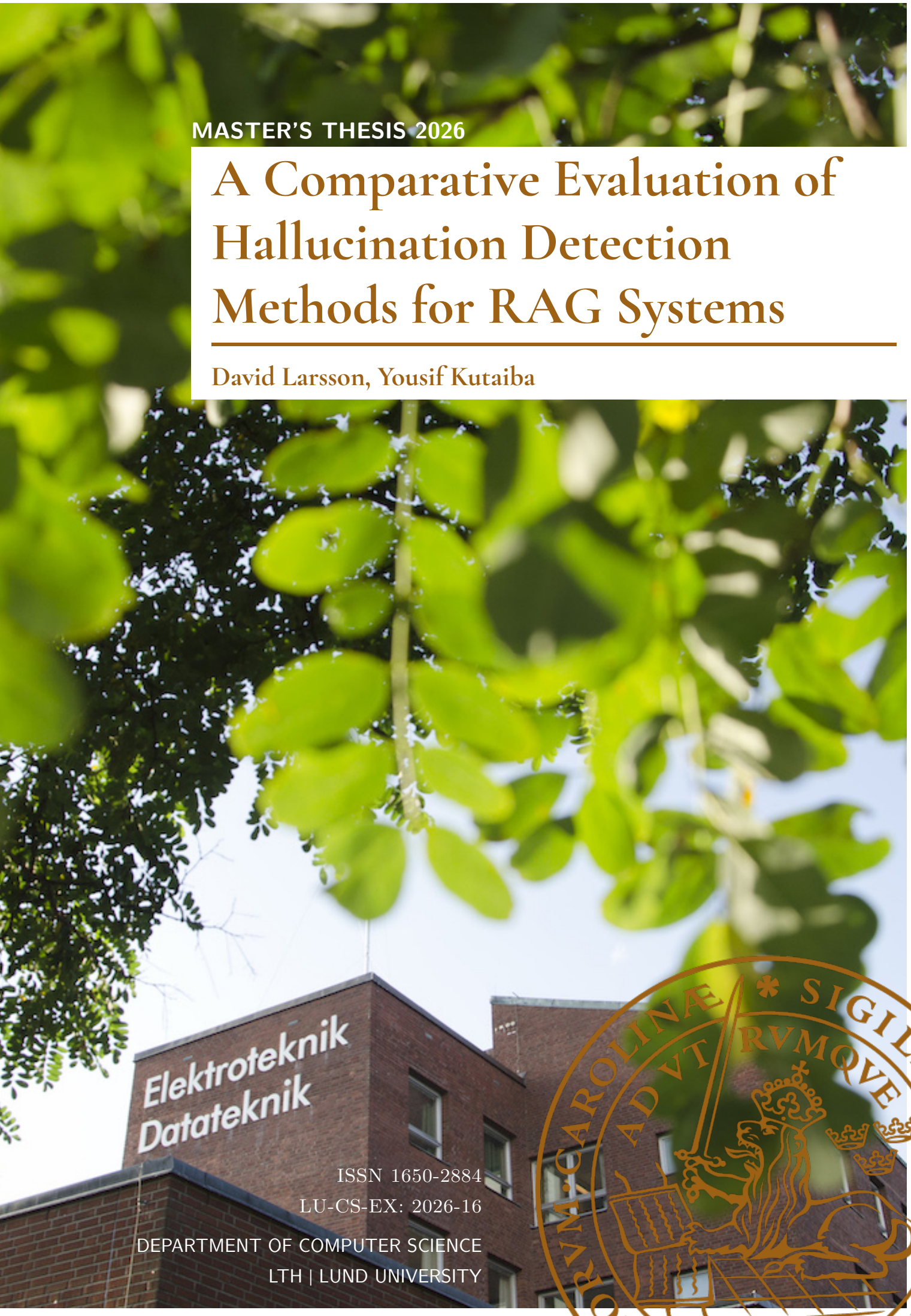


MASTER'S THESIS 2026

A Comparative Evaluation of Hallucination Detection Methods for RAG Systems

David Larsson, Yousif Kutaiba

ISSN 1650-2884
LU-CS-EX: 2026-16

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2026-16

A Comparative Evaluation of Hallucination Detection Methods for RAG Systems

En jämförande utvärdering av metoder för hallucinationsdetektion i RAG-system

David Larsson, Yousif Kutaiba

A Comparative Evaluation of Hallucination Detection Methods for RAG Systems

David Larsson
da7062la-s@student.lu.se

Yousif Kutaiba
yo8853ku-s@student.lu.se

June 16, 2026

LUND UNIVERSITY

Master's thesis work carried out at Robert Bosch AB.

Supervisors: Samir Jasarevic, Samir.Jasarevic@se.bosch.com
Marcus Klang, marcus.klang@cs.lth.se

Examiner: Xuan-Son Vu, xuan-son.vu@cs.lth.se

Abstract

Hallucination remains a challenge for large language models, occurring when a model produces a response that is coherent but factually incorrect or ungrounded. This study defines hallucination as any response that is not supported by a provided context, and evaluates several detection methods under this definition: LLM-as-a-Judge approaches, each tested across multiple underlying models, and specialized classifiers. All methods are benchmarked on the RAGTruth dataset, with performance measured using accuracy, execution time and monetary cost. Among the evaluated methods, a prompt-based evaluation approach combined with a large language model achieves the best detection performance (with respect to the AUROC metric) and demonstrates strong generalization across task domains and response lengths. In contrast, a lightweight classifier-based method achieves slightly lower detection performance but offers substantially reduced detection latency and cost. The two best-performing methods are combined into a pipeline configuration, which is then evaluated on a dataset of cybersecurity regulations and standards developed for this study. The pipeline achieves better performance than any of the best-performing detection methods used independently. Our results indicate that although LLM-as-a-Judge approaches achieve strong detection performance, encoder-based specialized classifiers can be significantly more efficient.

Keywords: Large Language Models, Retrieval-Augmented Generation, Hallucinations, Hallucination Detection, LLM-as-a-Judge, RAGAS, G-Eval, Lynx, LettuceDetect, HHEM, Cybersecurity Regulations and Standards, Cost–Performance Tradeoffs

Acknowledgements

We would like to express our sincere gratitude to Robert Bosch AB for providing an exciting thesis topic and for giving us the opportunity to carry out our master's thesis within their organization. We are especially thankful to our supervisor at Bosch, Samir Jasarevic, for his guidance, encouragement, and consistent support throughout the course of this project. We are equally grateful to our academic supervisor, Marcus Klang, for his valuable advice and thoughtful feedback on the content and structure of this report, as well as on the methodology used. We would also like to thank our examiner, Xuan-Son Vu, and the student reviewers for their feedback during the review process.

David Larsson & Yousif Kutaiba

Contents

Chapter 1

Introduction

Large language models (LLMs) are a type of artificial intelligence that processes and produces human language (Raza et al. 2025). They have demonstrated impressive performance in tasks such as language translation and text summarization (Raiaan et al. 2024).

Despite their progress, LLMs still struggle with hallucinations, leading them to produce unreliable content (Alansari and Luqman 2025) - sometimes with a high degree of certainty (H. Xu et al. 2025). Although Retrieval-Augmented Generation (RAG) offers a promising solution to this problem, LLMs may still favor their own embedded knowledge and generate ungrounded responses (Y. Gao et al. 2023; Kovács and Recski 2025; Ravi et al. 2024).

Hallucinations present a major obstacle to deploying LLMs in contexts where accuracy and reliability are essential. For instance, LLMs may generate inaccurate medical diagnoses, resulting in significant risks to patient safety (Y. Zhang et al. 2023). They may produce seemingly correct code that, while syntactically valid, can contain hidden flaws and security vulnerabilities (Twist et al. 2025). As a result, detecting hallucinated content has become an important research problem in the development of trustworthy language model systems.

A variety of approaches have been proposed for hallucination detection. Some methods rely on intrinsic signals from the language model itself, such as token probabilities (Alansari and Luqman 2025) or self-consistency measures (Manakul, Liusie, and Gales 2023), while others rely on external verification through retrieval systems (Alansari and Luqman 2025). Among the approaches proposed for hallucination detection, some use specialized classifiers designed to identify hallucinations at the token level (Kovács and Recski 2025), while others utilize fine-tuned large language models (Ravi et al. 2024). There has been a growing interest in using LLMs as evaluators for complex tasks (Gu et al. 2024). These methods leverage the reasoning capabilities of modern LLMs to evaluate generated text; however, like other hallucination detection approaches, they are not fully reliable (Alansari and Luqman 2025). Using an LLM

as a judge can sometimes produce variable assessments, as its evaluations may be influenced by prompt phrasing (Hwang et al. 2026), position bias that favors solutions based on their order in the prompt (L. Shi et al. 2025), and the model's inherent randomness (Schroeder and Wood-Doughty 2024).

Given the growing range of hallucination detection methods, it is important to systematically evaluate their performance and practical trade-offs. In real-world applications, detection methods must achieve high accuracy, rapid inference speed and cost efficiency. Furthermore, the choice of underlying models in LLM-as-a-Judge approaches, particularly between open-source/open-weight and closed-source LLMs, can significantly impact these trade-offs, since models vary in performance, accessibility and cost.

This thesis investigates multiple hallucination detection methods, systematically and empirically comparing their accuracy, cost and execution time. Execution time refers to the total time a detection method requires to produce a hallucination score for a given input. Cost denotes the monetary expense of running the detection methods, whether incurred through API usage fees or the resources required to run them locally. In particular, we evaluate LLM-as-a-Judge approaches like RAGAS and G-Eval using different underlying language models, including both open-source/open-weight and closed-source models. By analysing how the choice of detection method and underlying model affects performance, this work aims to provide a clearer understanding of the practical trade-offs involved in hallucination detection for LLM-based systems. Additionally, this thesis examines the encoder-only model LettuceDetect, the encoder–decoder model HHEM-2.1-Open, and the fine-tuned decoder-only hallucination detection model Lynx.

1.1 Problem Statement

As large language models become increasingly integrated into critical workflows, hallucination has emerged as a significant barrier to both trust and safe deployment. To the best of our knowledge, existing studies, including those by Kovács and Recski (2025), Ravi et al. (2024), Sardana (2025), and Valentin et al. (2024), evaluate detection performance on established benchmarks, but do not provide a systematic comparison that jointly considers execution time, monetary cost, and detection accuracy. Moreover, the effect of model selection in LLM-as-a-Judge approaches on this combination of factors has received little attention. Based on our review of the available literature, there is limited research on how hallucination detection methods perform in the context of cybersecurity regulations and standards.

1.2 Research Questions

This thesis is guided by the following research questions:

RQ1 How are hallucinations in RAG systems defined, categorized and evaluated in the literature, and how do these definitions influence which detection approaches are applicable?

RQ2 What methods have been established in the literature for detecting hallucinations in RAG systems?

RQ3 How do specialized hallucination classifiers compare to generative LLM-as-a-Judge frameworks in overall detection accuracy, and how does this performance vary across different task domains and generation characteristics?

RQ4 What are the trade-offs in latency, computational cost, and robustness (including threshold sensitivity and metric selection) across these approaches, and how does the choice of backbone LLM within LLM-as-a-Judge frameworks influence these trade-offs?

RQ5 How does a deployment pipeline derived from these findings perform when applied to a cybersecurity regulations and standards dataset?

RQ1 and RQ2 establish the conceptual and methodological foundations through the literature review, while RQ3–RQ5 are addressed empirically through the comparative evaluation and case study.

1.3 Related Work

Systematic evaluations of hallucination detection methods are well established in the literature. Sardana (2025) compares several approaches, including HHEM, Lynx, Prometheus, GPT-4o mini, and TLM, across six benchmarks using the AUROC metric. The results show that detection performance varies across domains, highlighting the importance of selecting methods based on task requirements.

Ravi et al. (2024) adopt accuracy as their evaluation metric to compare Lynx with LLM-as-a-Judge methods and RAGAS, finding that RAGAS performs worse than LLM-based evaluators. A similar pattern can be observed in Kovács and Recski (2025), in which LettuceDetect is benchmarked against RAGAS, a fine-tuned LLM, and Luna (an encoder-based model). The evaluation is carried out on the RAGTruth dataset, and the results show that LettuceDetect outperforms both the fine-tuned LLM and Luna in terms of F1 score.

Beyond raw detection performance, Valentin et al. (2024) investigate the cost-performance trade-off of hallucination detection methods and Tihanyi et al. (2024) create a multiple-choice benchmark for evaluating the cybersecurity knowledge of LLMs, drawn from sources such as NIST standards and RFCs.

1.4 Division of Work

Each author took on specific responsibilities corresponding to the thesis's distinct phases, ensuring a structured and balanced collaboration. David laid the theoretical foundations of the project by conducting background research and gathering relevant sources. David was also responsible for the cybersecurity case study, which involved improving and extending the Bosch cybersecurity compliance tool, integrating the hallucination detection methods into its pipeline, and designing and generating the domain-specific cybersecurity dataset. David authored Chapters 1 and 2 and contributed to the cybersecurity case study sections across Chapters 3–6. Yousif led the technical implementation and experimental analysis, covering the research, configuration, and execution of the hallucination detection methods both locally and through the Azure OpenAI API, as well as the design of the evaluation methodology, including dataset preparation, stratified sampling, and threshold calibration. Yousif authored Chapters 3–6. Throughout the project, both authors reviewed each other's work on an ongoing basis and exchanged constructive feedback.

1.5 The Use of AI

Generative AI was used as a supporting tool throughout this work. The models used were Claude Sonnet 4.6 and Gemini 3.0 Pro, both accessed through their respective browser-based chat interfaces. Google NotebookLM was additionally used for querying and cross-referencing information across multiple uploaded source documents.

These tools assisted with rephrasing, grammar correction, spell checking, and LaTeX formatting such as creating tables and figures. For the implementation of the hallucination detection methods, they were used to generate initial code based on existing reference implementations and documentation, which was then adapted and extended with further AI assistance to fit our experimental setup. They also supported the identification of relevant sources by efficiently scanning large volumes of literature and highlighting potentially relevant publications during the initial screening process.

All AI-generated output, including code and identified sources, was reviewed and validated to ensure correctness, relevance, and alignment with the research objectives of this thesis.

1.6 Scope

This thesis investigates automated methods for detecting hallucinations in large language models through a systematic evaluation conducted on two datasets: the well-established RAGTruth dataset and a domain-specific dataset of 91 samples covering cybersecurity regulations and standards. Through a structured comparative analysis, the study examines five existing detection methods, namely RAGAS, G-Eval, Lynx, LettuceDetect, and HHEM-2.1 Open, to evaluate their relative strengths and limitations in identifying hallucinated outputs. To further compare LLM-as-a-Judge approaches, four language models (GPT-5.2, GPT-4o

mini, Qwen3-32B and Gemma3-27B) are deployed as judges within the RAGAS and G-Eval evaluation frameworks.

1.6.1 Definition of Hallucination

The definition of hallucination adopted in this thesis is coupled with the definition provided by Ravi et al. (2024). For a given question x, a hallucination is defined as the case where a generated answer $P(C(x), x)$ is not supported by the retrieved context $C(x)$. Under this definition, hallucination is treated as a context-relative phenomenon: an answer $P(C(x), x)$ is considered faithful as long as it is grounded in $C(x)$, even if the retrieved context is irrelevant to the question x or factually incorrect with respect to x.

This framing is particularly relevant in RAG settings, where a language model is expected to generate answers that are grounded in a set of retrieved documents. In such settings, the model should prioritize closely following the retrieved context. A response that introduces unsupported claims, even plausible or technically accurate ones, undermines the reliability and trustworthiness of the system.

1.6.2 Out-of-Scope Aspects

The scope of this thesis is limited to the evaluation and comparison of existing hallucination detection methods, and does not include fine-tuning large language models. The study is further restricted to text-based hallucination detection. Topics that fall outside the scope of this work include multimodal hallucination detection and the mitigation or correction of hallucinations once detected.

1.6.3 Experimental Limitations

It should also be noted that all experiments are conducted using a limited set of models and evaluation data, due to constraints in computational resources, time and cost. The findings are therefore intended to provide comparative insights into the reliability and behavior of existing hallucination detection methods (including how certain methods perform when used with different LLM judges) rather than to establish general performance benchmarks across all available methods and models.

1.6.4 Execution Time and Measurement Constraints

The execution time of a hallucination detection method is defined as the total time elapsed from receiving input to producing a hallucination score or verdict. Frameworks such as G-Eval and RAGAS use an LLM-as-a-Judge approach and can be configured with different underlying language models. When these frameworks rely on models accessed through external APIs, execution time becomes an accumulated measure that reflects the combined latency of every API call made throughout the evaluation process. Since this report includes such frameworks, all execution time measurements inherently capture this accumulated API latency, along with any overhead introduced by the sequential or parallel calls the framework issues internally.

Variability arising from network conditions and server load is acknowledged as an inherent limitation of this setup. Measurements are therefore treated as approximate comparisons rather than definitive benchmarks. Beyond network factors, execution time is also significantly influenced by the hardware used to run local open-source and open-weight models, including the models that serve as the underlying components in frameworks such as RAGAS and G-Eval. On more capable hardware, such as high-end GPUs with large memory bandwidth, inference runs substantially faster. As a result, execution times reported for methods using locally hosted models should be interpreted in the context of our specific hardware setup, and any direct comparisons with API-based methods should be made with caution. To ensure a degree of consistency, all local model evaluations in this report are conducted on the same hardware environment, details of which are provided in Section 3.

Chapter 2

Background

This chapter provides the theoretical and technical foundations for the evaluation conducted in this thesis. It begins with an overview of large language models and the transformer-based architectures, introducing the four backbone models used in our experiments. The discussion then turns to retrieval-augmented generation and the RAGTruth benchmark dataset before examining hallucinations in detail, including their definition, causes, and the metrics used to quantify them. Finally, the chapter reviews the five hallucination detection methods selected for this study and the classification metrics used to compare their performance.

2.1 Large Language Models

Large Language Models (LLMs) are typically built using neural networks and can be trained to comprehend text, images, or audio (C. Chen et al. 2025; Kamath et al. 2025; Raiaan et al. 2024).

The datasets used to train large language models differ from one model to another, both in their composition and in volume. While some models are trained on only a few hundred gigabytes, others are trained on several terabytes of data. These datasets, combined with the use of deep learning, enable large language models to learn patterns and relationships (Raiaan et al. 2024).

Large language models can broadly be classified into three categories: open-source, hybrid and closed-source. Open-source models make configurations and weights publicly available, allowing users to inspect, fine-tune, and self-host them (Manchanda et al. 2024). Hybrid models, such as Qwen3-32B and Gemma3-27B, release trained parameters under permissive licenses, allowing practitioners to download, deploy, and adapt them locally, but typically withhold the training data and training code (Manchanda et al. 2024). One convenient way

to deploy these models locally is through Ollama, a tool designed for running large language models on local hardware (Ollama 2026). In contrast, closed-source models keep weights and training details private (Manchanda et al. 2024). Examples include GPT-5.2 and GPT-4o mini, which are accessible through the Azure OpenAI API and billed according to a token-based pricing model (Microsoft Azure 2026). Several closed-source models are additionally available through web interfaces, offering strong performance and managed infrastructure. The choice between open-source/hybrid and closed-source models generally involves trade-offs between performance, privacy, cost and customization.

The performance of a language model is influenced by several architectural and training factors: the model's depth (number of self-attention layers), its width (parameter size), and the volume of data it is trained on. Research shows that scaling up model width and training data, as well as extending training duration, can lead to meaningful performance gains. Increasing model depth can also lead to improvements, though this is not always guaranteed. For any specific task, fine-tuned models typically deliver the best results (Patil and Gudivada 2024).

2.2 Transformer-Based Architectures

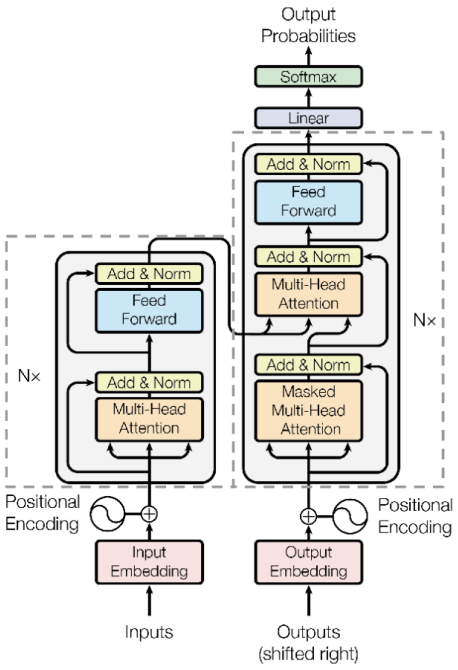


Figure 2.1: The transformer model. As illustrated, the encoder block is positioned on the left and is designed to process and represent an input sequence. The decoder block on the right facilitates the generation of the target output. The Figure is taken from Vaswani et al. (2017).

Most language models are built on the transformer architecture (Raiaan et al. 2024). As shown in Figure 2.1, the encoder-decoder architecture introduced by Vaswani et al. (2017) consists of an encoder and a decoder that work together, along with several supporting components.

The first step in text processing is tokenization, in which text is split into tokens. A token is a sequence of characters that typically corresponds to a whole word, a single character, or a subword, see Figure 2.2. The way text is split depends on the tokenizer used.

Figure 2.2: An illustration of tokenized text, with splits determined by the tokenizer in use.

Each token is converted into a numerical vector called an input embedding. Input embeddings are learned representations in which tokens with similar meanings are typically placed close to each other in a shared vector space. Positional embeddings are added to the input embeddings to represent the position of each token in a sequence. Together, these form the input to the encoder. A similar process is applied to the decoder's input, which is referred to as the output embedding (Raiaan et al. 2024).

At the core of the architecture is the attention mechanism, which allows the model to track dependencies and establish how tokens semantically relate to each other (Raiaan et al. 2024).

The encoder processes input through multiple self-attention and feed-forward layers, transforming it into a context-aware representation that the decoder can use in an encoder-decoder transformer architecture (Ansar et al. 2023; Raiaan et al. 2024). BERT is an encoder-only model built on the encoder block of the general transformer architecture (Devlin et al. 2019; Raiaan et al. 2024), and since encoder-only models produce one prediction per input, they are particularly well suited for classification tasks (Patil and Gudivada 2024). ModernBERT improves on BERT by delivering stronger performance on long-context inputs (Warner et al. 2025). These models rely on bidirectional attention, allowing each target token to attend to both its preceding and following tokens (Raiaan et al. 2024). They remain widely used because of their compact size and fast inference speed (Kovács and Recski 2025).

The primary function of the decoder is to generate a sequence of tokens, with each token conditioned on all preceding ones (Patil and Gudivada 2024; Raiaan et al. 2024).

A linear layer and SoftMax function convert the output from the decoder into a probability distribution over the model's vocabulary, from which a next token is selected or sampled using a decoding strategy (Raiaan et al. 2024).

2.2.1 Gemma3-27B

Gemma3-27B is a lightweight, open-weight language model developed by Google as part of the Gemma3 family. The model consists of 27 billion parameters, is built on a transformer-based decoder-only architecture and supports a large context window of 128K tokens (Kamath et al. 2025). The model is trained on 14 trillion tokens from a wide range of sources, such as web documents, code, and image data. The training data includes no information past its August 2024 knowledge cutoff date (Kamath et al. 2025; Google DeepMind 2025).

2.2.2 GPT-4o mini

The closed-source GPT-4o mini model, developed by OpenAI, is priced at $0.15 per one million input tokens and $0.60 per one million output tokens. The model supports a context window of up to 128K tokens and has a knowledge cutoff date of October 2023 (OpenAI 2024a). Similarly to GPT-4o, it is likely trained on a wide range of data, including publicly available internet sources and data obtained through collaborative partnerships with third parties (Hurst et al. 2024).

2.2.3 GPT-5.2

GPT-5.2 is a closed-source model series offered by OpenAI. GPT-5.2 is available in three versions: Instant, Thinking, and Pro. This project makes use of the Instant version, which is priced at $1.75 per one million input tokens and $14 per one million output tokens (OpenAI 2025a). GPT-5.2 supports a context window of 400K tokens and has a knowledge cutoff date of August 2025. The model is trained on publicly available internet sources and third-party data obtained through partnerships (OpenAI 2026; OpenAI 2025b).

2.2.4 Qwen3-32B

Qwen3 is a set of open-weight large language models developed by Alibaba Cloud. The series includes Qwen3-32B, a model with approximately 32 billion parameters that natively supports a 32K token context window, extendable to 128K tokens via YaRN, a technique for expanding the context window of transformer models (B. Peng et al. 2023). These models integrate two operating modes within a single architecture: a thinking mode for multi-step reasoning and a non-thinking mode for fast responses. They are trained on a large and diverse dataset of 36 trillion tokens, covering a wide range of content types, including reasoning tasks, books, and synthetic data (A. Yang et al. 2025).

2.2.5 Model Summary

Table 2.1 provides a summary of the language models used in this study. Of the four models, only Qwen3-32B features a built-in reasoning mode. Attempts to disable its built-in reasoning mode did not produce a measurable change in output behavior (see Section 3.4.6). The open-weight models were executed through Ollama, which provides Q4-K-M quantization as the default for these model sizes.

Table 2.1: Overview of the backbone language models used for LLM-as-a-Judge methods in this study.

Model	Provider	Parameters	Context	Cost (In / Out per 1M tokens)	Quantization	Reasoning
Closed-weight: Azure OpenAI API						
GPT-5.2 Instant	OpenAI	Undisclosed	400K	$1.75 / $14.00	Undisclosed	No
GPT-4o mini	OpenAI	Undisclosed	128K	$0.15 / $0.60	Undisclosed	No
Open-weight: Ollama						
Qwen3-32B	Alibaba	32.8B	32K	$0.00 / $0.00*	Q4_K_M	Yes (dual-mode)
Gemma3-27B	Google	27B	128K	$0.00 / $0.00*	Q4_K_M	No

* Local models incur no token-based API cost. Operational electricity costs are estimated separately in Section 3.5.2.

2.3 Retrieval-Augmented Generation

Large Language Models perform well across a wide range of tasks, but they have notable limitations, such as generating hallucinated content and relying on outdated knowledge. Retrieval-Augmented Generation (RAG) addresses this by providing LLMs with external knowledge sources (Y. Gao et al. 2023).

A standard RAG pipeline consists of three stages: indexing, retrieval, and generation. During indexing, documents are divided into chunks, encoded into vector representations using an embedding model, and stored in a vector database. When a user submits a query, the most relevant chunks are retrieved by comparing their vector representations to that of the query. The retrieved content is then combined with the query to generate a final response (Y. Gao et al. 2023).

The generation phase remains susceptible to hallucinations, where the model produces content that is not supported by the retrieved context (Y. Gao et al. 2023). Further difficulties arise when too much retrieved information is included, as redundant or long contexts can introduce noise and give rise to the "lost in the middle" problem, whereby important information positioned in the center of a long input tends to be overlooked (N. F. Liu et al. 2024). To address the limitations of RAG, advanced RAG methods have been proposed, with improved chunking strategies, metadata integration, reranking techniques, prompt compression, and iterative or recursive retrieval approaches (Y. Gao et al. 2023). In addition, a number of hallucination detection methods have been developed to identify cases where the generation phase of a RAG system produces content that is not grounded in the retrieved information. These methods are discussed in greater detail in Section 2.7.

2.4 The RAGTruth Dataset

RAGTruth is a benchmark dataset designed to evaluate hallucinations in RAG systems. The dataset contains 18K samples annotated at the span level across three tasks: question-answering, news summarization and data-to-text generation (Niu et al. 2024). A sample from the RAGTruth dataset can be seen in Table 2.2.

For the question answering task, data was drawn from the MS MARCO dataset (Bajaj et al. 2016). Questions were paired with contexts, and LLMs generated answers from the retrieved texts. For data-to-text generation, LLMs generated reviews for selected businesses from the Yelp Open Dataset (Yelp Inc. 2026). For news summarization, samples were randomly drawn from the CNN/Daily Mail training set (See, P. J. Liu, and Manning 2017), and LLMs generated summaries.

Each prompt in RAGTruth was submitted to six different LLMs, resulting in six independently annotated responses. Annotators reviewed every generated response and highlighted the specific spans that were unsupported by a given context. Each flagged span was also categorized by hallucination type, such as baseless information or contradiction (Niu et al. 2024). Only the presence of a hallucination in a response is of interest in this thesis, not its type.

Table 2.2: RAGTruth QA data sample regarding volcanic formations.

QUESTION	**How is a caldera different from a crater?**
CONTEXT	A large depression made because of volcanic activity is called a caldera. It is the result of a large cavity created underground when a chamber of magma and lava gets emptied. This cavity creates pressure and the over ground rocks collapse to create a... …
ANSWER	Based on the provided passages, a caldera is different from a crater in the following ways: 1. Formation: Calderas are formed as a result of the collapse of a large cavity created underground when a chamber of magma and lava emptie... …
ANNOTATION	**Labels:** evident conflict: 1, baseless info: 1 **Hallucination starts at:** 476 **Hallucination ends at:** 531 …

2.5 Hallucinations

LLMs can experience a phenomenon called hallucination, where they generate responses that sound convincing but are incorrect and unreliable (Alansari and Luqman 2025). These can be divided into factuality hallucinations, which arise when a model's output contradicts verifiable information or cannot be confirmed, and faithfulness hallucinations, which arise when a model's output strays from instructions, context, or internal consistency (L. Huang et al. 2025). While RAG systems leverage external knowledge to mitigate this issue, the risk of hallucinations still persists (L. Huang et al. 2025; Kovács and Recski 2025; Ravi et al. 2024).

