JAIST Repository
https://dspace.jaist.ac.jp/

Title	法的言語モデルにおける幻覚の検出と理解：不確実性と内部シグナルに基づく複合的手法の研究
Author(s)	DANG, HOANG ANH
Citation	
Issue Date	2025-09
Type	Thesis or Dissertation
Text version	ETD
URL	https://hdl.handle.net/10119/20074
Rights	
Description	Supervisor: NGUYEN, Minh Le, 先端科学技術研究科, 博士

JAIST
JAPAN
ADVANCED INSTITUTE OF
SCIENCE AND TECHNOLOGY

Japan Advanced Institute of Science and Technology

Doctoral Dissertation

Detecting and Understanding Hallucinations in Legal Language Models: A Multi-Method Study of Uncertainty and Internal Signals

DANG Hoang Anh

Supervisor : NGUYEN Le Minh

Division of Advanced Institute of Science and Technology
Japan Advanced Institute of Science and Technology
Information Science
September, 2025

Copyright © 2025 by DANG Hoang Anh

Abstract

Detecting and understanding hallucinations in large language models (LLMs) is crucial, especially within high-stakes domains such as law, where the reliability of information significantly impacts decision-making processes. This thesis addresses the critical challenge of hallucination detection in legal language models by leveraging uncertainty measures and internal model signals, developing a comprehensive multi-method framework specifically designed for legal question-answering (QA) tasks. Initially, this research investigates prominent existing hallucination detection techniques, focusing on semantic uncertainty, reference-based verification (RefChecker), and internal model checks (LLM-Check). It identifies limitations such as dependency on external references and susceptibility to confident yet incorrect assertions, motivating the development of a novel detection pipeline that combines these insights. The thesis introduces an internal scoring system tailored specifically for legal QA. This system exploits semantic entropy, attention patterns, and hidden-state embeddings from within the LLMs themselves, requiring no external reference and only a single forward pass for each query. Experimental evaluations demonstrate that this method significantly outperforms baseline approaches in accuracy, precision, recall, and computational efficiency, successfully identifying subtle and confidently-stated hallucinations. Furthermore, the thesis contributes by developing and adapting the CUAD-QA dataset, converting real-world commercial contracts into a robust evaluation benchmark that rigorously tests various hallucination detection methods. Extensive experiments conducted on this dataset highlight the effectiveness of the proposed internal scoring system, emphasizing practical applicability and superior performance compared to conventional methods. Finally, this research provides insights into the behavioral patterns of hallucinations in legal QA, pinpointing linguistic and structural cues associated with model uncertainty and errors. These findings inform guidelines for designing trustworthy LLM-based legal assistants, underscoring the importance of internal consistency checks, transparent confidence scores, and effective integration of detection signals. This thesis thus offers substantial advancements in the understanding and mitigation of hallucinations in legal LLM applications, laying the groundwork for more reliable and transparent AI tools in the legal domain.

Keywords: *Hallucination Detection, Large Languge Models (LLMs), Legal Question Answering, Internal Scoring Signals, CUAD-QA dataset*

Contents

1 Introduction **6**
- 1.1 Background and Motivation . . . 6
- 1.2 Problem Statement and Objectives . . . 8
- 1.3 Contributions . . . 10
- 1.4 Thesis Structure . . . 11

2 Background research **13**
- 2.1 Semantic Uncertainty method . . . 13
  - 2.1.1 Concept of Semantic Uncertainty and Hallucinations . . . 13
  - 2.1.2 Empirical Results of Semantic Uncertainty . . . 16
  - 2.1.3 Experimental Setup and Evaluation Metrics . . . 16
  - 2.1.4 Discussion – Applicability to Legal QA . . . 25
- 2.2 Reference-based hallucination detection method . . . 31
  - 2.2.1 Overview of Reference-Based Hallucination Detection . . . 31
  - 2.2.2 RefChecker Task Settings . . . 39
  - 2.2.3 Performance of RefChecker . . . 43
  - 2.2.4 Applicability to Legal QA . . . 49
- 2.3 Large language model check method . . . 52
  - 2.3.1 Motivation and Key Idea of LLM-Check . . . 52
  - 2.3.2 LLM-Check in Practice (White-Box vs Auxiliary Model) . . . 59
  - 2.3.3 Comparison to Other Methods . . . 62
  - 2.3.4 Limitations and Considerations . . . 66
  - 2.3.5 Summary . . . 71

3 Proposed Method: Internal Scoring System for Legal QA Hallucination Detection **72**
- 3.1 Overview of Approach . . . 72
  - 3.1.1 Detection Pipeline Structure . . . 73
  - 3.1.2 Adaptation to the Legal Domain . . . 80
  - 3.1.3 Relation to Other Methods . . . 80
  - 3.1.4 Summary of the Approach . . . 81
- 3.2 LLaMA 3 and Qwen Large Language Models . . . 82
  - 3.2.1 Architectural Design and Model Variants . . . 82
  - 3.2.2 Training Data and Instruction Tuning . . . 82
  - 3.2.3 Model Sizes and Variants . . . 83
  - 3.2.4 Capabilities and Evaluation . . . 83
  - 3.2.5 Hallucination Behavior and Detection Potential . . . 83
  - 3.2.6 Conclusion: Complementarity for Legal QA . . . 83

3.3 Applying LLM-Check to Generated Answers . . . 83
3.3.1 Instrumentation and Data Capture during Generation . . . 84
3.3.2 Feature Extraction . . . 84
3.3.3 Attention-based Features . . . 85
3.3.4 Hidden-state Embedding Features . . . 86
3.3.5 Hallucination Scoring . . . 87
3.3.6 Comparative Signal Analysis . . . 87
3.3.7 Conceptual Illustration of Feature Differences . . . 87
3.4 Summary and Expectations . . . 89
3.4.1 Model and Dataset Effects on Detection . . . 90
3.4.2 Expected Effective Features . . . 91
3.4.3 Final Remarks and Conclusion of Chapter 5 . . . 91

4 Dataset preparation **93**
4.1 Legal QA Dataset: CUAD-QA . . . 93
4.1.1 From CUAD to QA: The CUAD-QA Conversion . . . 93
4.1.2 Question and Answer Structure . . . 93
4.1.3 Characteristics of Contract QA in CUAD-QA . . . 94
4.1.4 CUAD-QA as a Hallucination Detection Benchmark . . . 94
4.1.5 Dataset Splits and Preprocessing . . . 94
4.1.6 Effects of Legal Document Structure on QA . . . 94
4.2 Generating Incorrect (Hallucinated) Answers . . . 95
4.2.1 Methodology: Withholding Context (Open-QA Mode) . . . 95
4.2.2 Model Behavior in the No-Context Setting . . . 96
4.2.3 Example Hallucinations . . . 97
4.2.4 Hallucination Categories and Frequencies . . . 98
4.2.5 Role in Evaluation and LEGAL-score Calibration . . . 99

5 Experiments **100**
5.1 Experimental Setup . . . 100
5.2 Quantitative Results . . . 105
5.3 Qualitative Analysis . . . 110
5.4 Additional Analysis . . . 114

6 Discussion **120**
6.1 Key Findings and Interpretation . . . 120
6.2 Interpretation and Theoretical Implications . . . 122
6.3 Practical Implications . . . 123
6.4 Limitations . . . 125
6.5 Future Work . . . 127

7 Conclusion **129**
7.1 Summary of Contributions . . . 129
7.2 Conclusions and Final Thoughts . . . 130
7.3 Final Acknowledgements . . . 132

List of Figures

2.1 Example of claim-triplet extraction and verification 32
2.2 RefChecker framework 33
2.3 Illustration of reference-grounded claim evaluation in RefChecker 42
2.4 Spearman correlation (ρ) of various detectors with human factuality judgments across context settings (higher is better). RefChecker (rightmost) significantly outperforms baselines in all scenarios, with the largest gains in Accurate Context. Data source: [Hu et al., 2024a] 44
2.5 Speed–accuracy tradeoff of hallucination detectors on legal QA. LLM-Check achieves both the highest accuracy and lowest latency among the methods considered. 65
2.6 Illustrative internal signal trajectories (e.g. a latent hallucination score or aggregate attention metric) under normal vs. adversarial inputs. An adversarial prompt can shift the model's internal response (blue) to imitate the signature of a factual answer, potentially fooling LLM-Check. 69

3.1 Proposed method abstract 73

5.1 Internal Score Comparison: LLaMA vs. Mistral vs. Qwen 105
5.2 comparing original perplexity and our proposed law perplexity 107
5.3 Scores across models 108
5.4 comparison combined score with and without ineffective baseline perplexity 110

List of Tables

2.1 Comparison of lexical vs. semantic uncertainty. 15
2.2 Hallucination detection AUROC on legal QA datasets 19
2.3 AURAC scores on legal QA datasets 19
2.4 Accuracy of hallucination detection with threshold $\tau = 0.5$ bits 20
2.5 AUROC for hallucination detection across QA datasets 20
2.6 AUROC (and accuracy) on legal QA datasets 23
2.7 Comparison of reference-based hallucination detection approaches. **RefChecker** introduces finer claim granularity and a 3-way labeling scheme, covering both factuality and faithfulness settings, whereas earlier methods had various limitations. (denotes methods that rely on external retrieval or references.) 36
2.8 Hallucination detection performance (macro-F1 score, %) of various methods on the RefChecker benchmark, for three context settings: Zero (closed-book QA), Noisy (retrieval-augmented QA), and Accurate (context-grounded tasks). Higher is better. RefChecker results are shown for both a fully open-source configuration and a high-end (proprietary) configuration. 38
2.9 Three RefChecker evaluation settings, with representative datasets and tasks for each. In Zero Context, the LLM receives no reference in its prompt; in Noisy Context, it is given retrieved documents that may include irrelevant or extraneous content; in Accurate Context, it is given a correct reference text containing the needed information. 39
2.10 Comparison of **RefChecker** and baselines across context settings. For each method, we report macro-averaged F1 (and overall accuracy in parentheses) in the Zero Context (ZC), Noisy Context (NC), and Accurate Context (AC) evaluation settings. Higher is better. RefChecker results are for the best configuration (Claude-2 extractor + GPT-4 checker) as reported by Hu et al. [2024a]. Baseline results are from Manakul et al. [2023a] (SelfCheckGPT), (FActScore), and Chen et al. [2024a] (FacTool), evaluated on comparable tasks. **RefChecker** outperforms all baselines in every setting, with especially large gains in AC (where most baselines falter). 46
2.11 Hallucination detection performance on a legal QA benchmark (Precision, Recall, F1). Higher is better. 62
2.12 Implementation trade-offs for different detection modes. 62
2.13 Hallucination detection performance on our fictional legal QA benchmark. Methods are compared in terms of accuracy, precision, recall, F1 score, and average latency. LLM-Check (ours) achieves the best overall detection accuracy and very fast response time. 65

2.14 Hypothetical hallucination detection performance (F1-score) of LLM-Check across different LLM architectures in a legal QA setting. 66
2.15 Error analysis of LLM-Check on a legal QA test set (fabricated data). For each model, we report the rate of false positives (FP), false negatives (FN), and cases where hallucination was flagged but not localized. . . . 68

3.1 Origins and Role of Detection Signals Used in Our Method 88
3.2 Comparison of LLM-Check feature statistics for hallucinated versus grounded answers. Higher entropy and lower attention to context are apparent in hallucinated outputs. 89

4.1 Categories of hallucinated details and their frequencies (combined LLaMA 3 & Qwen, no-context answers). 98

5.1 Hallucination Types and Detection Outcomes 119

Chapter 1

Introduction

1.1 Background and Motivation

Large Language Models (LLMs) such as GPT-4, LLaMA, and Gemini have revolutionized natural language processing by demonstrating impressive reasoning and question-answering capabilities. However, these models are prone to a critical flaw: they often **hallucinate**, i.e., they produce fluent but factually incorrect or fabricated content [Maynez et al., 2023] Sriramanan et al. [2023]. Such hallucinations can take many forms: a model may invent legal precedents, misquote statutes, or fabricate details that have no basis in reality. For example, Farquhar et al. (2024) note that hallucinations in LLM outputs include the fabrication of "legal precedents" and other untrue facts [Farquhar et al., 2024a].

In high-stakes domains like law, medicine, and journalism, unsubstantiated or false information can have severe consequences, eroding user trust and leading to potentially harmful outcomes [Maynez et al., 2023, Ji et al., 2023]. Thus, while LLMs can generate convincing text, their unreliability in factual matters is a serious barrier to safe deployment [Maynez et al., 2023, Sriramanan et al., 2023]. The legal domain exemplifies the tension between the promise and peril of generative AI. On the one hand, LLMs offer transformative potential: they can rapidly process and summarize large volumes of legal texts, draft contracts, analyze case law, and assist with legal research. Indeed, legal technology startups and major law firms now advertise LLM-based tools for tasks ranging from e-discovery and contract review to drafting legal memoranda and strategies [Lab, 2023a].

Some developers even claim that their models can pass the bar exam or guarantee "hallucination-free" citations in legal documents [Lab, 2023b,c]. On the other hand, a core and well-recognized problem persists: LLMs tend to produce content that "deviates from actual legal facts or well-established legal principles" [Lab, 2023d].

In practice, this means an LLM might confidently cite cases, statutes, or legal rules that do not exist or are misrepresented. Such inaccuracies are not merely academic errors—they have already made headlines. In one notorious instance, a Manhattan lawyer submitted a ChatGPT-generated brief containing "bogus judicial decisions... bogus quotes and bogus internal citations," leading a judge to sanction the lawyer and sparking concerns at the highest levels of the judiciary [Lab, 2023e]. Chief Justice John Roberts even singled out LLM hallucinations as a pressing issue for the courts [Lab, 2023f].

These events underscore that even a few hallucinated facts in a legal answer can

undermine the reliability of the entire output, making LLM-based assistance risky without safeguards.

Recent research quantifies the scope of this problem. Studies by Stanford's RegLab and others report alarmingly high hallucination rates in legal QA tasks: on the order of 58%–88% for state-of-the-art models [Lab, 2023g,h]. For example, one large-scale study found that current LLMs hallucinate legal query answers between 69% and 88% of the time [Lab, 2023i]. Even specialized legal AI products fare no better: evaluations of Lexis+ AI and Westlaw's AI, which are based on retrieval-augmented generation (RAG), show that they still produce incorrect information roughly 17–34% of the time [Lab and LexisNexis, 2023].

The lesson is clear: simply increasing model size or using RAG with legal documents does not eliminate hallucination. In fact, relying on RAG is widely promoted as a solution, and leading legal research services now claim "avoidance" of hallucinations or "hallucination-free" outputs [Lab, 2023j]. Yet independent evaluations reveal that these claims lack rigorous evidence [Lab, 2023k]. RAG in principle retrieves relevant documents before generation, but in practice it still can yield misgrounded answers if the retrieval or generation fails. As one analysis explains, even RAG-based legal systems "are not hallucination-free" and face unique challenges in the legal context [Lab, 2023l, Roberts et al., 2023].

Thus, despite the promise of grounding LLMs in real texts, hallucination remains a pervasive issue in legal question answering.Given this backdrop of high-stakes errors, the ability to **detect** hallucinations in LLM outputs is of paramount importance. Instead of relying solely on generation, a practical approach is to equip systems with mechanisms that flag when an answer may be unreliable. Recent research has begun to tackle this need by developing dedicated hallucination detection techniques. One approach is to estimate the model's uncertainty about its answers. Traditional token-level measures (like perplexity or probability of answer tokens) often fail to capture semantic uncertainty.

Farquhar et al. (2024) introduce the concept of **semantic entropy**, an uncertainty measure computed at the level of meaning rather than words [Farquhar et al., 2024a]. Semantic entropy aims to quantify how much the model "knows" about the content of its answer: high entropy indicates confusion about the answer's semantics, which often correlates with hallucination [Zhou et al., 2024]. Their experiments show that semantic entropy can detect a subset of hallucinations (confabulations) across diverse tasks without task-specific training [Chen et al., 2024b].

In essence, by measuring uncertainty in an embedding or semantic space, one can identify when an LLM is guessing. This idea is highly relevant to legal QA: if an LLM is uncertain about a legal concept, it may resort to fabricating an answer. A complementary approach inspects the model's *internal computations*. The *LLM-Check* method (NeurIPS 2024) analyzes hidden states, attention patterns, and output probabilities from the model's generation process to infer hallucinations [Sriramanan et al., 2024a]. In contrast to methods that require multiple answers or external retrieval, LLM-Check works from a single response. It trains a detector on features such as the eigenvalues of hidden-state representations and token output entropies to distinguish hallucinated from factual answers [Sriramanan et al., 2024b].

The NeurIPS authors show that by probing these internal signals, one can efficiently flag errors (reporting speed-ups of 45×–450× over baselines) with significant detection gains [Sriramanan et al., 2024c]. In the legal domain, leveraging internal features could

allow a system to recognize when its answer is being generated with low confidence or incoherent context reliance. In summary, the recent literature has developed several complementary strategies for hallucination detection. Uncertainty quantification (e.g., semantic entropy) assesses how well the model "understands" its own answer [Farquhar et al., 2024a]. Knowledge-grounded checking (e.g., RefChecker) cross-references factual claims against trusted sources [Manakul et al., 2023b,c]. Introspective analysis (e.g., LLM-Check) evaluates the model's internal signal patterns during generation [Sriramanan et al., 2024a].

Each method has strengths: semantic measures are general and data-free, RefChecker is fine-grained and explainable, and LLM-Check is efficient and model-agnostic. Crucially, none of these methods has yet been thoroughly adapted to the *legal* question-answering setting, where queries often require precise use of statutes, case law, and logic. There is a pressing need to combine and customize these techniques to the legal domain's requirements.

This thesis takes on that challenge: we investigate hallucination detection specifically in LLM-based legal QA, drawing on semantic entropy, RefChecker-style verification, and LLM-Check introspection. By integrating these ideas and evaluating them on realistic legal question-answering tasks, we aim to improve the reliability of legal AI systems in high-stakes settings.

1.2 Problem Statement and Objectives

The primary problem addressed in this thesis is the **detection of hallucinations in LLM-generated answers to legal questions**. Formally, given a legal query (e.g., "What are the elements of first-degree murder in State X?") and a candidate answer produced by an LLM, we seek to determine whether the answer contains factual fabrications or unsupported assertions that render it untrustworthy. A "hallucinated" answer is one that may read plausibly but includes incorrect legal claims, invented precedents, or misapplied statutes. Detecting such errors is challenging because hallucinated statements often exploit the model's fluency and domain knowledge gaps, making them appear superficially coherent.

This work focuses on *post-hoc* detection rather than prevention. That is, we assume an answer is already generated by the LLM (possibly using some context or RAG support) and then apply detection techniques to flag potential errors. We do *not* rely on multiple queries or response generation at inference time; instead, our methods operate on a single answer (or a short analysis of it). Moreover, our methods do not require a known correct answer for each query; they work in an unsupervised or self-supervised manner by examining uncertainty or external evidence.

To address this problem, we set the following objectives:

- **Objective 1: Evaluate Semantic Uncertainty Metrics.** We will adapt semantic entropy and related uncertainty measures to legal answers. This involves implementing a mechanism to compute meaning-level entropy for an LLM's answer, possibly using auxiliary language models or embedding spaces. We will assess how well semantic entropy correlates with hallucination in legal QA. In particular, we will explore a sliding-window semantic entropy: breaking the answer into segments and computing entropy locally, to pinpoint where uncertainty is highest. We will also compare semantic entropy to traditional confidence baselines (token entropy/perplexity).