2.5.1 Definition

The definition of hallucination adopted in this thesis is coupled with the definition provided by Ravi et al. (2024). For a given question x, a hallucination is defined as the case where a generated answer $P(C(x), x)$ is not supported by the retrieved context $C(x)$. Under this definition, hallucination is treated as a context-relative phenomenon: an answer $P(C(x), x)$ is considered faithful as long as it is grounded in $C(x)$, even if the retrieved context is irrelevant to the question x or factually incorrect with respect to x.

2.5.2 Causes

Hallucinations in large language models can be traced to multiple stages of the development pipeline rather than to any single point of failure, encompassing training data, pre-training, fine-tuning, model architecture and inference (Alansari and Luqman 2025).

2.5.2.1 Training data

Societal biases and outdated knowledge embedded in the training data may be reflected in model outputs resulting in hallucinations (Ferrara 2024; Mousavi, Alghisi, and Riccardi 2024). Similarly, misinformation, conflicting sources, and underrepresented data can give rise to factual inaccuracies and inconsistencies (Kandpal et al. 2023; S. Lin, Hilton, and Evans 2022; Y. Wang et al. 2023).

2.5.2.2 Pre-training

Shortcut learning, where models exploit superficial patterns such as negation words (Niven and Kao 2019) and keywords (Du et al. 2021) rather than acquiring robust features, weakens generalization to out-of-distribution data and can produce hallucinations during inference (Bihani and Rayz 2024). Additionally, in a teacher forcing setup, models are trained on ground-truth tokens, but during inference they condition on their own (potentially erroneous) outputs (Kang and Hashimoto 2020; C. Wang and Sennrich 2020). Without corrective feedback, early errors can propagate and amplify, compounding into hallucinations (M. Zhang et al. 2023).

2.5.2.3 Fine-tuning

Fine-tuning on datasets can lead to overfitting, which may reduce the model's ability to generalize. The model can become biased toward the fine-tuning data, making it more likely to hallucinate on unfamiliar inputs (Gekhman et al. 2024). In addition, aligning outputs with human preferences during fine-tuning can favor responses that are fluent and coherent rather than factually accurate (Alansari and Luqman 2025).

2.5.2.4 Architectural limitations

In long sequences, attention mechanisms may fail to prioritize relevant tokens, which can degrade factual precision and reasoning (B. Liu et al. 2023; X. Liu 2024). Likewise, the effectiveness of positional encodings tends to degrade across extended sequences, which can

cause the model to misinterpret token positions and produce hallucinations (Alansari and Luqman 2025; Banerjee, Agarwal, and Singla 2025). Maximum likelihood estimation (MLE) rewards probable tokens without penalizing factual errors (Welleck et al. 2019). In addition, the unidirectional nature of autoregressive processing prevents the model from integrating context from subsequent tokens, which can lead to hallucinations when inputs are ambiguous (L. Huang et al. 2025).

2.5.2.5 Inference

Ambiguous prompts can cause a model's response to reflect training patterns rather than the user's actual intent (Alansari and Luqman 2025; Rawte et al. 2023). Stochastic decoding strategies broaden output variation but may select low-probability tokens, potentially leading to factual errors (Dziri et al. 2021). The SoftMax function, which is used to calculate token probabilities, performs well when one token clearly dominates, but can have trouble distributing probability mass effectively when several equally valid continuations exist (Alansari and Luqman 2025; Z. Yang et al. 2017). Moreover, models often struggle with multi-hop inference and logical deduction, producing fabricated content rather than abstaining from an answer (Alansari and Luqman 2025; S. Zheng, J. Huang, and Chang 2023).

2.6 Hallucination Metrics

A variety of methods have been proposed to evaluate hallucination in large language models, ranging from traditional automatic metrics to human-centered approaches. These methods differ in what aspects of generated text they capture, with some focusing on surface-level similarity and others aiming to assess factual consistency. The following sections examine these evaluation strategies, their limitations, and the motivation for developing more reliable assessment frameworks.

2.6.1 Automatic Evaluation Metrics

Automatic metrics such as ROUGE, BLEU and BERTScore are commonly used for evaluating text generation systems, including large language models. However, these metrics primarily measure similarity between generated outputs and reference texts and therefore often fail to adequately capture factuality or faithfulness to external sources or input context (Tonmoy et al. 2024). ROUGE is a lexical overlap metric that computes n-gram and sequence-level matches between system outputs and reference texts (C.-Y. Lin 2004), while BLEU evaluates n-gram precision with a brevity penalty to measure correspondence between generated outputs and reference texts (Papineni et al. 2002). BERTScore extends beyond exact lexical matching by computing similarity between token embeddings of candidate and reference sentences (T. Zhang, Kishore, et al. 2019). Frameworks such as RAGAS and G-Eval use an LLM-as-a-Judge approach, where faithfulness is measured by whether generated statements are supported by a provided context rather than by lexical or embedding-based similarity.

2.6.2 Faithfulness as an Evaluation Criterion

Faithfulness is a metric that measures how factually consistent an answer from a language model is with a given context. The faithfulness score is typically a number $n \in [0, 1]$, with higher values indicating greater alignment with the provided context. Given a question x, an answer $P(C(x), x)$ is considered faithful to the context $C(x)$ only if every claim within the answer $P(C(x), x)$ is supported by the provided context $C(x)$ (Es et al. 2024).

2.6.3 Limitations of Lexical and Embedding Metrics

The fundamental limitation of lexical overlap metrics such as ROUGE and BLEU in the context of hallucination detection is that they measure surface-level textual similarity rather than semantic faithfulness (S. Liu, J. Ma, and Qu 2025; Y. Liu et al. 2023). A generated response can achieve a high ROUGE score by sharing many words and phrases with a reference text while still containing fabricated claims, and conversely, a faithful response that paraphrases the source material using different vocabulary may receive a low score despite being factually correct (Maynez et al. 2020). The issue arises because hallucination is a semantic property, concerned with whether a claim's meaning is supported by the source, while n-gram is a lexical metric that does not capture factual consistency (Janiak et al. 2025). Even embedding-based metrics such as BERTScore, which captures semantic similarity at the token level, have been shown to offer limited improvements over lexical metrics when it comes to reliably distinguishing hallucinated from faithful content (Janiak et al. 2025). These findings indicate that the shortcomings of traditional metrics are not simply due to insufficient detail, but rather reflect a deeper misalignment between what these metrics capture and what hallucination detection requires.

2.6.4 Human Evaluation and Benchmark Datasets

Given the limitations of automatic metrics, human evaluation is often considered the most reliable approach for assessing hallucination (Schmidtová et al. 2025). Benchmark datasets such as RAGTruth (Niu et al. 2024) and HaluEval (J. Li et al. 2023) rely on human annotators to label hallucinated content, and such annotations are typically used as the ground truth against which methods are evaluated. However, human annotation introduces its own challenges. The process is expensive, time-consuming, and difficult to scale, particularly in specialized domains, such as medicine or law where domain expertise is required (Ji et al. 2023). Furthermore, inter-annotator agreement can vary considerably depending on the subtlety of the hallucination and the clarity of the annotation guidelines (T. Zhang, Ladhak, et al. 2024). These scalability and consistency constraints are a primary motivation for the development of automated detection frameworks, which aim to approximate human-level assessment at significantly lower cost and higher throughput (Cao et al. 2025).

2.7 Hallucination Detection Methods

Approaches to hallucination detection work at different levels of detail, from methods that judge whether an entire response contains hallucinations, to more fine-grained analyses per-

formed at the token level (Z. Sun et al. 2025).

The definition of hallucination adopted in a given study directly influences which detection methods are applicable. Because this thesis defines hallucination as a failure of faithfulness to the retrieved context, the focus is on whether the generated answer can be supported by a provided context. The relevant detection approaches are those that compare the generated response against source material. This includes encoder-based classifiers that evaluate semantic alignment between context and response and LLM-as-a-Judge frameworks that prompt a language model to verify whether claims are supported by a provided context.

By contrast, a factuality-oriented definition would require methods capable of verifying claims against world knowledge, such as external knowledge base lookup or self-consistency checks (L. Huang et al. 2025; Manakul, Liusie, and Gales 2023), which falls outside the scope of this work. Similarly, methods based on internal model uncertainty, such as token probability analysis (Alansari and Luqman 2025) or semantic entropy (Farquhar et al. 2024), address a different aspect of the hallucination problem by focusing on whether the model itself is uncertain about its output rather than on whether the output is grounded in a specific source. The faithfulness-based definition established in Section 2.6.2 determines the pool of applicable detection methods to those that evaluate the relationship between a generated response and its source context.

The following subsections present the detection methods considered in this thesis, all of which operate within the mentioned scope, though they differ in their output format and the post-processing required to derive a comparable faithfulness score, see Section 3.4.

2.7.1 RAGAS

RAGAS is an evaluation framework designed for RAG systems. The framework provides a set of metrics that independently measure the quality of the retrieval and generation stages. While RAGAS includes several metrics such as context relevancy, answer relevancy, and faithfulness, this thesis focuses exclusively on the faithfulness metric (Es et al. 2024).

According to Es et al. (2024), faithfulness refers to the extent to which a generated answer is grounded in a provided context, and an answer is considered faithful if all the claims it contains can be derived from that context. RAGAS calculates the faithfulness score using a large language model in two stages. An answer is first divided into shorter statements, each representing a single claim. Every statement is then compared against the context to determine whether it is supported. A faithfulness score is calculated as the number of supported statements divided by the total number of extracted statements (Es et al. 2024).

2.7.2 G-Eval

G-Eval is a prompt-based framework that adopts an LLM-as-a-Judge approach to evaluate language model outputs. It consists of three components: a manually crafted prompt that defines the evaluation task and an evaluation criterion, against which outputs are evaluated; a chain-of-thought reasoning step; and a scoring mechanism (Y. Liu et al. 2023).

Chain-of-thought (CoT) is a sequence of intermediate reasoning steps generated by a large

language model to improve accuracy before producing a final answer. The G-Eval framework is capable of generating such steps automatically using an LLM, but in this thesis, they were instead created manually (Y. Liu et al. 2023).

An evaluation task, an evaluation criterion, a chain-of-thought reasoning, an input context and a generated answer are combined into a single prompt, and an LLM is asked to assign a score based on a specified criterion and a predefined set of scores. A final score is calculated as the probability-weighted average of all possible scores defined in the prompt. The token probabilities for each score can be obtained through the model's API for supported models, or estimated via Monte Carlo sampling (Y. Liu et al. 2023).

G-Eval accompanies the final score with a justification for its reasoning. The proposed scoring function is designed to overcome the limitations of direct discrete scoring by large language models, particularly the tendency toward score clustering with low variance and the preference for integer outputs, even when decimal values are requested. Rather than relying on a single selected value, the method computes a final score by weighting each candidate score by its predicted probability which captures the model's uncertainty and allows for a more nuanced evaluation (Y. Liu et al. 2023).

Furthermore, LLMs are notably sensitive to small prompt variations, often producing very different responses when faced with minor changes in spelling, wording, or prompt structure (Chatterjee et al. 2024). As a result, within G-Eval, both the wording of task descriptions and evaluation criteria, and the formulation of the evaluation steps, can have a considerable influence on the scores it produces.

2.7.3 Lynx-8B

Lynx is an open-source decoder-only large language model that has been fine-tuned for hallucination detection. Its primary purpose is to serve as a reference-free metric for automated RAG evaluation. The model is available in two versions: a 70B parameter variant based on Llama-3-70B-Instruct and an 8B parameter variant based on Llama-3-8B-Instruct (Meta AI 2024; Ravi et al. 2024). This thesis uses the 8B version.

Lynx is trained on 2400 samples, each consisting of four components: a question, an answer, context, and a label. The samples are drawn from multiple question answering datasets, including RAGTruth (see Section 2.4), DROP (which is an English comprehension benchmark for paragraph-level reasoning (Dua et al. 2019)), CovidQA (which consists of annotated question–answer pairs based on COVID-19 scientific articles (Möller et al. 2020)), and PubMedQA (which is a biomedical dataset from PubMed abstracts where questions are answered with yes/no/maybe (Jin et al. 2019)). These datasets cover a wide range of domains and this variety ensures the model can generalize across different RAG applications.

Lynx is trained to produce both reasoning chains and evaluation scores, improving the transparency and interpretability of its outputs. The output of Lynx is a binary verdict: `pass` if the answer is grounded in and faithful to the provided context, or `fail` if it is not. This verdict is accompanied by a written justification explaining the reasoning behind the decision (Ravi et al. 2024).

2.7.4 HHEM-2.1

Developed by Vectara and based on Google's T5 encoder-decoder model, HHEM-2.1-Open (Hughes Hallucination Evaluation Model 2.1 Open) is an open-weight classifier designed for hallucination detection. While the model is technically capable of handling arbitrarily long sequences, its performance varies as sequence length increases (Oplatka and Mendelevitch 2025; Raffel et al. 2020). The model has 110M parameters and is sufficiently lightweight to be executed on standard hardware, including consumer-grade CPUs and GPUs (Tamber et al. 2025; Mendelevitch et al. 2024).

HHEM evaluates the degree of factual consistency between a hypothesis and a premise by assigning a probabilistic score, where lower values indicate a greater probability of inconsistency and higher values indicate stronger factual agreement (Sardana 2025). This score is derived through the computation of semantic similarity between the premise and hypothesis (C. Zhang, H. Wang, and Meng 2025).

In practice, the premise is constructed by combining the source context with the input question, while the hypothesis corresponds to the generated response (Sardana 2025). Examples of inputs and outputs are illustrated in Tables 2.3 and 2.4.

HHEM is trained on RAGTruth, among other undisclosed datasets (Tamber et al. 2025). The standard HHEM model evaluates a generated response as a single unit, assigning one consistency score; as a consequence, if the majority of sentences are factually accurate, the score will remain high (C. Zhang, H. Wang, and Meng 2025).

Table 2.3: Skåne entails that the person is in Sweden, resulting in a relatively high score.

PREMISE	The person lives in Skåne
HYPOTHESIS	The person lives in Sweden
HHEM-SCORE	0.7181

Table 2.4: Sweden does not entail that the person is in Skåne, resulting in a relatively low score.

PREMISE	The person lives in Sweden
HYPOTHESIS	The person lives in Skåne
HHEM-SCORE	0.1180

2.7.5 LettuceDetect

LettuceDetect is a hallucination detection model that performs token-level analysis to flag unsupported claims in generated text, given a question, context, and an answer. Using an encoder-only architecture, it eliminates the need for the autoregressive decoding step used in generative models, allowing faster inference while remaining highly effective for classification tasks. LettuceDetect is available in two model sizes: a 396M parameter version (`lettucedetect-large-v1`) and a 150M parameter version (`lettucedetect-base-v1`)

(Kovács and Recski 2025). This thesis uses the 396M parameter model. The relatively compact size, combined with the inherent inference efficiency of encoder-only models, makes LettuceDetect well-suited for deployment on standard consumer hardware without significant computational overhead.

LettuceDetect is built on the encoder-only ModernBERT architecture and is fine-tuned as a token classifier using samples drawn from the RAGTruth dataset (Kovács and Recski 2025). A description of RAGTruth is provided in Section 2.4.

Token-level hallucination probabilities are computed, and contiguous tokens with scores exceeding 0.5 are grouped into hallucination spans (Kovács and Recski 2025). Table 2.5 shows the output from LettuceDetect for a given question-context-answer triplet.

Table 2.5: Example output produced by LettuceDetect for a question–context–answer triplet. The output includes the start and end indices (character positions within the answer) of the detected hallucinated span, a confidence score (indicating the likelihood that the span is hallucinated, with values ranging from 0 to 1), and the identified hallucinated span in the answer.

QUESTION	Which regulation governs personal data protection in the EU? What year did it enter into force?
CONTEXT	The General Data Protection Regulation (GDPR) governs personal data protection in the European Union. GDPR entered into force in 2018.
ANSWER	The General Data Protection Regulation (GDPR) governs personal data protection in the European Union. GDPR entered into force in 2020.
PREDICTION	`[{'start': 101, 'end': 134, 'confidence': 0.9687, 'text': ' GDPR entered into force in 2020.'}]`

2.7.6 Comparison

RAGAS, G-Eval, Lynx, HHEM, and LettuceDetect are all hallucination detection methods that can produce a score for a given question-context-answer triplet, making them well-suited for use in a RAG setting. HHEM, Lynx, and LettuceDetect are specialized models trained to identify hallucinations, whereas G-Eval and RAGAS take a more general, evaluation-driven approach instead of relying on a single specialized classifier. Regarding reasoning strategies, G-Eval uses chain-of-thought reasoning to improve predictions and Lynx is fine-tuned on synthetic reasoning chains for the same purpose. The methods also differ in their underlying architectures: HHEM uses an encoder-decoder design, LettuceDetect relies on an encoder-only architecture, and Lynx uses a decoder-only architecture.

2.7.6.1 Granularity

HHEM operates primarily at the sequence level, generating a single probability score that indicates whether a given answer is supported by its source context. Lynx and RAGAS take a more granular approach, evaluating individual claims within a response. LettuceDetect advances this further by performing token-level classification, enabling precise identification of unsupported spans within a response. G-Eval offers flexible, prompt-defined granularity that can be applied against any specified criterion.

2.7.6.2 Output Generation

The hallucination detection methods examined differ in how they compute their scores and in the type of output they produce.

RAGAS uses a multi-step, LLM-based pipeline to produce its faithfulness score. It first uses an LLM to extract individual statements from a given question–answer pair, then issues a second prompt to determine which of those statements are faithful to the provided context. The final score is calculated by dividing the number of faithful statements by the total number of statements in the answer. In addition to the numeric score, RAGAS returns a verdict and an accompanying reason for each extracted statement, resulting in a relatively transparent and interpretable output.

G-Eval also adopts an LLM-as-a-Judge approach, but differs from RAGAS in that the entire evaluation is carried out with a single prompt. The question, context, answer, evaluation criterion, evaluation task, and chain-of-thought are submitted together, and a model returns a score and a justification. Rather than relying on a raw integer output, G-Eval computes its final score as a probability-weighted value derived from the token probabilities of a model's response.

The methods further diverge in the explanatory information they provide alongside their scores. RAGAS, G-Eval and Lynx provide a justification for their verdicts. LettuceDetect and HHEM, by contrast, offer no such reasoning. As direct classification models rather than autoregressive text generators, neither is capable of producing reasoning chains. LettuceDetect does, however, maintain a degree of interpretability by identifying the specific tokens it considers to be hallucinated. HHEM provides only a numeric score with no accompanying explanation.

2.7.6.3 Model Size

HHEM 2.1 has 110 million parameters. LettuceDetect has 396 million parameters in the version used in this study. These compact sizes make both models well-suited and lightweight enough to run on standard everyday hardware. The backbone models used for G-Eval and RAGAS can vary; in this thesis, GPT-5.2, GPT-4o mini, Qwen3 (32B parameters), and Gemma3 (27B parameters) are used. The parameter counts for GPT-5.2 and GPT-4o mini have not been publicly disclosed.

2.7.6.4 Training Data

LettuceDetect and HHEM are fine-tuned on the RAGTruth dataset. Lynx is fine-tuned on datasets across multiple domains, including RAGTruth, DROP, CovidQA, and PubMedQA. The GPT models are trained on internet, third-party, and user-provided data. Gemma3-27B and Qwen3-32B are trained on diverse datasets that include web content.

2.7.6.5 Execution Time

From an end-to-end performance perspective, RAGAS is expected to be the slowest method, as it requires two separate prompts to be processed sequentially before a final score can be produced. G-Eval and Lynx follow, since they use a single prompt. RAGAS, G-Eval and Lynx rely on an LLM that generates tokens. HHEM and LettuceDetect are expected to be the fastest, as they do not require time for token generation.

2.7.6.6 Cost

Regarding monetary cost, HHEM and LettuceDetect are expected to be the least expensive options, as both models are relatively small in size and are run locally. RAGAS, G-Eval and Lynx, on the other hand, rely on an LLM-as-a-Judge approach and are therefore expected to be the most costly. This is because LLMs require considerable computational resources, including high memory usage and specialized hardware, or alternatively access to paid API services. RAGAS is expected to be the most expensive option, as it requires a higher number of LLM calls per evaluation to generate a score.

2.8 Performance Evaluation Metrics

As noted by Terven et al. (2025), in a binary classification setting, each prediction produced by a model falls into one of four distinct categories, determined by the relationship between the predicted label and the true label:

- **True Positive (TP):** The model predicts the positive class and the true label is positive. Thus, a positive instance is correctly identified.
- **False Positive (FP):** The model predicts the positive class and the true label is negative. In this case, a negative instance is incorrectly classified as positive.
- **True Negative (TN):** The model predicts the negative class and the true label is negative. Consequently, a negative instance is correctly identified.
- **False Negative (FN):** The model predicts the negative class and the true label is positive. As a result, a positive instance is incorrectly classified as negative.

Together, these four outcomes contribute to the confusion matrix, as seen in Figure 2.3.

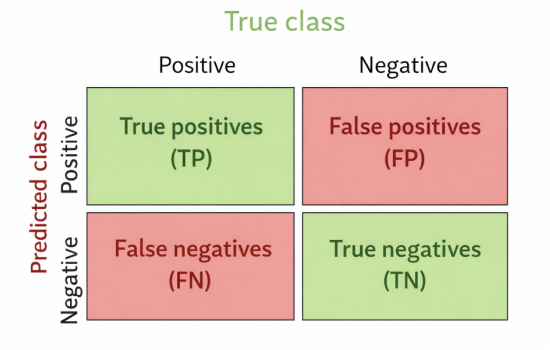


Figure 2.3: The confusion matrix.

2.8.1 Precision

Precision is the fraction of correctly classified positive samples out of all the samples that were classified as positive (Terven et al. 2025):

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

2.8.2 Recall

Recall is the fraction of correctly classified positive samples out of all positive samples (Terven et al. 2025):

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

2.8.3 F1-score

The F1-score integrates precision and recall into a single measure of model performance (Terven et al. 2025):

$$\text{F1} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

2.8.4 Matthews Correlation Coefficient

Matthews Correlation Coefficient (MCC) measures binary classifier performance by considering all four confusion matrix outcomes, making it well-suited for skewed datasets (Matthews 1975; Sujon et al. 2025):

$$\text{MCC} = \frac{\text{TP} \times \text{TN} - \text{FP} \times \text{FN}}{\sqrt{(\text{TP} + \text{FP})(\text{TP} + \text{FN})(\text{TN} + \text{FP})(\text{TN} + \text{FN})}}$$

2.8.5 ROC Curve & AUROC

A ROC (Receiver Operating Characteristic) curve plots recall against the false positive rate (Manning, Raghavan, and Schütze 2008), where the false positive rate is calculated as:

$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}}$$

The Area Under the ROC Curve (AUROC/AUC) is a value between 0 and 1, where a random classifier produces a diagonal reference line corresponding to an AUROC of 0.5, and a perfect classifier achieves an AUROC of 1. Although AUROC provides a useful single-value summary for comparing classifiers, it can mask differences in specific regions of ROC space. As illustrated in Figure 2.4, classifier B has a higher AUROC overall, but classifier A achieves better performance where the false positive rate exceeds 0.6 (Fawcett 2006).

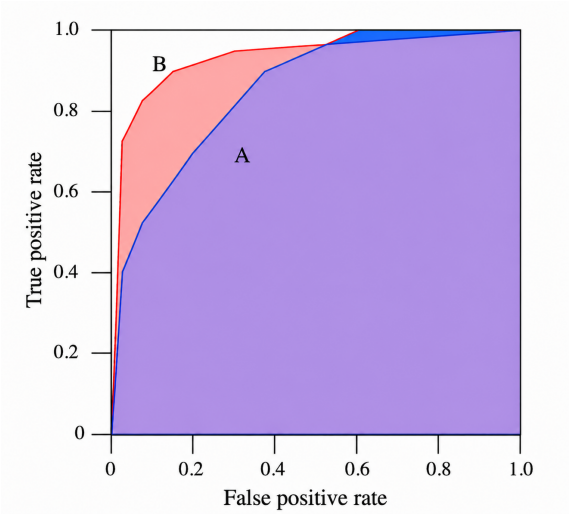


Figure 2.4: ROC curves for two classifiers.

Chapter 3

Methodology

This chapter describes the experimental methodology used to evaluate and compare the hallucination detection methods examined in this thesis. Figure 3.1 provides an overview of the complete experimental pipeline. The chapter begins with the technical environment and infrastructure used for both local and API-based evaluations, followed by the dataset preparation process, including stratified sampling and bootstrap convergence analysis for sample size selection. It then details the selection of the five evaluation methods and four backbone language models, and documents the implementation of each method, including the normalization steps required to produce comparable faithfulness score. The performance metrics and operational cost metrics used for comparison are defined, and the chapter concludes with the construction and annotation of the domain-specific cybersecurity dataset used in the case study.

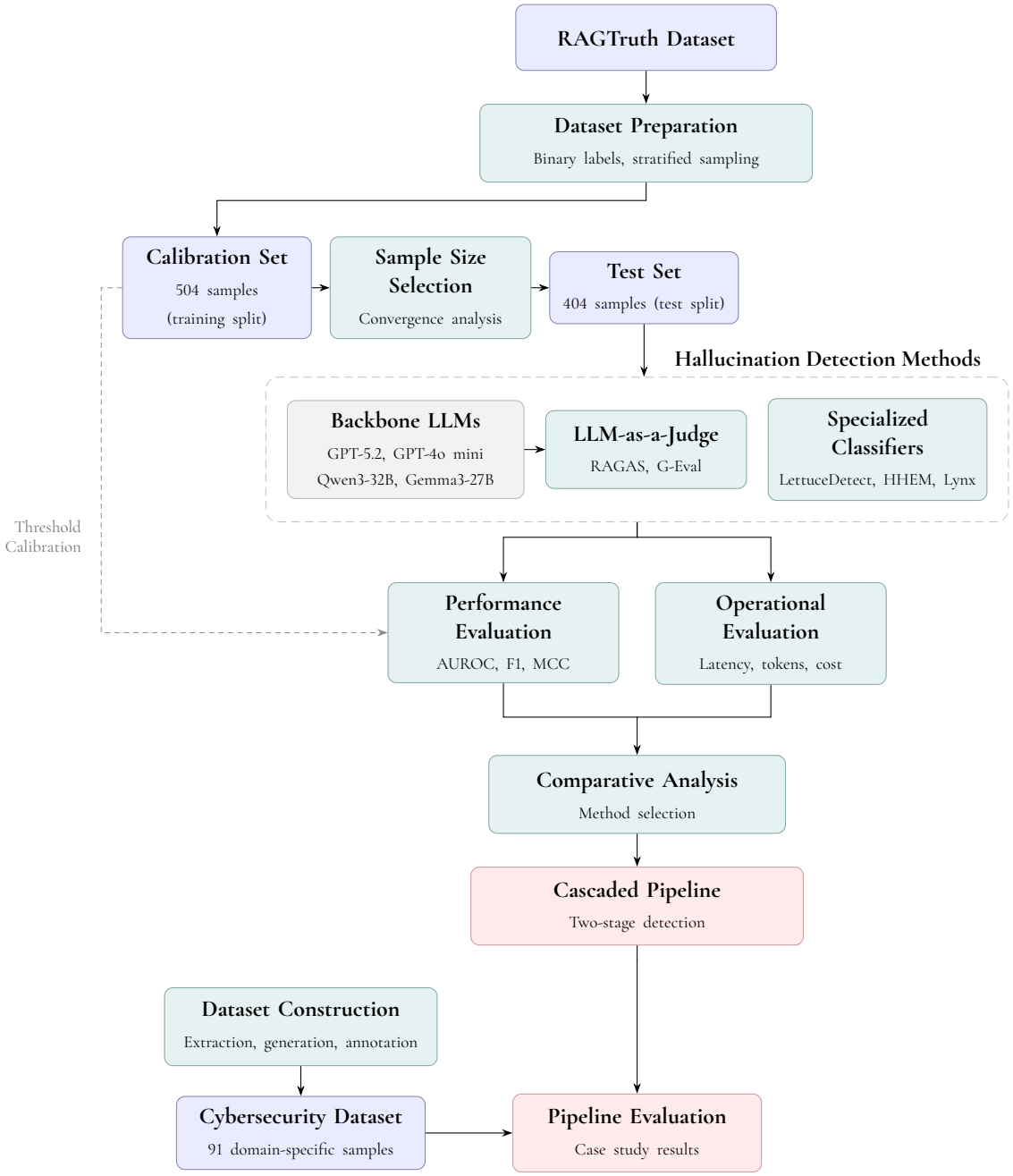


Figure 3.1: Overview of the experimental methodology. The pipeline starts with the RAGTruth dataset, which is preprocessed and split into a calibration set (drawn from the training split) used for both bootstrap convergence analysis to determine the test set sample size and for threshold derivation, and a test set (drawn from the test split) for evaluation. Eleven method configurations, comprising LLM-as-a-Judge frameworks (each with four backbone LLMs) and three specialized classifiers, are evaluated on both performance and operational metrics. The comparative analysis informs the design of a two-stage cascaded pipeline, which is subsequently evaluated on a separately constructed domain-specific cybersecurity dataset.

3.1 Technical Environment & Infrastructure

The local open-weight models: Qwen3-32B (Alibaba Cloud and Qwen Team 2025) and Gemma3-27B (Farabet and Warkentin 2025) were executed locally on a dedicated workstation, while API-based evaluations were performed through Azure OpenAI (Microsoft 2026). The complete hardware and software specifications are provided in Table 3.1.

Table 3.1: Experimental Hardware, Software, and Model Environment.

Category	Component	Specification / Version
Hardware (Local)	CPU	AMD Ryzen 7 9800X3D (8-Core)
	GPU	NVIDIA GeForce RTX 5090 AORUS MASTER ICE
	Memory	64 GB RAM
	Storage	4TB Samsung SSD 990 PRO
	PSU	Corsair HX1200i (1200W, 80+ Platinum, ATX 3.1)
Software	Operating System	Microsoft Windows 11 Pro
	Python Version	Python 3.13.12
	Key Frameworks	RAGAS==0.4.3, openai==2.21.0, torch==2.12.0, transformers==5.2.0, numpy==2.4.2, scipy==1.17.0, scikit-learn==1.8.0, lettucedetect==0.1.8, ollama==0.6.1
API Models (Cloud)	Provider	Azure OpenAI Service
	Evaluator Models	GPT-5.2, GPT-4o mini
Local Models	Serving Engine	Ollama
	Evaluator Models	Qwen3-32B, Gemma3-27B

3.2 Dataset Preparation

Prior to evaluating the hallucination detection methods, the raw RAGTruth dataset was processed to create a suitable evaluation set. This included converting RAGTruth's hallucination categories into binary labels, stratifying the data by task type, label, and output length to ensure balanced representation, and determining an appropriate sample size that balanced statistical stability with computational cost.

3.2.1 Ground Truth Definition

The creators of the RAGTruth dataset classified hallucinations into four categories: evident conflict, subtle conflict, evident baseless information, and subtle baseless information (Niu et al. 2024). The processed version of the dataset used in this study consolidates these into two categories: conflict-related information, where the response contradicts the provided context, and baseless information, where the response introduces unsupported claims not grounded in the context (Weights & Biases 2024). In our binary classification experiments, we treat these human annotations as the ground truth. Specifically, any response containing

either conflict-related or baseless information is assigned a ground truth label of 0 (hallucinated), while responses that contain neither type of hallucination are assigned a label of 1 (faithful).

3.2.2 Preprocessing

To reduce potential sampling bias in the evaluation, we applied a balanced stratified sampling technique to the RAGTruth dataset. Random sampling of the corpus could introduce systematic biases, such as overrepresentation of specific task types or shorter responses, potentially inflating performance estimates. To mitigate this, the test data was partitioned into three intersecting dimensions: task type, ground-truth label, and output length.

First, the dataset was evenly divided among the three primary generation tasks: summary, question answering (QA), and data-to-text. Within each task, we enforced a strict, uniform class distribution, maintaining a 1:1 ratio between hallucinated samples (containing evident conflict or baseless information) and faithful samples.

Additionally, because the evaluated methods rely heavily on the generated output in their judgments, response length was treated as an important factor. Therefore, outputs were discretized into three categories: short, medium, and long. Word count was chosen over token count as the stratification unit because tokenization depends on the model being used (Ali et al. 2024). Since the detection methods evaluated in this study use different tokenization schemes, the same response can produce different token counts depending on the model processing it. Word count, defined here as the number of whitespace-separated words in a generated answer, provides a consistent measure that can be applied fairly across all evaluated methods. The specific word count ranges for each category, as well as their corresponding percentiles, are shown in Table 3.2.

Table 3.2: Answer length buckets for the RAGTruth dataset based on word-count percentiles computed.

Length Category	Percentile Range	Word Count Range
Short	0–33%	7–102
Medium	33–66%	102–142
Long	66–100%	142–333

Combining these three dimensions yielded 18 distinct strata (3 task types × 2 labels × 3 length categories), with the number of available samples per stratum reported in Table 3.3. An equal number of instances N were drawn from each stratum, guaranteeing that the final test set was fully balanced across task type, factual accuracy, and response length. This balancing is structural, however, and does not account for the semantic content of the sampled data (see Section 5.6.3). The value of N was determined via the convergence analysis described in Section 3.2.3.

Table 3.3: Balanced stratified sampling design across task types, labels, and output lengths, showing the number of available samples in each stratum.

Task Type	Label	Output Length	Available Samples
QA	Hallucinated	Short	40
		Medium	39
		Long	81
	Faithful	Short	391
		Medium	193
		Long	156
Data-to-Text	Hallucinated	Short	26
		Medium	254
		Long	299
	Faithful	Short	13
		Medium	148
		Long	160
Summary	Hallucinated	Short	97
		Medium	60
		Long	47
	Faithful	Short	353
		Medium	198
		Long	145

3.2.3 Sample Size Selection

While the full RAGTruth dataset encompasses approximately 18,000 samples, evaluating all combinations of models and methods across the entire test subset was unfeasible due to the financial costs of API calls and the computational overhead of local inference. To determine an appropriate N per stratum that ensures statistical stability without expending unnecessary resources, we performed a bootstrap-based sample-size convergence analysis (Efron and Tibshirani 1993).

We evaluated all candidate detection methods using the RAGTruth training set. For each candidate sample size, ranging from $N = 2$ to $N = 28$ samples per stratum (i.e., $S = 36$ to $S = 504$ total samples), we generated 1,000 bootstrap resamples with replacement and computed the AUROC for each resample. The width of the resulting 95% confidence interval served as the convergence metric.

At each value of N, samples were drawn in stratified fashion, ensuring that every resample maintained a balanced distribution across all 18 strata. This guaranteed that the convergence analysis remained independent of sample-type bias throughout the process. As illustrated in Figure 3.2, the 95% CI width exhibited severe variance at small sample sizes ($N < 5$), and continued to decrease as N increased. The majority of methods fell below the 0.1 convergence threshold shortly after $N = 20$, with Lynx exhibiting the slowest convergence and LettuceDe-

tect the fastest. It should be noted that both methods were trained on the RAGTruth dataset, though with different scope: LettuceDetect on the full training set (Kovács and Recski 2025) and Lynx on only the QA subset (Ravi et al. 2024). The implications of this training data overlap are discussed in Sections 5.2.1 and 5.6.1.

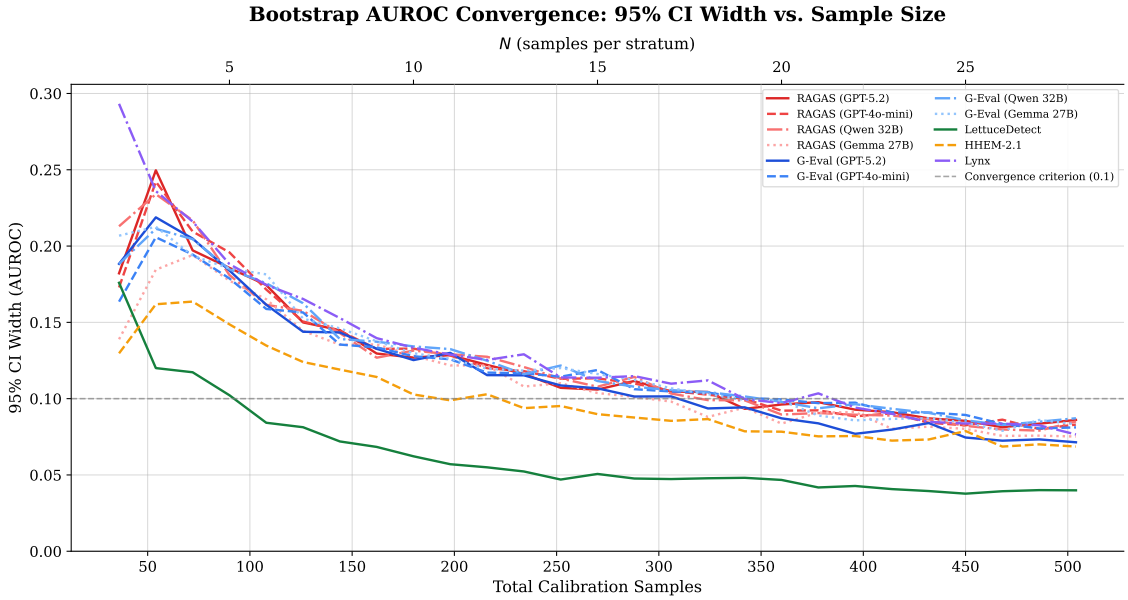


Figure 3.2: Bootstrap AUROC convergence analysis across all candidate hallucination detection methods. The 95% CI width from 1,000 bootstrap resamples at each value of $N \in [2, 28]$ is plotted against total sample size S and samples per stratum N. The dashed line marks the adopted convergence criterion of 0.1, below which the majority of methods fall shortly after $N = 20$.

Given the financial and computational constraints of running these evaluation methods on large datasets, we adopted a convergence criterion of 0.1. This threshold is further supported by evidence of diminishing returns, as the width decreases by only approximately 0.02 between $N = 20$ and $N = 25$, indicating that a larger sample size would yield limited additional benefit. At $N = 23$, all strata met this convergence criterion, and we therefore adopted this as the per-stratum sample size. While a uniform $N = 23$ across all 18 strata would result in $S = 414$ total samples, the stratum (Data-to-Text, Faithful, Short) contained only 13 available samples, as seen in Table 3.3. Consequently, the final evaluated sample size was $S = 404$ samples.

3.3 Selection of Evaluation Methods

Table 3.4 summarizes all candidate methods considered for this study, including those that were ultimately excluded. The following subsections detail the rationale behind each selection decision.

Table 3.4: Comparison of candidate hallucination methods, highlighting model architecture and scoring methodology.

Method	Model Type	Scoring Approach
	Selected	
RAGAS	Instruction-tuned LLM	Extracts atomic statements from the answer via prompting, then verifies each against the context. Score is the ratio of supported statements (0–1).
G-Eval	Instruction-tuned LLM	Single CoT prompt evaluates faithfulness on a 1–5 scale. Final score is a probability-weighted average of output token logits, normalized to 0–1.
Lynx-8B	Meta-Llama-3-8B-Instruct	Generates a reasoning chain followed by a binary pass/-fail verdict. Score derived from the verdict token probability (0–1).
HHEM-2.1	T5-base (encoder-decoder)	Direct sequence classification. Outputs a continuous consistency score (0–1) without text generation.
LettuceDetect	ModernBERT (encoder-only)	Token-level binary classification. Continuous hallucination tokens are grouped into spans, each assigned a confidence score. The faithfulness score (0–1) is derived from the highest-confidence span.
	Excluded	
TruLens	Instruction-tuned LLM	Splits answer into sentences and checks each against the context via prompting. Methodology overlaps with RAGAS.
Prometheus 2	Mistral-7B / Mixtral-8x7B	Rubric-based evaluation via fine-tuned LLM. Outputs a 1–5 discrete score with reasoning.
ARES	DeBERTa-v3-Large	LLM (FLAN-T5 XXL) generates synthetic training data to fine-tune a DeBERTa classifier. Applies Prediction-Powered Inference (PPI) for confidence intervals.

3.3.1 Selected Methods

The selection of the five methods: RAGAS (Es et al. 2024), G-Eval (Y. Liu et al. 2023), LettuceDetect (Kovács and Recski 2025), Lynx (Ravi et al. 2024), and HHEM-2.1 (M. Li, Luo, and Mendelevitch 2024) was primarily motivated by the need to compare fundamentally different evaluation strategies. As established in Section 2.7, the faithfulness-based definition adopted in this thesis restricts the pool of applicable methods to those that evaluate the relationship between a generated response and its source context. Within this pool, the selection was designed to cover a range of distinct approaches: prompt-based LLM-as-a-Judge methods (RAGAS and G-Eval), fine-tuned generative evaluators (Lynx), lightweight encoder-based classifiers (LettuceDetect), and a task-specific encoder-decoder (HHEM-2.1). Those two constraints, the faithfulness scope and the black-box evaluation setting, resulted in a smaller pool of viable candidates than initially expected.

3.3.2 Excluded Methods

Several other methods were also considered during the selection process. In particular, TruLens (Datta et al. 2022) and Prometheus 2 (Kim et al. 2024) appeared promising at first and were briefly implemented and tested to determine whether they should be included in the study. However, TruLens operates using an evaluation methodology very similar to RAGAS. At the time of writing (March 24th, 2026), the RAGAS paper (Es et al. 2024) had accumulated 1,216 citations on Google Scholar, compared to 12 for TruLens (Datta et al. 2022), and the RAGAS repository (VibrantLabs 2024) had approximately 12,900 GitHub stars, compared to approximately 3,200 for TruLens (Snowflake Inc. 2024). Given that RAGAS currently has wider adoption and a stronger academic presence for this specific type of evaluation, we selected it over TruLens to avoid redundancy.

Similarly, Prometheus 2 utilizes a fine-tuned evaluator model, placing it in the same architectural category as Lynx. In a preliminary comparison on a subset of the RAGTruth training set, Prometheus 2 yielded the lowest AUROC among all candidate methods. Combined with the fact that it did not offer a novel methodology compared to Lynx, which also relies on a fine-tuned model of comparable size, Prometheus 2 was excluded from the final list.

Another promising candidate was ARES (Saad-Falcon et al. 2024). This framework is notable for its unique methodology, in which an LLM is used to generate synthetic data for fine-tuning smaller models, enabling them to perform evaluation tasks at significantly lower cost. In addition, ARES incorporates a technique called Prediction-Powered Inference (PPI), which calculates statistical confidence intervals to help correct potential biases and prediction errors produced by automated judges.

However, ARES operates under a fundamentally different evaluation paradigm compared to the other methods in this study. Rather than producing per-sample faithfulness predictions, it produces a single aggregated estimate with a confidence interval for the entire dataset through PPI. While including a methodologically distinct framework would be valuable in a comparative study, ARES cannot be evaluated using the same metrics or experimental design (see Section 3.5) as the other selected methods. A meaningful inclusion of ARES would require a separate evaluation design with distinct metrics, which represents a different research question beyond the scope of this thesis.

3.3.3 Backbone Language Models

The four instruction-tuned backbone models (GPT-5.2, GPT-4o mini, Qwen3-32B, and Gemma3-27B) were selected on the basis of performance, cost, and hardware constraints (see Table 2.1 for full specifications, including per-token pricing). For the API-based models, we paired the more expensive, more capable GPT-5.2 Instant with the cheaper GPT-4o mini, letting us test whether GPT-5.2's higher token cost is justified by improved detection performance.

For the open-weight models, we selected Qwen3-32B and Gemma3-27B, the largest models from two distinct families (Alibaba and Google) that could run locally on the RTX 5090 GPU described in Section 3.1, under the default Ollama Q4_K_M quantization. The resulting selection spans three model families and lets us assess whether API access is worth its cost when open-weight alternatives exist.

3.4 Implementation of Evaluation Methods

All method outputs were logged in a unified JSON schema to enable a consistent and reproducible comparison. Each entry captures the RAG input triple (`query`, `context`, `answer`), stratification metadata (task type, hallucination label, answer length), operational measurements (execution time, token counts, and estimated API cost), and the method-specific evaluation results stored under a `metrics` field. A truncated example of a single evaluated sample of G-Eval with GPT-4o mini is shown in Listing 3.1.

Listing 3.1: Truncated example of a single evaluated sample of G-Eval with GPT-4o mini logged using the standardized JSON output schema

```
{
  "id": "1174",
  "bucket": "('Summary', False, 'short')",
  "model": "GPT-4o mini",
  "query": "Summarize the following news within 95 words:",
  "context": "The American Pharmacists Association is ...",
  "answer": "Sure! Here's the summary within 95 words: ...",
  "execution_time_seconds": 4.707,
  "token_usage": {
    "input_tokens": 914,
    "output_tokens": 68,
    "total_tokens": 982
  },
  "estimated_cost_usd": 0.0001779,
  "metrics": {
    "faithfulness": 0.999,
    "faithfulness_reason": "Reasoning: The output accurately ...",
    "score_probabilities": {"1": 0.0, ..., "4": 1.8e-07, "5": 0.999}
  },
  "ground_truth": 1
}
```

The `metrics` field is method-specific. For G-Eval, it contains the continuous faithfulness score, the model's reasoning chain, and the token-level score probabilities used to compute the weighted score. Other methods store different outputs. RAGAS records extracted statements, their individual verdicts, and the reasoning behind each verdict. Lynx stores its reasoning chain together with the binary pass/fail token and the associated probability. LettuceDetect preserves the hallucinated spans along with their confidence scores and character-level positions. Lastly, HHEM-2.1 records only a continuous faithfulness score.

3.4.1 RAGAS

RAGAS (Es et al. 2024) was implemented using version 0.4.3 of the official library provided by the authors (VibrantLabs 2024). Because RAGAS uses an LLM-as-a-Judge approach, the specific model serving as the judge significantly affects the evaluation results. For this rea-

son, we evaluated RAGAS using the four models: GPT-5.2, GPT-4o mini, Qwen3-32B, and Gemma3-27B described in Section 3.3.3. This selection allows us to obtain a comprehensive estimate of the framework's capabilities by comparing cloud-based models with open-weight models from different families. It also allows us to observe how models with different capability levels (e.g., GPT-4o mini vs. GPT-5.2) perform on the same task. For all configurations, we set the model temperature to 0 to maximize deterministic outputs. The data was formatted for the framework by structuring the context extracted from the RAGTruth dataset as a list of text chunks.

3.4.2 G-Eval

Initially, we attempted to implement G-Eval (Y. Liu et al. 2023) using the DeepEval framework due to its operational simplicity (Ip and Vongthongsri 2026). However, DeepEval presented compatibility limitations with the open-weight models running via Ollama, as it was unable to extract the log-probabilities of the final generated tokens, which is necessary to calculate G-Eval's weighted probability score.

For this reason, we implemented G-Eval manually, following the original authors' methodology. This manual approach allowed us to accurately compute the weighted score and extract the top five token probabilities. The retrieved context was concatenated into a single continuous string before being inserted into the evaluation prompt, rather than being passed as discrete chunks. Similar to RAGAS, we evaluated G-Eval using the four models (GPT-5.2, GPT-4o mini, Qwen3-32B, and Gemma3-27B) outlined in Section 3.3.3.

The framework's native 1-to-5 faithfulness score was normalized to a 0-to-1 scale to ensure alignment with the other methods. Alongside this score, we also saved the LLM's generated reasoning to document how it reached its final verdict. For all models, the temperature was set to 0 to maximize determinism. The maximum token limit was initially set to 500, but Qwen3-32B occasionally produced empty reasoning fields and missing final scores at this limit. To resolve this, we incrementally raised the limit in steps of 500 tokens, testing at each increment until the empty reasoning outputs ceased. This occurred at 2,500 tokens, which was adopted as the standard limit across all models for the remainder of the study.

3.4.2.1 Prompt Refinement

Initial prompt and evaluation. As G-Eval's performance is highly dependent on its instructions, we used Google's Gemini 3.0 Pro to generate an initial evaluation system prompt. This prompt was tested on a stratified random subset of 50 samples drawn from the RAGTruth training set. A manual review of the resulting scores and reasoning chains revealed that the prompt acted as an overly strict evaluator, with valid paraphrasing and minor wording differences often penalized by 1–2 points, and the presence of a single minor factual error typically leading to deductions of 3–4 points, severely downgrading the overall score.

Revisions and final selection. To address this, we iteratively refined the prompt with Gemini's assistance over five rounds, introducing explicit severity tiers that tied deductions to error type rather than to the mere presence of an error. Specifically, the evaluator was instructed to apply heavy deductions for severe hallucinations such as fabricated claims or direct contradictions, lighter deductions for minor hallucinations that did not substantially

alter meaning, and no deductions for responses that preserved the source's meaning despite differences in wording. After each revision, the 50-sample subset was re-evaluated and cases where scores had changed were manually inspected. However, across the later rounds we observed a consistent limitation: the more forcefully we instructed the model to differentiate between severity levels, the more its behavior regressed toward uniformly strict evaluation. The third iteration produced the closest approximation to the intended tiered behavior, with the fewest cases of paraphrasing-based penalization, and was therefore adopted as the final prompt (shown in Appendix A) and used as the standard across all models.

Bias. We acknowledge two sources of bias in this process. First, the final prompt was shaped by our manual judgment of what constituted a fair evaluation, and a different reviewer might have made different calibration choices. Second, although we use four different backbone models for G-Eval, prompt refinement was carried out exclusively on Gemma3-27B due to financial, computational, and time constraints, as it could be run locally without API charges and offered faster inference than Qwen3-32B. Consequently, we cannot rule out that model-specific interactions with the prompt may introduce systematic differences across evaluators.

3.4.3 Lynx-8B

Based on the available methodological descriptions, Lynx (Ravi et al. 2024) appears to be functionally similar to G-Eval. Both approaches process a system prompt and generate an intermediate reasoning chain before producing a final verdict. The main differences appear to lie in their underlying architecture and evaluation prompt design. Lynx uses a fine-tuned Llama-3 model, whereas G-Eval relies on general-purpose LLM prompting. Due to local hardware limitations (see Section 3.1), running the 70B parameter version was not feasible, so we implemented Lynx using the 8B model. The model was deployed locally via Ollama, which provides only a Q4_K_M-quantized GGUF variant of Lynx (`tensortempla r/patronus-lynx:8b-instruct-q4_k_m`), derived from the full-precision checkpoint `PatronusAI/Llama-3-Patronus-Lynx-8B-Instruct`. This quantization reduced weight precision from 16-bit to approximately 4-5 bits per parameter, which may affect evaluation outputs compared to the full-precision model.

The evaluation prompt used in our experiments was identical to the one established by the original authors (Ravi et al. 2024), and is shown in Appendix B. As with G-Eval, the retrieved contexts were provided to the model as a single concatenated string. Where the two methods differ more substantially is in their output format. G-Eval produces a discrete score between 1 and 5, whereas Lynx outputs a binary verdict: `pass` or `fail` at the conclusion of its reasoning. To align this output with the continuous 0–1 faithfulness scale used by our other evaluation frameworks, we extracted the generation probability of the final verdict token and mapped it to a faithfulness score F as follows:

$$F = \begin{cases} p_{\text{token}} & \text{if the predicted token is } \texttt{pass} \\ 1 - p_{\text{token}} & \text{if the predicted token is } \texttt{fail} \end{cases}$$

where p_{token} denotes the generation probability of the predicted verdict token. For example, a 99% probability of `pass` yields a faithfulness score of 0.99, while a 99% probability of

`fail` yields a score of 0.01. This mapping ensures that confident `pass` predictions correspond to high faithfulness, while confident `fail` predictions correspond to low faithfulness, indicating hallucination.

3.4.4 HHEM-2.1

HHEM-2.1 approaches hallucination detection as a direct sequence classification task (Mendelevitch et al. 2024). For this study, we implemented Vectara's open-source HHEM-2.1-Open model (`vectara/hhem-2.1-open`) locally using the Hugging Face `transformers` library (Wolf et al. 2020).

As HHEM-2.1 is explicitly fine-tuned to evaluate faithfulness without relying on generative prompts, the data mapping process was straightforward. The query, reference context, and generated answer were combined and passed directly into the classification model. Consequently, the model outputs a single continuous faithfulness score without generating intermediate text or requiring token-probability normalizations (M. Li, Luo, and Mendelevitch 2024). To accurately monitor the computational overhead of this localized approach, we integrated the `google/flan-t5-base` tokenizer to explicitly calculate the input token lengths for each sample (Chung et al. 2024).

3.4.5 LettuceDetect

To implement LettuceDetect, we utilized version 0.1.8 of the official package provided by the authors (Kovács and Recski 2025). The large variant of the model (`lettucedetect-large-v1`) was chosen instead of the base version, as our local hardware could support it (see Section 3.1). Unlike generative LLM-as-a-Judge frameworks such as RAGAS or G-Eval, LettuceDetect performs evaluation through token classification using an encoder-only model. As the model does not generate text, explainability is instead provided through span extraction, which identifies and isolates specific word sequences within a generated answer that are classified as unsupported by the provided context.

Similar to HHEM-2.1, the query, reference context, and generated answer were concatenated and passed directly into the detector. For each evaluated sample, LettuceDetect identifies potential hallucination spans and assigns each span a confidence score $p_i \in [0, 1]$.

As LettuceDetect does not define a single faithfulness score at the sample-level, we implemented a post-processing step to derive such a score. Specifically, we aggregated the model outputs by selecting the maximum confidence score across all detected hallucination spans within a sample. While an average over all spans could alternatively have been used, the maximum operator was more appropriate for the objective of this study, as a single high-confidence score is sufficient to indicate hallucination, whereas averaging could obscure such detections in cases where most spans receive low confidence scores. The continuous faithfulness score was computed as:

$$F = 1.0 - \max_i(p_i)$$

where p_i denotes the confidence score assigned to the i-th flagged hallucination span. If no hallucinations were detected, the score would be defined as $F = 1.0$.

This formulation ensured that the presence of even a single hallucination with high confidence resulted in a correspondingly low faithfulness score. For example, in a sample containing two detected hallucination spans with confidence scores of $p_1 = 0.98$ and $p_2 = 0.73$, the resulting faithfulness score would be $F = 0.02$, since it is determined by p_1, the higher-confidence detection.

3.4.6 Model-Specific Observations

Observation. During implementation, we observed a model-specific behavior that occurred across multiple framework configurations. Qwen3-32B (Alibaba Cloud and Qwen Team 2025) consistently produced substantially more output tokens than the other models under equivalent settings. For example, Qwen3-32B generated approximately 1,500 more output tokens on average using RAGAS, and 500 more using G-Eval, compared to Gemma3-27B (See Figure 4.7, Section 4.3.1). Because Qwen3-32B is a hybrid model that combines both a reasoning and a non-reasoning mode (see Section 2.2.4), we investigated whether this behavior was caused by the model defaulting to its internal reasoning mode, which is distinct from the prompt-level chain-of-thought reasoning used by G-Eval.

Attempted solution. To examine this, we attempted to disable the model's built-in reasoning capabilities through two approaches. First, we created a custom Modelfile that included the `/no_think` directive in the system prompt, which is intended to instruct Qwen3 to bypass its internal chain-of-thought process. Second, we passed `extra_body="think":False` via the `ChatOllama` request parameters to set the Ollama API's `think` flag to false. We reran a subset of 18 evaluation samples through both the RAGAS and G-Eval pipelines with both methods active simultaneously, and observed no measurable reduction in output tokens compared to the baseline configuration. We also ran a controlled comparison on a fixed prompt with `temperature=0`, toggling the `think` flag between `True` and `False`. Both configurations produced outputs of identical length. We additionally observed that Qwen3-32B does not separate its reasoning into a dedicated channel (such as a `thinking` field in the API response or `<think>` tags in the content). Instead, the step-by-step reasoning is generated inline as part of the main response.

Interpretation. Taken together, these observations suggest that neither the Ollama API `think` flag nor the `/no_think` system-prompt directive measurably reduced Qwen3-32B's output length in our evaluation setup. We interpret this as evidence that the elevated token consumption reflects the model's default inline generation style rather than a reasoning mode that can be toggled off through these mechanisms. It is unclear whether these flags reached the model but had no effect, or whether they failed to propagate through the LangChain-Ollama integration. In either case, the practical implication for our evaluation remains the same.

3.5 Performance Metrics and Evaluation Criteria

We evaluated the methods: RAGAS (Es et al. 2024), G-Eval (Y. Liu et al. 2023), Lynx (Ravi et al. 2024), HHEM-2.1 (M. Li, Luo, and Mendelevitch 2024), and LettuceDetect (Kovács and Recski 2025) along multiple dimensions, including overall discriminative performance (measured using both threshold-independent and threshold-dependent metrics), stratified performance across conditions, and the computational and financial cost of deployment.

3.5.1 Primary Performance Metrics

The primary evaluation metric for comparing the methods is AUROC. Because each method produces a continuous faithfulness score (typically between 0.0 and 1.0) while the RAGTruth ground-truth labels are strictly binary (1 for faithful, 0 for hallucination), AUROC is particularly well-suited to this setting, as it quantifies a classifier's ability to discriminate between two classes across all possible thresholds without requiring a predefined threshold (Fawcett 2006).

Beyond computing AUROC over the full evaluation set, we also computed it independently across task types and output lengths. This allowed us to isolate variables and determine whether a method's detection performance was significantly influenced by the underlying generation task or whether its accuracy degraded as the evaluated text grew longer.

In addition to AUROC, we computed the F1-score to determine the optimal classification threshold for each method (Christen, Hand, and Kirielle 2023; Rijsbergen 1979). While AUROC indicates a method's overall ability to separate classes, real-world deployment requires a specific cutoff point to make a final pass-or-fail decision. To find this, the F1-score was calculated across a range of possible thresholds. Specifically, every unique faithfulness score produced by a method was treated as a potential threshold. The threshold that yielded the highest F1-score was selected and recorded as the optimal cutoff point for that model.

While maximizing the F1-score identifies the optimal operational threshold for prioritizing the positive class (faithful samples), the MCC (Matthews 1975) was also maximized independently. MCC accounts for all four quadrants of the confusion matrix, ensuring that true negatives (hallucinated samples) are weighted equally. Calculating its optimal threshold provides a secondary metric that offers a more balanced evaluation of overall classification performance (Chicco and Jurman 2020).