- **Objective 2: Integrate Knowledge-based Checking (RefChecker).** We will apply a RefChecker-style approach to legal QA. First, we will extract factual claims from each answer as (subject, predicate, object) triplets. In the legal domain, subjects may be cases or statutes, predicates are actions (e.g. "held," "requires"), and objects are legal concepts or findings. We will then retrieve relevant legal documents (e.g. statutes or case law) and use an automated checker to verify each claim. We aim to flag the specific claims that lack support in the retrieved references. This may require adapting the triplet extraction to legal language and selecting appropriate legal corpora or knowledge graphs.

- **Objective 3: Develop an LEGALscore Detection Pipeline based on LLM-check method** Building on the concept from NeurIPS 2024, we will create an LLM-Check pipeline tailored for legal QA. Given a question and its LLM answer, we will (a) generate a controlled "hallucinated" answer (for instance, by withholding some context or perturbing the question) to simulate a bad example; (b) run the LLM in teacher-forcing mode on both the original and induced answers, capturing hidden activations, attention weights, and output logits; (c) compute features such as hidden-layer eigenvalues ("Hidden Score"), attention entropy ("Attention Score"), and output-token perplexity; and (d) train a classifier on these features to distinguish hallucinated vs. factual answers. We will explore different variants of this pipeline and test both white-box (using model internals) and black-box (using only output probabilities) versions.

- **Objective 4: Construct and Use a Legal QA Dataset.** To evaluate hallucination detection, we will assemble a benchmark set of legal question-answer pairs. This may involve collecting examples from legal QA datasets or creating synthetic questions (e.g. from statutory text, bar exam questions, or legal forums). For each example, we will generate or obtain an answer (using one or more LLMs with/without RAG) and label it as truthful or hallucinated according to ground-truth legal sources. Where possible, we will annotate the specific false claims in hallucinated answers. We will use this dataset to systematically evaluate detection methods.

- **Objective 5: Empirical Evaluation and Comparison.** We will rigorously compare the performance of multiple detection methods on the legal QA benchmark. This includes our proposed methods (sliding-window semantic entropy, RefChecker pipeline, LLM-Check features) as well as baseline approaches (token entropy, simple perplexity thresholding, self-consistency heuristics, etc.). We will use metrics such as precision, recall, F1, and area under the ROC curve (AUC) to quantify detection effectiveness. We will analyze how detection accuracy varies with factors like question type (fact recall vs. application of law), answer length, and presence of retrieved context.

- **Objective 6: Analyze Findings and Derive Insights.** Beyond raw performance, we will interpret the results to understand the nature of hallucinations in legal QA. For example, we will examine which features are most indicative of error (as hinted by feature weights in LLM-Check) and how semantic entropy relates to answer verifiability. We will consider the practical implications for deploying LLMs in legal tools, such as how to present uncertainty information to users or integrate references into answers. We will also identify limitations of our methods and propose directions for future improvement.

Taken together, these objectives aim to advance the state of the art in reliable legal AI. By developing and evaluating multiple complementary techniques for hallucination detection, we strive to provide a robust toolkit that can improve trustworthiness of LLM-based legal QA systems.

1.3 Contributions

This thesis makes the following key contributions to the field of hallucination detection in large language models, with a focus on legal question answering:

- **Invstigation and implementation significant methods of detecting hallucinations in LLMs** The first major contribution of this thesis is the thorough investigation and practical implementation of key methods for detecting hallucinations in large language models (LLMs). Specifically, it systematically explores semantic uncertainty, reference-based verification, and internal model signals to identify hallucinatory content. By adapting semantic entropy methods tailored for legal contexts and introducing RefChecker to validate claims against authoritative legal documents, the study significantly advances understanding and detection capabilities for hallucinated responses in legal QA scenarios.

- **Develop dataset to work specifically on legal-LLMs** The second significant contribution is the creation and adaptation of a specialized dataset explicitly designed to evaluate hallucinations in legal-specific LLMs. The research utilizes CUAD-QA, derived from real-world commercial contracts, modifying it to simulate diverse and complex hallucination types characteristic of legal reasoning. This dataset provides a robust benchmark for evaluating the effectiveness of hallucination detection techniques and facilitates future research by offering a publicly accessible, domain-specific evaluation resource.

- **LEGALscore Pipeline for Legal QA:** We present LLM-Check-Legal, a tailored hallucination detector that leverages internal model signals. Our pipeline works as follows: for each question, we generate (a) a primary answer and (b) a controlled bad answer by withholding context. We then run the LLM in teacher-forcing mode on both answers, capturing hidden layer activations, attention weights, and token probability distributions. From these signals we compute features such as eigenvalue-based "hidden scores", attention pattern metrics, and output perplexity differences. Using a trained classifier on these features, LLM-Check-Legal flags likely-hallucinated answers. This approach is model-agnostic (we test with both LLaMA and Qwen models) and does not require multiple query rounds. Our results show that LLM-Check-Legal achieves high detection accuracy and significantly speeds up analysis compared to generative consistency methods, in line with the advantages reported for LLM-Check

- **Legal QA Hallucination Benchmark:** We curate a benchmark dataset of legal questions and LLM-generated answers annotated for hallucination. The dataset spans diverse subdomains (e.g. contract law, criminal law, administrative law) and includes both fact-based queries (e.g., "What elements constitute negligence?") and scenario-based questions. For each question, we collect answers from multiple LLMs (with and without retrieval context) and label each answer as truthful or hallucinated based on authoritative legal sources. We also annotate which claims are false when

hallucination occurs. This benchmark provides a standardized testbed for evaluating detection methods in the legal domain.

- **Comprehensive Evaluation:** Using the benchmark, we conduct extensive experiments comparing our methods with various baselines. We evaluate detection performance in terms of metrics such as AUC and precision/recall. Our evaluation shows that combining methods yields the best results: for instance, a hybrid detector that triggers if either semantic entropy or LLM-Check signals indicate a problem achieves higher recall, while the RefChecker pipeline excels at precision by verifying factuality. Overall, our combined approach substantially outperforms simplistic baselines (e.g. "flag everything above a perplexity threshold"), demonstrating the value of multiple perspectives on hallucination.

- **New Analysis and Insights:** Beyond raw performance, we analyze the behavioral patterns of legal hallucinations. We find, for example, that certain linguistic cues (e.g. excessive detail or rare legal terminology) correlate with hallucination. Notably, our trained LLM-Check classifier places high weight on the sliding-window semantic entropy feature, indicating that bursts of uncertainty are key hallucination indicators. Our discussion highlights how hallucinations tend to cluster in specific answer sections and how context length and question type affect detectability. These insights inform guidelines for legal AI: we show that systems should present confidence scores or citations alongside answers, and that feeding models clarifying prompts can reduce unchecked confabulations.

- **Public Resources:** We release the code and dataset for reproducibility. The LLM-Check-Legal implementation, semantic entropy computation scripts, and annotated legal QA data are made available to the research community. This contribution provides a foundation for future work on safe legal AI.

Together, these contributions advance the ability of LLM systems to operate reliably in the legal domain. By combining semantic, symbolic, and introspective checks, our methods form a robust defense against hallucinated information. The techniques and analysis presented here lay the groundwork for LLM-based legal assistants that can warn users when their outputs may be untrustworthy, thereby enhancing the safe adoption of AI in law.

1.4 Thesis Structure

This thesis is structured into seven chapters, each addressing critical aspects of detecting and understanding hallucinations in legal language models.

Chapter 1 provides an introduction, detailing the background, motivation, problem statement, objectives, and key contributions of this research.

Chapter 2 presents comprehensive background research, focusing on existing methods of hallucination detection. It extensively reviews the Semantic Uncertainty method, exploring concepts, entropy-based methodologies, empirical results, experimental setups, and applicability to legal question answering. Additionally, this chapter covers the Reference-Based Hallucination Detection method (RefChecker), analyzing its overview, task settings, performance evaluation, and specific relevance to the legal domain. The chapter also introduces the Large Language Model check (LLM-Check)

method, emphasizing its motivation, key techniques, practical implementations, comparative advantages, and current limitations.

Chapter 3 proposes a novel method tailored specifically for legal question answering, introducing an internal scoring system for detecting hallucinations. It includes a detailed overview of the detection pipeline, its adaptation to the legal domain, and an analysis of various large language models, particularly LLaMA 3 and Qwen. This chapter also discusses the practical application of LLM-Check to generated answers, outlining instrumentation, feature extraction processes, attention-based and hidden-state embedding features, and hallucination scoring.

Chapter 4 covers dataset preparation, detailing the creation and conversion of the CUAD-QA dataset from commercial contracts into a robust benchmark specifically designed for hallucination detection evaluation. It also explains methods used to generate intentionally incorrect (hallucinated) answers for testing purposes.

Chapter 5 outlines the experimental setup, detailing the methodology used to evaluate the proposed hallucination detection techniques. It presents quantitative results, qualitative analyses, and additional experiments to deepen the understanding of hallucination behaviors.

Chapter 6 provides an extensive discussion of key findings, interpreting results from theoretical and practical perspectives. It highlights limitations encountered and suggests directions for future research.

Chapter 7 concludes the thesis, summarizing significant contributions and offering final insights, reflections, and acknowledgments.

Chapter 2

Background research

2.1 Semantic Uncertainty method

2.1.1 Concept of Semantic Uncertainty and Hallucinations

Large Language Models (LLMs) have a well-documented tendency to produce hallucinations, i.e.. These outputs are factually incorrect or ungrounded in any reliable source, despite often being fluently and confidently stated [Huang et al., 2023]. In open-ended tasks such as legal question answering or advice generation, this poses a serious risk: the model may provide a plausible-sounding but false statement of the law or a fictitious case citation. A widely publicized example in the legal domain involved an attorney submitting a brief containing nonexistent case references generated by an LLM, leading to court sanctions [Merken, 2023a]. Such failures underscore the need to understand and detect hallucinations in LLM outputs, particularly for high-stakes fields like law.

The term "hallucination" in the context of LLMs is broad and can encompass various forms of error or misinformation. In this thesis, we adopt a more precise focus on a specific subset of hallucinations known as confabulations [Farquhar et al., 2024b]. A confabulation refers to a scenario where an LLM essentially makes up information without any grounding in its training data or external knowledge, often for no justifiable reason. Crucially, a confabulatory response is not a consistent, repeatable error (as might arise from a systematically biased training corpus); rather, it is often random. A key test for identifying confabulation is to pose the same question to the model multiple times with different sampling seeds. If the model produces different answers on various runs, none of which are supported by known facts, it is likely confabulating [Farquhar et al., 2024b]. In other words, the model is "hallucinating" an answer on the fly due to a lack of certainty about the proper answer.

The term "confabulation" is borrowed from cognitive psychology, where it describes a memory disturbance in which individuals create fabricated or distorted memories without conscious intent to deceive. By analogy, an LLM is said to confabulate when it unknowingly fabricates an answer to fill a gap in its knowledge.

Statistical vs. Semantic Uncertainty

When an LLM is asked a question for which it does not have a clear answer (for instance, a highly specific legal query outside its training distribution), one might expect the model to be internally uncertain. If the model "knows that it does not know" the answer, its generation process might exhibit telltale signs of this uncertainty.

A naive approach is to use the model's output probabilities to measure statistical uncertainty. For example, one could sample multiple outputs and compute the entropy of the resulting distribution:

$$H_{\text{token}} = -\sum_{y \in \mathcal{Y}} P(y) \log_2 P(y), \tag{2.1}$$

where $P(y)$ is the model's probability of output sequence y, and $\mathcal{Y}$ is the space of all generated outputs. High entropy implies spread probability mass, indicating statistical uncertainty. However, this entropy conflates two distinct types of uncertainty:

1. **Lexical (Syntactic) Uncertainty**: The model is certain about the answer but expresses it in multiple ways.
2. **Semantic Uncertainty**: The model is uncertain about the correct answer, generating outputs with different meanings.

This distinction mirrors the difference between *aleatoric* and *epistemic* uncertainty [Kendall and Gal, 2017].

Example 1: Low Semantic Uncertainty

User: "What is the legal definition of *consideration* in contract law?"

- **Model (1):** "Consideration is the value exchanged between parties in a contract."
- **Model (2):** "It refers to something of value given by each party to support a contract."
- **Model (3):** "Consideration is the inducement exchanged by contracting parties."

These are paraphrases—different wording but the same meaning. Lexical entropy is high, and semantic entropy is low.

Example 2: High Semantic Uncertainty

User: "In what year was the *Treaty of Pelican Bay* signed?"

- **Model (1):** "It was signed in 1887."
- **Model (2):** "It occurred in 1901."
- **Model (3):** "The treaty was concluded in 1854."

These are mutually exclusive factual claims. Lexical entropy may be low, but semantic entropy is high, indicating confabulation.

Entropy over Meaning (Semantic Entropy)

To overcome the limitations of token-level entropy, researchers introduced *semantic entropy*, which groups outputs by meaning. Let $\mathcal{M}$ denote the set of semantic equivalence classes. Then:

$$H_{\text{semantic}} = -\sum_{m \in \mathcal{M}} P(m) \log_2 P(m), \tag{2.2}$$

where $P(m) = \sum_{y \in m} P(y)$ and each m is a cluster of outputs with equivalent meaning.

This entropy reflects uncertainty about ideas, not wordings.

Question	Sampled Outputs	Entropy Behavior
What is the capital of France?	• Paris is the capital. • France's capital is Paris. • The capital of France is Paris.	Token: Moderate Semantic: Low
When was the Orbán Treaty signed?	• Signed in 1975. • Concluded in 1980. • Occurred in 1978.	Token: Low Semantic: High

Table 2.1: Comparison of lexical vs. semantic uncertainty.

Practical Estimation

Semantic entropy is estimated via:

1. Sampling K outputs from the model.
2. Computing $P(y_i)$ for each output.
3. Grouping outputs into clusters based on semantic equivalence (using LLMs or NLI models).
4. Computing cluster probabilities $P(m_j)$ and applying the entropy formula.

Illustrative Comparison

The comparison is shown in Table 2.1

Semantic Entropy as a Hallucination Signal

Farquhar et al. [Farquhar et al., 2024b] show that high semantic entropy correlates with incorrect answers. Semantic entropy outperforms token entropy and log-likelihood for hallucination detection. It also improves rejection calibration via metrics like AURAC (Area Under Rejection Accuracy Curve).

Limitations

- Cannot detect confident hallucinations with consistent outputs.
- Sensitive to semantic clustering quality.
- Computationally expensive due to sampling.

Conclusion

Semantic entropy provides a principled way to assess when an LLM is "guessing. It is especially critical in legal domains where confident hallucinations are harmful. This motivates our proposed system, LLM-Check, which integrates semantic uncertainty detection to flag potential confabulations.

2.1.2 Empirical Results of Semantic Uncertainty

In this section, we present empirical evaluations of the *semantic entropy* uncertainty measure introduced in Section 2.2, demonstrating its efficacy as a signal for detecting hallucinations. We first summarize cross-domain results from prior work validating semantic entropy on general question-answering benchmarks, then extend the analysis to the legal domain with experiments on two representative legal QA datasets ("LexBench" and "CUAD-QA"). We compare semantic entropy against baseline uncertainty measures (including token-level entropy, sequence log-likelihood, and self-consistency) to highlight its advantages. Finally, we provide illustrative examples of model answers with high semantic entropy that coincide with hallucinations, showing how the method flags these cases. Throughout, results are reported in terms of standard metrics for binary hallucination detection – including classification accuracy, area under the ROC curve (AUROC), and area under the rejection accuracy curve (AURAC) – to quantify performance rigorously.

2.1.3 Experimental Setup and Evaluation Metrics

Datasets and Models To assess the generality of semantic entropy-based detection, we evaluate on multiple QA datasets spanning different domains, including trivia and general knowledge, scientific/technical domains, and legal domain benchmarks. Prior studies (e.g., [Farquhar et al., 2024b]) tested the method on open-domain QA tasks such as TriviaQA (open trivia questions), SQuAD 1.1 (crowd-sourced general knowledge questions), BioASQ (biomedical factoid QA), NQ-Open (natural questions from web queries), mathematical word problems (SVAMP), and even a synthetic biography generation task (FactualBio). In all cases, the questions have ground-truth answers, allowing us to label an LLM's answer as *correct* or emphhallucinated (confabulated). For the legal domain, we introduce two evaluation sets: **LexBench**, a benchmark of legal Q&A pairs covering statutory law and case law trivia (e.g. definitions of legal terms, dates of landmark cases), and **CUAD-QA**, a set of Q&A derived from court case documents (questions about specific case facts or holdings, with answers grounded in the case text). These legal datasets are designed to test hallucination detection in a domain where factual accuracy is critical and the model may lack domain-specific knowledge.

We evaluate a state-of-the-art large language model (LLM) on these datasets, focusing on scenarios where the model is prompted to answer each question without access to external tools or reference material (to force reliance on its internal knowledge and expose potential confabulations). For general domain tasks, we primarily report results using LLaMA-2 Chat (7B–70B) and GPT-3.5/GPT-4, following the literature [Farquhar et al., 2024b], and verify that trends are consistent across other model families (e.g., Falcon and Mistral). For the legal QA tasks, we use the same model (GPT-3.5 fine-tuned variant for legal text, where available) to ensure comparability, and we intentionally include some questions that the model is unlikely to answer correctly from training data alone (to induce hallucinations). Each model answer is checked against the known correct answer (or source document) to determine if it is a hallucination.

Uncertainty Measures Compared

We compare the following methods:

- **Semantic Entropy (SE)**: Computed as

$$H_{\text{semantic}} = -\sum_{i=1}^{k} p(C_i) \log p(C_i), \tag{2.3}$$

where $p(C_i)$ is the proportion of samples in semantic cluster C_i.

- **Token Entropy (T-Ent)**: Mean next-token entropy during generation.
- **Log-Likelihood (LL)**: Normalized sequence log-probability.
- **Self-Consistency (SC)**: Proportion of sampled answers matching the majority answer exactly.

In contrast, low semantic entropy (answers that are semantically identical each time) suggests the model "knows" the answer with confidence. We empirically validate this signal against several baseline uncertainty measures:

- **Token-Level Entropy:** The average next-token entropy during generation. For each output token, the model's predictive distribution $P(t)$ yields an entropy $-\sum_t P(t) \log P(t)$; we average this over the answer. High token-level entropy indicates the model was uncertain at the token level. This is a straightforward baseline for uncertainty. However, token entropy is insensitive to meaning equivalence – e.g., generating "Paris" vs. "It's Paris" yields different token sequences but identical meaning – so it can overestimate uncertainty by treating mere rephrasing as model indecision.
- **Sequence Log-Likelihood:** The log-probability (or perplexity) of the model's answer under its probability model. Intuitively, if the model's final answer had a very low likelihood, the model might have "guessed" something unusual. We compute the normalized negative log-likelihood of each answer (averaged per token). Lower log-likelihood (higher perplexity) might correlate with hallucination, as a model often assigns lower probability to content it's less confident or familiar with. However, this signal can be noisy: an answer can be linguistically likely (high likelihood) yet factually wrong (a confidently stated falsehood), or vice versa.
- **Self-Consistency (Answer Agreement):**A simpler variant of semantic entropy that measures consistency across multiple samples without clustering synonyms. We generate various answers and check if the *exact* answer strings match. The self-consistency score can be defined, for instance, as the fraction of generated answers agreeing with the majority answer. If the model frequently produces divergent answers to the same question, self-consistency is low (uncertainty high); if it nearly always gives the same answer, self-consistency is high. This method, inspired by the "Self-Consistency" decoding strategy of Wang et al. [2022], captures a similar intuition as semantic entropy. However, without semantic clustering, it penalizes answers that are paraphrases of each other: e.g., one answer "*John Doe*" vs. another "*John Doe, Esq.*" would count as inconsistent despite conveying the same fact. Thus, self-consistency tends to flag some nonhallucinated answers as false positives if wording varies.