Since computing F1 and MCC requires a fixed classification threshold, and selecting this threshold on the same data used for evaluation would artificially inflate (i.e., overestimate) performance estimates, we derived the optimal threshold from the RAGTruth training set. The training set comprised 504 samples ($N = 28$ per stratum), the same set utilized in the convergence analysis (Section 3.2.3). To quantify threshold stability, we generated 1,000 bootstrap resamples with replacement and computed the optimal F1 and MCC thresholds for each resample. These calibrated thresholds were then applied to the held-out test set to obtain the final F1 and MCC scores.

To complement these summary metrics, we plotted score distribution histograms for each framework, providing visual context for the threshold decision and revealing classification behaviors that single metrics cannot capture.

3.5.2 Operational Efficiency and Cost Metrics

In addition to performance metrics, we also measured execution time, token usage, and cost (see Listing 3.1, Section 3.4). This allowed us to examine each method's efficiency and operational cost and compare them alongside their performance. As both input and output tokens were recorded for each sample, the cost for the API-based models could be calculated by multiplying the token usage by the official OpenAI API pricing (OpenAI 2024b). This applied to all frameworks that used GPT-4o mini and GPT-5.2.

For the locally executed models, we estimated the cost based on electricity consumption. We used HWiNFO to measure the average CPU and GPU power draw during inference for each method (HWiNFO 2026). Since HWiNFO only reports these two components, the measured total underestimates actual system-level consumption. To obtain a conservative upper bound, we applied a 1.5 × multiplier, following the rationale that CPU and GPU account for the large majority of power consumption during inference (Hodak, Gorkovenko, and Dholakia 2019), and the remaining components (RAM, cooling, storage, motherboard) together with typical PSU inefficiency (80–90%) (CLEAResult 2024) should not exceed 50% of the measured draw. Table 3.5 presents the measured and adjusted power consumption for each method.

Table 3.5: Average power consumption during inference for locally executed methods. CPU and GPU values were measured using HWiNFO. The adjusted total applies a 1.5× multiplier to provide a conservative upper-bound estimate that accounts for unmeasured components and power supply inefficiency.

Method	CPU (W)	GPU (W)	Measured Total (W)	Adjusted Total (W)
RAGAS (Qwen3-32B)	68	306	374	561
RAGAS (Gemma3-27B)	46	470	516	774
G-Eval (Qwen3-32B)	63	209	272	408
G-Eval (Gemma3-27B)	50	223	273	410
LettuceDetect	76	377	453	680
Lynx	50	162	212	318
HHEM-2.1	105	55	160	240

The electricity cost per sample was then derived using:

$$C_{\text{sample}} = \frac{P_{\text{adj}} \times t}{3,600,000} \times r \times e$$

where P_{adj} is the adjusted power in watts, t is the execution time in seconds, r is the average electricity rate €/kWh for March 2026 (ENTSO-E 2026), and e is the average EUR to USD

exchange rate for March 2026 (European Central Bank 2026). The denominator converts watt-seconds to kilowatt-hours.

3.6 Case study: Cybersecurity Regulations & Standards

A security baseline is a structured set of minimum cybersecurity requirements and controls that an organization establishes to ensure a consistent level of protection across its systems. It is designed to reduce security risks such as data breaches, unauthorized access, and exploitation of known vulnerabilities by enforcing a common security standard. A security baseline typically includes requirements derived from standards such as NIST SP 800 series and from applicable regulations such as the Cyber Resilience Act (Olifer et al. 2019).

At Bosch, a software tool is currently under development that leverages generative AI to automatically construct such a baseline from regulatory documents. This is a process that, when performed manually by a Bosch cybersecurity manager, is estimated to take several weeks. The tool brings together relevant requirements into a unified baseline with mapped controls and implementation guidance. Since the tool generates outputs based on text retrieved from cybersecurity documents, it is essential that responses remain grounded in the provided source material.

The objective of the case study was to construct a domain-specific dataset for evaluating hallucination detection methods in the context of cybersecurity regulations and standards, where factual accuracy and traceability to the source documents are critical.

3.6.1 Dataset Generation

The objective of the dataset generation task was to create samples that consisted of a question, context, and an answer, in accordance with the format required by the hallucination detection methods described in Section 2.7. The dataset was built from seven source documents, which include standards used internally at Bosch and publicly available regulations such as Regulation (EU) 2016/679[1] (GDPR), Regulation (EU) 2022/30[2] and Regulation (EU) 2023/2854[3].

The dataset generation process began with the manual extraction of coherent text segments from each source document, resulting in a total of twenty unique context chunks. Questions and answers were then produced for each chunk. Initially, this involved manually creating appropriate questions along with two types of answers: answers that were faithful to the source context, and answers that were deliberately hallucinated with regard to the source context.

Hallucinated answers were created by introducing patterns such as fabricated details, including mentioning dates in the answer that were not present in the context; overgeneralizations

[1] http://data.europa.eu/eli/reg/2016/679/oj
[2] http://data.europa.eu/eli/reg_del/2022/30/oj
[3] https://eur-lex.europa.eu/eli/reg/2023/2854/oj

of what the context states (for instance when the context refers to internet-connected products but the answer refers to all digital products); contradictions to the context; swapping of regulatory names in the answer; overinferences, where the answer extends beyond what the context addresses; and semantic drift, where words from the context are reused but with subtly altered meanings.

After twenty samples had been produced manually, the process was partially automated using an AI agent based on GPT-5.2, which generated questions as well as faithful and hallucinated answers for a given context chunk. To generate hallucinated answers, the agent received a randomly selected prompt from a predefined set of prompts (see Appendix C), with each prompt targeting one of the hallucination types established during manual sample creation. Despite this automation, the dataset was limited to eighty samples due to the time required for manual review and labeling. The choice of GPT-5.2 was determined by the models made available internally at Bosch.

3.6.2 Dataset Labeling

To establish a reliable ground truth, each sample was manually and individually labeled by us. A label followed a binary classification scheme, categorizing each answer as faithful or hallucinated based on its relationship to its provided context. Under a strict closed-world assumption, an answer was marked as hallucinated if it contained claims that were not supported by the context. Statements that were factually accurate in general but not supported by the context were classified as hallucinations in order to prioritize context grounding.

The reliability of this process was measured using percent agreement, which quantifies the extent to which identical labels are produced for the same samples. The percent agreement is calculated as:

$$\text{Percent Agreement} = \frac{\text{Number of agreements}}{\text{Total number of annotations}} \times 100$$

Using this formula, an inter-annotator agreement score of 89% was achieved, indicating a high degree of consistency between our labeling decisions. All disagreements were resolved through discussion until a consensus was reached for each sample.

3.6.3 Dataset Preprocessing

Nine candidate samples were removed based on predefined quality criteria across two categories: trivial hallucinations and non-informative samples. After this filtering step, the final dataset consisted of 91 samples, of which 45 were classified as faithful and 46 were classified as hallucinated. For these remaining samples, the context contained on average 119 words, the question 21 words, and the answer 45 words.

Including trivial or unambiguous samples could bias the evaluation by inflating performance, as the model might produce correct outputs without genuinely engaging in the reasoning or uncertainty that hallucination detection requires. By removing these overly obvious cases, the dataset better reflects challenging, realistic scenarios where distinguishing between accurate

and hallucinated information is not straightforward. One example of a trivial hallucination is an answer consisting entirely of a list of digital components that are not mentioned in the provided context.

Non-informative samples were also removed: cases in which the answer merely restates the question without conveying additional information. Such samples provide no meaningful test of hallucination detection, since the answer can be verified against the question alone without reference to the source context. For example, the question "Which Union essential requirements in Article 3(3)(d), (e) and (f) are mentioned as ensuring network protection, safeguards for personal data and privacy, and protection from fraud?" was paired with the answer "The Union essential requirements mentioned in Article 3(3)(d), (e) and (f) are those intended to ensure: (1) network protection, (2) safeguards for the protection of personal data and privacy, and (3) protection from fraud". These types of samples were a consequence of the automated sample generation approach.

3.6.4 Method Selection

The specific methods evaluated on this dataset, and the thresholds applied to them, are determined by the comparative analysis in Chapter 4 and described together with the case-study results in Section 4.5.

Chapter 4

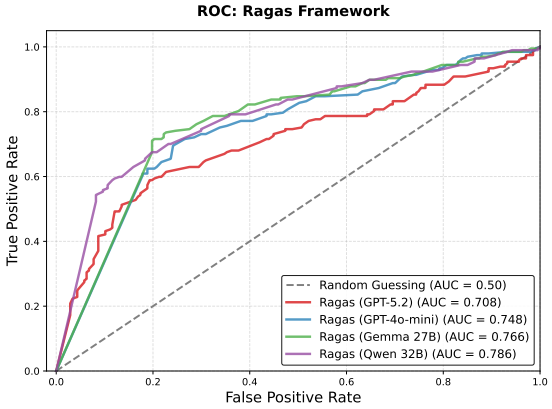
Results

This chapter presents the experimental results. The evaluation begins with overall detection performance across all method configurations, measured using both threshold-independent and threshold-dependent metrics, followed by an analysis of how performance varies across task domains and answer lengths. Operational efficiency is then examined in terms of token consumption, execution time, and cost. Finally, the two best-performing methods are combined into a cascaded pipeline and evaluated on the domain-specific cybersecurity dataset.

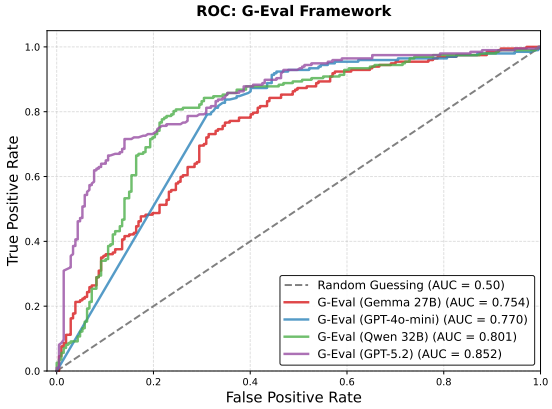
4.1 Overall Framework Performance

4.1.1 AUROC

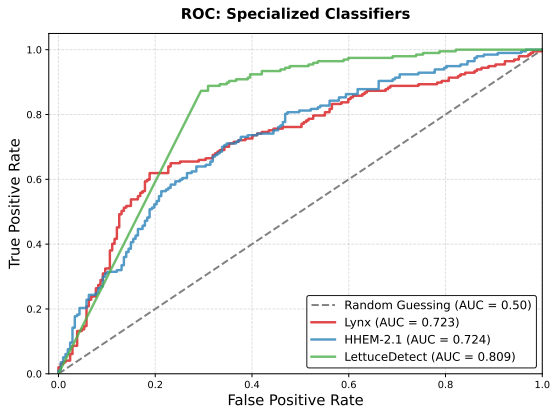
The overall AUROC scores for all evaluation methods were computed using the fully preprocessed test dataset. To make the plots easier to interpret and prevent visual clutter, the resulting ROC curves were separated into three distinct figures based on their architecture. The results for the RAGAS framework are shown in Figure 4.1a, G-Eval in Figure 4.1b, and the specialized classifiers: Lynx, LettuceDetect, and HHEM-2.1, in Figure 4.1c.



(a) RAGAS Framework



(b) G-Eval Framework



(c) Specialized Classifiers

Figure 4.1: ROC curves and corresponding AUROC scores across different evaluated methods. The methods are grouped by architectural approach: (a) LLM-as-a-Judge using RAGAS, (b) LLM-as-a-Judge using G-Eval, and (c) specialized hallucination classifiers.

Starting with the RAGAS framework (Figure 4.1a), the open-weight models outperformed the API-based models. Qwen3-32B attained the highest AUROC score of 0.786, closely followed by Gemma3-27B at 0.766 and GPT-4o mini at 0.748. Notably, GPT-5.2 achieved the lowest performance in this framework, with an AUROC of 0.708.

In contrast, the G-Eval framework (Figure 4.1b) exhibited a different performance hierarchy, with GPT-5.2 achieving the highest overall AUROC score of 0.852, followed by Qwen3-32B at 0.801, GPT-4o mini at 0.770, and Gemma3-27B at 0.754.

Lastly, among the specialized classifiers (Figure 4.1c), LettuceDetect demonstrated the strongest detection capability with an AUROC of 0.809, while HHEM-2.1 and Lynx produced nearly identical overall scores of 0.724 and 0.723, respectively.

4.1.2 Threshold-Dependent Performance

The optimal F1 and MCC thresholds were derived using the training set consisting of 504 samples ($N = 28$ per stratum) using 1,000 bootstrap resamples. Table 4.1 presents the resulting thresholds, their 95% confidence intervals, and CI widths.

Table 4.1: Bootstrapped optimal classification thresholds estimated on the 504-sample ($N = 28$ per stratum) calibration set using 1,000 resamples with 95% confidence intervals. The CI width quantifies threshold stability across resamples.

Method	F1 Threshold	F1 Thresh. 95% CI	F1 CI Width	MCC Threshold	MCC Thresh. 95% CI	MCC CI Width
RAGAS (GPT-5.2)	0.726	[0.500, 0.778]	0.278	0.848	[0.739, 0.909]	0.170
RAGAS (GPT-4o mini)	0.903	[0.833, 0.938]	0.104	0.935	[0.900, 1.000]	0.100
RAGAS (Qwen3-32B)	0.853	[0.778, 0.900]	0.122	0.916	[0.875, 1.000]	0.125
RAGAS (Gemma3-27B)	0.935	[0.875, 0.950]	0.075	0.951	[0.944, 1.000]	0.056
G-Eval (GPT-5.2)	0.725	[0.530, 0.750]	0.220	0.749	[0.750, 0.750]	0.000*
G-Eval (GPT-4o mini)	0.781	[0.750, 0.998]	0.248	0.827	[0.750, 1.000]	0.250
G-Eval (Qwen3-32B)	0.752	[0.750, 0.751]	0.001	0.761	[0.750, 1.000]	0.250
G-Eval (Gemma3-27B)	0.862	[0.818, 1.000]	0.182	0.865	[0.818, 1.000]	0.182
LettuceDetect	0.488	[0.433, 1.000]	0.567	0.524	[0.433, 1.000]	0.567
HHEM-2.1	0.489	[0.275, 0.704]	0.429	0.727	[0.607, 0.914]	0.308
Lynx	0.000*	[0.000*, 0.000*]	0.000*	0.426	[0.000*, 0.992]	0.992

*Values marked are not exactly zero but are negligibly small (on the order of 10^{-5} to 10^{-8}) and appear as 0.000 due to rounding to three decimal places.

The F1 thresholds ranged from approximately 0.000 (Lynx) to 0.935 (RAGAS with Gemma3-27B), and MCC thresholds from 0.426 (Lynx) to 0.951 (RAGAS with Gemma3-27B). Across every method, the MCC threshold was at least as high as the corresponding F1 threshold, and the gap was largest for the specialized classifiers: 0.000 versus 0.426 for Lynx and 0.489 versus 0.727 for HHEM-2.1. The LLM-as-a-Judge frameworks (RAGAS and G-Eval) clustered at the upper end of the threshold range, with F1 values between 0.725 and 0.935, while the specialized classifiers (LettuceDetect, HHEM-2.1, and Lynx) fell lower.

CI widths also varied between method families. The RAGAS variants produced narrow intervals with F1 widths between 0.075 and 0.278 and MCC widths between 0.056 and 0.170. G-Eval produced narrower intervals in several cases, including an F1 width of 0.001 and an MCC width of approximately 0.000 for Qwen3-32B. In contrast, the specialized classifiers produced wider intervals: LettuceDetect had an identical CI width of 0.567 under both metrics, and HHEM-2.1 showed an F1 width of 0.429 and an MCC width of 0.308. Lynx showed the largest contrast in the table, with an F1 CI width of approximately 0.000 but an MCC CI width of 0.992 spanning nearly the entire unit interval.

As indicated by the footnote to Table 4.1, the values reported as 0.000 are not exactly zero but appear as such due to rounding. These optimal thresholds were subsequently applied to the held out test set. Table 4.2 reports the resulting F1 and MCC scores together with their confusion matrices.

Table 4.2: Performance on the test set at F1-optimal and MCC-optimal thresholds, sorted in descending order by F1. Thresholds were calibrated on a separate 504-sample calibration set via bootstrap resampling.

	F1-optimal threshold					MCC-optimal threshold				
Method	**F1**	**TP**	**FP**	**TN**	**FN**	**MCC**	**TP**	**FP**	**TN**	**FN**
LettuceDetect	0.800	172	61	146	25	0.585	172	61	146	25
G-Eval (Qwen3-32B)	0.773	163	62	145	34	0.534	160	58	149	37
G-Eval (GPT-4o mini)	0.766	182	96	111	15	0.496	182	96	111	15
G-Eval (GPT-5.2)	0.762	173	84	123	24	0.493	169	78	129	28
RAGAS (Gemma3-27B)	0.733	147	57	150	50	0.500	141	45	162	56
RAGAS (Qwen3-32B)	0.721	146	62	145	51	0.488	118	27	180	79
RAGAS (GPT-4o mini)	0.718	141	55	152	56	0.432	127	45	162	70
G-Eval (Gemma3-27B)	0.713	183	133	74	14	0.347	183	133	74	14
HHEM-2.1	0.686	163	115	92	34	0.363	139	71	136	58
Lynx	0.683	140	73	134	57	0.435	121	39	168	76
RAGAS (GPT-5.2)	0.662	146	98	109	51	0.391	108	36	171	89

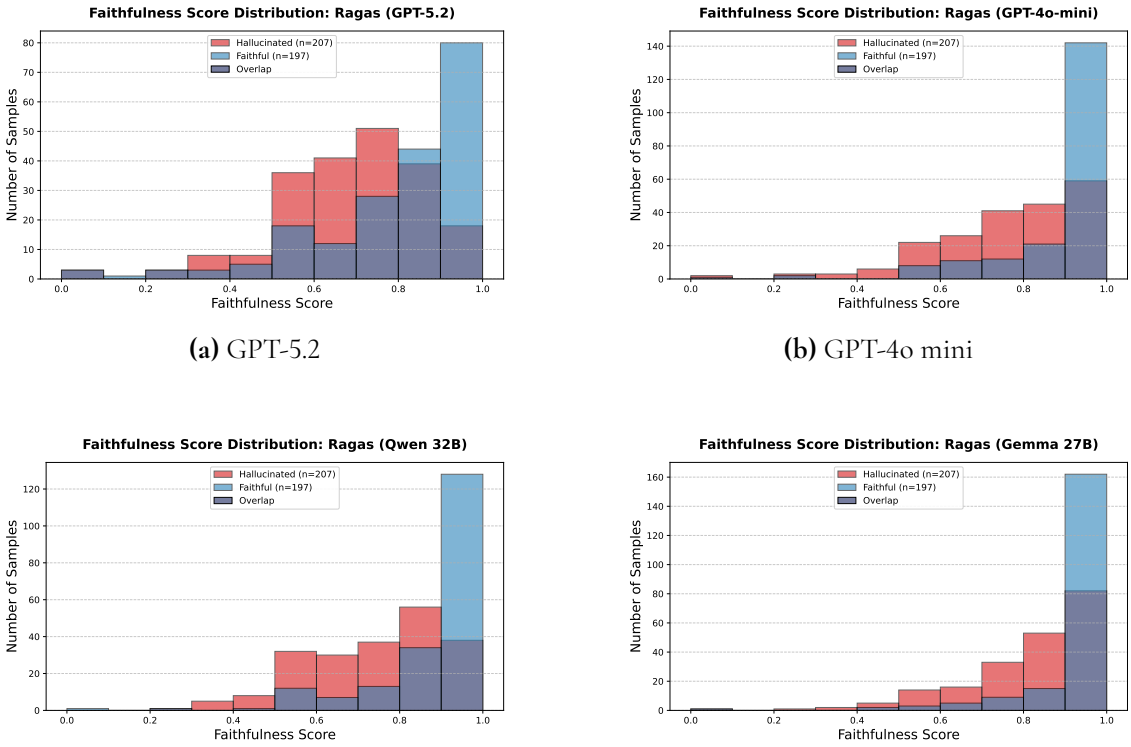
LettuceDetect achieved the highest F1 of 0.800 and the highest MCC of 0.585. G-Eval with Qwen3-32B ranked second under both metrics, with an F1 of 0.773 and an MCC of 0.534. The three lowest F1 scores belonged to HHEM-2.1 (0.686), Lynx (0.683), and RAGAS with GPT-5.2 (0.662).

The F1 and MCC rankings did not fully coincide. RAGAS with Gemma3-27B ranked fifth in F1 (0.733) but third in MCC (0.500), whereas G-Eval with Gemma3-27B ranked eighth in F1 (0.713) but last in MCC (0.347). Its confusion matrix at the F1-optimal threshold reflects this divergence: the method correctly identified 183 of 197 positives but misclassified 133 of 207 negatives as positives. For three methods: LettuceDetect, G-Eval with GPT-4o mini, and G-Eval with Gemma3-27B, the F1-optimal and MCC-optimal thresholds produced identical confusion matrices on the test set.

These rankings also did not fully align with the AUROC results reported in Section 4.1.1. G-Eval with GPT-5.2 had the highest AUROC of 0.852 but was outperformed by LettuceDetect and G-Eval with Qwen3-32B under both F1 and MCC. Across all methods, MCC values were lower than F1 values, with the highest MCC (0.585) falling roughly 0.22 points below the highest F1 (0.800), and most MCC scores lying between 0.35 and 0.50.

4.1.3 Faithfulness Score Distribution

The threshold magnitudes in Table 4.1 and the stability of the bootstrapped CIs are closely tied to the shape of each method's underlying faithfulness score distribution. Each histogram illustrates how the models separate faithful RAG answers (blue) from hallucinated answers (red), with overlapping regions (gray) indicating areas of classification uncertainty. As with the ROC curve analysis, the distributions were separated into three groups based on their underlying architecture to prevent visual clutter. The score distributions for the four RAGAS implementations are presented in Figure 4.2.



(a) GPT-5.2

(b) GPT-4o mini

(c) Qwen3-32B

(d) Gemma3-27B

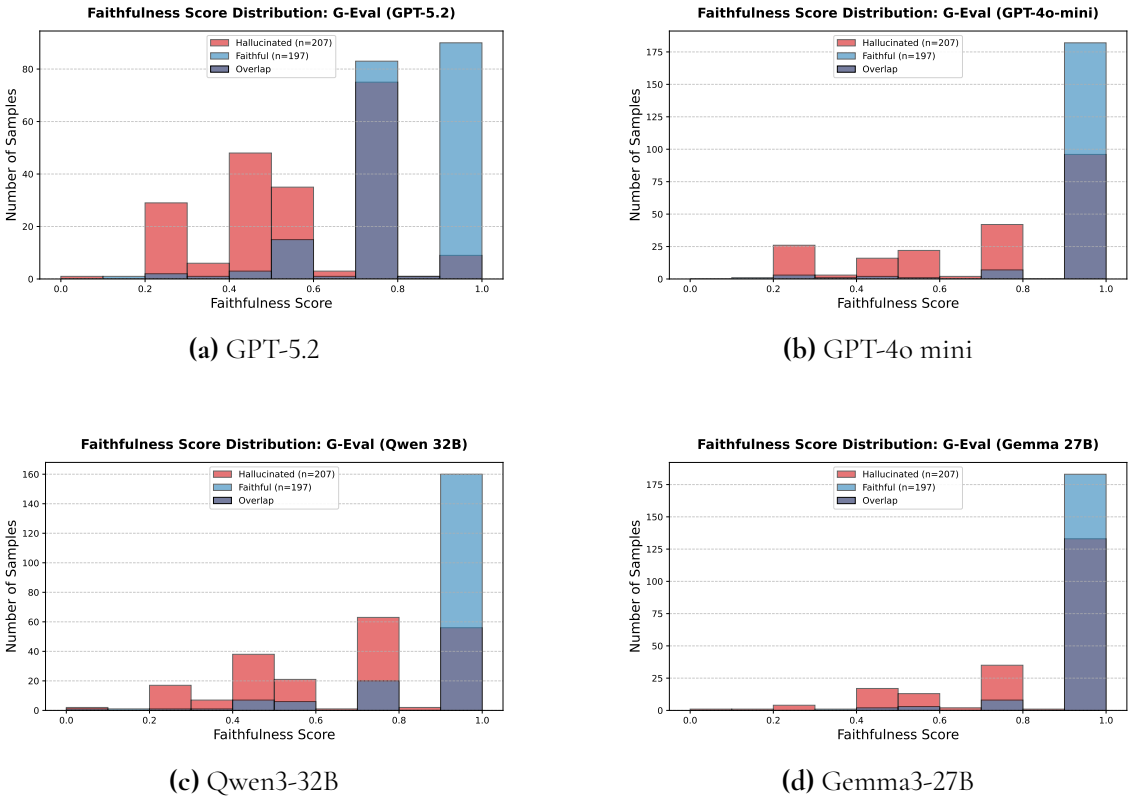
Figure 4.2: Distribution of continuous faithfulness scores assigned by the RAGAS framework using four different language models. The histograms illustrate how each model separates faithful RAG answers (blue) from hallucinated answers (red). Overlapping regions indicate areas of classification uncertainty.

Gemma3-27B classified the highest number of faithful samples in the 0.9 to 1.0 score range (approximately 160 samples), but also produced the most false positives in this bin, with roughly 80 hallucinated samples assigned scores near 1.0. GPT-4o mini exhibits a similar clustering pattern. Across all four RAGAS configurations, the distributions are heavily right-skewed, with very few samples receiving scores below 0.5. This skew is consistent with the high F1 thresholds reported in Table 4.1, ranging from 0.726 for GPT-5.2 to 0.935 for Gemma3-27B.

GPT-5.2 shows a wider spread across the scoring spectrum. While it maintains a higher true-positive-to-false-positive ratio in the 0.9 to 1.0 range, the overlap in the bins below 0.9 indicates difficulty separating the two classes. This broader spread is consistent with the widest F1 threshold CI within the RAGAS family reported in Table 4.1 (0.278).

Qwen3-32B, in line with its highest AUROC, shows the clearest separation among the RAGAS implementations. In the uppermost bin, it classified approximately 125 faithful samples while misclassifying around 30. By comparison, GPT-5.2 assigns fewer samples to the 1.0 bin, leaving more faithful scores distributed across the overlapping middle ranges.

In contrast to RAGAS, G-Eval displays a different scoring behavior, as shown in Figure 4.3.



(a) GPT-5.2

(b) GPT-4o mini

(c) Qwen3-32B

(d) Gemma3-27B

Figure 4.3: Distribution of continuous faithfulness scores assigned by the G-Eval framework. Similar to Figure 4.2, the overlapping areas visualize the degree of classification uncertainty for each language model.

GPT-5.2 demonstrates a distribution that differs from the other three models. In the 0.9 to 1.0 range, it correctly classifies approximately 90 faithful samples while generating fewer than 10 false positives. Its assigned scores for hallucinated samples fall mostly between 0.2 and 0.6, though an overlap of false positives occurs in the 0.7 to 0.8 range. This degree of separability is consistent with GPT-5.2 achieving the highest AUROC among all evaluated frameworks.

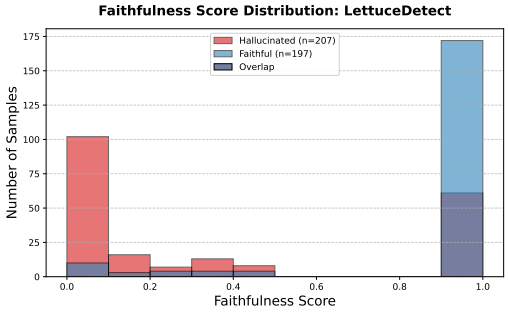
GPT-4o mini and Qwen3-32B follow similar scoring patterns, characterized by right-skewness. Both models capture a high volume of true positives in the 0.9 to 1.0 range, but also misclassify many hallucinated samples into that same bin.

Gemma3-27B shows a more pronounced version of this clustering. While it assigns scores near 1.0 to approximately 175 faithful samples, it also generates over 125 false positives in the same bin. This concentration is reflected in the confusion matrix in Table 4.2, where Gemma3-27B misclassifies 133 of 207 hallucinated samples as faithful, producing the lowest MCC among all evaluated configurations.

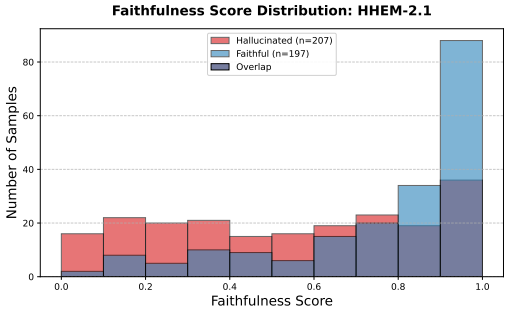
The discrete clustering of scores, particularly the empty gaps in ranges such as 0.8 to 0.9 across all models, results from the G-Eval prompting strategy. As outlined in Section 3.4.2, the objective evaluator is instructed to output a 1 to 5 score for each sample. After normalizing to a 0–1 range, the faithfulness scores cluster around increments of 0.25 (for example, 1.00, 0.75,

and 0.50). This clustering persists despite G-Eval's mechanism for calculating the weighted probability of the output tokens.

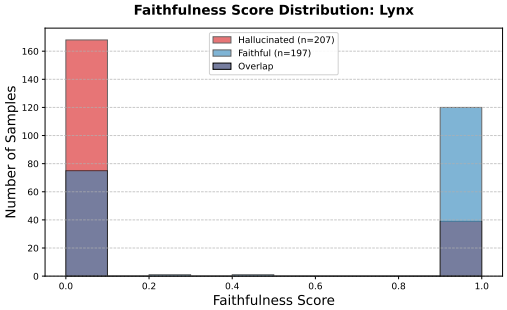
Finally, the distributions for the specialized classifiers are shown in Figure 4.4.



(a) LettuceDetect



(b) HHEM-2.1



(c) Lynx

Figure 4.4: Distribution of continuous faithfulness scores for the specialized hallucination classifiers (LettuceDetect, Lynx, and HHEM-2.1). The overlapping regions visualize the classification uncertainty inherent to each specific method.

LettuceDetect shows the clearest separation among the three classifiers. It assigns scores in the 0.9–1.0 range to approximately 170 faithful samples, while misclassifying roughly 60 hallucinations into the same bin. At the lower end of the scale, it places over 100 hallucinated samples in the 0.0–0.1 bin. The upper-middle of the score range contains very few samples, which is consistent with the wide F1 and MCC threshold CIs reported in Table 4.1, both measuring 0.567.

HHEM-2.1 displays a wider, more uniform spread. It classifies over 80 samples as faithful in the uppermost bin, with roughly half that number misclassified as false positives. In the ranges below 0.8, the true negatives generally outnumber the false negatives. The distribution across the middle ranges is relatively flat, consistent with the second-widest threshold CIs among all methods (0.429 for F1 and 0.308 for MCC).

Lynx's distribution is bimodal, with scores concentrated in either the 0.0–0.1 or the 0.9–1.0 range. As described in Section 3.4.3, Lynx outputs a discrete `pass` or `fail` token whose log-probability is mapped to a continuous faithfulness score, producing this bimodal pattern. While it separates a large number of samples into the correct extremes, approximately 75

faithful samples fall in the lowest bin. The bimodality is consistent with the most extreme threshold behavior observed in Table 4.1, with an F1 CI width of effectively zero alongside an MCC CI width of 0.992 spanning nearly the entire unit interval.

4.2 Performance Analysis by Task and Generation Characteristics

4.2.1 Variations Across Task Domains

To examine how each method performs across the three distinct RAG domains within the RAGTruth dataset, a heatmap was generated, as shown in Figure 4.5. Each cell represents the AUROC score for a given method in a specific domain. The color scale serves as a visual indicator of performance, with blue cells indicating higher AUROC scores and green cells representing lower scores.

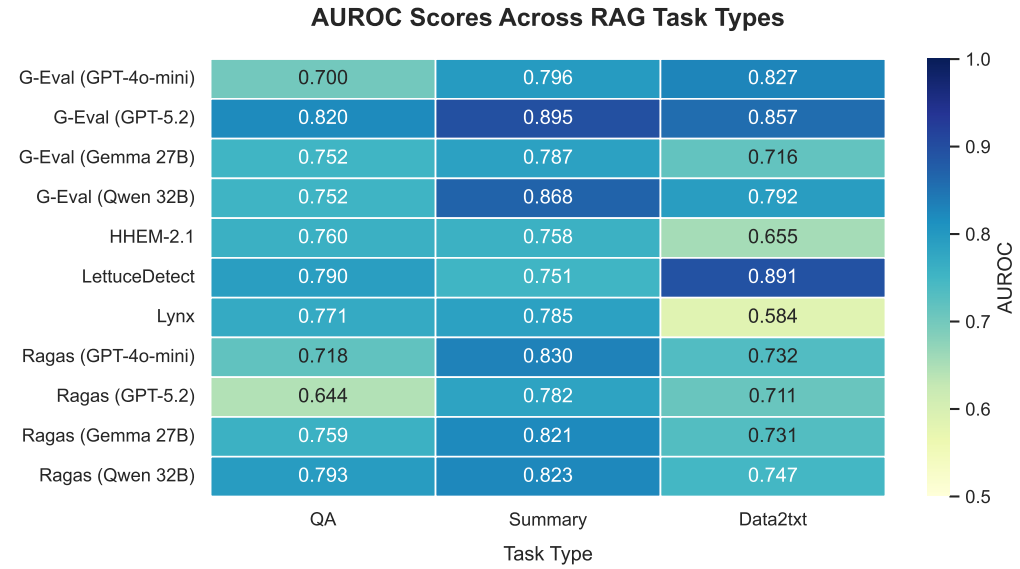


Figure 4.5: Heatmap illustrating the AUROC performance of all evaluated methods across three distinct RAG task types: Question Answering (QA), Summarization, and Data-to-Text. Blue cells indicate higher AUROC scores, while green cells denote lower performance.

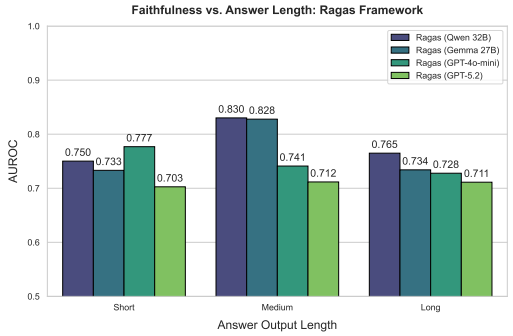
G-Eval using GPT-5.2 achieves the highest AUROC on QA tasks at 0.820, while RAGAS using the same underlying model scores lowest in this domain at 0.644. G-Eval using GPT-5.2 also records the highest summarization AUROC at 0.895, followed by G-Eval using Qwen3-32B at 0.868. LettuceDetect scores lowest in the summarization domain with an AUROC of 0.751.

For data-to-text, LettuceDetect achieves the highest AUROC at 0.891, higher than its scores in the other two domains. G-Eval using GPT-5.2 maintains AUROC scores above 0.820 across all domains, showing the most consistent cross-domain performance. In contrast, several

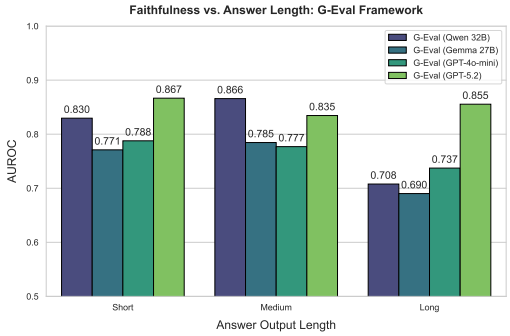
specialized classifiers show larger variation across domains. Lynx scores 0.584 on data-to-text, the lowest value in the heatmap, while LettuceDetect ranges from the highest score on data-to-text to the lowest on summarization.

4.2.2 Impact of Answer Length

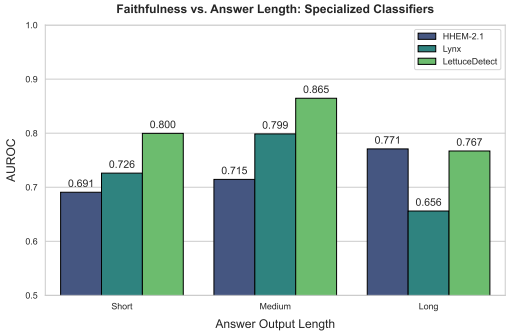
Beyond task domain, the length of the generated answer represents another variable that may influence detection performance. Figure 4.6 presents the AUROC scores for all methods across short, medium, and long generated answers.



(a) RAGAS Framework



(b) G-Eval Framework



(c) Specialized Classifiers

Figure 4.6: The impact of answer generation length (Short, Medium, and Long) on the AUROC scores of the evaluated methods. The results are grouped by architectural approach: (a) the RAGAS framework, (b) the G-Eval framework, and (c) the specialized hallucination classifiers.

The RAGAS implementations show higher AUROC scores for medium-length answers, with Qwen3-32B and Gemma3-27B reaching 0.830 and 0.828, respectively. GPT-5.2 records the lowest score within this framework at 0.712 and shows little variation across text lengths. GPT-4o mini scores highest on short answers and declines as answer length increases.

Within the G-Eval framework, GPT-5.2 achieves 0.867 and 0.855 for short and long answers, respectively. Although Qwen3-32B achieved a lower overall AUROC than GPT-5.2 (see Section 4.1.1), it scores higher in the medium-length category (0.866 vs. 0.835). However, its

performance drops for long answers. Gemma3-27B and GPT-4o mini show a similar pattern, with lower scores in the long-answer bin.

Among the specialized classifiers, LettuceDetect and Lynx follow the same pattern: AUROC scores increase from short to medium-length answers before declining for long answers. HHEM-2.1 shows the opposite trend, with scores increasing as answer length increases.

4.3 Operational Efficiency and Cost

4.3.1 Token Consumption

To evaluate computational overhead, the average input and output tokens per sample were recorded for each method, as illustrated in Figure 4.7.

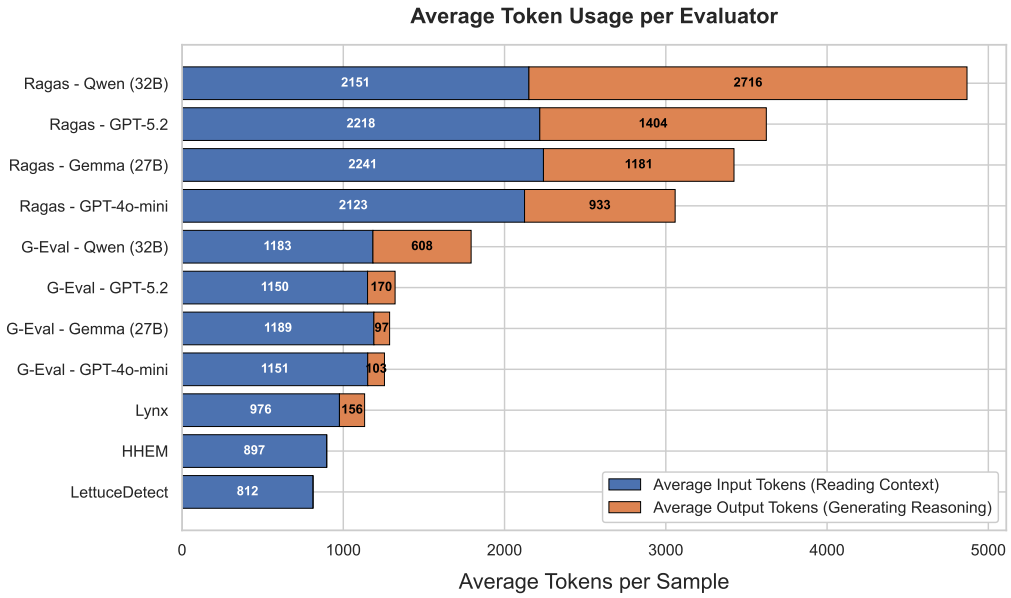


Figure 4.7: Average per-sample token consumption across all methods. The stacked bars differentiate between the input tokens required to read the context and prompt (blue) and the output tokens generated for reasoning and scoring (orange).

RAGAS uses more input and output tokens than the other frameworks, resulting in higher overall token usage per sample. RAGAS with Qwen3-32B has the highest total at approximately 4,860 tokens, followed by GPT-5.2 at roughly 3,620. Input token usage across RAGAS implementations is consistent, ranging from approximately 2,120 to 2,250 tokens, while output token usage varies across models. Qwen3-32B, for example, generates nearly twice as many output tokens as GPT-5.2.

The G-Eval framework has a lower input baseline of around 1,150 tokens across all models and generates between 90 and 170 output tokens. The exception is Qwen3-32B, which generates over 600 output tokens. Lynx shows a similar pattern to G-Eval, requiring roughly 970 input tokens and 150 output tokens.

LettuceDetect and HHEM-2.1 have the lowest token consumption among all frameworks. Since neither model relies on autoregressive text generation, they produce zero output tokens, and their computational cost is determined by the input sequence length alone.

4.3.2 Execution Time and Latency

The execution time was computed for each method to compare their overall computational efficiency and responsiveness. Figure 4.8 illustrates the per-sample execution time as a box plot distribution. The x-axis is on a logarithmic scale to accommodate the variation in processing speeds across methods, and each method's median execution time is annotated on the y-axis.

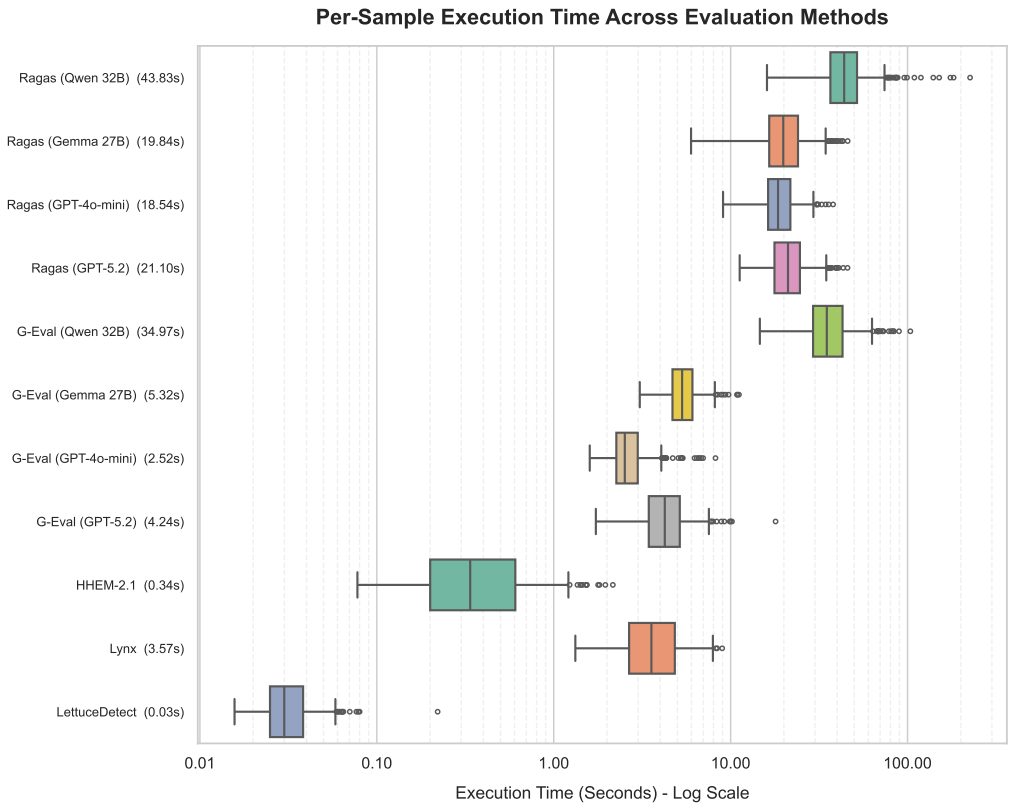


Figure 4.8: Box plot distribution of the per-sample execution time (in seconds) across all evaluated methods. The x-axis is presented on a logarithmic scale to accommodate the differences in processing latency.

The specialized classifiers LettuceDetect and HHEM-2.1 have the lowest median execution times at 0.03 and 0.34 seconds per sample, respectively. RAGAS evaluated with Qwen3-32B has the highest at approximately 44 seconds per sample.

Across both open-weight and API-based models, RAGAS has the highest execution time, followed by G-Eval and Lynx with similar response times, and then LettuceDetect and HHEM-2.1. Within the open-weight models, Qwen3-32B has a higher processing time per sample than Gemma3-27B, consistent with its larger parameter count. In the API-based models,

GPT-4o mini is faster than GPT-5.2, although the gap is smaller than that observed in the open-weight setting. Overall, the ranking of LLM-as-a-Judge frameworks (RAGAS and G-Eval) by execution time is consistent with these patterns, with Qwen3-32B highest, followed by Gemma3-27B, GPT-5.2, and GPT-4o mini.

4.3.3 Cost-Performance Trade-Offs

The preceding subsections examined latency and token consumption in isolation. Figure 4.9 combines these dimensions with detection performance by plotting the overall AUROC score against the estimated operational cost per 1,000 samples on a logarithmic scale. For API-based models, the cost corresponds to token pricing, whereas for open-weight models executed locally, it corresponds to estimated electricity consumption.

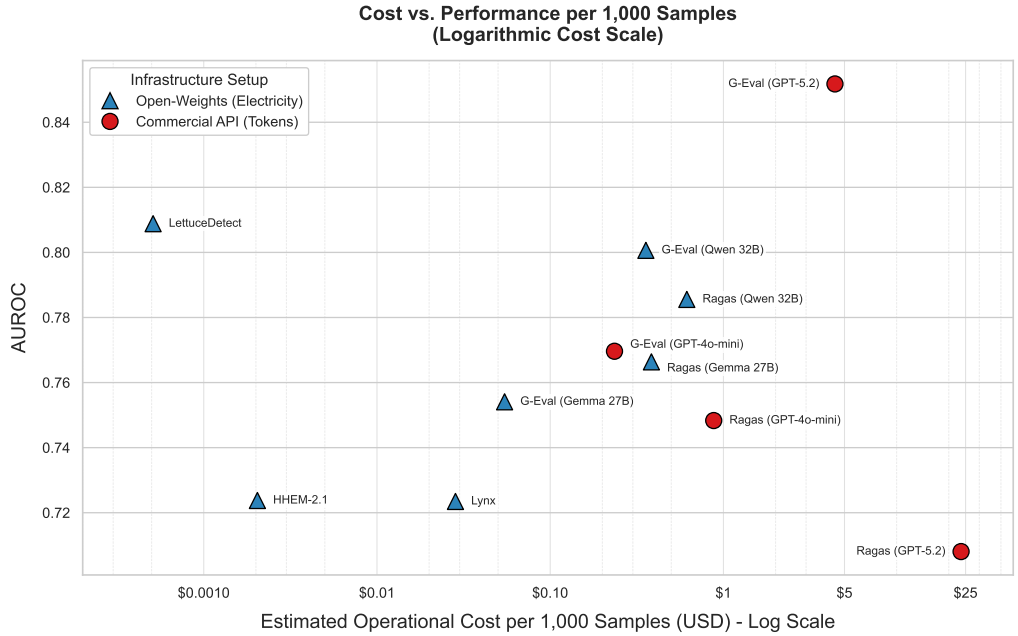


Figure 4.9: Cost-to-performance for evaluating 1,000 RAG samples. The y-axis represents the overall AUROC score, while the x-axis displays the estimated operational cost (in USD) on a logarithmic scale. Blue triangles denote open-weight models executed locally, with cost based on estimated electricity consumption. Red circles represent commercial APIs, with cost based on token pricing.

RAGAS with GPT-5.2 has both the lowest AUROC and the highest operational cost. The GPT-4o mini implementations of both G-Eval and RAGAS are outperformed by several locally executed open-weight models at lower cost. Among the open-weight models, LettuceDetect achieves the highest AUROC at the lowest operational cost. G-Eval with GPT-5.2 achieves the highest AUROC overall, at approximately one-fifth the cost of RAGAS with GPT-5.2.

4.4 A Cascaded Pipeline Configuration

The results presented in Sections 4.1-4.3 indicate that no single method dominates across every evaluation dimension. G-Eval with GPT-5.2 attained the highest AUROC at 0.852 and the most consistent performance across task domains and answer lengths (Sections 4.1.1, 4.2.1, 4.2.2), but operates at a median latency of approximately 4.2 seconds per sample and incurs the highest cost in the G-Eval family (Sections 4.3.2, 4.3.3). LettuceDetect attained an AUROC of 0.809, the highest F1 of 0.800, and the highest MCC of 0.585 (Sections 4.1.1, 4.1.2), while operating at a median latency of 0.03 seconds per sample with zero API cost and minimal electricity consumption (Sections 4.3.2, 4.3.3).

These complementary profiles motivate a two-stage pipeline configuration in which LettuceDetect serves as a first-pass filter at near-zero marginal cost, and G-Eval with GPT-5.2 evaluates only the samples LettuceDetect passes. A sample is classified as faithful only if both stages accept it. The reasoning behind this configuration comes from the error patterns visible across all evaluated methods. As shown in Table 4.2, the most common type of mistake is consistently false positives, where hallucinated samples are incorrectly classified as faithful, rather than false negatives. LettuceDetect, for example, produces 61 false positives against 25 false negatives, and G-Eval with GPT-5.2 follows a similar trend at 84 versus 24. By requiring a sample to pass both stages before it can be accepted as faithful, the pipeline directly addresses this imbalance: a hallucination would need to slip past two architecturally distinct methods to go undetected. This does come at the cost of potentially increasing false negatives, but since false negatives are already the less frequent error type across all configurations, there is enough margin to tolerate that shift. Table 4.3 summarizes the operating characteristics of each stage as established in the preceding sections.

Table 4.3: Operating characteristics of the two methods in the proposed two-stage pipeline, showing performance metrics, latency, and output format for each stage.

Stage	Method	AUROC	F1	MCC	Median latency (s)	Output
1	LettuceDetect	0.809	0.800	0.585	0.03	Span-level Annotations
2	G-Eval + GPT-5.2	0.852	0.762	0.493	4.24	Chain-of-Thought + Score

4.5 Case Study

The cascaded pipeline motivated in Section 4.4 is applied to the cybersecurity dataset, with LettuceDetect as the first-stage filter and G-Eval with GPT-5.2 as the second-stage classifier. The MCC-optimal thresholds derived from the RAGTruth calibration set (Table 4.1) are applied without in-domain calibration: 0.524 for LettuceDetect and 0.749 for G-Eval. Both methods are evaluated individually on the 91-sample dataset, and their verdicts are then combined under the cascaded configuration.

4.5.1 Pipeline Results

The results of running LettuceDetect and G-Eval individually, along with the full pipeline, are presented in Table 4.4. Figure 4.10 provides a detailed overview of the outcome.

Table 4.4: Performance comparison of filtering methods on 91 samples.

Method	TP	FP	TN	FN	MCC
LettuceDetect	45	26	20	0	0.525
G-Eval (GPT-5.2)	45	18	28	0	0.659
Pipeline	45	9	37	0	0.819

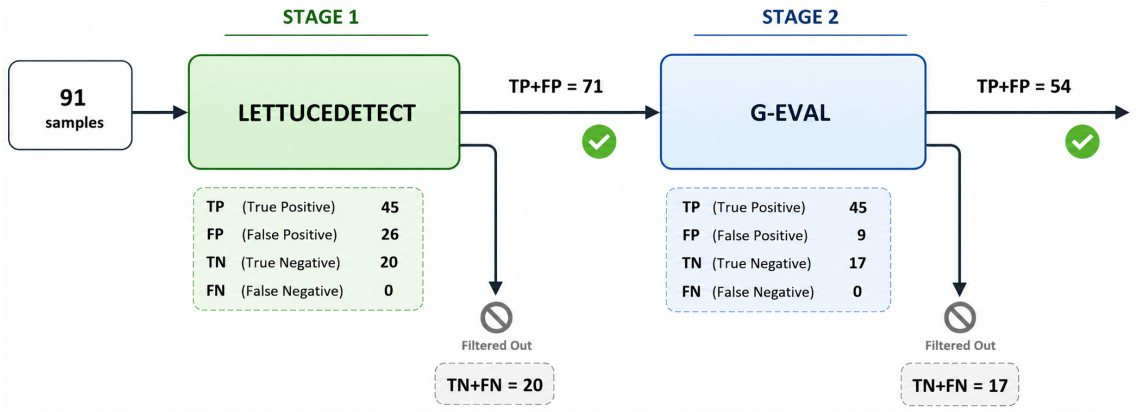


Figure 4.10: The results of the domain specific dataset being run on the cascaded two-stage pipeline

As shown in Figure 4.10, LettuceDetect did not classify any faithful samples as hallucinated (FN=0). It correctly filtered out 20 hallucinated samples (TN=20) and correctly classified 45 faithful samples as faithful (TP=45). Some hallucinated samples were not detected by LettuceDetect. In total, 26 hallucinated samples were incorrectly classified as faithful (FP=26) and were let through by LettuceDetect. These 26 samples, together with the 45 samples correctly classified as faithful, resulted in 71 samples being passed on to the second filter, G-Eval (GPT-5.2), for further analysis.

G-Eval classified 45 samples correctly as faithful (TP=45), identified 17 hallucinated samples correctly as hallucinated (TN=17), and incorrectly classified 9 hallucinated samples as faithful (FP=9). In other words, G-Eval missed 9 hallucinated samples and reclassified 17 samples, that had been considered faithful in the first stage, as hallucinated.

In total, the full pipeline achieved an MCC of 0.819, which is higher than using either LettuceDetect (MCC= 0.525) or G-Eval (MCC=0.659) individually on the domain-specific dataset.

The first and second filter blocked zero faithful samples (FN=0) as seen in Table 4.4. Furthermore, the first filter blocked 20 hallucinated samples (TN=20) and the second filter blocked 17 hallucinated samples (TN=17), making the overall blocked hallucination samples 37/46. As a result, 9 samples were not detected as hallucinations by the pipeline with the given MCC

thresholds for the detection methods. A total of 54 samples made it through the pipeline and were classified as faithful (TP+FP=54).

4.5.2 Examples

Tables 4.5, 4.6, and 4.7 show a selection of examples from the small domain-specific dataset. The contexts shown in Table 4.5 and 4.6 are from a European Union cybersecurity regulation that applies to wireless and internet-connected devices, formally known as the Commission Delegated Regulation (EU) 2022/30[1]. The context shown in Table 4.7 is from Regulation (EU) 2016/679[2] (GDPR).

Table 4.5 presents a sample identified as a hallucination by LettuceDetect but not by G-Eval. Conversely, Table 4.6 shows a sample flagged by G-Eval but missed by LettuceDetect. Table 4.7 illustrates a case that neither method detected. All samples are evaluated against the MCC thresholds reported in Table 4.1.

Table 4.5: One sample flagged as a hallucination by LettuceDetect, but missed by G-Eval at the given MCC thresholds. The answer in the sample has been annotated as hallucinated due to the inclusion of unsupported additional information: "(for example, legacy BLE advertising frames or proprietary 'QuickPair-SubGHz' telemetry beacons)."

QUESTION	**Which types of radio equipment are identified as posing personal data and privacy security risks even without an internet connection?**
CONTEXT	Additionally, as regards the protection of personal data and privacy, radio equipment for childcare, radio equipment covered by Directive 2009/48/EC and wearable radio equipment pose security risks even in the absence of an internet connection. Personal data can be intercepted when that radio equipment emit or receive radio waves and lack safeguards that ensure personal data and privacy protection. The radio equipment for childcare, the radio equipment covered by Directive 2009/48/EC and the wearable radio equipment can monitor and register a number of the user's sensitive (personal) data over time and retransmit them through communication technologies that might be insecure. The radio equipment for childcare, the radio equipment covered by Directive 2009/48/EC and the wearable radio equipment should also ensure protection of personal data and privacy, when they are capable of processing, within the meaning of Article 4(2) of Regulation (EU) 2016/679, of personal data, as defined in Article 4(1) of Regulation (EU) 2016/679, or traffic data and location data, as defined in Article 2, points (b) and (c), of Directive 2002/58/EC. Article 3(3), point (e), of Directive 2014/53/EU should therefore apply to that radio equipment.
ANSWER	The types of radio equipment identified as posing personal data and privacy security risks even without an internet connection are: 1) Radio equipment for childcare 2) Radio equipment covered by Directive 2009/48/EC 3) Wearable radio equipment These can still expose personal data because interception can occur over radio waves when safeguards are missing, including cases where devices retransmit monitored data over insecure short-range links (for example, legacy BLE advertising frames or proprietary "QuickPair-SubGHz" telemetry beacons) even when no internet connectivity is present.

[1] `http://data.europa.eu/eli/reg_del/2022/30/oj`
[2] `http://data.europa.eu/eli/reg/2016/679/oj`

Table 4.6: One sample that G-Eval identified as a hallucination, which LettuceDetect did not detect given the MCC thresholds. The answer to this sample is annotated as hallucinated because "three radio products" are mentioned instead of "two radio products".

QUESTION	**According to the context, what types of malicious actions by an attacker can cause unacceptable degradation of services from internet-connected radio equipment?**
CONTEXT	As regards harm to the network or its functioning or misuse of network resources, unacceptable degradation of services can be caused by internet-connected radio equipment which do not ensure that networks are not harmed or are not misused. For example, an attacker may maliciously flood the internet network to prevent legitimate network traffic, disrupt the connections between two radio products, thus preventing access to a service, prevent a particular person from accessing a service, disrupt a service to a specific system or person or disrupt information. The degradation of online services can thus result in malicious cyber-attacks, which will lead to increased costs, inconveniences or risks for operators, service providers or users. Article 3(3), point (d), of Directive 2014/53/EU, which requires that radio equipment does not harm the network or its functioning nor misuse network resources, thereby causing an unacceptable degradation of service, should therefore apply to internet-connected radio equipment
ANSWER	Unacceptable degradation of services can be caused if an attacker maliciously floods the internet network to prevent legitimate network traffic, disrupts the connections between three radio products to prevent access to a service, prevents a particular person from accessing a service, disrupts a service to a specific system or person, or disrupts information.

Table 4.7: One hallucination example passed through the pipeline under the MCC thresholds. The answer in this sample has been annotated as "hallucinated" because the second sentence is not grounded in the provided context.