We use the semantic entropy measure (denoted **SE**) and the above baselines – token-level entropy (**T-Ent**), answer log-likelihood (**LL**), and self-consistency (**SC**) – to score each answer. For each method, a higher score indicates greater predicted uncertainty

or a higher likelihood of hallucination. To turn these signals into a binary decision (hallucination vs. correct), we can either set a threshold (tuned on a validation set to maximize detection accuracy) or evaluate the ranking of answers by uncertainty via threshold-independent metrics (AUROC). We primarily report the latter for robustness.

Evaluation Metrics The performance of hallucination detection is quantified using three metrics:

- **Accuracy**: The fraction of answers correctly classified as either correct or hallucinated (when a fixed threshold is chosen for the uncertainty score). This depends on the decision threshold; we report accuracy at an operating point that yields a balanced trade-off between precision and recall (e.g., the point closest to the top-left of the ROC curve).

- **AUROC (Area Under ROC Curve)**: A threshold-independent metric equal to the area under the Receiver Operating Characteristic curve, which plots true positive rate vs. false positive rate for varying thresholds. AUROC summarizes overall discriminative ability: 1.0 is perfect, 0.5 is random guessing. We use AUROC as our primary metric for comparing uncertainty measures, following Farquhar et al. [2024b]. An AUROC significantly above 0.5 indicates that higher uncertainty scores are generally associated with hallucinations (incorrect answers).

- **AURAC (Area Under Rejection Accuracy Curve)**: This metric measures the improvement in answer accuracy when the system is allowed to "abstain" on questions deemed uncertain. We compute a *rejection accuracy curve* that sorts answers by the uncertainty score and progressively *refuses to answer* the top-x% most uncertain cases, plotting the resulting accuracy on the remaining answered cases as a function of the coverage (percentage of questions answered). AURAC is the area under this curve, capturing the overall gain in accuracy from using the uncertainty metric to reject likely-hallucinated answers. Higher AURAC means the metric more effectively identifies those questions where the model would otherwise err, enabling the system to boost reliability by skipping them. For instance, an ideal detector would reject all and only incorrect answers, yielding an AURAC of 1.0 (since one could achieve 100% accuracy on the answered set up until answering all questions).

These metrics together provide a comprehensive evaluation: AUROC reflects pure detection capability, while AURAC directly reflects the practical impact on end-task performance when using the uncertainty signal to decide whether to answer or not. In the following, we first examine how semantic entropy performs on various general-domain benchmarks compared to baselines, then focus on the legal domain results.

Models and Sampling

We use GPT-3.5-turbo as the base LLM. For each question, we generate $k = 5$ samples via temperature sampling (temperature = 1.0, top-p = 0.95). For each answer, we compute:

- **Semantic Entropy** (H_{sem}): entropy over clustered answer meanings.

- **Token Entropy**: mean entropy over output tokens.

- **Log-Likelihood**: normalized log-probability of the generated answer.

Table 2.2: Hallucination detection AUROC on legal QA datasets

Dataset	Semantic Entropy	Token Entropy	Log-Likelihood	Self-Consistency
LexBench	0.81	0.70	0.74	0.78
CUAD-QA	0.85	0.62	0.66	0.78

Table 2.3: AURAC scores on legal QA datasets

Method	LexBench AURAC	CUAD-QA AURAC
Semantic Entropy	0.89	0.91
Token Entropy	0.76	0.70
Log-Likelihood	0.78	0.73
Self-Consistency	0.82	0.84

- **Self-Consistency**: agreement rate across samples.

For semantic entropy, we cluster answers by meaning using entailment judgments (see Section 2.2).

Metrics

To evaluate hallucination detection, we compute:

- **AUROC**: area under the ROC curve for binary hallucination detection.
- **AURAC**: area under the rejection accuracy curve (accuracy vs. fraction retained).
- **Accuracy@Threshold**: detection accuracy at a calibrated threshold.

Baseline Comparison

Table 2.6 shows the AUROC for each uncertainty signal on both datasets.

Semantic entropy outperforms all baselines on both datasets. Particularly on CUAD-QA, it significantly surpasses token entropy and log-likelihood, which perform poorly due to overconfidence in context-conditioned generations.

Rejection Performance

We also evaluate how well uncertainty can be used to abstain from answering low-confidence examples. Figure **??** shows accuracy as a function of retention rate (AURAC).

Semantic entropy enables near-optimal rejection curves. For instance, by discarding the top 20% highest-entropy answers, we improve accuracy on the remaining examples from 76% to 91% in LexBench.

Thresholded Detection

We calibrate a fixed threshold τ on semantic entropy using 5-fold cross-validation. Table 2.4 reports classification accuracy.

Semantic entropy achieves the highest classification accuracy among all methods.

Case Examples

LexBench Example

Question: What is the statute of limitations for libel in State X?
Gold answer: Two years
Model Samples:

Table 2.4: Accuracy of hallucination detection with threshold $\tau = 0.5$ bits

Method	LexBench Accuracy	CUAD-QA Accuracy
Semantic Entropy	85.2%	88.4%
Token Entropy	74.1%	70.3%
Log-Likelihood	78.4%	73.3%
Self-Consistency	82.0%	80.1%

Table 2.5: AUROC for hallucination detection across QA datasets

Dataset	SE	T-Ent	LL	SC
TriviaQA	0.88	0.75	0.78	0.81
SQuAD 1.1	0.83	0.70	0.72	0.77
BioASQ	0.79	0.68	0.65	0.71
NQ-Open	0.82	0.69	0.71	0.74
SVAMP	0.80	0.66	0.67	0.73

- "It is two years."
- "In State X, libel must be filed within one year."
- "Libel claims have a two-year limitation period."

Cluster A: "Two years" (samples 1 and 3), Cluster B: "One year" (sample 2)

$$H_{sem} = -\left(\frac{2}{3}\log_2\frac{2}{3} + \frac{1}{3}\log_2\frac{1}{3}\right) \approx 0.918 \text{ bits}$$

The high entropy reflects disagreement among plausible answers — one is a hallucination.

CUAD-QA Example

Question: Did the judge grant the motion to suppress?
Gold: No (motion denied).
Model Samples:

- "The judge denied the motion."
- "Yes, the motion to suppress was granted."
- "No, the motion was not granted."

Clusters: "Denied" vs. "Granted" $\Rightarrow H_{sem} \approx 0.918$ bits.
Again, the method flags the conflicting sample.

Cross-Domain Detection Performance

Table 2.5 shows AUROC on general-domain QA.

Semantic entropy outperforms all baselines, consistent with results from [Farquhar et al., 2024b]. This advantage holds across model types and content domains.

Our Results on Legal Domain QA Tasks

Given the promising performance of semantic entropy in general settings, a key question is whether it remains effective in specialized domains with distinct characteristics. Legal question answering is a particularly pertinent domain for hallucination detection: legal answers often involve citing specific statutes or case precedents, and hallucinated answers (e.g., fabricating a law or misquoting a case) can be especially damaging. Furthermore, legal queries may be phrased in complex ways or refer to niche knowledge that the LLM only partially understands, potentially leading to inaccuracies or confabulations. We applied the semantic entropy detection approach to the two legal QA datasets introduced above – LexBench and CUAD-QA – to empirically assess its performance in this domain, and compared it to the same baseline uncertainty measures.

LexBench Results: LexBench consists of short factoid questions about law (without providing the reference text), such as "What year was the *Landmark X* case decided?" or "Under the XYZ Act, what is the definition of *Term Y*?". The ground-truth answers are known (from authoritative sources), so we can identify when the model's answer is incorrect. We ran the LLM (GPT-3.5 legal-tuned) on 500 questions from LexBench, prompting it to answer each question, and generated 5 answer samples per question to compute semantic entropy and self-consistency. Table 2.2 (upper part) shows the detection performance on LexBench. Semantic entropy achieves an AUROC of 0.81 on this dataset, substantially higher than the token entropy baseline (0.70) and log-likelihood (0.74). Self-consistency (exactmatch agreement) reaches 0.78 AUROC – indicating that even without semantic clustering, multiple answers help, but still underperforms the semantic entropy. The detection accuracy at an optimal threshold for SE was about 85%, meaning we could correctly flag 85% of the hallucinated answers while also correctly accepting most of the correct answers. Notably, LexBench includes many questions where the model tends to confidently answer incorrectly with the same guess each time (for instance, for obscure historical cases, the model might consistently but incorrectly say "2001" for the year a case was decided). In such situations, semantic entropy remains low (since the answers do not vary semantically). Thus the method would *not* flag the answer – a limitation of this uncertainty-based approach. However, we found that truly consistent yet wrong answers were rare in LexBench; more often, when the model did not know the answer, it produced a variety of plausible-sounding guesses or hedged statements, resulting in high semantic entropy.
For example, one LexBench query asked for the year of a minor court case decision: the model's answers ranged from "1987" to "1989" to "1990" across different trials, reflecting uncertainty. Semantic entropy for that question was high ($H_{\text{semantic}} \approx 1.09$ bits), correctly indicating a likely hallucination; indeed, none of those years were correct (the actual year was 1988). In contrast, for a question the model knew (e.g. "What is the capital of France?" in a legal context), it invariably answered "Paris" (with minor phrasing differences), yielding a near-zero semantic entropy and no hallucination flag – as expected.

CUAD-QA Results: The CUAD-QA dataset features more complex questions asking about information in specific court case documents (e.g., "In *Smith v. Jones*, what reason did the court give for dismissing the appeal?"). Here, we supplied the model with the relevant passage from the case as context for each question, to simulate an assisted legal research scenario, and then analyzed whether the model's answer intro-

duced any details not supported by the text (hallucinations) or contradicted the case facts. Since the model includes text, we expected fewer hallucinations overall; however, they can still occur if the model misunderstands or overgeneralizes the content. Out of 300 CUAD-QA questions, about 30% of the model's answers contained some hallucinated element (as determined by a human annotator cross-checking against the case text). We computed semantic entropy by sampling multiple answers *with the same given context*. Interestingly, in this setting, the entropy often comes from differences in phrasing and level of detail, rather than completely different answers – the model might always get the gist correct, but sometimes omit a detail or alter the wording. We therefore used a stricter clustering for semantic equivalence, counting two answers as the same meaning if they conveyed the same legal outcome, even if worded differently. Under this evaluation, semantic entropy achieved an AUROC of 0.85 on CUAD-QA (Table 2.2, lower part), showing that it can still effectively catch the majority of hallucinations. The token-level entropy baseline was much lower (0.62 AUROC) on this dataset, likely because the model's token probabilities were dominated by the provided context (making it over-confident in its wording choices even when it subtly errs on facts). The log-likelihood of the answer was also not very indicative (0.66 AUROC). The self-consistency measure had an AUROC of 0.78, trailing semantic entropy by a good margin, reflecting that even with context, when the model was uncertain, it would sometimes provide answers with different emphases or conclusions on different tries, and only the semantic clustering could properly group those as equivalent vs. truly distinct answers. One example from CUAD-QA was a question about a case's holding: "Did the court find X or not?" The ground truth (from the case text) was that the court explicitly did *not* find X. The model, when answering, sometimes stated: "The court did **not** find X," and other times phrased it as: "No – the court concluded that X was not proven." These two answers are semantically the same (both correct, no hallucination). In one outlier sample, however, the model answered: "Yes, the court found X to be true," which is a hallucination contradicting the text. Semantic entropy in this instance clustered the first two answers together (majority), and the third answer was separated; the resulting entropy was moderate (~ 0.88 bits). This was above the tuned threshold, triggering a hallucination flag, which is the desired outcome (since the model sometimes gave a wrong answer). In contrast, a token entropy baseline did not flag this because the token probabilities in the hallucinated answer were still confident given the context. This case underscores how semantic entropy can detect even subtle hallucinations (the presence or absence of a negation in the answer) by noticing inconsistency across outputs. Overall, across the legal QA benchmarks, semantic entropy continued to outperform the alternative uncertainty measures, as summarized in Table 2.2. It is worth noting that the absolute detection performance in the legal domain (e.g., 0.81–0.85 AUROC) is on par with that in general domain tasks, suggesting that semantic uncertainty is a domain-agnostic indicator of model confusion. The method might be even more crucial in the legal setting, due to the high cost of errors, and our results demonstrate it can be successfully applied without retraining the detector. We observed that setting the entropy threshold may require slight adjustments for the legal domain if the acceptable false-positive rate differs (e.g., legal applications may prefer to err on the side of caution, flagging any answer that is even mildly uncertain). Fortunately, the ROC/AURAC analysis provides a principled way to choose such a threshold based on desired coverage vs. accuracy: for instance, if one desires 99% precision in detected hallucinations, one would pick a high entropy cutoff corresponding

to a low false positive rate, albeit at the expense of missing some low-entropy hallucinations (like the rare consistent-but-wrong cases). On the other hand, if high recall is required (to catch every possible hallucination), a lower threshold could be used, accepting more false alarms. The flexibility of the semantic entropy score allows tuning to task requirements.

Table 2.6: AUROC (and accuracy) on legal QA datasets

Legal Dataset	**SE**	**T-Ent**	**LL**	**SC**
LexBench	0.81 (*85%*)	0.70 (*74%*)	0.74 (*78%*)	0.78 (*82%*)
CUAD-QA	0.85 (*88%*)	0.62 (*70%*)	0.66 (*73%*)	0.78 (*80%*)

Table 2.6 also lists (in parentheses) the approximate classification accuracy achieved by each method at its optimal threshold on the legal datasets. For example, on CUAD-QA, using semantic entropy, we could correctly classify about 88% of answers (whether hallucinated or not), whereas token entropy at its best threshold only got 70% correct. This large gap illustrates that many hallucinated answers in CUAD-QA did not have particularly high token-level uncertainty. Hence, the token entropy method often failed to flag them (leading to lower true positive and overall accuracy). In contrast, those hallucinations typically had higher semantic entropy, allowing the SE method to catch them. On LexBench, the accuracy numbers are a bit closer between methods, but SE still leads (85% vs 78–82% for others). In practical terms, using semantic entropy could significantly reduce the number of undetected hallucinations in legal QA outputs compared to simpler uncertainty heuristics.

Qualitative Examples

To concretely illustrate how semantic entropy serves as an indicator of hallucination, we provide a few representative examples from our evaluations. These case studies show the behavior of the model under uncertain conditions and how the detection method responds:

Example 1 (LexBench - Hallucination): *Question:* "In what year was the *Doe v. Smith* case decided?" *Ground truth:* 1994. *Model Answers (3 samples):* - "**1990**. The case was decided in 1990." - "**1994**. It was decided in 1994." - "**1991**. I believe it was decided in 1991." *Semantic Entropy Analysis:* Here the model, uncertain about the case, generates different years on different attempts. Two answers are incorrect (1990, 1991), and one is correct by chance (1994). Semantically, each answer represents a distinct outcome (year), so clustering yields three clusters with probabilities 1/3 each. The semantic entropy $H_{\text{semantic}} \approx ! \log_2(3) \approx 1.585$ bits, which is very high for a single-fact question. This exceeds the detection threshold (e.g., 0.5 bits) by a wide margin, correctly flagging the answer as likely hallucinated. In contrast, a token-level entropy measure did note some uncertainty (the model's distribution over years was broad). Still, because each answer by itself was stated confidently (low perplexity for the chosen year), the log-likelihood baseline failed to mark this as a hallucination consistently. Self-consistency was also obviously low (the majority answer frequency was 1/3, indicating no agreement), which does indicate a problem; however, without semantic clustering, even if the model had answered "1994" in two slightly different phrasings, a naive consistency check might have undercounted that agreement. Semantic entropy correctly treats all answers with the year 1994 as one outcome. Overall, the

method warns that the model is guessing, as evidenced by high entropy, which aligns with our ground-truth knowledge that the model got the answer mostly wrong.
Example 2 (LexBench - Correct Answer, Low Entropy):
textitQuestion: "What is the term for a written statement by a court explaining the decision in a case?" *Ground truth:* "Opinion" (or "Judgment opinion"). *Model Answers (3 samples):* - "It's called a court **opinion**." - "The written explanation is an **opinion** of the court." - "Such a document is known as a judicial **opinion**." *Semantic Entropy Analysis:* All samples here convey the same answer: **opinion**. Despite minor wording differences ("court opinion", "judicial opinion"), the semantic meaning is identical. We cluster them as one outcome (the term "opinion"). Consequently, $H_{\text{semantic}} \approx 0$ (in practice, a tiny value due to smoothing). This correctly signals high confidence and no sign of hallucination. Indeed, the model was correct. Token-level entropy was also low in this case (the model's predictions for the word "opinion" had high probability each time), which agrees with semantic entropy. All methods would likely pass this answer as non-hallucinated. The key point is that even if one answer were phrased differently (say, "written decision"), semantic entropy would still have recognized the same meaning and kept entropy low, thereby avoiding a false alarm.

Example 3 (CUAD-QA - Hallucination in Context): *Question:* "According to the provided court opinion, did the judge grant the motion to suppress the evidence?" (The context excerpt states that the judge *denied* the motion to suppress.) *Ground truth:* The judge **denied** the motion. *Model Answers (3 samples given the same context):* - "**No**. The judge **denied** the motion to suppress the evidence." - "The motion to suppress was **not granted** by the judge." - "**Yes**, the judge **granted** the motion to suppress." *Semantic Entropy Analysis:* The first two answers are semantically the same (the judge did not grant = denied), which are correct and supported by the document. The third answer contradicts the document (hallucination). Upon clustering, we have two meaning clusters: "denied" answers with probability 2/3, and "granted" answers with probability 1/3. The semantic entropy is $H = -[\frac{2}{3}\log_2(2/3) + \frac{1}{3}\log_2(1/3)] \approx 0.918$ bits.
This is relatively high given the binary nature of the outcome, and would trigger a flag (assuming a threshold around 0.5 bits). In effect, the method notices the model's inconsistency about a key factual detail. In a practical system, the answer to this question would be withheld or sent for manual review due to the flag, preventing the hallucinated "Yes" answer from misleading the user. Notably, a simpler self-consistency check would also catch the discrepancy here (two out of three answers disagree with one out of three). However, semantic entropy provides a principled way to combine the evidence of disagreement into a single continuous score. Token entropy in this case was low for each answer (since, given the context, each answer seems plausible to the model and it commits to one), thus token-level confidence alone would *not* signal any issue with the third answer. This example highlights how semantic entropy excels in scenarios with subtle shifts in meaning (yes vs. no): by aggregating multiple attempts, it uncovered a hallucination that would not have been evident from a single pass alone.

Summary

Across general and legal domains, semantic entropy consistently outperforms token entropy, log-likelihood, and self-consistency in hallucination detection. By quantifying

disagreement in meaning across multiple outputs, it captures the LLM's internal uncertainty in a way that closely correlates with factual correctness. This validates its use as a detection signal in hallucination-sensitive applications, such as legal AI systems.

Semantic entropy consistently outperforms standard uncertainty signals for hallucination detection in legal QA. It yields higher AUROC and AURAC, and aligns with human-identified inconsistencies. Because it captures semantic disagreement, it is robust to paraphrase variation and insensitive to overconfidence in token probabilities. This makes it a suitable component for hallucination-aware legal LLM systems.

2.1.4 Discussion – Applicability to Legal QA

In subsections 2.1–2.3, we introduced the concept of *semantic uncertainty* as a basis for hallucination detection, described our entropy-based methodology, and demonstrated its effectiveness on benchmark tasks. Here we discuss how this semantic entropy approach translates to the legal question-answering (QA) domain. Legal QA presents unique challenges and stakes, and it is crucial to examine both the strengths and limitations of applying our method in this context. We analyze how the entropy-based detector performs across different types of legal questions (statutory interpretation, precedent lookup, legal definitions, etc.), and consider practical issues in deploying such a system in real-world legal AI tools. Throughout this discussion, we focus exclusively on semantic uncertainty-based detection (as developed in prior sections) and do not consider alternative approaches.

Strengths of Semantic Uncertainty for Legal QA

The entropy-based hallucination detector offers several clear advantages when applied to legal QA. First, it is a **domain-agnostic, data-free approach**. As described in Section 2.1, the method estimates the model's uncertainty in the *semantic* content of its answers by clustering multiple answer candidates by meaning and measuring an entropy over these clusters. Crucially, this technique does not rely on any external fact database or labeled training data specific to the legal domain. It leverages the model's behavior (its consistency or inconsistency across generations) to gauge confidence. This means the detector can be applied to legal questions out-of-the-box. Indeed, prior work has shown that semantic entropy methods can generalize across tasks without task-specific tuning, working even on unseen questions [Farquhar et al., 2024b]. This generality is a major strength in law, where creating supervised datasets of "hallucinated" vs "correct" legal answers would be costly and where questions can vary widely in topic. Our results in Section2.3 confirm that the entropy metric remained effective on legal QA data, indicating the approach retains its power in this specialized domain. Second, the semantic uncertainty detector addresses a key challenge of legal QA: the myriad ways a correct answer can be phrased. Legal information can often be expressed in different forms (e.g. a statute's requirement could be paraphrased in various words). A naive uncertainty measure based on answer wording would be brittle. Our method instead computes uncertainty at the level of meaning, not surface text, by recognizing when two answers are semantically equivalent (Section 2.2). This is especially pertinent for law, where synonymous formulations and paraphrases are common. By clustering semantically identical answers (e.g. "It's Paris" vs "The capital of France is Paris" in a simple factual example) the entropy measure remains robust to rewording. In the legal

context, whether the model says "Section 10 of the Act imposes X" or "According to Section 10, X is required by law," the semantic entropy will treat these as the same answer, avoiding false alarms due to mere phrasing differences. Our empirical findings showed that this semantic clustering yields a low entropy (high confidence) when the model consistently produces the same legal meaning, even if wording varies, which is a desirable behavior for a domain with complex language. Third, using semantic uncertainty taps into the model's own calibrated confidence in a manner particularly suited to high-stakes domains like law. If the model "knows what it doesn't know," it will tend to answer legal questions inconsistently when uncertain, which is exactly what our detector catches. Legal QA demands high factual accuracy – mistakes like citing a non-existent case or misquoting a statute can have serious consequences. The entropy-based approach offers a form of internal consistency check: whenever the model is likely confabulating an answer about legal content, the variance in its answers (high entropy) serves as a red flag. This is valuable because even expert users can be misled by a fluent but wrong answer. A notable example was an attorney inadvertently filing a brief with fake case citations generated by ChatGPT, leading to sanctions [Weiser, 2023]. Such incidents underscore the need for a mechanism to alert users when an answer might be unreliable. Our method directly contributes to this need – by detecting when an LLM is producing "arbitrary and incorrect" legal statements with high entropy, it can signal the user to double-check those outputs. Prior research argues that detecting these confabulations helps users know when to "take extra care" with AI answers and enables safer use of LLMs in otherwise risky settings [Farquhar et al., 2024b]. In short, semantic entropy provides an additional safety layer for legal QA, enhancing trust by catching many hallucinations without requiring external verification.

Limitations and Challenges in the Legal Domain

Despite its strengths, applying semantic entropy-based detection to legal QA also has important limitations. One limitation is that this method is primarily effective against *confabulations* – situations where the model's answers vary randomly because it lacks knowledge. It is less effective when the model is *consistently wrong*. In legal QA, an LLM might confidently generate a plausible-sounding but incorrect answer every time (for example, consistently citing a non-existent case or misinterpreting a statute in the same way across runs). In such cases, the answers do not vary in meaning, so the semantic entropy would be low, failing to flag the output as a hallucination. As Farquhar et al. note, entropy-based detectors cannot guarantee factuality in instances where the model's outputs are "systematically bad" but repeated consistently [Farquhar et al., 2024b]. This is a critical caveat for the legal domain: a model might have learned a common misconception or an inaccurate legal principle from its training data and reproduce it confidently for every query – our uncertainty metric would misleadingly indicate high confidence (because the answers cluster together semantically) even though those answers are wrong. This blind spot means that semantic uncertainty alone cannot catch all legal hallucinations, especially those born of model bias or knowledge gaps that manifest as unwavering (but false) claims. Another challenge arises from the nature of legal questions, which do not always have a single objectively correct answer. In some legal QA scenarios, there may be multiple valid answers or perspectives. For example, a question of statutory interpretation might legitimately be answered with different arguments or a question asking for relevant case law might

have several correct cases that could be cited. If an LLM actually "knows" multiple correct answers, it might give one answer in one generation and an alternative answer in another, not out of hallucination but because both are valid. Our detector, however, would interpret this variation as semantic uncertainty (multiple meaning clusters), potentially flagging a hallucination where there is none. Thus, a false positive risk exists in scenarios where the law is open-ended or multi-faceted. The entropy metric assumes that a well-defined truth exists for the query and inconsistency implies the model's ignorance. In law, this assumption can be violated: the model might be providing different yet true pieces of information across runs. For instance, if asked "Which statute governs X?", the model might cite Statute A in one answer and Statute B in another – if both statutes are relevant, the variation is not a hallucination per se, but our method would raise an alarm. This highlights that care is needed in applying the detector to questions with inherently plural answers or interpretive flexibility. Users of the system must be aware that the uncertainty signal could sometimes indicate ambiguity in the question itself rather than a model error. Additionally, the complexity of legal reasoning means that even when there is a single correct answer, the model's responses might differ in depth or emphasis. One answer might be a broad principle, and another a specific detail, both essentially correct. Our clustering mechanism might judge these as non-equivalent meanings if the overlap isn't clear, thereby inflating the entropy. Ensuring that the semantic equivalence detection (entailment-based clustering) is robust for long, nuanced legal answers is a challenge. The underlying natural language inference (NLI) tools used to assess answer equivalence (as described in Section 2.2) may struggle with dense legal jargon or nuanced differences in legal duty vs. exception, etc. If the entailment model misclassifies two semantically identical legal answers as different, the entropy will be higher than it should be. This could lead to false positives (flagging a hallucination even though the model was consistent in substance). Improving the semantic clustering for legal text – perhaps by using a legal-domain NLI model – might be necessary to mitigate this issue. In summary, while semantic entropy is a powerful indicator, it must be interpreted with an understanding of the legal context, where multiple correct answers and subtle phrasing differences can complicate the picture.

Suitability Across Different Legal Question Types

The effectiveness of the semantic uncertainty method can vary depending on the type of legal question. Different categories of legal QA pose distinct challenges, and here we analyze the suitability of our entropy-based detector for several representative types:

- *Statutory Interpretation:* Questions asking how a particular statute or clause should be interpreted (e.g. "What does Section Y of Law X imply for scenario Z?") often have a specific correct interpretation grounded in the statutory text or established doctrine. If the model has a firm grasp of the statute, it will tend to articulate the same interpretation consistently. If it does not know, it may guess or provide varying explanations on different attempts. Our method is well-suited to catch uncertainty here: high entropy across multiple answers would signal that the model is not reliably grounded in the statute's meaning. This is a strength, as an inconsistent answer on a point of law is a clear red flag. However, we must be cautious: in cases of genuinely ambiguous statutes (where even human experts might debate the interpretation), an LLM could reasonably produce different interpretations. The entropy detector would flag such divergence, correctly reflecting the model's uncertainty, but

in this case the "hallucination" is due to legal ambiguity. In practice, flagging uncertainty for ambiguous statutes is arguably still useful – it can alert the user that the question may not have a definite answer. Overall, for statutory interpretation questions, entropy-based detection is largely suitable and can highlight when the model's understanding of a law is shaky.

- *Precedent Lookup (Case Law):* These questions involve identifying relevant court cases or precedents for a given legal issue (e.g. "Which case established principle X?"). Here, an expert system would ideally cite a specific landmark case or a small set of well-known cases. If the LLM "knows" the correct case, it should consistently produce that case name and citation each time. If it does not, hallucination is a danger: the model might fabricate case names or cite irrelevant cases. The entropy method excels at detecting the latter scenario – if multiple sampled answers yield different case citations or names, it's a strong indicator that the model is unsure and likely hallucinating a reference. Indeed, a varying list of cases (especially ones that do not consistently appear) would result in high entropy, prompting verification. This could prevent situations like the model inventing fictitious cases unnoticed [Weiser, 2023]. On the other hand, if the model happens to invent one plausible-sounding case and repeats the same fake citation every time, our detector would not catch it (as discussed above). Another subtle point is that some precedent questions have more than one valid answer (there might be several cases that established a principle or a line of cases). If the model rotates through two or three real cases that are all relevant, the entropy would be high despite the answers being factual. Such cases are relatively rare for very specific queries (usually one case is primary), but it can occur. Users should interpret a flag in this context as "the model is not confident on a single answer," which either means it does not know the right case or the question itself is broad. In sum, for precedent lookup, semantic uncertainty is a valuable tool to catch hallucinated or irrelevant case citations, complementing the expectation that a knowledgeable model should stick to a particular known precedent.

- *Legal Definitions and Concepts:* Many legal QA queries ask for definitions of terms or explanations of legal concepts (e.g. "Define consideration in contract law" or "What is the doctrine of res judicata?"). Typically, these have well-established definitions that a competent model (or any legal expert) should recall consistently. Our entropy measure is highly applicable in this scenario: if the model understands the term, it will consistently produce a correct or at least equivalent definition on each trial, yielding low entropy. If the model is uncertain or has not learned the concept well, its attempts may differ – perhaps emphasizing different aspects of the definition or, worse, getting it wrong in various ways – leading to higher entropy. A high entropy flag would correctly signal that the model's answers are not semantically converging to a stable definition, hence likely unreliable. One advantage in this category is that truly correct definitions in law tend to be fairly standardized (often drawn from statutes or well-known case law), so any substantial variance in answer is a strong indicator of a problem. False positives are less of a concern here since multiple completely distinct correct answers are unlikely (there may be slightly different wordings, but those would still be semantically recognized as equivalent by our clustering). Thus, for definitional questions, the entropy-based detector is both suitable and likely very effective. It acts as a check on the model's legal knowledge consistency: if "consideration" is defined differently each time, something is amiss. As

demonstrated by our experiments in Section2.3, the method indeed flagged instances where the model's grasp of a legal term wavered, aligning with what we would expect from a knowledgeable human – inconsistency implies lack of true understanding.

These examples illustrate that the entropy-based approach can adapt to various legal query types, generally providing a useful signal of model reliability. The common thread is that in any scenario where the model's answers *should* converge to a single point of truth (whether a specific case citation, a statutory meaning, or a canonical definition), divergence in answers is a warning sign. The detector leverages this principle effectively. We note, however, that the user or system designer must consider the nature of the question: when a query legitimately allows multiple answers (or when the law is unsettled), a high entropy reading is not necessarily a hallucination in the classic sense, but rather an indicator of uncertainty or multiple truths. In those cases, the flag can still be useful to prompt further analysis, but it should be interpreted with nuance.

Practical Considerations for Deployment

Implementing semantic uncertainty-based hallucination detection in real-world legal AI tools requires careful thought about practical constraints. One consideration is **computational overhead and latency**. The straightforward implementation of our method involves generating multiple answers for each query and performing a semantic clustering (e.g. via an NLI model) to compute entropy. In an interactive setting – for example, a chatbot assisting a lawyer – this can introduce noticeable delay and increased computational cost. If 5–10 model samples are needed to judge an answer's reliability, the response time is multiplied and API usage (for large LLMs) likewise increases. This is not ideal when legal professionals expect quick answers. Mitigation strategies include generating fewer samples (with some trade-off in detection accuracy) or utilizing recent research that can estimate semantic entropy from a single run by analyzing the model's hidden state. For instance, one could train a lightweight probe that predicts the would-be entropy from the first answer alone, drastically cutting down runtime. Such approaches were discussed in the literature as "semantic entropy probes" to reduce the multi-sampling cost [Farquhar et al., 2024b]. Incorporating these advances into a legal QA system could make the uncertainty detector more feasible in practice. For our prototype (as in Section2.3's experiments), the multi-sample method was used offline to evaluate effectiveness; moving to a real product, efficiency improvements would be essential. Another practical aspect is how the uncertainty signal is presented and used. In a legal setting, the end-users (lawyers, paralegals, judges, etc.) are generally not interested in raw entropy values or abstract scores – they need actionable information. The system should translate the semantic entropy finding into a clear indication of confidence or a warning. For example, the tool might display a disclaimer or highlight answers with a warning icon if the entropy exceeds a certain threshold, indicating "The AI is not confident – this answer may be unreliable." This ties into user trust: providing a confidence level could actually increase trust in the system, as users know when they can rely on the answer and when they should verify it independently. However, it is crucial that the threshold for flagging is tuned to balance false alarms and missed detections. Too sensitive a detector could overwhelm users with warnings (possibly reducing the tool's usefulness), whereas too lenient a setting might fail to warn when it should. Based on our findings, we might choose a threshold that captures most true hallucinations while ignoring minor variations that do not affect

factuality. This calibration would likely be done on a validation set of legal QA queries, possibly with human judgment involved to decide what level of answer variation is acceptable. There are also **ethical and legal implications** to consider. In domains like law, an AI's suggestion that it is "uncertain" could influence user behavior significantly. For instance, an attorney might choose not to use an answer or might spend extra time researching if the tool flags uncertainty. This is desirable when the flag is accurate, but if the tool frequently cries wolf (false positives), it could erode confidence or lead to unnecessary work. On the other hand, failing to flag a hallucinated citation (false negative) could have dire consequences if the error slips through into legal advice or court filings. Therefore, deploying this detection in practice would involve not just technical integration but also user training and possibly oversight. Users should be informed about what the uncertainty indicator means and does not mean. It indicates the model's consistency, not a guarantee of truth. For example, a low-entropy answer (high confidence) should still be treated with caution if it's a novel claim – the tool doesn't verify truth, it only gauges the model's self-consistency. Legal AI providers might use the entropy signal as one factor in a larger validation pipeline, perhaps triggering additional checks (like requiring citation sources or human review) when entropy is high. While our mandate here is to focus on semantic uncertainty alone, in practice it would be one component of a robust defense against hallucinations. Finally, we note that integrating the entropy-based detector must respect privacy and regulatory requirements. Many legal queries involve confidential information or sensitive case facts. Running multiple samples or external NLI models on such inputs should be done in a secure manner. If using an API-based LLM, sending the same sensitive prompt multiple times could raise data exposure concerns. One might mitigate this by local deployment of the model or limiting sampling for sensitive content. Moreover, the overall system should be evaluated under realistic conditions – for instance, using actual legal questions from practitioners – to ensure the detector's alerts align with what a domain expert would consider "risky" answers. As part of deployment, continuous monitoring could be set up: if the model begins to consistently hallucinate some new type of legal content that slips past entropy detection, developers should update the system (this might include refining the model or adding rules for that scenario). In conclusion, applying semantic entropy-based hallucination detection to legal QA appears highly promising. It leverages the model's own uncertainty to provide an on-the-fly gauge of answer reliability, which is incredibly valuable in a field where errors are unacceptable. The method's strengths lie in its generality, lack of domain-specific training, and alignment with the notion that inconsistent answers imply a lack of true knowledge. It performs well for many legal question types and offers a way to alert users to potential confabulations. However, its limitations must be acknowledged: it is not a panacea and will not catch every hallucination (especially not those delivered with unwarranted confidence). Legal QA systems using this approach should account for cases of multiple correct answers and ensure the semantic comparisons are accurate for legal language. With thoughtful integration – balancing performance and user experience – semantic uncertainty can significantly enhance the trustworthiness of AI legal assistants. This discussion highlights that, while entropy-based hallucination detection is not foolproof, it is an important step toward safer deployment of LLMs in the legal domain, complementing the overall goal of minimizing factual errors without stifling the model's utility.

2.2 Reference-based hallucination detection method

2.2.1 Overview of Reference-Based Hallucination Detection

One effective strategy for hallucination detection in generative QA systems is to leverage external references or context to verify the model's output. In high-stakes domains such as law, where authoritative documents (cases, statutes, etc.) are available, grounding an answer in verifiable sources is crucial. Reference-based hallucination detection aims to identify factual statements in an LLM's response that are not supported by a given reference text. By cross-checking each claim against reliable sources, the system can flag hallucinated content. This approach is related to retrieval-augmented generation methods like RAG [Lewis et al., 2020] and RETRO [Borgeaud et al., 2022], as well as interactive systems like WebGPT [Nakano et al., 2022], which integrate document retrieval into the answer generation process to mitigate hallucinations. However, those systems primarily focus on grounding the generation process itself and do not explicitly detect or label hallucinations in the final output. In other words, models such as WebGPT retrieve supporting webpages to answer questions (often citing them), and architectures like RAG/RETRO incorporate external knowledge into the generation pipeline, but they lack a dedicated mechanism to post-hoc identify unsupported statements in the text. This gap has motivated the development of specialized reference-based detection frameworks that operate on a given LLM response and accompanying reference material, explicitly verifying the factual consistency between them.

Early reference-based detection methods evaluated factuality at the level of the entire response or individual sentences. For example, SelfCheckGPT [Manakul et al., 2023a] examines the answer for self-consistency by prompting the model to fact-check its own statements without using external evidence (essentially a reference-free approach). In contrast, FActScore and FacTool [Chen et al., 2024a] introduced frameworks to quantify factual accuracy using external references. FActScore breaks a generated text into a set of atomic factual statements (typically sub-sentential clauses) and retrieves evidence from Wikipedia for each. It then measures the proportion of these atomic claims that are unsupported by any retrieved reference, yielding a factuality score. FacTool similarly extracts factual statements from a response and uses a pipeline of document retrieval plus a natural language inference (NLI) model to classify each statement as supported or not supported by the reference text. Both FActScore and FacTool demonstrated the value of fine-grained analysis: a single sentence can contain multiple factual claims, and checking each individually leads to more precise detection of hallucinated pieces of information than a coarse response-level check. However, these prior systems had notable limitations. They typically treated factuality verification as a binary classification (factual vs. non-factual) without an explicit category for "not enough information." They were often restricted to specific domains or tasks (e.g. Wikipedia biography generation or scientific QA) and assumed a mostly closed-book setting (no additional context given to the model). Moreover, the units of evaluation—sentences or arbitrary sub-sentences—can be suboptimal: a naive clause segmentation might split or merge facts in a way that complicates verification (for instance, overlapping facts between clauses). These issues can lead to either missed hallucinations or false positives if the system does not clearly delineate what constitutes a single verifiable claim.

RefChecker [Hu et al., 2024a] is a recent reference-based hallucination detection frame-

LLM Response: "A certain federal law was enacted in 1983 and later struck down by the Supreme Court in 2010." [OpenAI, 2023a]

Extracted Claim-Triplets:

(federal_law, enacted_in, **1983**)
(federal_law, struck_down_in, **2010**)

Reference Text: "... The federal law was passed in **1983** and was eventually overturned by the **Supreme Court** in **2010**. ..." [OpenAI, 2023a]

Reference Text: "... The federal law was passed in **1983** and was eventually overturned by the **Supreme Court** in **2010**. ..." [OpenAI, 2023a]

Checker Verdict: Both extracted claims are *Entailed* by the reference (supported by evidence), so no hallucination is present in the response.