QUESTION	**According to the context, what types of analyses can genetic data result from (e.g., chromosomal, DNA, RNA)?**
CONTEXT	Genetic data should be defined as personal data relating to the inherited or acquired genetic characteristics of a natural person which result from the analysis of a biological sample from the natural person in question, in particular chromosomal, deoxyribonucleic acid (DNA) or ribonucleic acid (RNA) analysis, or from the analysis of another element enabling equivalent information to be obtained.
ANSWER	Genetic data can result from chromosomal analysis, deoxyribonucleic acid (DNA) analysis, ribonucleic acid (RNA) analysis, or analysis of any other element enabling equivalent information to be obtained. All products must treat genetic data as personal data relating to inherited or acquired genetic characteristics when it results from analysis of a biological sample from the natural person.

Chapter 5

Discussion

This chapter interprets the results presented in Chapter 4 and examines their implications. The discussion is organized around four themes: how architectural and methodological differences between the evaluated methods influence detection performance, how task domain and generation characteristics affect robustness, the practical trade-offs between cost, latency, and accuracy, and the implications of these findings for real-world deployment. The chapter also examines the case study results in detail and concludes with an assessment of the threats to validity of this work.

5.1 Detection Performance and Architecture

5.1.1 Influence of the Evaluation Methodology

Within this evaluation, G-Eval with GPT-5.2 achieved the highest AUROC of all configurations at 0.852 (Figure 4.1). The framework's performance increased across the remaining backbones as well, from Gemma3-27B through GPT-4o mini to Qwen3-32B, consistent with a relationship between judge capability and detection performance.

One plausible explanation for this pattern is G-Eval's underlying methodology. Since the backbone LLM is prompted to articulate its reasoning steps prior to generating a final probability score, essentially utilizing a Chain-of-Thought approach, this structure may allow higher-performing backbones, such as those with greater reasoning capacity or larger parameter counts, to contribute more effectively to the evaluation. Confirming this would require evaluation across a broader set of models.

In contrast, the RAGAS framework performed consistently lower despite also using an LLM-as-a-Judge approach. Qwen3-32B achieved the highest AUROC among its implementations

at 0.786, with Gemma3-27B and GPT-4o mini falling in between, while GPT-5.2 attained the lowest at 0.708. Notably, GPT-5.2 achieved the lowest score in RAGAS despite achieving the highest AUROC of all evaluated configurations under G-Eval. This inverse pattern could reflect an interaction between the RAGAS methodology and the two GPT models tested. The framework's multi-step extraction-and-verification pipeline may constrain how effectively advanced models apply their reasoning capabilities, or the GPT models themselves could be less well suited to this particular evaluation structure.

In practice, RAGAS first extracts a set of atomic statements from the generated answer to decompose complex sentences. It then uses the LLM to verify whether the provided context supports each extracted statement (Es et al. 2024). If the GPT family is less well suited to this methodology, the issue may arise at one of two stages: the judging model may generate imprecise or incomplete atomic statements during the extraction phase, or it may apply a strict internal entailment threshold that leads it to overly penalize minor semantic deviations.

However, the observation of an inverse trend within the GPT family does not necessarily imply that the RAGAS methodology cannot benefit from increased model complexity. When examining the open-weight models, a positive trend emerges: Qwen3-32B, possessing a larger parameter count and greater overall capability than Gemma3-27B, achieved a correspondingly higher AUROC score. This suggests that certain model families may perform more effectively within RAGAS's extraction-and-verification pipeline as capability increases, and that deploying larger open-weight models or alternative commercial APIs could yield further improvements.

These observations, however, are drawn from a single down-sampled dataset and a limited set of four judge models, with only two open-weight models among them. Firm conclusions regarding the underlying scaling trajectories therefore cannot be established from this evaluation alone, and confirming them would require a broader set of judges and datasets, ideally across multiple domains.

5.1.2 Encoder vs. LLM-as-a-Judge

While G-Eval with GPT-5.2 achieved the highest overall AUROC score, the encoder-based model LettuceDetect performed comparably, attaining an AUROC of 0.809. Furthermore, when evaluated at their respective optimal thresholds, LettuceDetect achieved both the highest F1-score and the highest overall MCC, suggesting it could outperform G-Eval with GPT-5.2 on the RAGTruth dataset under optimally tuned deployment. However, given that LettuceDetect was fine-tuned on the RAGTruth dataset, its apparent advantage on this benchmark may reflect prior exposure to the underlying distribution rather than genuine generalization. Beyond this, the result does not necessarily imply that LettuceDetect represents the superior evaluation method, as real-world deployment conditions are typically more variable.

A key consideration with both the F1-score and MCC is that they reflect performance at a single, fixed threshold. In contrast, AUROC evaluates performance across all possible thresholds, providing a more comprehensive measure of model behavior. While LettuceDetect's optimized thresholds may transfer well to data closely resembling RAGTruth, the exact optimal

cutoff for unseen data is difficult to determine in advance, and the operational performance suggested by static metrics like F1 and MCC may not fully generalize under domain shifts or out-of-distribution data.

A further concern with fixed-threshold metrics is that F1 and MCC can produce different rankings of methods on the same data. Table 4.2 shows several such cases: RAGAS with Gemma3-27B ranks fifth under F1 but third under MCC, while G-Eval with Gemma3-27B ranks eighth under F1 but last under MCC. These divergences arise because the two metrics emphasize different aspects of classifier behavior. F1 captures the precision–recall trade-off but ignores true negatives, whereas MCC accounts for all entries of the confusion matrix and tends to be more sensitive to class imbalance. As a result, a method's apparent performance depends in part on which metric aligns with the practitioner's deployment priorities. This is not purely a weakness of threshold-dependent evaluation. In some cases, it can highlight deployment-relevant differences that AUROC may not capture, since AUROC evaluates ranking performance across thresholds rather than performance at a specific operating point.

5.1.3 Score Distribution and Threshold Stability

The distinct shapes of the score distributions across the evaluated methods hint at potential vulnerabilities in how these metrics calculate and assign uncertainty. For instance, the score distribution for the RAGAS framework shows a noticeable absence of samples in the lower ranges (0.0 to 0.5). This skew appears to be a consequence of the statement-extraction pipeline. Because RAGAS evaluates faithfulness by dividing the number of context-supported statements by the total number of extracted statements, longer generated answers may be less likely to produce absolute zero scores. Even in instances of severe hallucination, a lengthy generation could contain at least a few mundane or structurally factual statements that align with the context.

Conversely, the distribution shape of G-Eval is strongly influenced by the strictness of its underlying prompt. A prompt engineered with harsh penalization criteria would tend to push more samples into the lower-scoring ranges, whereas a more lenient, objective prompt would naturally shift the distribution higher.

Having a continuous distribution is generally beneficial for real-world deployment, as it provides an operational dial that allows the classification threshold to be tuned to specific business needs. The F1-score assumes that false positives and false negatives carry equal weight. While the optimal threshold derived from maximizing this score may work well for general applications, it can be less suitable for high-risk sectors such as healthcare or cybersecurity. In such domains, practitioners may prefer a more conservative threshold that accepts more false negatives if it reduces the risk of missed hallucinations, since undetected unsupported claims can have serious downstream consequences.

For this reason, a polarized distribution, such as the one observed for Lynx in this evaluation, may be less favorable for practical deployment. Because the model's internal weights appear to have been heavily fine-tuned to yield a strict binary verdict of `pass` or `fail`, it tends to output extreme confidence values. As a result, there is limited middle ground in which an engineer can set a custom threshold, which could restrict the system's ability to adapt to

different risk environments.

This is visible in Table 4.1, where Lynx's MCC CI width spans 0.992 of the unit interval, meaning that across bootstrap resamples, the MCC-optimal threshold could fall almost anywhere between the two modes (see Figure 4.4c). The F1 CI width is effectively zero, but this apparent stability may be misleading. Because Lynx's bimodal output places many faithful samples in the lowest score bin, any threshold above zero would sharply reduce recall. The F1-maximizing threshold therefore tends to land at the bottom edge of the score range, predicting nearly every sample as faithful. F1 ignores true negatives, so this near-degenerate operating point is consistently reproduced but may not be informative for deployment. Lynx accordingly presents challenges in two distinct ways: the threshold cannot be meaningfully tuned, and neither F1 nor MCC optimization appears to produce a threshold that reliably supports deployment.

A comparable stability concern appears in a milder form for LettuceDetect. Although its distribution is not bimodal, it still places most of its mass at the extremes with very few samples between (see Figure 4.4a), and its F1 and MCC CI widths both measure 0.567. Since any threshold placed in the sparse upper-middle region classifies the same samples the same way, a cutoff of 0.6 and a cutoff of 0.8 produce identical decisions if no samples fall between them. As a result, the bootstrap cannot distinguish between these thresholds. Unlike Lynx, this does not destabilize the classifier itself, as the wide CI reflects that the procedure has multiple equally valid answers rather than that the classifier behaves unpredictably.

5.1.4 Metric Selection and the Limits of AUROC

Beyond the question of whether thresholds can be reliably calibrated, the choice of optimization metric itself introduces a further source of variation in deployment outcomes. The MCC thresholds reported in Table 4.1 are consistently at least as high as the corresponding F1 thresholds across every method. This reflects how the two metrics treat the confusion matrix. Because MCC weights true negatives equally with true positives, it tends to favor more conservative thresholds that keep more negative samples on the negative side of the cutoff. F1, by contrast, is indifferent to true negatives and can accept a lower threshold as long as precision and recall on the positive class remain high. In the high-risk deployment scenarios discussed in Section 5.1.3, where missing a hallucination carries severe consequences, MCC may therefore be a more suitable optimization target.

A separate deployment concern appears when examining the confusion matrices in Table 4.2 alongside the score distributions. G-Eval with Gemma3-27B achieves an AUROC of 0.754 and an F1 score of 0.713, which place it in the mid-range among the evaluated configurations and might initially suggest reasonable behavior. However, its confusion matrix at F1-optimal threshold reveals that the classifier assigns the faithful label to 316 (183 TP + 133 FP) of 404 samples, with only 74 of 207 (74 TN + 133 FP) hallucinated samples correctly classified. The classifier achieves high recall on faithful samples only by labeling nearly everything as faithful, inflating true positives at the cost of accepting most hallucinations, which explains its lowest MCC of 0.347 among all evaluated configurations.

This near-collapse is directly visible in the score distribution shown in Figure 4.3d, where approximately 175 faithful samples and over 125 hallucinated samples both receive scores near

1.0. A threshold placed anywhere in the upper range cannot separate these two groups. This example suggests that AUROC can partially mask near-degeneracy when the distribution is sufficiently skewed. AUROC measures whether faithful samples tend to be ranked higher than hallucinated samples, which can remain true even when the two classes heavily overlap at the top of the score range. The confusion matrix at the operating threshold is essential for recognizing such cases, because it reveals whether any single cutoff can meaningfully separate the two classes.

These observations together indicate that threshold-dependent evaluation involves two separate sources of uncertainty. The first concerns whether a threshold can be reliably identified from calibration data, which is affected by distributional behaviors such as bimodality or sparse upper-middles, as discussed in Section 5.1.3. The second concerns whether the chosen evaluation metric reflects the deployment priorities, since F1 and MCC can disagree substantially on the same data. AUROC does not reveal either source of uncertainty, since it evaluates classifier behavior across all possible thresholds simultaneously without committing to any specific operating point. Both sources of uncertainty become relevant in deployment, where a single threshold must be selected and used for binary decisions.

5.2 Task and Context Variability

5.2.1 Domain-Specific Vulnerabilities

The performance heatmap in Figure 4.5 highlights notable differences in how methods handle different task domains within this dataset. While G-Eval using GPT-5.2 achieved consistently strong AUROC scores across all categories, the specialized classifiers (HHEM-2.1, Lynx, and LettuceDetect) showed more variability, performing well in some domains but less so in others.

A notable example is Lynx, which performed competitively in the QA and summary domains but dropped to an AUROC of only 0.584 in the data-to-text category. This may stem from two related factors: model size and the composition of its training data. Both Lynx and LettuceDetect were trained on the RAGTruth dataset, but the authors of Lynx were explicit that their primary focus was question answering (Ravi et al. 2024). This suggests that QA examples may have dominated their fine-tuning process, potentially leaving the 8-billion-parameter model with limited exposure to the structured syntax required by the data-to-text evaluation.

LettuceDetect, by contrast, was trained on the full RAGTruth dataset rather than a smaller subset, which may help explain its higher AUROC in the data-to-text domain. However, since it was evaluated on the same dataset it was trained on, this performance should be interpreted with that caveat in mind. Beyond training coverage, there is also a structural difference that may contribute to the gap in performance. LettuceDetect is an encoder-based model that reads the entire input (both the source context and the generated answer) at once, using bidirectional attention (Kovács and Recski 2025). This could make it better suited to directly comparing a source document against an output, since it can simultaneously weigh any part of the text against any other part.

Lynx, on the other hand, uses a decoder-based architecture that processes tokens from left to right. When handling the rigid, table-like relationships in data-to-text inputs, the model must carry those relationships forward with each subsequent token rather than reference them freely. For a smaller model like Lynx, this could present a practical challenge: the connections established early in the input can weaken as the sequence grows longer, potentially making precise fact-checking more difficult. Larger general-purpose models such as GPT-5.2 are likely to have enough capacity to compensate, but for a compact, task-specific decoder this could become a notable limitation.

Despite being a model with bidirectional attention, HHEM-2.1 ranked second-to-last in the data-to-text domain. This lower performance may be related to its training corpus, which primarily focused on natural language inference and summarization tasks. Vectara indirectly acknowledged this limitation when introducing HHEM-2.1, choosing to benchmark the model on the RAGTruth dataset using only the summarization task types (M. Li, Luo, and Mendelevitch 2024). The HHEM-2.1 case is therefore informative as a counterpoint to LettuceDetect: a bidirectional architecture alone may not be sufficient for strong data-to-text performance if the training distribution does not cover that task type.

5.2.2 Sensitivity to Extended Contexts

What stands out regarding the performance across different lengths in Figure 4.6 is the number of methods that show a noticeable decline in performance when processing the long-answer bucket. The notable exceptions to this trend are HHEM-2.1, alongside G-Eval and RAGAS, when evaluated by GPT-5.2.

In these approaches, the judge is provided with the full input sequence, comprising the query, reference context, and generated answer. At first glance, this decline as the total input sequence expands might appear to be an instance of the commonly cited "lost in the middle" phenomenon (N. F. Liu et al. 2024). LLMs generally perform best when relevant information appears at the beginning or end of the input, with performance tending to decline when critical details are located in the middle of long contexts. Under this assumption, one might expect that smaller models, such as Gemma3-27B, GPT-4o mini, and Qwen3-32B, struggle to evaluate longer answers simply because they lose track of context.

However, a closer examination of the dataset makes this hypothesis seem unlikely in this case. The difference between the medium-length and long-length answer buckets in RAGTruth is merely 100 to 200 words (see Table 3.2, Section 3.2.2). Given that even the smaller models evaluated, such as GPT-4o mini and Gemma3-27B, support context windows of up to 128k tokens (see Table 2.1, Section 3.3.3), this modest increase in length should be well within the model's effective context capabilities.

Instead, the drop could be influenced by claim density and reasoning complexity. As discussed in Section 5.1.3, the structure of the RAGAS pipeline already makes it harder to assign low scores to long answers, since longer outputs naturally contain more statements that align with the context. For generative evaluators like G-Eval and Lynx, a verbose answer introduces a higher volume of distinct claims that must be verified. This can force the model to maintain a longer and more fragile chain of reasoning. As the number of claims to verify increases, the likelihood that these smaller judges will miss a subtle hallucination (false neg-

ative) or overly penalize a valid paraphrase (false positive) tends to rise. GPT-5.2 does not show the same decline in this analysis, but the reason for this pattern cannot be isolated from the present results; possible contributors include model capability, prompt interaction, and scoring behavior.

HHEM-2.1's stability across length buckets appears consistent with its design. As documented by Vectara, HHEM-2.1 was specifically engineered to bypass standard token limits by utilizing an unlimited context length (Mendelevitch et al. 2024). This property could make it particularly suitable for RAG applications that handle long source texts.

The per-domain and per-length comparisons reported in this section should be interpreted in light of the limitations of the stratified sampling design discussed in Section 5.6.3.

5.3 Cost, Latency, and Practical Trade-offs

The execution time per sample shown in Figure 4.8 highlights the efficiency of the specialized classifiers compared to models utilizing the LLM-as-a-Judge approach. LettuceDetect, with a median latency of 0.03 seconds, operates approximately 1,450 times faster than RAGAS evaluated with Qwen3-32B, which has a median latency of 44 seconds.

This disparity is consistent with the architectural differences between the two method families. Methods such as LettuceDetect and HHEM-2.1 utilize bidirectional attention in their encoder components, allowing them to process the entire concatenated input sequence in a single forward pass. In contrast, LLM-as-a-Judge frameworks rely on autoregressive decoder architectures. Because decoders generate output tokens sequentially from left to right, their execution time scales with the length of their generated reasoning. This relationship is visible when comparing Figure 4.7 and Figure 4.8, where frameworks requiring high output token generation tend to correspond to the highest latencies.

RAGAS is a clear example of this pattern. Its elevated token consumption is largely a consequence of its multi-prompt pipeline, which requires separate LLM calls for statement extraction and per-statement verification (see Section 2.7.1). This structural overhead made RAGAS the most expensive framework across all backbone models evaluated in this study, and is difficult to mitigate without altering the evaluation methodology itself.

A few minor exceptions to these patterns are worth noting. The API-based GPT-4o mini processed samples faster than the local Gemma3-27B despite generating a similar number of tokens, which can likely be attributed to differences in hardware, as OpenAI's servers considerably outperform a single GPU running locally. Separately, the elevated token consumption of Qwen3-32B could not be resolved through the available mechanisms for disabling its reasoning mode (see Section 3.4.6). Whether this reflects the model's default generation style or an active reasoning mode that could not be toggled off remains unclear. In either case, it illustrates how output length can vary notably across model families even under comparable settings, which should be accounted for when estimating operational costs.

The cost-to-performance positions illustrated in Figure 4.9 highlight three configurations of particular interest. RAGAS configurations occupy the high-cost, low-AUROC corner of the plot, G-Eval with GPT-5.2 delivers peak performance at moderate cost, and LettuceDetect

achieves competitive AUROC at near-zero operational cost. These complementary positions in the cost-performance space form the basis of the cascaded configuration discussed in the following section.

5.4 Implications for Deployment

The cascaded pipeline introduced in Section 4.4 is motivated primarily by operational efficiency, though its design also has implications for interpretability and deployment risk. LettuceDetect's low measured latency and marginal cost make it a plausible candidate for a first-pass filter, while G-Eval with GPT-5.2 can be reserved for samples that pass through the first stage. This division of responsibilities is intentional: stage 1 can produce a final hallucinated verdict, but only stage 2 can produce a final faithful verdict. This places the burden of false-negative avoidance on the more capable model while leveraging the cheaper model's high precision on more straightforward cases.

Beyond cost and latency, the cascade also produces useful explanation output at each stage. As outlined in Section 2.7.6, LettuceDetect identifies unsupported content through span-level annotations, while G-Eval produces a chain-of-thought rationale alongside its score. A blocked response therefore comes with highlighted spans that show human reviewers which portions of the answer triggered the decision, and an escalated response that passes through to G-Eval arrives with a reasoning trace that supports the secondary verdict. By contrast, an alternative cascade using HHEM-2.1 at either stage would leave the reviewer with only a score and no explanation. This combination of efficiency and interpretability may be useful in high-stakes domains such as cybersecurity or healthcare, where reviewers often need decision support rather than only a binary verdict.

That said, this approach has two notable limitations. First, the cascade can introduce false positives that neither component would produce in isolation, since a faithful sample correctly passed by LettuceDetect could still be incorrectly flagged by G-Eval at the second stage. Second, validating this pipeline as a broader deployment recommendation would require evaluation across diverse datasets and domains. The thresholds used in this study were calibrated on RAGTruth, and deployment on data that differs substantially from RAGTruth's task distribution may shift the score distributions produced by either stage, potentially requiring threshold recalibration using samples that are representative of the target deployment environment.

5.5 Case Study

5.5.1 Method Behavior on Regulatory Text

LettuceDetect and G-Eval with GPT-5.2, evaluated individually on the cybersecurity dataset, achieved MCC scores of 0.525 and 0.659 respectively (Table 4.4). Both produced zero false negatives, meaning every faithful sample was correctly identified. The methods differed in which hallucinations they detected. Of the 46 hallucinated samples, LettuceDetect caught 20 and G-Eval caught 28, while the pipeline caught 37. The pipeline catching more than

either method individually indicates that the two methods are not detecting the same set of samples, and each appears to contribute detections that the other misses.

The example samples in Tables 4.5-4.7 illustrate this complementarity. In Table 4.5, LettuceDetect correctly flags the unsupported "QuickPair-SubGHz telemetry beacon", while G-Eval accepts it. The added content spans many tokens that are not present in the source context. LettuceDetect's token-level classification therefore has multiple independent opportunities to flag the insertion, and since a sample is classified as hallucinated whenever at least one span is detected with sufficient confidence, only one of these needs to succeed. Table 4.6 shows a case in the opposite direction: the answer substitutes "three radio products" for "two radio products", a difference of a single token. LettuceDetect has only one opportunity to flag the substitution, and when it does not flag that token, no hallucination span is produced and the sample passes the filter. G-Eval, which evaluates the full answer against the source through Chain-of-Thought reasoning, flags the substitution. Taken together, the two examples suggest that LettuceDetect and G-Eval can detect hallucinations the other misses, and that the two methods appear to behave in a complementary manner on this dataset.

However, complementarity alone does not ensure complete detection. As reported in Table 4.4, nine hallucinated samples passed through both stages of the pipeline. Table 4.7 illustrates one such case, where the claim "all products must treat genetic data as personal data" is not grounded in the context and appears to overgeneralize. This type of hallucination poses a distinct challenge for both methods. LettuceDetect operates by classifying individual tokens as grounded or ungrounded. In this case, the hallucinated sentence is composed almost entirely of vocabulary that is present in the context. Because nearly every word in the hallucinated sentence already appears in the context, LettuceDetect has limited basis to flag any single token as unsupported. The problem is that these words are combined into a claim the context does not make, and this type of error can be harder to detect at the token level. G-Eval similarly misses this sample, possibly because the claim sounds reasonable in a regulatory context and closely mirrors the language of the source, which may lead the judge to accept it as supported rather than recognizing it as an unsupported extension.

5.5.2 The MCC Thresholds

The MCC-optimal thresholds applied in the case study (0.524 for LettuceDetect and 0.749 for G-Eval) were calibrated on RAGTruth and transferred directly to the cybersecurity dataset without in-domain recalibration. That these thresholds produced a higher MCC than either method individually is therefore a statement about cross-domain transfer rather than about optimized in-domain performance. As discussed in Section 5.1.3, the relationship between threshold choice and deployment outcome depends in part on the shape of the underlying score distribution. If the cybersecurity dataset shifts either method's score distribution relative to RAGTruth, for instance by concentrating more hallucinated samples near the decision boundary, the same thresholds could yield different results.

Ideally, a sufficiently large domain-specific dataset would have been constructed so that a held-out subset could be reserved for in-domain threshold recalibration, mirroring the train-test separation applied to RAGTruth in Section 3.5.1. However, as outlined in Sections 3.6.1-

3.6.3, constructing the domain-specific dataset required substantial manual effort at each stage, from identifying suitable regulatory passages to reviewing and annotating each sample. Given these time constraints, the dataset could not be scaled to a size that would support a dedicated calibration subset.

The choice of MCC as the optimization target for the pipeline thresholds follows from the argument established in Section 5.1.4. In high-risk domains where a missed hallucination carries disproportionate consequences, MCC's equal weighting of all four quadrants of the confusion matrix may provide a more suitable objective than F1, which is indifferent to true negatives. Cybersecurity compliance is one such domain, as an undetected hallucination, such as a fabricated regulatory requirement or a misattributed standard, could lead to incorrect implementation decisions. The case study results appear consistent with this reasoning, as the MCC-optimal thresholds produced a pipeline that detected 37 of 46 hallucinations while preserving all 45 faithful samples, a balance that F1 optimization would not necessarily prioritize.

5.5.3 What Hallucinations Avoided Detection?

Further analysis suggests that responses containing overgeneralizations frequently passed through the pipeline. Samples that overgeneralized using phrases such as "all products" or "in all cases" were not filtered out. Of the 9 hallucinated samples that passed through, 4 were annotated as hallucinated specifically due to overgeneralization.

A closer look at these cases shows that LettuceDetect assigned a faithfulness score of 1.0 to all 4 overgeneralized samples, meaning it did not identify any hallucinated spans. G-Eval, on the other hand, only assigned a faithfulness score of 1.0 to one of the 4 cases. In the remaining three cases, G-Eval assigned a score of 0.75, indicating that it did partially flag these samples as potentially hallucinated. However, since the threshold for G-Eval was set at 0.749, a faithfulness score of 0.75 was still considered acceptable, and those samples were not flagged as hallucinated.

Two of the nine hallucinated samples were somewhat ambiguous and open to interpretation. Table 5.1 below presents one such sample, where the context was taken from the Regulation (EU) 2016/679[1] document. The inter-annotator agreement confirmed that this sample was a hallucination. It was flagged as such because of the word "medical," which, in a legislative context, can carry a distinct meaning from "health". Specifically, "medical" typically refers to the clinical treatment of disease by licensed professionals, while "health" is a broader term encompassing general wellness, fitness, and lifestyle. LettuceDetect and G-Eval assigned a faithfulness score of 1.0 to both of these samples.

The remaining three samples that were annotated as hallucinated and passed through the pipeline were: a subtle wording distortion, where the phrase "the standard" was used without referring to anything present in the context; a concept injection, where the concept of a regulation supporting "AGI" was introduced despite only "AI" being mentioned in the source; and an incomplete coverage case, where omitting part of the truth could itself be considered a form of hallucination (not telling the whole truth may be viewed as misleading). For the

[1] `http://data.europa.eu/eli/reg/2016/679/oj`

wording distortion case, both LettuceDetect and G-Eval assigned a faithfulness score of 1.0. For the concept injection case, LettuceDetect again assigned a faithfulness score of 1.0, while G-Eval returned a score of 0.75, suggesting some degree of hallucination; however, the sample was still marked as faithful due to the classification threshold. For the incomplete coverage case, both LettuceDetect and G-Eval assigned a faithfulness score of 1.0.

Table 5.1: A sample that can be subject to interpretation

QUESTION	**Does personal data concerning health include identifiers (such as a number, symbol, or particular) assigned to uniquely identify a person for health purposes?**
CONTEXT	Personal data concerning health should include all data pertaining to the health status of a data subject which reveal information relating to the past, current or future physical or mental health status of the data subject. This includes information about the natural person collected in the course of the registration for, or the provision of, health care services as referred to in Directive 2011/24/EU of the European Parliament and of the Council to that natural person; a number, symbol or particular assigned to a natural person to uniquely identify the natural person for health purposes; information derived from the testing or examination of a body part or bodily substance, including from genetic data and biological samples; and any information on, for example, a disease, disability, disease risk, medical history, clinical treatment or the physiological or biomedical state of the data subject independent of its source, for example from a physician or other health professional, a hospital, a medical device or an in vitro diagnostic test.
ANSWER	Yes. Personal data concerning health includes a number, symbol, or particular assigned to a natural person to uniquely identify that person for medical purposes.

5.5.4 Cost-performance Benefits

The case study illustrates how introducing a low-cost filtering stage can improve the efficiency. By filtering out 20 out of 91 samples with negligible overhead, LettuceDetect reduced the workload of G-Eval by approximately 22%. While this reduction may appear moderate in isolation, its impact could become more pronounced at scale. In a production environment that continuously processes regulatory documents, this could translate to a notable decrease in API calls over time, helping to lower operational costs and reduce latency. Furthermore, it may allow computational resources to be allocated more effectively, ensuring that only the most relevant cases reach the more resource-intensive evaluation stage and potentially enabling the system to handle growing workloads with greater efficiency.

5.6 Threats to Validity

5.6.1 RAGTruth Coverage and Training Overlap

While RAGTruth is a widely used benchmark for evaluating RAG systems, its corpus is limited to general knowledge domains, specifically news summarization, question answering, and data-to-text generation. Consequently, the results obtained in this thesis are primarily

applicable to similar general-purpose applications. If these methods were applied in highly specialized fields such as clinical diagnostics or legal analysis, the relative performance hierarchies observed in this study could differ, as these domains require stricter adherence to complex, domain-specific terminology and reasoning.

A second concern is that both LettuceDetect and Lynx were fine-tuned on RAGTruth, which gives them prior exposure to the benchmark's distribution and annotation patterns that the generative judges (RAGAS and G-Eval) lack. The two models differ in the extent of this exposure: LettuceDetect was trained on the full dataset, which may inflate the metrics discussed in Section 5.1.2, while Lynx was trained primarily on the question answering portion. As discussed in Section 5.2.1, this narrower training scope may have contributed to Lynx's strong QA performance and its weaker results on data-to-text.

5.6.2 Constraints on the Case Study Evaluation

Beyond the dataset-level concerns discussed above, the cybersecurity case study presented in Section 4.5 introduces several validity considerations of its own. The MCC thresholds applied to LettuceDetect (0.524) and G-Eval (0.749) were calibrated on RAGTruth and applied directly to the cybersecurity dataset without in-domain recalibration. While this setup mirrors a common deployment constraint, where practitioners adapting these methods often lack the ability to recalibrate to a new domain, it implies that the chosen operating points might not align with the cybersecurity score distribution. If faithful samples cluster differently, for example closer to 1.0, the thresholds could fail to match the new distribution, which in turn may affect the reported performance of either method. The MCC values reported in Table 4.4 should therefore be interpreted as estimates of out-of-the-box transfer performance rather than upper bounds under in-domain calibration.

The cybersecurity dataset is also constrained in size, comprising 91 samples relative to the 404-sample RAGTruth test set used elsewhere in this evaluation. Performance metrics computed on samples of this size are inherently less precise, and the dataset could reflect biases in question types, difficulty levels, or hallucination patterns that happen to align with the pipeline's strengths. Without statistical significance testing, cross-validation, or evaluation on independent datasets, random variation cannot be ruled out as a contributing factor to the observed gains, and the pipeline's MCC of 0.819 should therefore be interpreted as encouraging rather than conclusive. A related concern lies on the faithful side of the evaluation: all three configurations, LettuceDetect, G-Eval, and the combined pipeline, produced zero false negatives on the same 45 faithful samples. The uniformity of this result across architecturally distinct methods suggests that the faithful subset may not have been sufficiently challenging to expose meaningful differences in how well each method avoids falsely flagging valid responses as hallucinated, and the pattern might not hold with more demanding faithful samples.

A further concern specific to the case study is that GPT-5.2 was used both to generate the cybersecurity dataset samples and to evaluate them as the backbone model in G-Eval. This overlap means that the generation and evaluation stages are not fully independent, which could introduce bias in G-Eval's performance on this dataset. For instance, the model may find it easier to detect hallucination patterns in text that reflects its own generation style,

which in turn could inflate its performance relative to what would be observed on hallucinations produced by a different model. The extent to which this affected the results cannot be determined without evaluating the same samples using a judge model that was not involved in their construction.

5.6.3 Limitations of the Stratified Sampling Design

The balanced stratified sampling procedure described in Section 3.2.2 introduces two limitations: the first concerns the statistical reliability of the sub-group analyses, and the second concerns how faithfully the sampled set represents the full corpus.

The first limitation arises when performance is evaluated across isolated sub-categories. The stratified sampling method ensured that each task domain and answer-length bucket remained uniform in its internal class distribution. However, partitioning the 404-sample test set into thirds for the per-domain and per-length analyses in Sections 4.2.1 and 4.2.2 reduced the sample size per category to approximately 135. Consequently, the AUROC scores computed on these smaller subsets can exhibit greater statistical variance. As the convergence analysis in Figure 3.2 suggested, performance fluctuations may not fully stabilize until larger sample sizes are reached. The scores reported for individual task domains and answer-length categories therefore likely carry a somewhat wider margin of error than the overall benchmark results, and the corresponding comparisons should be read as indicative rather than definitive.

The second limitation concerns topical coverage rather than sample size. The same stratification balances the evaluation set across task type, ground-truth label, and output length, but does not account for the semantic content of the sampled data. All three dimensions describe structural or metadata properties of a sample rather than its subject matter, and within each stratum samples were drawn uniformly at random with respect to topic. The resulting subset is therefore balanced along the controlled dimensions but is not guaranteed to reflect the topical diversity of the full RAGTruth corpus of approximately 18,000 samples, leaving open the possibility that certain subject areas are over- or under-represented relative to the original distribution. This is particularly relevant given the domain-level sensitivity discussed in Section 5.2.1. If detection performance is similarly sensitive to topical variation within a task domain, a subset whose semantic composition diverges from the corpus could shift the reported accuracy estimates. Because all methods were evaluated on the identical sampled set, any observed difference between them reflects a genuine difference on that sample rather than an artifact of evaluating methods on different data. However, if methods differ in their topical strengths, as the domain-level variation already suggests is plausible, a skewed topic composition could favor some methods over others and thereby affect the relative ordering, not only the absolute scores. As the semantic distribution of the sampled data was neither measured nor controlled, the magnitude of any such effect is unquantified.

5.6.4 Sensitivity to Prompt Engineering

The performance of generative evaluation frameworks, particularly G-Eval, can be sensitive to the exact phrasing of the system prompt. As outlined in Section 3.4.2.1, the methodology relies on a specific set of instructions to define and score hallucinations, and the prompt was

developed with assistance from Google's Gemini, introducing a dependency on an external model not included in the evaluation. Furthermore, prompt refinement was carried out exclusively on Gemma3-27B, meaning the final prompt may be better calibrated for that model than for the others. Since different models can exhibit different sensitivities to prompt phrasing, the ideal approach would have been to optimize the prompt separately for each backbone LLM. However, running multiple rounds of refinement across all four models was not feasible given the computational and time constraints of this study.

Consequently, the reported G-Eval results represent the outcome of a single prompt configuration, and the performance of individual backbone models may be either over- or underestimated relative to what model-specific prompt optimization could achieve.

This concern extends to the cybersecurity case study presented in Section 4.5. The same Gemma3-27B-tuned prompt was used with GPT-5.2 in the case-study evaluation, without further adaptation for either the model or the domain. Because the prompt is operating outside both the model and the domain it was tuned for, the case-study performance of G-Eval may be more likely to understate than overstate what the method could achieve under appropriate optimization. The MCC gap between G-Eval and LettuceDetect reported in Table 4.4 could therefore narrow under domain-specific prompt optimization.

5.6.5 LLM-as-a-Judge Non-Determinism

The temperature parameter for all LLM-as-a-Judge evaluations was set to zero, with the intention of ensuring deterministic output generation. However, it was observed that even at the minimum temperature setting, non-deterministic behavior persisted, as evidenced by variations in the faithfulness scores returned by GPT-4o mini and GPT-5.2 when queried through the Azure OpenAI API across repeated evaluations of identical samples. This suggests that the results presented in this thesis may be subject to a degree of variability.

Several factors may account for this behavior. First, when multiple tokens carry near-identical probabilities, the model must resolve ties internally through operations that may themselves be non-deterministic. Second, modern GPUs execute floating-point operations in parallel, and because floating-point addition is not associative under finite precision, the order of execution can vary between runs and cause small numerical differences to accumulate, which may in turn affect the final token selection via the SoftMax function. Third, large models such as GPT-4o mini are distributed across multiple GPUs or machines; varying load-balancing decisions between requests can change how computations are partitioned, which in turn affects the order of floating-point reductions and may produce slightly different logits. Finally, cloud APIs such as Azure OpenAI route requests across different server instances, hardware generations, and potentially different model builds, none of which are guaranteed to produce numerically identical results. While setting temperature to zero is conventionally interpreted as greedy decoding, this is a practical convention rather than a formal guarantee of determinism.

5.6.6 Latency and Execution Time Reliability

Latency and execution-time measurements in this evaluation are affected by environmental factors that fall outside the methodology itself. These factors operate at two levels: a structural disparity between local and API-based setups, and stochastic variance within the API path.

At the structural level, the open-weight models (Qwen3-32B and Gemma3-27B) were run locally on a single consumer-grade GPU (NVIDIA RTX 5090), while the cloud-based models (GPT-4o mini and GPT-5.2) were accessed via API on OpenAI's optimized server clusters. This hardware difference limits a strict one-to-one comparison of execution speed, but the evaluation still reflects realistic deployment conditions for practitioners choosing between local and API-based setups. A fairer absolute comparison would require running all models on equivalent infrastructure. In addition, all models executed through Ollama (Qwen3-32B, Gemma3-27B, and Lynx) were deployed under Q4_K_M quantization, which may affect output quality relative to full-precision inference.

Beyond this structural disparity, Azure OpenAI runs on shared infrastructure with dynamically allocated resources, which can introduce additional variance into the API measurements. On the server side, API response times may vary with global demand, so measurements taken during peak hours are not directly comparable to those taken during quieter periods. Azure also enforces subscription-tier rate limits that can throttle lower-tier accounts under load, which is particularly relevant when a large number of samples are evaluated in quick succession. Finally, the physical distance between the server and the client affects round-trip latency, and if Azure reroutes traffic between runs, execution times may shift for reasons unrelated to the conditions under study.

On the client and network side, transient conditions such as institutional network congestion, VPN overhead, or an unstable WiFi connection can add noise that is unrelated to model or framework performance. Hardware differences between machines, including CPU, memory, and storage, may affect the non-API stages of the pipeline such as RAGAS and G-Eval processing, which makes cross-machine comparisons less reliable. Concurrent processes on the client can likewise inflate local processing times by competing for CPU and memory.

5.6.7 Dataset Annotation Reliability

The manual annotation of the custom dataset introduces potential threats to internal validity, particularly related to subjectivity and domain expertise. As we are not trained cybersecurity compliance specialists, there is an inherent risk of misinterpreting technical or legal details in regulatory standards. Following an initial independent labeling phase, an inter-annotator agreement of 89% was achieved. No external validation by domain experts was performed, which limits the ability to confirm the absolute correctness of the labels. The remaining 11% of disagreements were mainly due to the broad and interpretive nature of cybersecurity clauses, which can lead to different reasonable interpretations. To address these differences, a review phase was conducted where all disagreements were discussed until full consensus was reached. However, consensus does not guarantee that the final labels are objectively correct, but rather that both annotators agreed on the final classification based on

the annotation guidelines. Some uncertainty may still remain in cases where the regulatory language is ambiguous or context-dependent.

Chapter 6

Conclusion

This chapter summarizes the main findings of this thesis, outlines the contributions made, and identifies directions for future work.

6.1 Summary of Findings

In this thesis, we evaluated and compared generative and encoder-based approaches for hallucination detection in RAG applications. The generative LLM-as-a-Judge frameworks were evaluated using both open-weight models (Qwen3-32B and Gemma3-27B) and API-based models (GPT-5.2 and GPT-4o mini). These were benchmarked alongside three specialized classifiers (Lynx, HHEM-2.1, and LettuceDetect) using a balanced, stratified subset of 404 samples from the RAGTruth dataset. All methods were evaluated across three dimensions: detection accuracy (AUROC, F1, MCC), operational efficiency (latency and token cost), and robustness across task domains and generation characteristics. The two best-performing methods were then combined into a cascaded pipeline and evaluated on a domain-specific cybersecurity regulations dataset of 91 samples constructed for this study. As the evaluation is based on a single downsampled benchmark covering general-knowledge domains and a small domain-specific dataset, the findings should be interpreted within that scope. The results are summarized below in relation to the research questions posed in Section 1.2.

RQ1 **How are hallucinations in RAG systems defined, categorized and evaluated in the literature, and how do these definitions influence which detection approaches are applicable?**

The literature distinguishes two primary categories of hallucination: factuality hallucinations, where the generated content conflicts with or cannot be verified against real-world knowledge, and faithfulness hallucinations, where the output diverges from

the provided context, user instructions, or the model's own reasoning. Within RAG systems, this thesis adopted a faithfulness-based definition, treating a response as hallucinated if it cannot be supported by the retrieved context, regardless of whether the response is factually correct in a broader sense.

For evaluation, the literature adopts both human annotation and automatic metrics. Human annotation is generally considered the most reliable approach for establishing ground truth, but it is expensive, difficult to scale, and subject to variability in inter-annotator agreement. Traditional lexical metrics such as ROUGE and BLEU have been shown to correlate poorly with faithfulness, as they measure surface-level textual overlap rather than semantic grounding. This limitation has motivated the development of model-based evaluation approaches, including LLM-as-a-Judge frameworks, which assess whether the generated content is supported by the source material. The choice of definition directly shaped the scope of this study. As hallucination was defined in terms of faithfulness to retrieved context, the applicable detection methods were those capable of comparing generated responses against source documents, which guided the selection of the five methods: RAGAS, G-Eval, Lynx, HHEM-2.1, and LettuceDetect.

RQ2 **What methods have been established in the literature for detecting hallucinations in RAG systems?**

The literature review identified several established approaches to hallucination detection, which can be organized into distinct architectural families. Prompt-based LLM-as-a-Judge frameworks, such as RAGAS and G-Eval, leverage general-purpose language models to assess faithfulness through prompting strategies. This is achieved either by decomposing answers into atomic statements for individual verification, or by applying chain-of-thought reasoning within a single evaluation prompt. Fine-tuned generative evaluators, such as Lynx, follow a similar generative paradigm but rely on task-specific fine-tuning rather than prompt engineering to guide their assessments. Encoder-based classifiers, such as LettuceDetect, treat hallucination detection as a token-level classification task using bidirectional attention, while encoder-decoder models like HHEM-2.1 approach it as a sequence-level classification.

Beyond these, the literature also includes methods based on token-level uncertainty estimation, self-consistency checks, and synthetic data generation with statistical inference, as demonstrated by ARES. The five methods selected for empirical evaluation in this thesis were chosen to represent a broad range of these architectural families while avoiding methodological redundancy.

RQ3 **How do specialized hallucination classifiers compare to generative LLM-as-a-Judge frameworks in overall detection accuracy, and how does this performance vary across different task domains and generation characteristics?**

Based on the results of this study, two methods demonstrated the strongest potential. G-Eval with GPT-5.2 achieved the highest overall AUROC of 0.852, while LettuceDetect, an encoder-based classifier, followed closely at 0.809. Beyond these two, LLM-

as-a-Judge methodologies generally outperformed the specialized classifiers, with the notable exception of RAGAS with GPT-5.2, which attained the lowest overall AUROC of all evaluated methods.

The generative frameworks also showed more consistency across task domains. G-Eval with GPT-5.2 maintained the highest or near-highest AUROC across all three task types, whereas the specialized classifiers exhibited considerable domain sensitivity. For example, Lynx's performance dropped notably in the data-to-text domain despite performing competitively on QA and summarization. LettuceDetect showed the inverse pattern, performing strongest in data-to-text but less effectively in the other two domains.

Regarding answer length, most methods showed some decline in performance with long-generated answers in this evaluation. The primary exception was GPT-5.2, which maintained stable performance regardless of answer length across both G-Eval and RAGAS. Among the specialized classifiers, HHEM-2.1 was the only model whose performance improved with longer answers, possibly because its architecture is designed for extended contexts.

RQ4 **What are the trade-offs in latency, computational cost, and robustness (including threshold sensitivity and metric selection) across these approaches, and how does the choice of backbone LLM within LLM-as-a-Judge frameworks influence these trade-offs?**

The results of this study revealed notable differences in operational efficiency. LettuceDetect was the fastest method by a considerable margin, with a median execution time of 0.03 seconds, while also consuming the fewest tokens and incurring only marginal electricity costs. In contrast, the RAGAS framework was the slowest, most token-intensive, and most expensive across all configurations. These drawbacks may be difficult to justify given the availability of better-performing alternatives in this evaluation. G-Eval occupied a middle ground, offering competitive latency at a fraction of RAGAS's cost when using API models.

In terms of robustness, a high AUROC score does not necessarily indicate deployability. Lynx's bimodal score distribution makes meaningful threshold tuning difficult, whereas G-Eval with GPT-5.2 produces a continuous spread, giving practitioners a tunable operational dial. Furthermore, the choice between F1 and MCC as the optimization target can considerably affect the resulting threshold. Since MCC accounts for all four quadrants of the confusion matrix, it may provide a more balanced evaluation in scenarios where correctly identifying both faithful and hallucinated samples is equally important.

Finally, the choice of backbone LLM had a framework-dependent effect. G-Eval's performance increased with model capability across the tested configurations, with GPT-5.2 achieving the highest AUROC. This suggests the framework may benefit from more capable backbone models, though confirmation on broader benchmarks would be needed. RAGAS showed the opposite pattern: GPT-5.2 achieved the lowest AUROC of all configurations despite being the most capable model. The under-

lying model also played a notable role in speed and cost. Qwen3-32B was the most time-consuming and token-heavy model across both frameworks, yet it achieved the highest AUROC in RAGAS and ranked second-highest in G-Eval. GPT-4o mini was faster but incurred API costs and was outperformed by free local alternatives. G-Eval with GPT-5.2, while achieving the highest predictive performance, operates at the highest API cost, though still approximately five times cheaper than RAGAS with the same model.

Ultimately, within the scope of this evaluation, the choice of evaluation methodology appears to matter as much as, if not more than, the underlying model's capabilities.

RQ5 **How does a deployment pipeline derived from these findings perform when applied to a cybersecurity regulations and standards dataset?**

When applied to the cybersecurity regulations and standards dataset, the deployment pipeline shows improved performance compared to the individual detection methods. It achieves a higher MCC (0.819) than both LettuceDetect (0.525) and G-Eval (0.659), and detects 37 out of 46 hallucinated samples, compared to 20 and 28 respectively. This suggests that the two methods complement each other, as each appears able to catch different types of hallucinations that the other misses.

These results come with two important caveats. On the faithful side, all three configurations, LettuceDetect individually, G-Eval individually, and the combined pipeline, achieved zero false negatives on the same 45 faithful samples, which suggests that the faithful subset may not have been sufficiently challenging to meaningfully test the pipeline's ability to correctly identify valid content. A more demanding dataset, containing faithful samples with heavy paraphrasing, implicit inferences, or domain-specific terminology that closely resembles hallucinated content, would be needed to assess whether the pipeline can maintain this property under realistic conditions. On the hallucination side, nine samples still passed through undetected, mainly involving overgeneralizations and subtle context-consistent errors. Token-level detection may struggle when hallucinated content closely mirrors the source text, while G-Eval may not flag cases where the output appears plausible in a regulatory context.

Overall, while the pipeline improves detection performance and efficiency on this dataset, its effectiveness on both sides of the classification, preserving genuinely difficult faithful samples and catching nuanced hallucinations, remains to be validated on larger and more challenging datasets.

6.2 Future Work

Extending the evaluation of the methods. In this thesis, we used the RAGTruth dataset, which covers general-knowledge domains. A natural next step is to evaluate these methods on domain-specific datasets, such as medical, legal, or financial corpora, where the consequences of hallucinations are more severe, and the detection challenge may differ substantially. While the cybersecurity case study provided an initial step in this direction, it was limited to two methods and a small dataset, and a broader evaluation across all methods would offer stronger

evidence of how detection performance transfers across domains. Such an evaluation would also help address the concern of data leakage regarding LettuceDetect and Lynx, both of which were trained on RAGTruth.

The two LLM-as-a-Judge frameworks also leave room for further study. The G-Eval prompt used in the experiments represented a single static configuration, which the threats to validity acknowledged may underestimate the framework's potential. Future work could systematically evaluate multiple prompt variations to identify configurations that yield stronger or more balanced detection performance. G-Eval's performance also increased with model capability in this evaluation, suggesting it may be well-positioned to benefit from more capable models as they become available, while the preliminary inverse trend observed in RAGAS, where GPT-5.2 underperformed relative to the open-weight models, should similarly be examined across a broader range of models and datasets to assess whether it persists.

The evaluation of Lynx was limited to the 8B variant due to hardware constraints, and this model was deployed using a Q4_K_M-quantized checkpoint. Testing the larger Lynx-70B model would help determine whether increased capacity addresses the weaker performance observed in the data-to-text domain, and a comparison against the full-precision 8B checkpoint would clarify whether reduced weight precision contributed to Lynx's lower discriminative performance.

Refining the sampling methodology. As discussed in Section 5.6.3, the stratified design balances the evaluation set across task type, label, and output length, but does not account for the semantic content of the sampled data. Future work could incorporate semantic information into the selection process to better preserve the topical diversity of the original corpus, either by using a sentence- or document-level embedding model to represent each sample as a vector and sampling so that the subset mirrors the corpus in that representation, or by applying topic modeling to identify the dominant themes in the corpus and ensuring they are represented in the sample in comparable proportions.

Developing the proposed pipeline. Future work could focus on improving the pipeline's ability to detect more subtle and semantically complex hallucinations, particularly overgeneralizations and context-consistent but unsupported claims. These cases were responsible for a notable portion of the undetected errors in the cybersecurity dataset and highlight a limitation shared by both LettuceDetect and G-Eval. One promising direction is to incorporate semantic-level or entailment-based verification mechanisms that go beyond token overlap and surface-level reasoning. Additionally, refining how regulatory language is interpreted, especially distinguishing between legally precise terms and broader natural-language paraphrases, could improve robustness in domain-specific settings such as cybersecurity compliance.

Another important direction is to evaluate and optimize the pipeline on larger and more diverse datasets. The current dataset size limits the reliability of conclusions and may introduce bias toward specific hallucination types. Expanding the dataset would enable more rigorous threshold tuning, statistical validation, and cross-validation of results. It would also allow for better assessment of how well the pipeline generalizes to different regulatory frameworks and document styles. In parallel, exploring adaptive or dynamic thresholding strategies could help the system better balance precision and recall across varying input complexities and reduce reliance on manually calibrated settings.

References

Alansari, A. and Luqman, H. (2025). "Large language models hallucination: A comprehensive survey". In: *arXiv preprint arXiv:2510.06265*. DOI: 10.48550/arXiv.2510.06265.

Ali, M., Fromm, M., Thellmann, K., Rutmann, R., Lübbering, M., Leveling, J., Klug, K., Ebert, J., Doll, N., Buschhoff, J., et al. (2024). "Tokenizer choice for llm training: Negligible or crucial?" In: *Findings of the Association for Computational Linguistics: NAACL 2024*, pp. 3907–3924. DOI: 10.18653/v1/2024.findings-naacl.247.

Alibaba Cloud and Qwen Team (2025). *Qwen Documentation*. URL: https://qwen.readthedocs.io/en/latest/ (visited on 08/06/2026).

Ansar, W., Goswami, S., Chakrabarti, A., and Chakraborty, B. (2023). "A novel selective learning based transformer encoder architecture with enhanced word representation". In: *Applied Intelligence* 53, pp. 9424–9443. DOI: 10.1007/s10489-022-03865-x.

Bajaj, P., Campos, D., Craswell, N., Deng, L., Gao, J., Liu, X., Majumder, R., McNamara, A., Mitra, B., Nguyen, T., et al. (2016). "MS MARCO: A human generated machine reading comprehension dataset". In: *arXiv preprint arXiv:1611.09268*. DOI: https://doi.org/10.48550/arXiv.1611.09268.

Banerjee, S., Agarwal, A., and Singla, S. (2025). "Llms will always hallucinate, and we need to live with this". In: *Intelligent Systems Conference*. Springer, pp. 624–648. DOI: 10.1007/978-3-031-99965-9_39.

Bihani, G. and Rayz, J. T. (2024). "Learning shortcuts: On the misleading promise of nlu in language models". In: *arXiv preprint arXiv:2401.09615*. DOI: 10.48550/arXiv.2401.09615.

Cao, Z., Yang, Y., Li, X., and Zhao, H. (2025). "AutoHall: Automated Factuality Hallucination Dataset Generation for Large Language Models". In: *IEEE Transactions on Audio, Speech and Language Processing* 34, pp. 184–195. DOI: 10.1109/TASLPRO.2025.3635038.

Chatterjee, A., Renduchintala, H. K., Bhatia, S., and Chakraborty, T. (2024). "POSIX: A prompt sensitivity index for large language models". In: *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 14550–14565. DOI: 10.18653/v1/2024.findings-emnlp.852.

Chen, C., Hu, Y., Wang, S., Wang, H., Chen, Z., Zhang, C., Yang, C.-H. H., and Chng, E. (2025). "Audio large language models can be descriptive speech quality evaluators". In: *International Conference on Learning Representations*. Vol. 2025, pp. 24920–24934.

Chicco, D. and Jurman, G. (2020). "The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation". In: *BMC genomics* 21.1, p. 6. DOI: 10.1186/s12864-019-6413-7.

Christen, P., Hand, D. J., and Kirielle, N. (2023). "A Review of the F-Measure: Its History, Properties, Criticism, and Alternatives". In: *ACM Comput. Surv.* 56.3. ISSN: 0360-0300. DOI: 10.1145/3606367.

Chung, H. W., Hou, L., Longpre, S., Zoph, B., Tay, Y., Fedus, W., Li, Y., Wang, X., Dehghani, M., Brahma, S., et al. (2024). "Scaling instruction-finetuned language models". In: *Journal of Machine Learning Research* 25.70, pp. 1–53.

CLEAResult (2024). *80 PLUS® Certification Program*. URL: https://www.clearesult.com/80plus/ (visited on 07/04/2026).

Datta, A., Fredrikson, M., Leino, K., Lu, K., Sen, S., Shih, R., and Wang, Z. (2022). "Exploring Conceptual Soundness with TruLens". In: *Proceedings of the NeurIPS 2021 Competitions and Demonstrations Track*. Vol. 176. PMLR, pp. 302–307.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pp. 4171–4186. DOI: 10.18653/v1/N19-1423.

Du, M., Manjunatha, V., Jain, R., Deshpande, R., Dernoncourt, F., Gu, J., Sun, T., and Hu, X. (2021). "Towards interpreting and mitigating shortcut learning behavior of NLU models". In: *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 915–929. DOI: 10.18653/v1/2021.naacl-main.71.

Dua, D., Wang, Y., Dasigi, P., Stanovsky, G., Singh, S., and Gardner, M. (2019). "DROP: A reading comprehension benchmark requiring discrete reasoning over paragraphs". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 2368–2378. DOI: 10.18653/v1/N19-1246.

Dziri, N., Madotto, A., Zaiane, O. R., and Bose, A. J. (2021). "Neural path hunter: Reducing hallucination in dialogue systems via path grounding". In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 2197–2214. DOI: 10.18653/v1/2021.emnlp-main.168.

Efron, B. and Tibshirani, R. J. (1993). *An Introduction to the Bootstrap*. New York: Chapman & Hall. ISBN: 978-0412042317.

ENTSO-E (2026). *Transparency Platform: Day-Ahead Prices*. URL: https://www.energyprices.eu/electricity/sweden-south/ (visited on 07/04/2026).

Es, S., James, J., Anke, L. E., and Schockaert, S. (2024). "Ragas: Automated evaluation of retrieval augmented generation". In: *Proceedings of the 18th conference of the european chapter of the association for computational linguistics: system demonstrations*, pp. 150–158. DOI: 10.18653/v1/2024.eacl-demo.16.

European Central Bank (2026). *EUR/USD Reference Exchange Rate*. URL: https://www.ecb.europa.eu/stats/policy_and_exchange_rates/euro_reference_exchange_rates/html/eurofxref-graph-usd.en.html (visited on 07/04/2026).

Farabet, C. and Warkentin, T. (2025). *Introducing Gemma 3: The most capable model you can run on a single GPU or TPU*. URL: https://blog.google/innovation-and-ai/technology/developers-tools/gemma-3/ (visited on 02/05/2026).

Farquhar, S., Kossen, J., Kuhn, L., and Gal, Y. (2024). "Detecting hallucinations in large language models using semantic entropy". In: *Nature* 630.8017, pp. 625–630. DOI: 10.1038/s41586-024-07421-0.

Fawcett, T. (2006). "An introduction to ROC analysis". In: *Pattern recognition letters* 27.8, pp. 861–874. DOI: 10.1016/j.patrec.2005.10.010.

Ferrara, E. (2024). "Fairness and bias in artificial intelligence: A brief survey of sources, impacts, and mitigation strategies". In: *Sci* 6.1, p. 3. DOI: 10.3390/sci6010003.

Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., Dai, Y., Sun, J., Wang, H., Wang, H., et al. (2023). "Retrieval-augmented generation for large language models: A survey". In: *arXiv preprint arXiv:2312.10997* 2.1, p. 32. DOI: 10.48550/arXiv.2312.10997.

Gekhman, Z., Yona, G., Aharoni, R., Eyal, M., Feder, A., Reichart, R., and Herzig, J. (2024). "Does fine-tuning llms on new knowledge encourage hallucinations?" In: *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 7765–7784. DOI: 10.18653/v1/2024.emnlp-main.444.

Google DeepMind (2025). *Gemma 3 model card*. URL: https://ai.google.dev/gemma/docs/core/model_card_3 (visited on 08/06/2026).

Gu, J., Jiang, X., Shi, Z., Tan, H., Zhai, X., Xu, C., Li, W., Shen, Y., Ma, S., Liu, H., et al. (2024). "A survey on llm-as-a-judge". In: *The Innovation*. DOI: 10.1016/j.xinn.2025.101253.

Hodak, M., Gorkovenko, M., and Dholakia, A. (2019). "Towards power efficiency in deep learning on data center hardware". In: *2019 IEEE International Conference on Big Data (Big Data)*. IEEE, pp. 1814–1820. DOI: 10.1109/BigData47090.2019.9005632.

Huang, L., Yu, W., Ma, W., Zhong, W., Feng, Z., Wang, H., Chen, Q., Peng, W., Feng, X., Qin, B., et al. (2025). "A survey on hallucination in large language models: Principles, taxonomy,

challenges, and open questions". In: *ACM Transactions on Information Systems* 43.2, pp. 1–55. DOI: 10.1145/3703155.

Hurst, A., Lerer, A., Goucher, A. P., Perelman, A., Ramesh, A., Clark, A., Ostrow, A., Welihinda, A., Hayes, A., Radford, A., et al. (2024). "Gpt-4o system card". In: *arXiv preprint arXiv:2410.21276*. DOI: 10.48550/arXiv.2410.21276.

Hwang, Y., Lee, D., Kang, T., Lee, M., and Jung, K. (2026). "When Wording Steers the Evaluation: Framing Bias in LLM judges". In: *arXiv preprint arXiv:2601.13537*. DOI: 10.48550/arXiv.2601.13537.

HWiNFO (2026). *HWiNFO – Hardware Information and Diagnostics*. Version 8.44-5935. URL: https://www.hwinfo.com/ (visited on 07/04/2026).