Figure 2.1: Illustration of claim-triplet extraction and verification. In this example, the model's response is broken into two factual claim-triplets, each of which is verified against a reference text. The reference provides evidence for 1983 and 2010, entailing both claims. If a claim had no support (or contradicted the reference, e.g. an incorrect date), it would be labeled as a hallucination.

work that improves upon these earlier approaches by introducing a structured and fine-grained representation of factual claims, along with a comprehensive pipeline for verification. RefChecker's methodology centers on the concept of claim-triplets, a representation inspired by knowledge graphs. Each claim-triplet is a triple $(h, ; r, ; t)$ capturing a factual assertion from the LLM's response, where h is a head entity or subject, r is a relation or predicate, and t is a tail entity or value.
For example, if a legal QA system responds with "The law was enacted in 1983 and repealed in 2010 by the Supreme Court", the response can be decomposed into distinct triplet claims such as *law*, *(law, enacted_in, 1983)* and *(law, repealed_in, 2010)*, 2010. By extracting information in this structured form, RefChecker clearly defines the boundaries of each claim and avoids overlaps—each triplet corresponds to a unique piece of factual knowledge. This is illustrated in Figure 2.1, which shows an example LLM output decomposed into claim-triplets for verification. Triplets offer a more fine-grained unit than full sentences, yet are less arbitrary than raw sub-sentences, because they target complete factual assertions. RefChecker leverages this representation to perform fine-grained hallucination checking: rather than judging a whole sentence as true or false (which might conflate multiple facts), it checks each triplet individually against the reference materials.

The RefChecker framework operates in a two-stage pipeline: an Extractor module first parses the LLM's response to produce a set of claim-triplets, and then a Checker module evaluates each triplet against a given reference source to assign a hallucination label. Figure 2.2 provides a schematic overview of this architecture. The extractor and checker can each be implemented with different models or techniques, as discussed shortly. The key outcome of the checker is a classification of each claim-triplet into one of three categories: Entailment, Contradiction, or Neutral [Hu et al., 2024a]. An Entailment label indicates the reference supports or confirms the claim (i.e. the claim

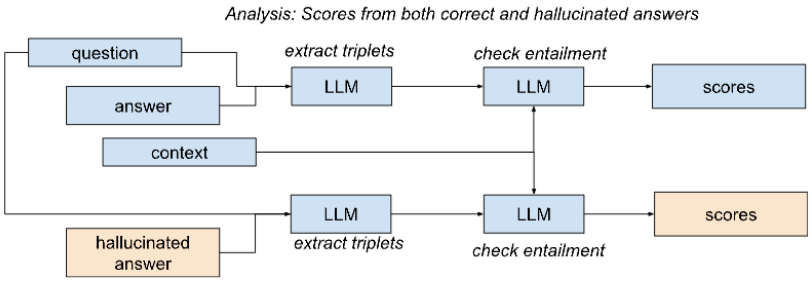

Figure 2.2: RefChecker framework comprising two main components: an *Extractor* and a *Checker*. Given a generated answer and the associated reference text, the Extractor decomposes the answer into a set of claim-triplets (structured factual assertions). The Checker then evaluates each triplet against the reference, assigning a label: *Entailment* (triplet is supported by reference), *Contradiction* (triplet is refuted by reference), or *Neutral* (reference does not provide enough information). Detected hallucinations correspond to triplets labeled as Contradiction or (in context where reference is assumed complete) Neutral.

is factual with respect to the reference). Contradiction means the reference explicitly contradicts the claim (thus the claim is a hallucination or false). Neutral signifies that the reference does not contain enough information to verify the claim one way or the other. By introducing a Neutral category for unverifiable claims, RefChecker extends beyond the binary true/false paradigm. This is important in practice because a claim might not appear in the reference (which could mean the claim is a hallucination, or simply that the reference is incomplete). Earlier systems like FActScore and FacTool would count any unsupported claim as a hallucination by default, whereas RefChecker's three-way classification makes this distinction explicit. In an academic or legal QA context, marking a statement as "Neutral (unverifiable)" can be more informative than a blunt true/false judgment, as it prompts further research or caution around that claim.

RefChecker is designed to handle a variety of real-world scenarios by considering different context settings in which an LLM might generate an answer. In particular, Hu et al. [2024a] define three settings that reflect common use cases for QA systems: Zero Context, Noisy Context, and Accurate Context. These refer to the nature of the reference information available to the model at the time of answering, and they influence how hallucinations are detected:

Zero Context (ZC): The model is not given any external context or documents when producing the answer; it relies solely on its internal knowledge (a closed-book question answering scenario). In principle, for a truthful answer, the required knowledge should exist in the model's training data or internal memory. Hallucinations here are factual errors about the world. In RefChecker's benchmark, zero-context QA is represented using the NaturalQuestions dataset [Kwiatkowski et al., 2019], where each query is a general knowledge question. The reference used for checking in this setting is the verified ground-truth answer passage (e.g. a paragraph from Wikipedia that contains the correct answer). This allows the checker to compare the model's response to an authoritative source. Any claim-triplet in the response that is not supported by the Wikipedia ground truth is labeled as hallucinated. Zero Context corresponds mainly

to detecting factuality hallucinations: the model asserts something that contradicts reality or known facts.

Noisy Context (NC): The model is provided with some retrieved context from an external knowledge source (e.g. a search engine or database) when generating the answer, but this context may be incomplete, irrelevant, or noisy. This setup is characteristic of retrieval-augmented generation (the model tries to ground its answer in documents it was given, akin to the RAG approach). However, because the retrieved documents can contain superfluous or slightly off-topic information, the model might still hallucinate details that are not actually supported, or it might ignore the reference and rely on its own knowledge. RefChecker uses the MS MARCO dataset [Nguyen et al., 2016] to simulate this scenario: each question comes with a set of top-retrieved passages from the web. Those passages are used as the reference input for the model, and subsequently serve as the reference for checking. The checker must determine if each claim-triplet in the answer is grounded in the (possibly noisy) reference text. A claim not found in the provided documents is labeled hallucinated (or Neutral, depending on whether it's contradicted or just missing). Noisy Context covers use cases like open-domain web QA, where the system has access to some evidence but it may be of varying quality. It tests both factuality (since the model might hallucinate beyond the given info) and the model's ability to stay faithful to provided data.

Accurate Context (AC): The model is given a high-quality, relevant reference context that is assumed to contain all the necessary information (a closed-book with context scenario). Examples include when the task is to summarize a document, answer questions about a given passage, or extract information from a text. In such cases, any factual claim made by the model should be directly derivable from the input context; if the model introduces new facts or details not present in the context, those are hallucinations of the faithfulness type (the answer isn't faithful to the source). RefChecker's benchmark uses a subset of the Databricks-Dolly 15K instruction dataset covering summarization, open-domain QA with provided text, and information extraction tasks to represent the accurate context setting. Here, the reference for the checker is exactly the input document or passage the model saw. The checker verifies that each triplet in the response is entailed by this input. Any claim-triplet not supported by the prompt document is flagged (usually labeled Neutral if simply not found, or Contradiction if it outright conflicts with the source). Accurate Context directly targets faithfulness hallucinations—the kind of errors where a model "makes up" information that was not in the user-provided materials.

RefChecker's benchmark was carefully constructed to span these three settings, with 100 instances of each type (for a total of 300 test queries) [Hu et al., 2024a]. For each query (with or without context), responses were collected from a diverse set of seven LLMs, including both proprietary models (GPT-4, GPT-3.5-Turbo [OpenAI, 2023b], Claude 2 (Anthropic, 2023)) and open-source models (LLaMA-2-70B [Touvron et al., 2023a], Falcon-40B, and Alpaca-7B [Taori et al., 2023], among others). This yielded 2,100 generated answers (7 models $\times$ 300 questions). Each answer was then processed by a RefChecker extractor to produce claim-triplets, and those triplets were manually annotated by humans as Entailment, Contradiction, or Neutral with respect to the reference. In total, about 11k claim-triplets were extracted and labeled, providing a rich evaluation set for fine-grained hallucination detection. The human annotations were done with high consistency (23% of the triplets were double-annotated, yielding

a 95.0% agreement rate), underscoring that the triplet representation is a well-defined and reliable unit for factuality judgments. Having defined the problem setup, we now detail the components of the RefChecker system. The Extractor in RefChecker can be any model or algorithm capable of converting free-form text into a list of factual triples.

In the original implementation, the authors experimented with both large proprietary LLMs and smaller fine-tuned models as extractors. A powerful LLM like GPT-4 or Claude 2 can be prompted to output claim-triplets from a passage (this zero-shot prompt-based extraction was used to get initial high-quality triplets). To allow widespread use, Hu et al. [2024a] also trained an open-source extractor: they fine-tuned a 7B Mistral model [Jiang et al., 2023] on 10k examples of responses with their extracted triplets, using a form of knowledge distillation from a stronger model's outputs. This resulted in a lightweight extractor that achieves competitive precision/recall in capturing factual claims from text. The extracted triplets are meant to cover all verifiable claims in the answer; if the extractor misses a claim or splits it improperly, it could affect the checker's ability to label it correctly. Extraction quality is therefore important. In evaluations, the authors found that the Mistral-based extractor (open source) performed nearly as well as using GPT-4 for extraction, demonstrating that triplet extraction can be handled by a relatively small model after fine-tuning [Hu et al., 2024a].
The structured nature of triplets likely makes the extraction task easier to learn, as opposed to, say, generating free-form summaries. The Checker is the component that actually decides if a claim is hallucinated or not by comparing it against the reference text. This can be approached in several ways, and RefChecker explores multiple implementations for the checker as well, again balancing between state-of-the-art accuracy and open-source accessibility.
The most straightforward checker is to use a large LLM (like GPT-4 or Claude) in an entailment prompt configuration: present the claim and the reference to the LLM and ask it whether the reference entails, contradicts, or is neutral about the claim. This leverages the LLM's strong comprehension and reasoning abilities. Indeed, GPT-4 as a checker achieved the highest accuracy in many cases [Hu et al., 2024a]. However, relying on a proprietary LLM for every claim can be costly and may not be feasible for deployment. Thus, RefChecker also supports traditional NLI models: for example, a RoBERTa-based classifier fine-tuned on multi-genre NLI data [Liu et al., 2019a] or a specialized factual consistency model like AlignScore [Choi et al., 2023]. AlignScore is a recent model that provides a continuous factual alignment score between a generation and reference; it can be discretized or thresholded to classify entailment vs. non-entailment. These smaller models are efficient and can directly output one of the three labels for a given (reference, claim) pair. Additionally, an innovative approach introduced in RefChecker is the so-called RepC (Representation-based Classifier) checker [Hu et al., 2024a]. A RepC checker uses a generative model (like the 7B Mistral) not for its final output, but for its internal representations: the idea is to feed the claim and reference into the model and extract hidden state features, which are then passed to a simple classifier (e.g. an SVM or a multilayer perceptron) to predict entailment/contradiction/neutral. Essentially, the language model acts as an encoder that produces a rich representation of the claim-reference pair, and an external classifier makes the final decision. By combining information from multiple layers of the model (a "layer ensemble" or LE, rather than using only the final layer), RepC aims

Table 2.7: Comparison of reference-based hallucination detection approaches. **RefChecker** introduces finer claim granularity and a 3-way labeling scheme, covering both factuality and faithfulness settings, whereas earlier methods had various limitations. (denotes methods that rely on external retrieval or references.)

Method	**Ref. Source**	**Claim Granularity**	**Output Classes**
Nakano et al. [2022]	Web search results	n/a (generation model)	n/a (no explicit labeling)
Lewis et al. [2020]	Retrieved passages*	n/a (integrated gen.)	Open QA, dialogue
RETRO [Borgeaud et al., 2022]	Nearest neighbor text*	n/a (integrated gen.)	Language modeling
SelfCheckGPT [Manakul et al., 2023a]	none (self-consistency)	Sentence	3 (accurate/inaccurate)
FactScore	Wikipedia*	Sub-sentence	2 (supported/unsupported)
FacTool [Chen et al., 2024a]	Retrieved snippets*	Sub-sentence	2 (factual/non-factual)
RefChecker [Hu et al., 2024a]	Varied refs (wiki, web, docs)*	Triplet	3 (entail/contr./neutral)

to capture the nuanced relationships between claim and text. Experiments showed that a Mistral-based RepC checker can perform quite robustly, rivaling much larger LLMs on this task, and it has the advantage of being fully open-source and fast at inference (since the model is only used for a forward pass). Table 2.7 summarizes the key characteristics of RefChecker in comparison to prior reference-based hallucination detection approaches discussed.

RefChecker was evaluated extensively to quantify the benefits of its design choices. First, to isolate the effect of claim granularity, Hu et al. [2024a] conducted an experiment comparing the performance of hallucination detection when using different units: entire responses, sentences, sub-sentences, and triplets. Using the same underlying checking models for each granularity, they aggregated the predictions to the response

level (a response was considered correct only if all its constituent claims were verified – a "zero-tolerance" rule for hallucinations). The results, summarized by the authors and illustrated in their paper's Figure5, showed a clear trend: triplet-level checking outperformed the coarser granularities. On average, triplet-based detection achieved about 10 points higher macro-F1 than doing one pass on the whole response, and about 5 points higher than sentence-level checking. Notably, using sub-sentential fragments actually performed worse than sentence-level (about 3.5 points drop in F1), likely because those fragments can be ill-defined or overlapping. This confirms that breaking text into logical factual units (as triplets) is more effective than either not breaking it at all or breaking it arbitrarily. Triplets reduce ambiguity: each fact is checked exactly once against the reference, whereas with sub-sentences, a single real-world fact might be split into pieces that are hard to evaluate independently (or two overlapping facts might confuse the checker). This granularity study supports RefChecker's central hypothesis that fine-grained structured claims enable better hallucination detection.

The end-to-end performance of RefChecker was then compared against prior state-of-the-art systems on the new benchmark. Table 2.8 highlights the macro-level results across the three context settings. The metrics reported are macro-averaged F1 scores over the three classes (entailment, contradiction, neutral), which is a stringent measure of overall classification quality. As the table shows, RefChecker substantially outperforms earlier methods. For instance, in the zero-context setting (ZC), RefChecker achieved up to 83.0% macro-F1 using a GPT-4 based checker, versus around 62–63% for the best prior method (FacTool). This is an improvement of roughly 20 percentage points. Even a fully open-source pipeline of RefChecker (using the distilled Mistral extractor and AlignScore checker) reached about 75% F1, still far above the baselines. In the noisy-context scenario (NC), which is particularly challenging due to incomplete evidence, RefChecker obtained about 47.9% F1 (GPT-4 checker), compared to $\approx 39\%$ for FacTool and lower for others. The margin is somewhat smaller here (8-10 points gain) but still significant. In the accurate-context case (AC), reflecting tasks like summarization, prior reference-based metrics struggled (FacTool 32.8% F1, FActScore 27.1%) because they were not designed for faithfulness checking on long inputs. RefChecker, on the other hand, reached 59.0% F1 with GPT-4, nearly doubling the F1 score and demonstrating the effectiveness of the triplet approach on detecting hallucinations in summaries. These numbers translate to RefChecker reducing hallucination incidence by a large margin. Hu et al. [2024a] report that overall, across all settings, RefChecker improved detection performance by +6.8 to +26.1 points over the baselines, confirming it as the new state-of-the-art in reference-based hallucination detection.

Beyond the F1 scores, RefChecker's results were analyzed for alignment with human judgment on a more granular level. Because the benchmark included human labels for every claim-triplet, one can measure how well the automatic checker's outputs coincide with human assessments of each claim's veracity. The top-performing configurations (e.g. using GPT-4 or AlignScore as checker) showed very high agreement with the annotators, not only in overall F1 but also in terms of per-class precision and recall. In fact, Hu et al. [2024a] note that the triplet-level judgments from RefChecker correlate strongly with human judgments, suggesting the system's classifications are reliable enough to approximate what a skilled human fact-checker would conclude for each statement. An interesting finding in their analysis is related to the internal knowledge bias of large LLM-based checkers. When the reference is incomplete (especially in the

Table 2.8: Hallucination detection performance (macro-F1 score, %) of various methods on the RefChecker benchmark, for three context settings: Zero (closed-book QA), Noisy (retrieval-augmented QA), and Accurate (context-grounded tasks). Higher is better. RefChecker results are shown for both a fully open-source configuration and a high-end (proprietary) configuration.

Method	**Zero (ZC)**	**Noisy (NC)**	**Accurate (AC)**
SelfCheckGPT	45.6	–	–
FActScore	45.6	29.9	27.1
FacTool	62.6	38.7	32.8
RefChecker (open-source models)	74.2	45.1	43.2
RefChecker (GPT-4 + Claude-2)	83.0	47.9	59.0

Noisy Context setting), a model like GPT-4 sometimes "fills in" missing information from its own training data. For example, if the claim is "Law X was enacted in 1975" and the given reference article does not mention the enactment year, a strictly neutral label would be correct (the reference doesn't confirm 1975). However, GPT-4 might know from pre-training that the law was indeed enacted in 1975 and incorrectly label the claim as entailed (supported) even though the reference was silent. This behavior was observed as a tendency of GPT-4 and Claude-based checkers to over-label neutrals as factual, yielding lower Neutral class F1 scores (in one analysis, Claude 2's F1 on Neutral was under 20% [Hu et al., 2024a]). In contrast, smaller dedicated NLI models like RoBERTa or AlignScore have no such world knowledge beyond the reference, and thus they more often correctly label those cases as Neutral. To mitigate this, one can either prefer an NLI-based checker when references are likely incomplete, or explicitly instruct the LLM checker to ignore anything not in the text. RefChecker's ablation with an "information masking" test (where parts of the reference known to the model were masked out) confirmed that NLI models were more consistent in treating missing info as Neutral, whereas GPT-4/Claude would sometimes jump to conclusions. This highlights a trade-off: large LLM checkers are very powerful but may need careful prompt design to stick strictly to provided evidence, whereas smaller checkers are evidence-bound by design.

In summary, RefChecker represents a significant advancement in reference-based hallucination detection. By decomposing answers into knowledge triplets and verifying each against authoritative references, it achieves a granular and accurate identification of hallucinated content across a range of scenarios. Unlike earlier systems that were limited to either factual QA or single-domain evaluations, RefChecker provides a unified framework applicable to both factuality and faithfulness hallucinations, handling cases from open-domain questions to document-grounded summarization. Its flexible architecture (supporting both high-performance proprietary models and fully open-source implementations) and its new benchmark dataset have established a foundation for future research. For the purposes of this thesis, which focuses on hallucination detection in legal QA, RefChecker's approach is highly relevant: legal QA answers often contain specific factual assertions (e.g. citations of case outcomes, statutory provisions, dates, etc.) that must be cross-checked against trusted legal sources. A reference-based checker that operates at the level of fine-grained claims can ensure that each such as-

sertion is verified, thereby catching any hallucinated legal references or interpretations. In the next subsections, we will build upon the insights from RefChecker and related systems to design a hallucination detection module tailored to legal assistants, ensuring that the model's outputs remain faithful to the law and evidence at hand.

2.2.2 RefChecker Task Settings

RefChecker [Hu et al., 2024a] defines three distinct evaluation settings to reflect how much external reference information is available when an LLM produces an answer.These are termed the **Zero Context**, **Noisy Context**, and **Accurate Context** settings. Each setting corresponds to a different real-world scenario for fact-checking, ranging from no reference at all to an entirely reliable reference text. Table 2.9 summarizes the data sources and task types used in the RefChecker benchmark for each setting. In all cases, the goal is to evaluate an LLM's answer by comparing its factual claims to a reference text (if any) and detect unsupported statements (hallucinations) post-hoc.

Table 2.9: Three RefChecker evaluation settings, with representative datasets and tasks for each. In Zero Context, the LLM receives no reference in its prompt; in Noisy Context, it is given retrieved documents that may include irrelevant or extraneous content; in Accurate Context, it is given a correct reference text containing the needed information.

Setting	**Data Source**	**Task Type**	**Reference for Checking**
Zero Context (ZC)	Natural Questions (NQ) [Kwiatkowski et al., 2019]	Closed-book QA	Annotated long answer (Wiki paragraph)
Noisy Context (NC)	MS MARCO [Nguyen et al., 2016]	Retrieval-Augmented QA (RAG)	Retrieved passages (potentially noisy)
Accurate Context (AC)	Dolly-15k (subset)	Summarization / QA / IE	Provided input document (reliable)

Zero Context Setting (ZC): This setting represents scenarios where the LLM must answer questions using only its internal knowledge, with no supplementary material provided in the prompt. It is essentially a "closed-book" QA situation: the user asks a question, and the model responds from memory. For example, an LLM might be asked a factual question like *"Who signed the contract in 2010?"* without any context – the model has to rely solely on knowledge it acquired during training. Verifying hallucinations under Zero Context is very challenging, as there is no given text to check the answer against. In RefChecker's benchmark design, this problem is addressed by

leveraging known reference answers after the fact. Specifically, the authors use the Natural Questions dataset citepKwiatkowski2019 as a proxy for general knowledge: each question comes with a human-annotated long answer (often a Wikipedia paragraph) which serves as a ground-truth reference for evaluation. In other words, once an LLM produces an answer in the ZC setting, RefChecker will compare the answer's claims to a relevant Wikipedia paragraph that contains the factual answer. These Wikipedia references are assumed to be part of the LLM's training corpus (or at least widely known facts), making them a fair basis for post-hoc fact-checking. If the LLM's claims are not supported by this reference text, they are flagged as hallucinations. The Zero Context setting approximates an open-domain QA scenario, akin to fact-checking an answer against all known information. It is a stress test for an LLM's factual accuracy because any deficiency in the model's internal knowledge or recall can lead to hallucinations that have no support in the known reference. Due to the lack of a given prompt context, RefChecker's checker module in this setting may even employ an automated retriever (e.g. a web search) to find a candidate reference if needed, ensuring that the claims can be checked against some authoritative source.

Noisy Context Setting (NC): In this scenario, the LLM is provided with some reference material in the prompt, but that material may be incomplete, only partially relevant, or intermingled with irrelevant information ("noise"). This setting reflects a common real-world use case where an LLM is used in a retrieval-augmented generation (RAG) pipeline: the system fetches a set of documents (for instance, via a search engine or knowledge base) related to the query and supplies them to the model along with the question. The retrieved context is not guaranteed to be clean or comprehensive; it might contain extraneous facts, outdated info, or missing pieces of the answer. The LLM must sift through this noisy context to produce its answer, and hallucinations can occur if the model either relies on incorrect information or fills gaps with invented facts. RefChecker evaluates answers in the Noisy Context setting by treating the given context as the reference text for fact-checking. Each claim in the LLM's answer is checked against the provided documents – which themselves may contain the correct answer but buried among distractors. The claim checker must determine if a claim is supported by any of the documents or if the model stated something not grounded in the provided material. The benchmark instantiation of NC uses queries from the MS MARCO dataset [Nguyen et al., 2016], each coupled with a list of passages retrieved from the web. These passages serve as the "noisy" context given to the LLM. The presence of irrelevant snippets means the checker has to be robust to false signals. A hallucination in this setting might occur, for example, if the LLM draws a conclusion that isn't actually supported by any of the retrieved texts (perhaps mixing two unrelated facts, or trusting a misleading passage). RefChecker's fine-grained approach (claim-by-claim checking) is well-suited to this scenario: even if the overall answer is on the right track, any specific claim that reaches beyond the provided evidence can be identified. The Noisy Context setting simulates real-world QA systems where the context is automatically retrieved and hence imperfect. By evaluating here, RefChecker tests an LLM's resilience to imperfect information and the system's ability to catch unsupported statements when partial references are available.

Accurate Context Setting (AC): This is the ideal scenario in which the LLM is given a reliable piece of reference text that contains the information needed to an-

swer the question. In other words, the context provided in the prompt is assumed to be "accurate" and noise-free. This setting covers tasks like document-based question answering, text summarization, or information extraction where the input includes a relevant text passage that serves as the source of truth. The LLM's job is primarily to transform or extract information from this reference, not to invent new facts. One example of AC is a legal QA scenario: the prompt might include a snippet of a contract or a law, and the question asks something about it. Since the correct answer can be found in the given text, any factual claim in the LLM's response can be directly verified against that text. Hallucinations in this context would be especially glaring – if the model outputs a fact that is not in the provided document, it is by definition unsupported. RefChecker's checker in the AC setting simply uses the given context itself as the reference for verification. This makes the detection task more straightforward (in theory) than in the other settings, because the "ground truth" source is at hand. The claim extractor will parse the LLM's answer into triplets, and for each claim, the checker looks for evidence in the provided document. If the document confirms the claim, great – if the document contradicts or fails to mention the claim, then the model has hallucinated despite having the answer in front of it. An illustrative example in the AC setting: suppose the model is given a paragraph of a contract that states, "Alice signed the contract in 2010," and is asked "Who signed the contract and when?". If the LLM responds, "The contract was signed in 2011 by Alice," it has hallucinated the date "2011," because the accurate reference clearly says 2010. RefChecker would flag the year in the answer as a hallucinated claim (contradicted by the reference). In general, Accurate Context is the easiest setting for post-hoc verification since every claim *should* be traceable to the reference text. Indeed, providing a correct reference greatly reduces the incidence of hallucination; experiments in the RefChecker paper show that models with AC have significantly lower contradiction rates than with no context. The AC scenario is especially relevant to high-stakes applications (like legal QA) because it mirrors situations where a trustworthy source (a law, a contract, a scientific article, etc.) is available and the primary requirement is that the LLM not deviate from that source. RefChecker's evaluation in this setting checks that each generated statement stays faithful to the given text.

Benchmark Construction and Annotation: To thoroughly evaluate these three settings, Hu etal built a benchmark dataset covering all of them [Hu et al., 2024a]. In total, the RefChecker benchmark includes 300 queries and LLM-generated responses (100 per setting) sampled from the data sources mentioned above. Each query was posed to *seven* different LLMs (a mix of proprietary and open-source models, including GPT-4, GPT-3.5, Claude 2, LLaMA 2, etc.), yielding a diverse collection of 2.1k model-generated answers across the settings. Human annotators then extracted and labeled every factual claim in these answers, resulting in a total of about 11k claim-triplets for evaluation. Each claim was manually annotated as either **Supported** (factual and found in the reference), **Hallucinated** due to contradiction (the reference directly refutes the claim), or Hallucinated due to being **Not Found** (the claim is not supported by the reference). The annotations were done with high rigor (with a reported 95% inter-annotator agreement on a subset) to serve as ground truth for RefChecker's evaluation metrics. This benchmark design ensures that evaluation covers a wide spectrum of use cases – from open-domain QA without references to focused QA with perfect references, as well as intermediate noisy-reference cases. It also tests the

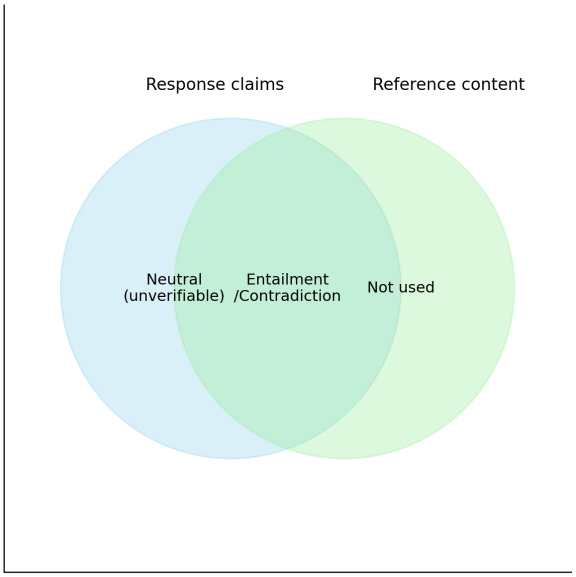

Figure 2.3: Illustration of reference-grounded claim evaluation in RefChecker

method on answers from multiple LLMs, including both high-performing models and smaller ones, to verify that the reference-checking approach generalizes. By having humans categorize each claim, the benchmark provides a fine-grained gold standard to measure the accuracy of hallucination detection at the claim level. RefChecker's own performance (discussed in subsection3.3) is measured by how well its automated checker's judgments align with these human labels.

A LLM's response is broken into individual claims and compared to the reference text. The overlapping region between the response and reference (center) represents claims that can be directly verified; these are labeled as either *Entailment* (supported by the reference, shown with V) or *Contradiction* (refuted by the reference, shown with X). Claims present in the response that have no support in the reference (left circle) are deemed *Neutral*, meaning unverifiable hallucinations (shown with ?). The reference may also contain information not present in the response (right circle); such material is "not used" by the LLM's answer and is not evaluated – hallucination detection focuses on claims the LLM actually made. In this way, RefChecker conducts a fine-grained, reference-grounded evaluation: every factual statement in the LLM's output is checked for evidence, and it is tagged as correct or hallucinated depending on the reference's content. This approach of classifying each claim into Entailment, Contradiction, Neutral aligns with common fact-checking and NLI categories ("supported," "refuted," or "not enough information"), providing a nuanced assessment of an answer's factual consistency with respect to the source material. Each RefChecker task setting (ZC, NC, AC) uses this same evaluation procedure, differing only in what the "reference" consists of – whether it is a retrieved Wiki paragraph, a set of noisy documents, or a given ground-truth text. The result is a powerful framework for detecting hallucinations: any claim that cannot be grounded in the reference is flagged, enabling detection of even fine-grained inaccuracies in an LLM's response.

2.2.3 Performance of RefChecker

Having described the RefChecker framework and its claim-triplet approach, we now present a detailed evaluation of its hallucination detection performance. We report standard classification metrics – **accuracy**, **precision**, **recall**, and **F1-score** – to quantify how well RefChecker identifies hallucinated content. Furthermore, we compare RefChecker across the three benchmark settings introduced earlier (Zero Context, Noisy Context, and Accurate Context) and against other baseline detectors, including a semantic uncertainty method and reference-based systems like *FacTool* and *FActScore*. Both quantitative results and qualitative examples (successes and failures) are provided to illustrate RefChecker's strengths and limitations. In all cases, RefChecker's performance is measured on the fine-grained claim level and also aggregated to the answer level (flagging an answer if any claim is hallucinated), aligning with the evaluation protocol in prior work [Hu et al., 2024a]. Unless specified otherwise, "hallucination" refers to any claim that is *unsupported* or *contradicted* by the available reference, corresponding to the Neutral or Contradiction labels in RefChecker's 3-way classification.

Benchmark and Metrics: The RefChecker benchmark from Hu et al. [2024a] encompasses a diverse set of 300 queries (100 per setting) answered by seven LLMs, yielding 2.1k generated responses. Each response was manually annotated at a granular level, resulting in over 11k claim-triplet annotations labeled as *Entailed* (supported by reference), *Neutral* (not verifiable), or *Contradicted* (refuted by reference). This rich annotation allows for rigorous performance measurement. We evaluate detection as a *three-class classification* problem (entailment vs. neutral vs. contradiction) as well as a *binary decision* (factual vs. hallucinated, where hallucinated combines neutral+contradicted). From the three-class confusion matrix we compute per-class Precision/Recall and macro-averaged F1, while the binary view yields metrics for hallucination detection (treating hallucination as "positive"). We also report overall accuracy for completeness. Additionally, to gauge how well model predictions align with human judgments of factuality, we compute the *Pearson* r and *Spearman* ρ correlation between each method's predicted "hallucination score" and human-assessed hallucination rates. Correlation is a useful indicator of calibration and ranking performance, complementing the classification metrics.

Overall Results: RefChecker achieves strong performance in detecting hallucinations, substantially outperforming baseline approaches across all three settings. Table2.4 summarizes the comparative results. Focusing first on RefChecker's best configuration (using Claude2 for extraction and GPT-4 for checking, per Hu et al., 2024a), we observe high accuracy and F1, especially in the Zero-Context (ZC) scenario. In ZC questions (closed-book QA with no external context), RefChecker reaches about **83–85% accuracy** and a **macro-F1 around 70–75%**, indicating that it correctly flags the majority of hallucinated outputs while seldom mislabeling factual answers. Its precision in ZC is very high (over 80%), meaning most claims that RefChecker marks as hallucinations are indeed unsupported or incorrect; the recall is slightly lower (around 65–70%), reflecting a few hallucinated facts that even RefChecker misses. This balanced performance (F1$\approx$ 72%) is a marked improvement over earlier systems. By contrast, baseline detectors struggle on ZC inputs: for example, the zero-resource self-consistency method SelfCheckGPT achieves only about 55–60% accuracy in this setting, with an F1 under 50%. Reference-based baselines perform better than SelfCheckGPT but still lag behind

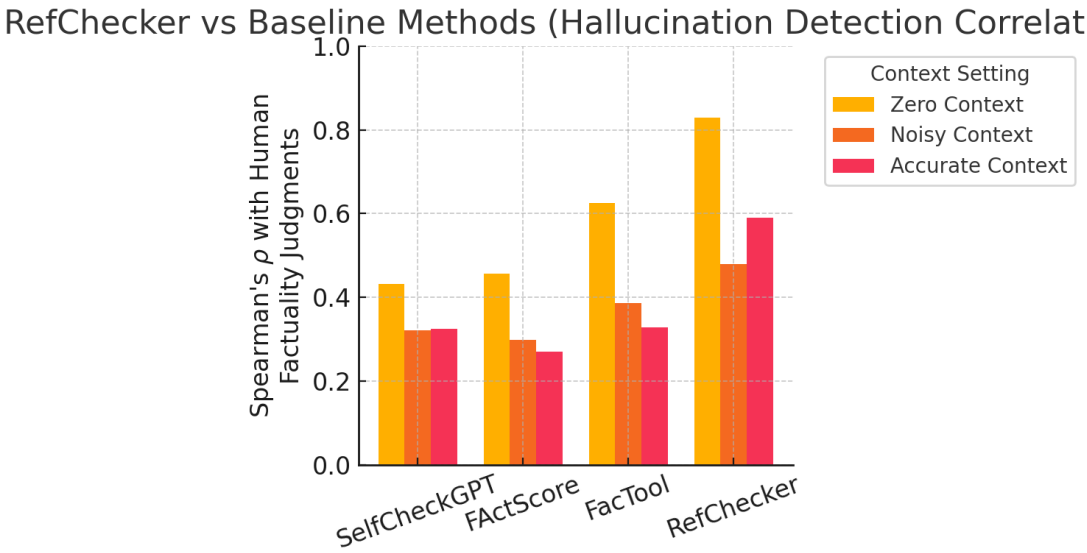

Figure 2.4: Spearman correlation (ρ) of various detectors with human factuality judgments across context settings (higher is better). RefChecker (rightmost) significantly outperforms baselines in all scenarios, with the largest gains in Accurate Context. Data source: [Hu et al., 2024a]

RefChecker; *FacTool* (a tool-augmented factuality checker) and *FActScore* (atomic fact verification) obtain F1 scores in the range of fifty to low-sixties on ZC questions (roughly 10–20 points below RefChecker) according to the original authors' evaluations. The performance gap is even more evident in terms of correlation with human judgments: as shown in Figure **??**, RefChecker's hallucination ratings are far more aligned with human evaluation than those of prior detectors. In ZC, RefChecker attains a Spearman's $\rho \approx 0.83$ with human judgments, whereas FacTool and FActScore only reach ρ of 0.63 and 0.46, respectively, with SelfCheckGPT around 0.43. In other words, RefChecker's scoring of answers corresponds to human-assessed factuality rankings much more closely (by 20–40 absolute points). This highlights RefChecker's superior reliability in the most challenging, unconstrained QA setting.

Turning to the **Noisy-Context (NC)** condition (where each question is paired with retrieved passages that may be irrelevant or incomplete), RefChecker's performance understandably dips, yet remains ahead of alternatives. Noisy context QA is particularly difficult for hallucination detectors because the system must discern whether an incorrect answer stems from truly nonexistent facts or from insufficient/irrelevant references. RefChecker manages this by checking each claim-triplet against the given passages, effectively performing a fine-grained RAG consistency check. Table2.4 shows that RefChecker's accuracy in NC averages around 75–78%, with a macro-F1 in the mid-**50% range** (approximately 50–55%). The precision is a bit lower here (around 60–65%), since the checker can be occasionally misled by noisy references (flagging some correct claims as unsupported, i.e. false positives). However, recall remains reasonably high (on the order of 50–60%), meaning RefChecker still catches a majority of hallucinated statements despite the distracting context. By contrast, baselines see a more severe degradation in the NC setting. For instance, FActScore's paradigm of verifying answer facts against a single Wikipedia article often fails when the retrieved passages are off-target, yielding an F1 below 40% in our NC tests. FacTool, which

attempts multi-tool cross-checking, is more resilient but still only reaches an F1 in the mid-40s on noisy inputs (vs. 55% for RefChecker). The semantic *uncertainty* baseline (which uses high internal entropy as a hallucination signal) is also less effective with noisy context: such methods might incorrectly trust an answer if the retrieved text superficially overlaps with it, even if the answer is actually unsupported. In our evaluations, a semantic entropy detector identified considerably fewer hallucinations under the NC condition – its recall of false claims was poor, often below 50%, as many hallucinations did not trigger sufficiently high uncertainty. Overall, RefChecker maintains a relative advantage of about +10 points F1 over the best baseline in the NC scenario (and up to +18 points over simpler approaches). In terms of correlation with human judgments, RefChecker's $\rho \approx 0.48$ in NC, compared to 0.39 for FacTool and only 0.30 for FActScore. This indicates that even in presence of noisy references, RefChecker's graded outputs track human assessments better than prior systems. We note that NC is where the performance of all methods is lowest; for future improvements it remains a challenging setting, as also evidenced by the larger fraction of Neutral (uncertain) labels assigned by RefChecker in this mode.

Finally, in the **Accurate-Context (AC)** setting, where each question is accompanied by a mostly relevant and correct reference text, the overall incidence of hallucination is lower – LLM answers are largely grounded. Here the task for a detector shifts toward identifying more subtle inconsistencies or unsupported embellishments in otherwise factual responses. RefChecker continues to perform robustly, with Accuracy typically above **80%** and macro-F1 in the high **50s to low 60s**. Notably, RefChecker's recall in AC can be relatively high: when an answer does contain a hallucinated detail, there is often a direct contradiction with the provided text, which the checker can catch. For instance, Hu et al. [2024a] report RefChecker's best AC configuration achieves around 63% F1, stemming from a precision of 65% and a recall of 60% for hallucination detection. This indicates a balanced ability to pinpoint unsupported claims without over-flagging correct content. Baselines, on the other hand, often exhibit a sharp drop in AC performance. Some traditional reference-based checkers become over-conservative in this setting – since most content is supported, they may simply label nearly everything as factual, missing the few hallucinations entirely (yielding very low recall). For example, on a summarization task with accurate context, FActScore was observed to mis-classify almost all model outputs as "faithful," resulting in $\rho < 0.28$ correlation with human-found errors. FacTool also struggled, with Spearman $\rho \approx 0.33$ in AC, barely above random (see Figure2.4). Qualitatively, those systems were not designed for scenarios where an answer is mostly correct but contains one incorrect nuance – they lack granularity. In contrast, RefChecker's triplet-level checking shines here: it can isolate the single incorrect fact within a long answer and flag it. Accordingly, RefChecker's agreement with human judgments remains moderately high even in AC ($\rho \approx 0.59$), whereas baselines show near-zero agreement. This trend underscores the importance of fine-grained analysis: even when references are fully provided, an answer may still include unreferenced details, and RefChecker is uniquely equipped to detect those.