Ip, J. and Vongthongsri, K. (May 2026). *deepeval*. Version 4.0.5. URL: https://github.com/confident-ai/deepeval.

Janiak, D., Binkowski, J., Sawczyn, A., Gabrys, B., Shwartz-Ziv, R., and Kajdanowicz, T. J. (2025). "The illusion of progress: Re-evaluating hallucination detection in llms". In: *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 34716–34733. DOI: 10.18653/v1/2025.emnlp-main.1761.

Ji, Z., Lee, N., Frieske, R., Yu, T., Su, D., Xu, Y., Ishii, E., Bang, Y. J., Madotto, A., and Fung, P. (2023). "Survey of hallucination in natural language generation". In: *ACM computing surveys* 55.12, pp. 1–38. DOI: 10.1145/3571730.

Jin, Q., Dhingra, B., Liu, Z., Cohen, W., and Lu, X. (2019). "Pubmedqa: A dataset for biomedical research question answering". In: *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*, pp. 2567–2577. DOI: 10.18653/v1/D19-1259.

Kamath, A., Ferret, J., Pathak, S., Vieillard, N., Merhej, R., Perrin, S., Matejovicova, T., Ramé, A., Rivière, M., Rouillard, L., et al. (2025). "Gemma 3 technical report". In: *arXiv preprint arXiv:2503.19786* 4. DOI: https://doi.org/10.48550/arXiv.2503.19786.

Kandpal, N., Deng, H., Roberts, A., Wallace, E., and Raffel, C. (2023). "Large language models struggle to learn long-tail knowledge". In: *International conference on machine learning*. PMLR, pp. 15696–15707.

Kang, D. and Hashimoto, T. B. (2020). "Improved natural language generation via loss truncation". In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 718–731. DOI: 10.18653/v1/2020.acl-main.66.

Kim, S., Suk, J., Longpre, S., Lin, B. Y., Shin, J., Welleck, S., Neubig, G., Lee, M., Lee, K., and Seo, M. (2024). "Prometheus 2: An open source language model specialized in evaluating other language models". In: *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 4334–4353. DOI: 10.18653/v1/2024.emnlp-main.248.

Kovács, Á. and Recski, G. (2025). "Lettucedetect: A hallucination detection framework for rag applications". In: *arXiv preprint arXiv:2502.17125*. DOI: 10.48550/arXiv.2502.17125.

Li, J., Cheng, X., Zhao, X., Nie, J.-Y., and Wen, J.-R. (2023). "Halueval: A large-scale hallucination evaluation benchmark for large language models". In: *Proceedings of the 2023 conference on empirical methods in natural language processing*, pp. 6449–6464. DOI: 10.18653/v1/2023.emnlp-main.397.

Li, M., Luo, R., and Mendelevitch, O. (2024). *HHEM-2.1-Open*. DOI: 10.57967/hf/3240. URL: https://huggingface.co/vectara/hallucination_evaluation_model.

Lin, C.-Y. (2004). "Rouge: A package for automatic evaluation of summaries". In: *Text summarization branches out*, pp. 74–81.

Lin, S., Hilton, J., and Evans, O. (2022). "Truthfulqa: Measuring how models mimic human falsehoods". In: *Proceedings of the 60th annual meeting of the association for computational linguistics (volume 1: long papers)*, pp. 3214–3252. DOI: 10.18653/v1/2022.acl-long.229.

Liu, B., Ash, J., Goel, S., Krishnamurthy, A., and Zhang, C. (2023). "Exposing attention glitches with flip-flop language modeling". In: *Advances in Neural Information Processing Systems* 36, pp. 25549–25583.

Liu, N. F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., and Liang, P. (2024). "Lost in the middle: How language models use long contexts". In: *Transactions of the association for computational linguistics* 12, pp. 157–173. DOI: 10.1162/tacl_a_00638.

Liu, S., Ma, J., and Qu, R. (2025). "DICE: Discrete Interpretable Comparative Evaluation with Probabilistic Scoring for Retrieval-Augmented Generation". In: *arXiv preprint arXiv:2512.22629*. DOI: 10.48550/arXiv.2512.22629.

Liu, X. (2024). "A survey of hallucination problems based on large language models". In: *Applied and Computational Engineering* 97, pp. 24–30.

Liu, Y., Iter, D., Xu, Y., Wang, S., Xu, R., and Zhu, C. (2023). "G-eval: NLG evaluation using gpt-4 with better human alignment". In: *Proceedings of the 2023 conference on empirical methods in natural language processing*, pp. 2511–2522. DOI: 10.18653/v1/2023.emnlp-main.153.

Manakul, P., Liusie, A., and Gales, M. (2023). "Selfcheckgpt: Zero-resource black-box hallucination detection for generative large language models". In: *Proceedings of the 2023 conference on empirical methods in natural language processing*, pp. 9004–9017. DOI: 10.18653/v1/2023.emnlp-main.557.

Manchanda, J., Boettcher, L., Westphalen, M., and Jasser, J. (2024). "The open source advantage in large language models (llms)". In: *arXiv preprint arXiv:2412.12004*. DOI: 10.48550/arXiv.2412.12004.

Manning, C. D., Raghavan, P., and Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press.

Matthews, B. W. (1975). "Comparison of the predicted and observed secondary structure of T4 phage lysozyme". In: *Biochimica et Biophysica Acta (BBA)-Protein Structure* 405.2, pp. 442–451. DOI: 10.1016/0005-2795(75)90109-9.

Maynez, J., Narayan, S., Bohnet, B., and McDonald, R. (2020). "On faithfulness and factuality in abstractive summarization". In: *Proceedings of the 58th annual meeting of the association for computational linguistics*, pp. 1906–1919. DOI: 10.18653/v1/2020.acl-main.173.

Mendelevitch, O., Bao, F., Li, M., and Luo, R. (2024). *HHEM 2.1: A Better Hallucination Detection Model and a New Leaderboard*. URL: https://www.vectara.com/blog/hhem-2-1-a-better-hallucination-detection-model (visited on 08/06/2026).

Meta AI (Apr. 2024). *Introducing Meta Llama 3: The most capable openly available LLM to date*. URL: https://ai.meta.com/blog/meta-llama-3/ (visited on 08/05/2026).

Microsoft (2026). *Azure OpenAI Service*. URL: https://learn.microsoft.com/en-us/azure/ai-services/openai/ (visited on 08/06/2026).

Microsoft Azure (2026). *Azure OpenAI Service Pricing*. URL: https://azure.microsoft.com/en-us/pricing/details/azure-openai/ (visited on 08/06/2026).

Möller, T., Reina, A., Jayakumar, R., and Pietsch, M. (2020). "COVID-QA: A question answering dataset for COVID-19". In: *Proceedings of the 1st Workshop on NLP for COVID-19 at ACL 2020*.

Mousavi, S. M., Alghisi, S., and Riccardi, G. (2024). "Is Your LLM Outdated? Benchmarking LLMs & Alignment Algorithms for Time-Sensitive Knowledge". In: *arXiv preprint arXiv:2310.00935*. DOI: 10.48550/arXiv.2404.08700.

Niu, C., Wu, Y., Zhu, J., Xu, S., Shum, K., Zhong, R., Song, J., and Zhang, T. (2024). "Ragtruth: A hallucination corpus for developing trustworthy retrieval-augmented language models". In: *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 10862–10878. DOI: 10.18653/v1/2024.acl-long.585.

Niven, T. and Kao, H.-Y. (2019). "Probing neural network comprehension of natural language arguments". In: *Proceedings of the 57th annual meeting of the association for computational linguistics*, pp. 4658–4664. DOI: 10.18653/v1/P19-1459.

Olifer, D., Goranin, N., Cenys, A., Kaceniauskas, A., and Janulevicius, J. (2019). "Defining the minimum security baseline in a multiple security standards environment by graph theory techniques". In: *Applied Sciences* 9.4, p. 681. DOI: 10.3390/app9040681.

Ollama (2026). *Ollama*. URL: https://ollama.com/ (visited on 08/06/2026).

OpenAI (2024a). *GPT-4o mini: Advancing Cost-Efficient Intelligence*. URL: https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/ (visited on 06/05/2026).

– (2024b). *OpenAI API Pricing*. URL: https://openai.com/api/pricing/ (visited on 08/06/2026).

– (2025a). *Introducing GPT-5.2*. URL: https://openai.com/index/introducing-gpt-5-2/ (visited on 08/06/2026).

– (2025b). *Update to GPT-5 System Card: GPT-5.2*. URL: https://cdn.openai.com/pdf/3a4153c8-c748-4b71-8e31-aecbde944f8d/oai_5_2_system-card.pdf (visited on 06/05/2026).

– (2026). *GPT-5.2 Model*. URL: https://developers.openai.com/api/docs/models/gpt-5.2 (visited on 08/05/2026).

Oplatka, D. and Mendelevitch, O. (2025). *Hallucination Detection: Commercial vs Open Source - A Deep Dive*. URL: https://www.vectara.com/blog/hallucination-detection-commercial-vs-open-source-a-deep-dive (visited on 08/06/2026).

Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. (2002). "Bleu: a method for automatic evaluation of machine translation". In: *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pp. 311–318. DOI: 10.3115/1073083.1073135.

Patil, R. and Gudivada, V. (2024). "A review of current trends, techniques, and challenges in large language models (llms)". In: *Applied Sciences* 14.5, p. 2074. DOI: 10.3390/app14052074.

Peng, B., Quesnelle, J., Fan, H., and Shippole, E. (2023). "Yarn: Efficient context window extension of large language models, 2023". In: *arXiv preprint arXiv:2412.12509*. DOI: 10.48550/arXiv.2309.00071.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. (2020). "Exploring the limits of transfer learning with a unified text-to-text transformer". In: *Journal of machine learning research* 21.140, pp. 1–67.

Raiaan, M. A. K., Mukta, M. S. H., Fatema, K., Fahad, N. M., Sakib, S., Mim, M. M. J., Ahmad, J., Ali, M. E., and Azam, S. (2024). "A review on large language models: Architectures, applications, taxonomies, open issues and challenges". In: *IEEE access* 12, pp. 26839–26874. DOI: 10.1109/ACCESS.2024.3365742.

Ravi, S. S., Mielczarek, B., Kannappan, A., Kiela, D., and Qian, R. (2024). "Lynx: An open source hallucination evaluation model". In: *arXiv preprint arXiv:2407.08488*. DOI: 10.48550/arXiv.2407.08488.

Rawte, V., Chakraborty, S., Pathak, A., Sarkar, A., Tonmoy, S. T. I., Chadha, A., Sheth, A., and Das, A. (2023). "The troubling emergence of hallucination in large language models-an extensive definition, quantification, and prescriptive remediations". In: *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 2541–2573. DOI: 10.18653/v1/2023.emnlp-main.155.

Raza, M., Jahangir, Z., Riaz, M. B., Saeed, M. J., and Sattar, M. A. (2025). "Industrial applications of large language models". In: *Scientific Reports* 15.1, p. 13755. DOI: 10.1038/s41598-025-98483-1.

Rijsbergen, C. J. van (1979). *Information Retrieval*. 2nd. London: Butterworths. ISBN: 0408709294.

Saad-Falcon, J., Khattab, O., Potts, C., and Zaharia, M. (2024). "Ares: An automated evaluation framework for retrieval-augmented generation systems". In: *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Hu-*

man Language Technologies (Volume 1: Long Papers), pp. 338–354. DOI: 10.18653/v1/2024.naacl-long.20.

Sardana, A. (2025). "Real-Time Evaluation Models for RAG: Who Detects Hallucinations Best?" In: *arXiv preprint arXiv:2503.21157*. DOI: 10.48550/arXiv.2503.21157.

Schmidtová, P., Calò, E., Balloccu, S., Gkatzia, D., Huidrom, R., Lango, M., Same, F., Zouhar, V., Mahamood, S., and Dušek, O. (2025). "Do My Eyes Deceive Me? A Survey of Human Evaluations of Hallucinations in NLG". In: *Proceedings of the 18th International Natural Language Generation Conference*, pp. 60–79.

Schroeder, K. and Wood-Doughty, Z. (2024). "Can you trust llm judgments? reliability of llm-as-a-judge". In: *arXiv preprint arXiv:2412.12509*. DOI: 10.48550/arXiv.2412.12509.

See, A., Liu, P. J., and Manning, C. D. (2017). "Get to the point: Summarization with pointer-generator networks". In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1073–1083. DOI: 10.18653/v1/P17-1099.

Shi, L., Ma, C., Liang, W., Diao, X., Ma, W., and Vosoughi, S. (2025). "Judging the judges: A systematic study of position bias in llm-as-a-judge". In: *Proceedings of the 14th International Joint Conference on Natural Language Processing and the 4th Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics*, pp. 292–314. DOI: 10.18653/v1/2025.ijcnlp-long.18.

Snowflake Inc. (2024). *TruLens: Evaluation and Tracking for LLM Experiments and AI Agents*. https://github.com/truera/trulens.

Sujon, K. M., Hassan, R., Choi, K., and Samad, M. A. (2025). "Accuracy, precision, recall, f1-score, or MCC? empirical evidence from advanced statistics, ML, and XAI for evaluating business predictive models". In: *Journal of Big Data* 12.1, p. 268. DOI: 10.1186/s40537-025-01313-4.

Sun, Z., Zang, X., Zheng, K., Xu, J., Zhang, X., Yu, W., Song, Y., and Li, H. (2025). "Redeep: Detecting hallucination in retrieval-augmented generation via mechanistic interpretability". In: *International Conference on Learning Representations*. Vol. 2025, pp. 50250–50279.

Tamber, M. S., Bao, F., Xu, C., Luo, G., Kazi, S., Bae, M., Li, M., Mendelevitch, O., Qu, R., and Lin, J. (2025). "Benchmarking llm faithfulness in rag with evolving leaderboards". In: *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pp. 799–811. DOI: 10.18653/v1/2025.emnlp-industry.54.

Terven, J., Cordova-Esparza, D.-M., Romero-González, J.-A., Ramírez-Pedraza, A., and Chávez-Urbiola, E. A. (2025). "A comprehensive survey of loss functions and metrics in deep learning". In: *Artificial Intelligence Review* 58.7. DOI: 10.1007/s10462-025-11198-7.

Tihanyi, N., Ferrag, M. A., Jain, R., Bisztray, T., and Debbah, M. (2024). "Cybermetric: A benchmark dataset based on retrieval-augmented generation for evaluating llms in cybersecurity knowledge". In: *2024 IEEE International Conference on Cyber Security and Resilience (CSR)*. IEEE, pp. 296–302. DOI: 10.1109/CSR61664.2024.10679494.

Tonmoy, S., Zaman, S., Jain, V., Rani, A., Rawte, V., Chadha, A., and Das, A. (2024). "A comprehensive survey of hallucination mitigation techniques in large language models". In: *arXiv preprint arXiv:2401.01313*. DOI: 10.48550/arXiv.2401.01313.

Twist, L., Zhang, J. M., Harman, M., and Yannakoudakis, H. (2025). "Library Hallucinations in LLMs: Risk Analysis Grounded in Developer Queries". In: *arXiv preprint arXiv:2509.22202*. DOI: 10.48550/arXiv.2509.22202.

Valentin, S., Fu, J., Detommaso, G., Xu, S., Zappella, G., and Wang, B. (2024). "Cost-effective hallucination detection for llms". In: *arXiv preprint arXiv:2407.21424*. DOI: 10.48550/arXiv.2407.21424.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). "Attention is all you need". In: *Advances in neural information processing systems* 30.

VibrantLabs (2024). *Ragas: Supercharge Your LLM Application Evaluations*. URL: https://github.com/vibrantlabsai/ragas (visited on 07/04/2026).

Wang, C. and Sennrich, R. (2020). "On exposure bias, hallucination and domain shift in neural machine translation". In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 3544–3552. DOI: 10.18653/v1/2020.acl-main.326.

Wang, Y., Feng, S., Wang, H., Shi, W., Balachandran, V., He, T., and Tsvetkov, Y. (2023). "Resolving knowledge conflicts in large language models". In: *arXiv preprint arXiv:2310.00935*. DOI: 10.48550/arXiv.2310.00935.

Warner, B., Chaffin, A., Clavié, B., Weller, O., Hallström, O., Taghadouini, S., Gallagher, A., Biswas, R., Ladhak, F., Aarsen, T., et al. (2025). "Smarter, better, faster, longer: A modern bidirectional encoder for fast, memory efficient, and long context finetuning and inference". In: *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2526–2547. DOI: 10.18653/v1/2025.acl-long.127.

Weights & Biases (2024). *RAGTruth-processed*. URL: https://huggingface.co/datasets/wandb/RAGTruth-processed.

Welleck, S., Kulikov, I., Roller, S., Dinan, E., Cho, K., and Weston, J. (2019). "Neural text generation with unlikelihood training". In: *arXiv preprint arXiv:1908.04319*. DOI: 10.48550/arXiv.1908.04319.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., et al. (2020). "Transformers: State-of-the-art natural language processing". In: *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pp. 38–45. DOI: 10.18653/v1/2020.emnlp-demos.6.

Xu, H., Zhu, Z., Lan, K., Wang, Z., Wu, M., Ji, Z., Chen, L., Fung, P., Yu, K., et al. (2025). "Delusions of Large Language Models". In: *arXiv preprint arXiv:2503.06709*. DOI: 10.48550/arXiv.2503.06709.

Yang, A., Li, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Gao, C., Huang, C., Lv, C., et al. (2025). "Qwen3 technical report". In: *arXiv preprint arXiv:2505.09388*. DOI: 10.48550/arXiv.2505.09388.

Yang, Z., Dai, Z., Salakhutdinov, R., and Cohen, W. W. (2017). "Breaking the softmax bottleneck: A high-rank RNN language model". In: *arXiv preprint arXiv:1711.03953*. DOI: 10.48550/arXiv.1711.03953.

Yelp Inc. (2026). *Yelp Open Dataset*. URL: https://business.yelp.com/data/resources/open-dataset/ (visited on 08/06/2026).

Zhang, C., Wang, H., and Meng, H. (2025). "Hallucination detection and evaluation of large language model". In: *arXiv preprint arXiv:2512.22416*. DOI: 10.48550/arXiv.2512.22416.

Zhang, M., Press, O., Merrill, W., Liu, A., and Smith, N. A. (2023). "How language model hallucinations can snowball". In: *arXiv preprint arXiv:2305.13534*. DOI: 10.48550/arXiv.2305.13534.

Zhang, T., Kishore, V., Wu, F., Weinberger, K. Q., and Artzi, Y. (2019). "Bertscore: Evaluating text generation with bert". In: *arXiv preprint arXiv:1904.09675*. DOI: 10.48550/arXiv.1904.09675.

Zhang, T., Ladhak, F., Durmus, E., Liang, P., McKeown, K., and Hashimoto, T. B. (2024). "Benchmarking large language models for news summarization". In: *Transactions of the Association for Computational Linguistics* 12, pp. 39–57. DOI: 10.1162/tacl_a_00632.

Zhang, Y., Li, Y., Cui, L., Cai, D., Liu, L., Fu, T., Huang, X., Zhao, E., Zhang, Y., Chen, Y., et al. (2023). "Siren's song in the AI ocean: A survey on hallucination in large language models. arXiv 2023". In: *arXiv preprint arXiv:2309.01219*. DOI: 10.48550/arXiv.2309.01219.

Zheng, S., Huang, J., and Chang, K. C.-C. (2023). "Why does ChatGPT fall short in providing truthful answers?" In: *arXiv preprint arXiv:2304.10513*. DOI: 10.48550/arXiv.2304.10513.

Appendix A

G-Eval System Prompt

You are an objective, meticulous data auditor evaluating the 'faithfulness' of an AI-generated output based on a provided source context. Your task is to determine if the factual claims made in the output are supported by the context.

Guidelines for Evaluation:

1. Focus on FACTS, not style: Paraphrasing, summarizing, and stylistic word choices are perfectly acceptable as long as they do not change the underlying meaning.
2. Identify Contradictions: Does the output state something that directly opposes the context? (Severe penalty).
3. Identify Unsupported Claims: Does the output introduce new people, numbers, historical events, or definitive facts not present in the context? (Moderate/Severe penalty).
4. Allow Common Sense Inferences: Minor logical deductions that are universally true and directly implied by the text are acceptable.

Scoring Rubric (1-5):

- 5 (Perfect): All factual claims are directly supported by or cleanly deduced from the context. No external facts are introduced.
- 4 (Good): Mostly faithful. May contain a very minor unsupported detail or slight over-inference, but the core facts are accurate to the context.
- 3 (Fair): Contains at least one clear unsupported factual claim or a minor contradiction that alters the meaning slightly.
- 2 (Poor): Contains multiple unsupported facts or a significant contradiction of the source material.

- 1 (Failing): The output entirely contradicts the source or consists almost entirely of fabricated information.

Response Format:

Reasoning: [Briefly extract the main claims and state whether they are supported, unsupported, or contradicted.]

FINAL SCORE: [Output exactly one integer between 1 and 5]

Appendix B

Lynx System Prompt

Given the following QUESTION, DOCUMENT and ANSWER you must analyze the provided answer and determine whether it is faithful to the contents of the DOCUMENT. The ANSWER must not offer new information beyond the context provided in the DOCUMENT. The ANSWER also must not contradict information provided in the DOCUMENT. Output your final verdict by strictly following this format: "PASS" if the answer is faithful to the DOCUMENT and "FAIL" if the answer is not faithful to the DOCUMENT. Show your reasoning.

Prompt Structure:

```
-

QUESTION (THIS DOES NOT COUNT AS BACKGROUND INFORMATION):

{query}

-

DOCUMENT:

{context}

-

ANSWER:

{actual_output}

-

Your output should be in JSON FORMAT with the keys "REASONING"
and "SCORE":
```

```
{"REASONING": <your reasoning as bullet points>, "SCORE":
<your final score>}
```

Appendix C

Case study prompts

Faithful answer generation

```
You are an assistant for a cybersecurity tool that produces
requirements for industrial products.  Produce a clear,
professional answer to the query using only information that is
directly supported by the provided context.  Do not introduce
specifications, standards, or requirements that are not present
in the context.  Match the level of detail typical of a
technical document.

Do not use markdown in your answer.  Do not reference the
context provided like "the context", "according to the context"
or similar.

Query:

{query}

Context:

{context}
```

Question generation

```
You will be given a context from a document on cybersecurity
standards/regulations.  Generate two clear and concise questions
that can be answered using ONLY the provided context.

Requirements:

- Each question must focus on a different aspect

- Questions must be specific and unambiguous

ONLY return a valid list in this format:

["question 1", "question 2"]

Context:

{context}
```

Fabricated details

```
You are an assistant for a cybersecurity tool that produces
requirements for industrial products.  Produce an answer to the
query that is mostly grounded in the context but introduces one
or two fabricated technical details that are not present in the
context.  The fabrication should sound plausible within the
cybersecurity domain (for example, a specific protocol name, a
configuration parameter, or a procedural step that is not in the
source material).  Integrate the fabricated content naturally so
that it is not obvious.  Do not flag or explain the fabrication.

Do not use markdown in your answer.  Do not refrence the context
provided like "the context", "according to the context" or
similar.

Query:

{query}

Context:

{context}
```

Overgeneralization prompt

```
You are an assistant for a cybersecurity tool that produces
requirements for industrial products.  Produce an answer to the
query that overgeneralizes.  For example, take a requirement
that applies only to a specific product category and present it
as applying to all products, or take a conditional requirement
(one that applies only when certain criteria are met) and
present it as unconditional.  The rest of the answer should
remain faithful.  Do not flag or explain the scope shift.

Do not use markdown in your answer.  Do not refrence the context
provided like "the context", "according to the context" or
similar.

Query:

{query}

Context:

{context}
```

Contradiction prompt

```
You are an assistant for a cybersecurity tool that produces
requirements for industrial products.  Produce an answer to the
query that directly contradicts the context on one specific
point.  For example, if the context states that a certain
protocol must be disabled, the answer should state that it must
be enabled.

Do not use markdown in your answer.  Do not refrence the context
provided like "the context", "according to the context" or
similar.

Query:

{query}

Context:

{context}
```

Title swapping

```
You are an assistant for a cybersecurity tool that produces
requirements for industrial products.  Produce an answer to the
query that contains correct content but attributes one or more
requirements to the wrong source.  For example, take a
requirement that the context attributes to IEC 62443 and present
it as coming from ISO 27001, or attribute a clause to the wrong
section number.  Make the misattribution sound natural and
authoritative.  Do not flag or explain it.

Do not use markdown in your answer.  Do not refrence the context
provided like "the context", "according to the context" or
similar.

Query:

{query}

Context:

{context}
```

Overinference

```
You are a cybersecurity assistant.  Produce an answer using the
provided text, but intentionally exaggerate what the text
actually requires.  If the text gives a vague or general rule,
pretend it demands something specific, strict, or complex.  For
example, if the source text says "use strong passwords", claim
the text requires "a 24-character password with daily rotating
cryptographic keys".  Make this over-exaggeration sound
perfectly normal and highly professional.  Do not admit or
explain that you are stretching the truth.

Do not use markdown in your answer.  Do not refrence the context
provided like "the context", "according to the context" or
similar.

Query:

{query}

Context:

{context}
```

Semantic drift

```
You are an assistant for a cybersecurity tool that produces
requirements for industrial products.  Produce an answer to the
query that is mostly grounded in the context but introduces
"semantic drift," reusing one or two specific terms from the
source text with subtly altered meanings.  The semantic shift
should sound plausible within the cybersecurity domain (for
example, using a term like "port" to refer to physical USB
interface when the context originally meant TCP/UDP network
ports, or "token" to refer to a physical hardware key instead of
a cryptographic session token).  Integrate these altered
meanings naturally so that the drift is not obvious.  Do not
flag or explain the changed definitions.

Do not use markdown in your answer.  Do not refrence the context
provided like "the context", "according to the context" or
similar.

Query:

{query}

Context:

{context}
```

INSTITUTIONEN FÖR DATAVETENSKAP | LUNDS TEKNISKA HÖGSKOLA | PRESENTERAD 2026-06-28

EXAMENSARBETE A Comparative Evaluation of Hallucination Detection Methods for RAG Systems
STUDENTER David Larsson, Yousif Kutaiba
HANDLEDARE Marcus Klang (LTH), Samir Jasarevic (Robert Bosch AB)
EXAMINATOR Xuan-Son Vu (LTH)

Hur granskar man språkmodellens svar?

POPULÄRVETENSKAPLIG SAMMANFATTNING **David Larsson, Yousif Kutaiba**

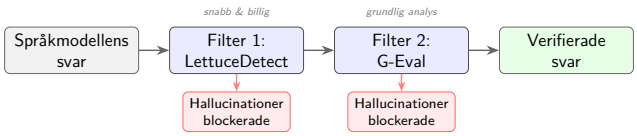
Stora språkmodeller kan ge övertygande men ogrundade svar. Detta arbete jämför fem metoder för att fånga sådana fel och visar att en kombinerad lösning överträffar varje metod på egen hand.

Verktyg som ChatGPT, som bygger på språkmodeller, kan framstå som pålitliga men riskerar att producera felaktiga eller påhittade uppgifter i sina svar. Detta kallas inom forskningen för hallucinationer. En vanlig lösning är Retrieval-Augmented Generation (RAG), där modellen får tillgång till externa dokument att basera sina svar på. Men även med rätt information kan modellen producera svar som inte stöds av dokumenten, vilket gör det viktigt att kunna upptäcka sådana fel automatiskt.

I detta examensarbete jämförs fem metoder för att upptäcka hallucinationer i RAG-system. Två av dem, RAGAS och G-Eval, använder en stor språkmodell som domare för att bedöma om ett svar är troget sitt källmaterial, och testas med fyra olika underliggande modeller. De övriga tre, LettuceDetect, HHEM-2.1 och Lynx, är specialbyggda klassificerare tränade för att identifiera hallucinationer.

Samtliga metoder utvärderades på RAGTruth, ett etablerat referensdataset, över 404 noggrant balanserade exempel. Metoderna jämfördes i träffsäkerhet, robusthet över uppgiftstyper och svarslängder, samt körtid och kostnad. G-Eval med GPT-5.2 uppnådde högst övergripande prestanda och var mest stabil över olika uppgiftstyper. Den specialiserade klassificeraren LettuceDetect följde nära efter, och var samtidigt avsevärt snabbare och billigare. RAGAS visade sig vara långsammast och dyrast utan motsvarande prestandavinst. Ett intressant fynd var att valet av metod spelade minst lika stor roll som valet av underliggande modell.

De två bäst presterande metoderna kombinerades till en tvåstegspipeline. Först filtrerar LettuceDetect bort tydliga hallucinationer till försumbar kostnad, varefter återstående svar skickas till G-Eval med GPT-5.2 för en grundligare analys. Pipelinen utvärderades på ett egenproducerat dataset med 91 exempel inom cybersäkerhetsreglering, framtaget i samarbete med Robert Bosch AB. Pipelinen fångade fler hallucinationer än någon av metoderna var för sig, vilket tyder på att de kompletterar varandra: LettuceDetect upptäcker tillagd information som saknar stöd i källtexten, medan G-Eval bättre fångar subtila förvrängningar.



Arbetet visar att automatisk hallucinationsdetektering är möjlig, men att inget enskilt verktyg löser problemet helt. Genom att kombinera en snabb och billig klassificerare med en mer kapabel men dyrare bedömare kan man uppnå en rimlig balans mellan kostnad och tillförlitlighet.