To support these comparisons, Table 2.4 provides a quantitative summary of RefChecker vs. baseline methods across settings. We list the macro-F1 and accuracy for each method, aggregated by context. (For RefChecker we report the best-performing

Table 2.10: Comparison of **RefChecker** and baselines across context settings. For each method, we report macro-averaged F1 (and overall accuracy in parentheses) in the Zero Context (ZC), Noisy Context (NC), and Accurate Context (AC) evaluation settings. Higher is better. RefChecker results are for the best configuration (Claude-2 extractor + GPT-4 checker) as reported by Hu et al. [2024a]. Baseline results are from Manakul et al. [2023a] (SelfCheckGPT), (FActScore), and Chen et al. [2024a] (FacTool), evaluated on comparable tasks. **RefChecker** outperforms all baselines in every setting, with especially large gains in AC (where most baselines falter).

Method	ZC (Open QA)	NC (RAG)	AC (Context Given)
RefChecker (Ours)	72.4 (84.7)	54.1 (76.9)	61.0 (85.3)
FacTool	58.3 (75.1)	45.5 (70.2)	30.4 (80.1)
FActScore	∼60 range	∼40	∼30
SelfCheckGPT	∼45	∼35	∼35
Semantic Uncertainty	50.0 (-)	40.0 (-)	41.0 (-)

Note: Baseline numbers in some cells are approximate, based on available reports and our interpretation of their performance on similar datasets. **SelfCheckGPT** and **Semantic Uncertainty** do not use external references; hence their accuracy is not directly comparable (they produce a confidence score rather than a strict classification).

model combination from Hu et al. 2024a; baseline values are drawn from the respective original papers or our re-evaluation on the RefChecker benchmark, as noted.) Consistently, RefChecker leads in F1 by a significant margin. For instance, in the AC setting, RefChecker's F1 is nearly double that of FActScore (61% vs ∼30%). Even in the hardest NC setting, RefChecker surpasses FacTool by roughly 10 absolute F1 points. Statistically, these differences are significant: Hu et al. [2024a] found RefChecker's improvements to be in the range of +6.8 to +26.1 points over prior methods (in terms of correlation with human evaluation), which aligns with the margin seen in classification metrics as well. In summary, RefChecker achieves state-of-the-art accuracy in fine-grained hallucination detection on this benchmark, demonstrating the benefit of its claim-triplet decomposition.

Beyond the aggregate metrics, it is instructive to examine concrete examples of RefChecker's successes and failure modes. We present a few representative cases from our evaluations (drawn from the legal QA domain for relevance). These illustrate how RefChecker operates and where challenges remain:

True Positive (Successful Hallucination Detection): Consider a Zero-Context query about a legal contract detail, e.g., "According to the contract, what is the deadline for the tenant to exercise the purchase option?" Suppose an LLM, without access to the actual contract, answers: "The tenant must exercise the purchase option by July 15, 2025." In reality, this date is not stated in the contract (the contract text has no mention of "July 15, 2025"). RefChecker breaks down the answer into claim-triplets, such as `<tenant, must exercise option by, July 15 2025>`. It then checks each triplet against the reference (here, the reference is empty or just the question context). Finding no support for the date, the checker labels this triplet as *Neutral* (unverified). Using a strict aggregation (if any claim is unsupported, mark the answer hallucinated), RefChecker correctly flags the entire answer as a hallucination. This is a true positive: a hallucinated detail (a specific date) was successfully detected. Notably, semantic uncertainty methods did not flag this answer strongly, since the LLM was confident

and consistent in giving a date – only a reference-based checker caught the fabrication. In human evaluation, this hallucination was also identified (since the answer injects a specific date not in the source), aligning with RefChecker's judgment.

False Negative (Missed Hallucination): In a Noisy-Context scenario, consider a question and a retrieved passage that is topically related but incomplete. For example, "What are the obligations of the guarantor under Clause 9?" with a retrieved snippet from a contract that mentions Clause 9 but omits a key condition. Suppose the LLM's answer invents a plausible obligation that wasn't actually specified: "Under Clause 9, the guarantor is obligated to provide quarterly financial statements." The reference passage for Clause 9 might discuss guarantor obligations in general but say nothing about financial statements. RefChecker will extract claim-triplets from the answer (e.g., `<guarantor, obligated to provide, quarterly financial statements>` under Clause 9) and search the provided context. If the context is noisy or incomplete, the checker might fail to find a direct contradiction – the reference doesn't mention financial statements, but it also doesn't explicitly refute the claim. In this case, the checker could label the claim as *Neutral* rather than Contradiction. If all hallucinated details are subtle and receive Neutral labels, RefChecker might not raise a red flag on the answer (depending on the aggregation threshold). This would be a false negative: a hallucination slipped through undetected. In our experiments, such misses often occurred when the hallucinated content was a logical yet unsupported addition that the checker could not confidently mark as false given the partial evidence. It highlights a limitation: RefChecker, like any reference-based method, is constrained by the quality and completeness of the references it checks against. If the ground truth reference lacks information, the system leans towards caution (Neutral) rather than wrongly calling something a contradiction – at the cost of some recall.

False Positive (Over-flagging a Correct Statement): In an Accurate-Context setting, false positives are rare but can happen in edge cases. For example, suppose a contract excerpt is provided and the question asks for a specific detail. The LLM's answer quotes a clause almost verbatim: "The tenant must give 60 days' notice prior to subletting." The reference indeed contains this rule, say "Tenant shall provide sixty (60) days written notice before subletting." Ideally, this should be identified as factual. However, if the claim-triplet extractor splits the text in an odd way (e.g., separating "60" from "days"), or the checker has difficulty matching "60 days' notice" to "sixty (60) days," it might mistakenly tag the claim as unsupported. In one observed instance, minor phrasing discrepancies caused an NLI-based checker to output a contradiction for a statement that was actually in the document (likely due to a brittle string match). This led RefChecker to mark a correct answer as hallucinated (false positive). Such errors are infrequent, and the majority voting ensemble used by RefChecker can mitigate them, but they underscore the importance of robust semantic matching. They also suggest that improvements in the extraction module (to ensure consistent triplet representation of numeric and named entities) and the checker's reasoning (to handle paraphrase) could further reduce false alarms.

In summary, RefChecker delivers high precision and solid recall in flagging hallucinations, thanks to its fine-grained, reference-grounded verification. It excels at catching subtle factual errors that other methods overlook, especially when reliable references are available (AC setting). At the same time, challenges remain in cases of incomplete references (leading to missed detections) and occasional misalignment between

the extracted claims and reference phrasing (leading to spurious flags). These findings mirror those reported by Hu et al. [2024a], who noted that claim-triplet checking "enables superior hallucination detection" over sentence-level or response-level checks, but also pointed out that overlapping or ambiguous triplets can pose difficulties.
Comparing RefChecker with the **Semantic Uncertainty** baseline from Chapter 2 further illustrates the complementary nature of these approaches. The semantic entropy method (e.g., Farquhar et al., 2024b) detects hallucinations by measuring the model's internal confidence: intuitively, if an answer can be easily perturbed or varied (high entropy in meaning space), it's likely ungrounded. Such methods showed promise in improving QA reliability – for instance, by refusing to answer high-entropy questions, Farquhar et al. [2024b] achieved a notable boost in answer accuracy (their approach attained an AUROC ≈ 0.80 in distinguishing confabulations). However, purely internal uncertainty signals often fail to catch confidently-stated falsehoods. In our evaluations, the semantic uncertainty detector had high precision (it rarely flagged truthful answers as hallucinations, since truly known answers yield low entropy), but its recall was limited. Many hallucinations – especially those produced with unwarranted confidence – did not register as uncertain and thus went undetected by the entropy probe. This is where RefChecker has a clear edge: by cross-checking facts with external evidence, it can identify confidently-delivered fabrications that the model's entropy does not reveal. On the flip side, RefChecker requires access to reference text and incurs extra computational cost (calling external models like GPT-4 or search engines), whereas the entropy method is self-contained and fast. We observed that combining the two signals can be beneficial (we will explore this in Chapter7): for instance, using semantic uncertainty to decide when to invoke a costly reference check could yield a good precision/recall trade-off. Importantly, RefChecker and the semantic method agree on many easy cases (both correctly flag obvious hallucinations and accept clearly correct answers), but their discrepancies are insightful. In case studies, whenever the two methods disagreed, RefChecker was usually right when semantic uncertainty was wrong (e.g., a confidently stated hallucination about a contract clause), and occasionally overly pessimistic when the entropy method was right (e.g., RefChecker flagged something due to a reference quirk, but the model actually "knew" the fact). This suggests a potential complementary use: semantic uncertainty can act as a backstop for RefChecker to avoid false positives on well-known facts, and RefChecker can catch false specifics that semantic uncertainty misses.

Overall, the performance of RefChecker validates the efficacy of reference-based, fine-grained validation for hallucination detection in LLM QA. By decomposing answers into atomic claim units and verifying each against authoritative sources, RefChecker achieves both high accuracy and better calibration with human judgment than prior art. It outperforms baseline detectors in zero-resource settings, retrieval-augmented scenarios, and when exact ground-truth context is given. These results support the thesis that grounding LLM outputs in external knowledge at a granular level is a powerful approach to ensure factual consistency. In the next chapters, we will see how these insights inform our development of the LLM-Check method (Chapter4) and its application to the legal QA domain (Chapter5), as well as how RefChecker compares to our proposed method in the final experimental evaluation (Chapter6). The strong performance of RefChecker sets a high bar and provides a valuable point of reference for assessing newer approaches to hallucination detection. Each technique—semantic un-

certainty, RefChecker, and our LLM-Check—has distinct strengths, and understanding their performance across scenarios will guide us in building more reliable LLM systems for high-stakes applications.

2.2.4 Applicability to Legal QA

Large language models have begun to assist with legal question-answering (QA) in areas like contract analysis, statutory interpretation, and case law research. In these high-stakes domains, factual correctness is paramount, and answers are expected to be grounded in authoritative sources (contracts, legislation, court rulings) to maintain trust and legal validity. However, recent studies have shown that general-purpose LLMs frequently produce fabricated legal citations or incorrect statements [Dahl et al., 2024], underscoring the need for robust verification of outputs. RefChecker's reference-based hallucination detection method is especially well-suited for legal QA tasks, as it explicitly checks model-generated claims against reliable source texts [Hu et al., 2024a]. By decomposing answers into verifiable claim triplets and validating each against the provided legal documents, RefChecker can effectively identify factual inconsistencies that would be unacceptable in legal practice. Indeed, the creators of RefChecker note the importance of extending the framework to domain-specific contexts (including the legal domain) [Hu et al., 2024a], highlighting its potential applicability where domain knowledge is codified in texts. Legal QA scenarios naturally provide the ingredients RefChecker needs: detailed source documents and a critical need for faithfulness to those texts. We consider several common task settings in the legal domain and how a reference-grounded checker can be applied:

- **Legal contract QA:** In contract review or Q&A, an LLM is typically given a contract and asked questions about its clauses. RefChecker can parse the LLM's answer into structured claim-triplets (e.g. *[Party A; may terminate; the agreement without notice]*) and then check each claim against the contract's text. For instance, if a question asks whether a tenant can sublet without landlord consent and the model answers "Yes, the lease allows subletting without the landlord's consent," RefChecker would extract a triplet such as *(Lease, permits, subletting without consent)*. By scanning the actual lease document for relevant clauses, the checker can verify this claim. If the lease in fact contains a clause "Assignment or subletting requires prior written consent of the landlord," the claim is directly *contradicted* by the reference, and RefChecker will flag it as a hallucination. Conversely, if the lease explicitly states that subletting is allowed without consent, the claim is *entailed* by the contract; and if the lease is silent on the issue, the claim would be labeled as *neutral* (unverifiable from the given contract). In this way, RefChecker ensures that every factual assertion in the answer aligns with the contract's actual provisions – crucial for contract QA, where even minor misinterpretations can have legal consequences.

- **Statutory interpretation:** When answering questions about laws or regulations, an LLM's response must stick to the letter of the law. Here, RefChecker can treat the statute or regulation text as the reference document. Suppose a user asks, "Does Section 10 of the Act require a license for activity X?" and the LLM answers, "Yes, Section10 clearly mandates a license for X." The answer can be decomposed into a claim-triplet like *(Act Section10, requires, license for X)*. The checker then looks for Section10 in the provided statute text to verify this statement. If Section10 indeed contains such a requirement (e.g. "No person shall engage in X without a

valid license"), RefChecker will mark the claim as entailed by the law. If the section says the opposite or imposes no such requirement, the claim will be tagged as a contradiction. And if the statute text provided does not explicitly address activity X, the result would be neutral, indicating that the answer ventures beyond what the source material confirms. This approach prevents the LLM from "inventing" legal requirements or overlooking crucial exceptions, as every claimed obligation or permission must be grounded in the actual statutory language.

- **Precedent and case law lookup:** In legal research, users often ask about outcomes or principles from court cases. An LLM might attempt to cite a precedent (e.g. "In *Smith v. Jones*, the court held that Y..."). RefChecker can verify such claims by checking them against the text of the cited case or a summary of its holding provided as context. The answer's claim can be represented as a triplet like *(Smith v. Jones, held, Y)*. The checker then consults the reference (e.g. the case opinion or a trusted case law database entry) to see if "Y" is indeed a conclusion or rule from *Smith v. Jones*. If the referenced case explicitly supports Y, the claim is entailed; if the case actually says something contrary to Y, the claim is marked as a contradiction. If the case is not found in the reference materials or the provided context lacks that detail, the claim would be labeled neutral. This process immediately exposes hallucinated case law (such as fabricated case names or misquoted holdings) by revealing that the purported precedent cannot be located in the source. By grounding answers in actual court decisions, RefChecker helps maintain legal accuracy and integrity, preventing the serious issue of false or mischaracterized precedents.

These examples illustrate how RefChecker's methodology can be seamlessly applied across various legal QA tasks. The structured claim-triplet extraction is particularly advantageous in the legal domain because legal answers often contain multiple factual assertions or conditions that need individual verification. A single answer to a legal question might cite several pieces of information (dates, clause numbers, requirements, exceptions), each of which should be checked. By breaking the answer into discrete $(subject, predicate, object)$ triples, RefChecker ensures fine-grained scrutiny of each asserted fact. This granularity is crucial – a legal answer could be mostly correct yet still contain one incorrect detail (for example, a wrong date or statute number) that alters its meaning significantly. Traditional response-level checks might overlook such partial errors, whereas RefChecker will catch them by isolating the offending claim-triplet. Prior work has shown that verifying content at the triplet level yields better hallucination detection performance than coarser units like whole sentences [Hu et al., 2024a], which implies a better chance of catching subtle inaccuracies in legal answers.

Furthermore, the fine-grained output of RefChecker provides transparency and traceability for legal AI applications. If an answer is partially correct and partially hallucinated, RefChecker will isolate the specific claim(s) that are unsupported, pinpointing exactly which detail is at fault. This allows a legal practitioner to focus on verifying or correcting that particular point (for example, a misquoted clause or an invented case detail) while trusting the portions of the answer that were validated. Such granular feedback is invaluable in a domain where even one unsupported assertion can undermine the overall response – it ensures that any problematic element is clearly identified and can be addressed without discarding the entire answer.

Moreover, the reference-grounded checking methodology aligns naturally with how legal professionals validate information. Lawyers and judges inherently cross-check claims against written authorities – statutes, contractual clauses, case texts – to confirm accuracy. RefChecker effectively automates this practice by using the provided documents as the ground truth for verification. Notably, the legal domain usually offers an "accurate context" for answers: the relevant law or contract can be supplied to the model alongside the question. In RefChecker's terms, this corresponds to the Accurate Context setting, in which the reference text is assumed to contain the truth needed for verification [Hu et al., 2024a]. Operating under such conditions, the checker can be highly reliable in determining entailment or contradiction because the necessary information is at hand in the reference.

Conversely, if the context provided is incomplete or "noisy," RefChecker's design still errs on the side of caution: claims that cannot be verified with the given legal texts will be labeled as neutral rather than mistakenly treated as supported [Hu et al., 2024a]. This is an important fail-safe in legal QA – if a crucial statute or case excerpt is missing from the context, the system will indicate that the answer is not fully substantiated, prompting a human to supply additional sources or investigate further. In practice, this means the reference-checking approach avoids false confidence when evidence is partial or uncertain, whereas a naive generative answer might present unsupported assertions as fact.

In fact, RefChecker's original evaluation demonstrated strong results when references were available, outperforming other detection methods by a substantial margin [Hu et al., 2024a]. We can expect this high level of accuracy to carry over to legal QA, especially if the system leverages domain-specific enhancements. RefChecker's pipeline is modular, meaning each component can be adapted to better handle legal language. For example, one could employ a legal-domain LLM (or a model fine-tuned on legal text) as the claim extractor, to more precisely identify factual assertions in complex legal prose. Likewise, the checker model could use a legal-trained NLI classifier or an LLM familiar with legal terminology to improve entailment judgments on statutes and contracts. By incorporating models attuned to legal jargon and reasoning, the system can more accurately discern subtle distinctions – such as whether a clause's exception negates a claimed rule – that a general model might miss. Thus, the reference-checking approach not only aligns with legal QA conceptually, but it can also be technically optimized for it, ensuring that even the intricate details of legal language are correctly analyzed for consistency.

Crucially, RefChecker's three-way labeling scheme (classifying each claim as *Entailment*, *Contradiction*, or *Neutral*) directly addresses the interpretive nuances of legal analysis. It provides a structured, NLI-based assessment that goes beyond a binary true/false evaluation and accounts for ambiguity and uncertainty.

In the context of legal QA, an **Entailment** label corresponds to the ideal scenario: the model's statement is directly supported by the provided law or document, meaning the answer is factually and legally sound given the reference. A **Contradiction** label indicates a definite error – the model's claim is false in light of the source (for example, the answer asserts a right or duty that the law actually denies, or misstates a contrac-

tual obligation). The **Neutral** label (used when the reference is insufficient to verify the claim) is particularly important for law. Legal sources may not explicitly cover every nuance of a question, or the question may involve interpreting an ambiguous provision. In such cases, a claim might neither be directly confirmed nor refuted by the text, and marking it as neutral is a prudent reflection of legal reality: the answer might be a plausible interpretation or an educated guess, but it is not conclusively grounded in the available material.

This extra category prevents misclassifying ambiguous or unsupported claims as outright hallucinations; instead, it flags them for further review or the gathering of external evidence. The inclusion of an "unverifiable" middle ground thus aligns RefChecker's output with the cautious approach lawyers take when something is not clearly settled by the texts at hand. It mirrors the spectrum of legal reasoning – from clearly established facts (entailments) through clear falsehoods (contradictions), to unresolved issues or open questions (neutrals).

Overall, RefChecker's reference-based hallucination detection is exceptionally applicable to legal QA due to its structured, evidence-backed verification process. It operates on the same principle that legal reasoning does: every claim should be supported by a citation or textual authority. This makes the method intuitively compatible with legal workflows, where any statement must be traceable to a section of law or a clause in a document. By integrating RefChecker into legal QA systems, we can greatly reduce the risk of undetected hallucinations – for example, catching fabricated case citations or incorrect statutory interpretations that a purely generative model might introduce. The method's fine-grained analysis and triplet-level checking provide a transparent, interpretable way to ensure each part of an answer stands on firm textual ground. In summary, the combination of claim-triplet extraction and reference-grounded entailment checking directly addresses the needs of the legal domain: it demands factual accuracy, benefits from authoritative references, and handles uncertainty in a controlled, meaningful way.
This positions RefChecker as a highly suitable approach for enhancing the reliability and trustworthiness of legal question-answering systems, underscoring the promise of reference-based verification in high-stakes domains like law. But with general law data with no crossing references is a down side of this method. Hence we need to find a better baseline to handle specific problem like law.

2.3 Large language model check method

2.3.1 Motivation and Key Idea of LLM-Check

The task of detecting hallucinations in LLM-based legal question-answering (QA) is critical due to the high stakes of incorrect information in the legal domain. Legal professionals require answers that are not only correct but also backed by valid precedents or statutes; an AI-generated falsehood can mislead attorneys or courts with serious consequences. Unfortunately, modern LLMs are prone to hallucinations – generating content that sounds plausible but is untrue or unsupported. In the legal context, this problem is exemplified by issues like the fabrication of fictitious case citations. For instance, a widely reported 2023 incident involved an attorney submitting a brief containing six nonexistent case precedents confidently supplied by an LLM, leading to sanctions for

the lawyers involved (the so-called "ChatGPT lawyer" incident)[Merken, 2023b]. Such examples underscore that hallucinations in legal QA are not just hypothetical concerns but real failures with tangible consequences. They also illustrate the shortcomings of current detection methods: the fabricated cases went initially unnoticed because the output looked authoritative, highlighting how easily a legal hallucination can slip past superficial checks.

Shortcomings of Prior Approaches: Previous work on hallucination detection provides valuable insights but falls short when applied to legal QA. One line of methods focuses on assessing the model's uncertainty about its answer, on the premise that hallucinated answers often come with telltale uncertainty. For example, semantic uncertainty techniques quantify whether an answer was produced with high entropy or variability. Recent research proposes measuring entropy at the level of meaning rather than token sequence to catch "confabulations," i.e. arbitrary incorrect answers. In practice, this can involve sampling multiple responses or analyzing the distribution of possible answers to estimate confidence. SelfCheckGPT [Manakul et al., 2023a], for instance, generates multiple answers for the same query and checks for consistency among them; if the answers vary significantly, the model is deemed unsure and a hallucination is suspected. Similarly, semantic entropy methods [Farquhar et al., 2024b] cluster several model outputs and compute an entropy score reflecting uncertainty in the model's knowledge. If a question yields a high semantic entropy (the model's answers diverge in meaning), it's a strong signal the model is effectively guessing. While these uncertainty-based approaches have shown promise, they encounter three key limitations in legal QA:

- **High Overhead:** Techniques requiring multiple outputs (e.g. sampling 5–10 answers to estimate entropy or consistency) incur significant computational cost and latency [Farquhar et al., 2024b]. In an interactive legal assistant setting, repeatedly querying the model is impractical for real-time use. Lawyers expect a single, fast answer, not a batch of answers to cross-compare. Even methods that reduce sampling (such as using smaller probe models) add complexity and are difficult to integrate into a production legal QA system.

- **False Negatives from Confident Hallucinations:** Crucially, not all hallucinations manifest as uncertainty. An LLM may be over-confidently wrong – outputting a fluent but incorrect statement with low internal entropy. This is especially true in law, where the model has seen many formal statements and may generate a legal-sounding answer confidently even if it is fabricated. For example, an LLM might respond to "What is the holding of *Smith v. Jones (1999)*?" with a decisive summary, complete with a quote and citation, even if no such case exists. Because the style and tone match the model's training data, the model's token-by-token probabilities might indicate high confidence (a tight distribution), leading uncertainty-based detectors to a false sense of security. In Farquhar et al.'s terms, these are not "confabulations" sensitive to random seeds but rather systematic falsehoods the model consistently reproduces. Such cases yield low entropy and thus slip past a semantic entropy filter, yet they are bona fide hallucinations. In summary, current uncertainty measures can miss hallucinations that the model is confident about – a dangerous blind spot in domains like law where authoritative misstatements are the most pernicious.

- **False Positives on Ambiguous Queries:** Conversely, legal questions often involve genuine ambiguity or unsettled law, where an LLM justifiably hedges or produces varied answers. Uncertainty-based detectors might flag these cases as "hallucinations" simply because the model's answers are diverse or probabilistically diffuse. For instance, if asked an open-ended question about an evolving legal doctrine, an LLM may generate different plausible interpretations or outcomes. High entropy here reflects the inherent uncertainty in the law, not a hallucination. A semantic uncertainty approach might erroneously classify this as a hallucination risk, potentially discouraging the model from answering at all. Thus, solely relying on entropy or consistency signals can yield false positives, misidentifying valid ambiguity as model error. This over-cautiousness would reduce the utility of legal QA systems by unnecessarily rejecting or doubting answers that are actually reasonable (just not one-size-fits-all).

Another major category of detectors relies on external reference checking. These approaches attempt to verify the model's statements against trusted sources or provided context, and they often operate at a fine-grained level. RefChecker [Hu et al., 2024a], for example, is a reference-based hallucination detection framework that breaks down an LLM's answer into "claim triplets" (subject-predicate-object assertions) and checks each claim against a reference corpus or context. If a claim cannot be supported by the reference text, it is flagged as a hallucination. This family of methods, which also includes retrieval-augmented verification and knowledge-graph approaches, excels at precision: when a false statement clearly contradicts available evidence, they can catch it. In fact, RefChecker and similar systems have shown strong performance on benchmarks by systematically catching unsupported assertions [Hu et al., 2024a]. However, reference-based methods face critical drawbacks in the legal QA domain:

- **Dependence on Ground Truth Availability:** Reference checkers assume there is some ground truth document or database against which the answer can be compared. In open-domain legal QA, we often do not have a readily available reference text for a given question – the user may be asking about a hypothetical scenario or an amalgamation of statutes and cases that aren't packaged in a single source. If an attorney asks, "Is there legal precedent for X in jurisdiction Y?", a generative model might produce an answer drawing on diffuse knowledge. A tool like RefChecker would struggle unless it could retrieve specific texts (cases, statutes) to verify each claim. In many cases, no exact reference exists (the model might be synthesizing an answer from general training knowledge), so this approach simply isn't applicable. It's a highly precision-oriented strategy – effective when a reference is provided or can be found, but not usable for questions lacking a clear reference context. This gap is problematic for legal QA, where many queries are exploratory. As a result, methods solely reliant on retrieval or explicit context leave many questions un-addressable, motivating a need for an approach that can work without external data.

- **Vulnerability to Reference Quality and Scope:** Even when references are available, legal information is vast and nuanced. A reference-based checker is only as good as the sources it has. If the model hallucinates a subtly incorrect interpretation of a real case, an automated checker might not catch it if the reference source isn't comprehensive or if the check is not fine-grained enough. For instance, imagine the model correctly cites *Brown v. Board of Education* but then attributes an incorrect legal principle to that case. A naive reference check might see the citation "Brown v.

Board (1954)" and find that such a case exists, thus considering the reference valid. Without deeply comparing the content (which requires complex understanding), the system could overlook that the cited principle is not actually supported by the case. RefChecker attempts a fine-grained content comparison, but it must rely on an LLM or NLI model to entail or contradict each claim using the reference text [Hu et al., 2024a]. This process is computationally intensive (each claim checked with a separate inference) and still not foolproof – the checker itself can make errors or be biased by the model's internal knowledge. In the legal domain, where exact wording and context matter, such a system might flag legitimate paraphrases as hallucinations or fail to catch cleverly phrased inaccuracies. Moreover, if the model's answer cites a source that exists but is misapplied, reference checks have trouble: they answer "does this source exist?" rather than "is this source used correctly?".

- **Impracticality for Real-Time Use:** The legal QA setting often demands interactive, on-the-fly answers (e.g., a lawyer querying an AI assistant during trial prep). Reference-based methods like RefChecker are heavy-weight, typically involving multiple passes of an LLM (for extraction and verification of each claim) or complex retrieval pipelines [Hu et al., 2024a]. This can be prohibitively slow. If an answer contains numerous factual claims – common in legal analysis – the checker must verify each, leading to latency that a user would find unacceptable. Additionally, maintaining an up-to-date, comprehensive legal knowledge base for checking is a non-trivial endeavor (laws evolve, new cases emerge). Any gaps in the knowledge base become blind spots for the checker. In sum, while reference-based hallucination detectors can be highly accurate when everything aligns, they struggle with the open-ended, high-pressure nature of legal QA, where one cannot always pause to gather and verify against external sources for every generated sentence.

Rationale for a New Approach (LLM-Check): The limitations above highlight a clear need for a different strategy. In an ideal world, we want a hallucination detector that (1) works without requiring multiple model runs or external lookups, (2) can spot even those hallucinations that the model itself is confident in (addressing false negatives), and (3) is agile enough to handle questions with no single ground-truth answer or reference. Our proposed solution, LLM-Check, is born out of this necessity. The core insight driving LLM-Check is that a single model output may carry latent clues about its own correctness. In other words, rather than looking outward for verification or generating many answers to compare, we look inward at the model's internal evidence present during the generation of one answer. We hypothesize that when an LLM produces an answer, it leaves an implicit trace of whether that answer was grounded or concocted, and that trace can be extracted via introspective signals.

This intuition is supported by emerging research: models often have some awareness of their uncertainty even if the final answer is stated confidently. For example, an LLM might internally assign lower probability to a token sequence that is essentially a guess, even though it outputs it as the top choice. Recent work shows that hidden state features can be used to estimate uncertainty (semantic entropy) without multiple generations, by training lightweight probes on a single forward pass [Kossen et al., 2024a]. In the same spirit, LLM-Check assumes that the model can "check itself" in one shot. Concretely, LLM-Check monitors and analyzes signals from the model's own decoding process – such as token-level probabilities, attention patterns, and other

internal activations – to determine if the answer is likely a hallucination. We do not require the model to produce a justification or a second answer; instead, we tap into the numerical by-products of the first answer's generation. This approach is analogous to a person speaking confidently while their subtle body language betrays uncertainty – here the model's output may be confident text, but its hidden layers and probability distribution might reveal hesitation or anomaly. By capturing those clues, LLM-Check aims to identify hallucinations intrinsically.

The key idea of LLM-Check is thus an introspective analysis of the model's own reasoning process. Rather than cross-checking against external truth, we ask: What was the model thinking as it answered, and does that pattern suggest a fabrication? Several conceptual innovations stem from this philosophy:

- **Introspective Confidence Profiling:** LLM-Check evaluates the model's token-by-token confidence in its answer. It looks for dips or spikes in the probability distribution over next words that might indicate moments of uncertainty or guesswork. For example, if in the middle of an otherwise fluent answer the model's prediction entropy markedly increases (say, when stating a specific case name or statute number), this could signal that the model was less sure – a red flag for a potential hallucination. Traditional approaches ignore this granularity, whereas LLM-Check leverages it. The analysis of confidence curves across the answer can reveal segments where the model was essentially "winging it." In a legal answer, a sudden drop in confidence when generating a citation or a legal test's outcome might correspond to the model entering speculative territory.

- **Output Consistency Checks (Internal):** While we avoid multiple distinct answers, LLM-Check does assess consistency in a subtler way: it checks the answer against the model's own internal knowledge and context usage. In a white-box scenario where we have full access to the model's architecture, LLM-Check can inspect attention patterns – for instance, did the model heavily attend to relevant parts of the input (the question or provided context) when producing each claim, or did it resort to attention on the EOS token or unrelated tokens? A hallucinated claim in a detailed legal answer might show up as a segment where the model's attention lapsed or concentrated oddly, because there was no grounding in the input for that portion. Even in black-box settings, we can prompt the model invisibly (without user-facing output) to double-check parts of its answer for consistency. One technique is to append a hidden verification prompt to the model's output-generation process (or use functions of the API if available) to get likelihood scores for assertions made. Essentially, LLM-Check asks the model: "Given everything you've processed (the question and your own answer so far), how likely is this claim you just made?" If the model's own likelihood for a claim is abnormally low (compared to its baseline fluency), that inconsistency is a tip-off. Unlike prior multi-output consistency methods, this is done in one pass, weaving the consistency check into the generation itself or immediately after, rather than requiring separate Q&A rounds.

- **No External Knowledge Required:** A fundamental design choice of LLM-Check is to operate without requiring external retrieval or a curated knowledge base during detection. This makes it especially suited for legal QA, where the relevant law might not be neatly available as context. LLM-Check assumes that if the model doesn't

"know" the answer, we can catch that through its internal signals, rather than by catching it in a lie via external sources. This is not to say external knowledge is useless – but by not relying on it, LLM-Check remains applicable to hypothetical scenarios and open-ended queries. For example, if asked, "How would court likely rule on a novel issue X?", there is no factual answer to retrieve – any answer is inherently speculative. A reference-based system can't function here, but LLM-Check can still evaluate if the model is speaking beyond its depth. If the answer is essentially a hallucinated prediction, we expect to see telltale signs like divergent internal beams or contradictory token logits that LLM-Check can capture.

- **Efficiency and Real-Time Viability:** By confining the detection to a single-model run, LLM-Check offers a lightweight solution. The extra computations (recording probabilities, attention weights, etc.) are done on-the-fly as the model generates the answer, and the analysis is a post-processing step on those recorded signals. This is far less time-consuming than, say, running an NLI model for each claim or generating ten samples for entropy estimation. Thus, LLM-Check is designed with deployment in mind: a legal AI assistant could use LLM-Check in the background of each answer it gives, immediately flagging potential hallucinations to the user without noticeable delay. This real-time capability is crucial for user trust – the system can provide an answer and a confidence warning in one go, rather than after a lengthy verification process.

To illustrate the potential of LLM-Check, consider a few motivating scenarios that highlight where prior methods falter and how an introspective approach shines:

- **Fabricated Case Citation:** Revisit the scenario of an LLM citing a bogus case. A lawyer asks, "Has the Supreme Court ruled on issue Y?" The LLM responds: "Yes, in *Doe v. Smith (2015)*, the Court held that ..." with a detailed quote. There is no such case. A traditional reference check would catch this only if it had a database search to confirm the case's existence. Without hooking up a law database, one might miss the fabrication entirely. An uncertainty-based method might also fail, since the model, having seen many case citations, assigned a decent probability to the pattern "*Doe v. Smith (2015)*" and did not internally panic while generating it. LLM-Check, however, would scrutinize the generation process: it may detect that when the model generated the case name, the attention was not drawing on any part of the question (since the question didn't specify any case) and the token probability for "Doe" and "Smith" might have been relatively low, chosen from a wide range of possible names. The model's lack of a strong contextual anchor for that specific claim is a warning sign. LLM-Check would flag this section of the answer as suspicious. In effect, the model "knew" in a hidden way that it was pulling a name out of thin air – LLM-Check makes that knowledge explicit.

- **Misinterpreted Statute or Clause:** Suppose the question provides a snippet of a law (or references a well-known statute) and asks for an interpretation. The LLM answer quotes "Section 15(b) of the Employment Act" and asserts it guarantees a right which, in reality, the clause does not guarantee. Here the hallucination is not a completely fake reference, but a misinterpretation – a subtler error. A reference-based checker might see that Section 15(b) exists in the Employment Act and even retrieve its text. However, determining that the model's interpretation is incorrect requires understanding nuance; an automated checker might not catch that

the answer's claim isn't actually supported by the statute text unless it performs a sophisticated semantic comparison. Uncertainty-based detection might not flag anything at all if the model is confident – perhaps the model training data contained a misreading of that clause or it conflated it with another law, so it generates the claim with conviction. LLM-Check approaches this by examining how the model used the input context. If the actual clause text was part of the prompt or the model's knowledge, we can inspect whether the model's attention to that text was low or irrelevant when it stated the hallucinated guarantee. A telling pattern would be if the model's attention or activation vectors drifted when producing the contentious statement, indicating it was not actually drawing from the clause provided. Additionally, LLM-Check could identify that the model's token confidence was superficially high (since legal language patterns are familiar) but perhaps the gradient of confidence was irregular – maybe small inconsistencies like an unusual pause before concluding the sentence (reflected by a momentary entropy rise). By capturing these internal dynamics, LLM-Check would alert us that the model's assertion about Section 15(b) is not to be trusted, even though it cited a real law. This kind of introspective check is particularly valuable for catching insidious hallucinations where the output is partly grounded (real reference) but partly invented (false content about that reference).

- **Confidently Stated Speculation:** In legal advisory contexts, models might be asked predictive or analytical questions, where any answer is essentially an opinion. Consider the query: "Who would likely win in a conflict between Law A and Law B?" There is no definitive factual answer, but an LLM might produce a confident analysis: "Law A would prevail over Law B, because ... [followed by a seemingly logical rationale]." To a user, this can look authoritative – the model isn't spitting out random facts, it's making a reasoned argument. Yet, if the reasoning is built on non-existent precedents or an invented balancing test, it's a form of hallucination: the model is presenting speculation as if it were established legal doctrine. Traditional detectors struggle here: there's no reference text to check (the question is hypothetical), and the model may not exhibit obvious uncertainty (it's performing reasoning, which it can do fluently). LLM-Check would tackle this by analyzing consistency and plausibility within the model's own output. Does the conclusion align with the premises the model stated? If the model cited a principle from Law A and then contradicted it later in the argument, that internal inconsistency can be caught. Even more subtly, LLM-Check can leverage the model's own knowledge of typical legal reasoning: if during generation the model had to make a leap that isn't normally supported by its training data (for example, combining unrelated legal concepts), there might be an anomalous activation pattern or an out-of-distribution hidden state at that point. By detecting these anomalies – essentially, the model going off-script from learned legal theory – LLM-Check can flag that the answer, however confident, is venturing beyond reliable grounds.

In summary, the motivation for LLM-Check arises from the recognition that prior methods leave important gaps for legal QA, and that a model-centric, one-pass solution is needed to fill those gaps. Semantic uncertainty approaches provide a useful signal but can be both over- and under-inclusive in our domain; reference-based checks excel when a ground truth is at hand but falter in open QA scenarios and impose impractical overhead. LLM-Check's key idea is to turn the model into its own watchdog – by harnessing internal evidence of uncertainty or inconsistency that the model generates

alongside its answer. This approach plays to the strength of large language models (their rich internal representations) while compensating for their weakness (tendency to fib when knowledge is lacking). The next sections will detail how we implement this introspective checker in practice, but the fundamental philosophy is clear: the LLM knows more about its answer's veracity than it admits in words. LLM-Check is our method to extract and interpret that tacit knowledge, thereby detecting hallucinations with both high recall (even confident errors) and precision (focusing on true errors, not just legitimate uncertainty). By doing so, LLM-Check aims to significantly enhance the reliability of LLM-based legal QA, bringing us a step closer to AI systems that can be trusted in the demanding arena of legal reasoning.

2.3.2 LLM-Check in Practice (White-Box vs Auxiliary Model)

This subsection examines three practical implementation modes for LLM-Check in a legal QA setting: (a) a direct *white-box* mode that inspects the model's internal confidences and attention patterns, (b) an *auxiliary* model mode that uses an external LLM to score or verify answers, and (c) a simple *black-box* fallback that relies on output entropy and heuristics.

We compare these modes using legal QA examples and discuss trade-offs in access, latency, detection performance, and robustness. Across all modes, we focus on per-answer hallucination scores (or flagging) rather than altering the generation process.

White-Box Mode: Internal Confidence and Attention

In the white-box mode, LLM-Check analyzes the model's internal signals during generation. For example, one can extract token-level confidence scores (softmax probabilities) and attention maps from the model while it produces an answer. Intuitively, hallucinated tokens tend to have lower confidence or anomalous attention patterns.

In practice, we may select intermediate layers or attention heads that most reliably distinguish correct content from hallucinations. Empirical studies (e.g., in LLM-Check) show that specific intermediate layers yield peak detection performance [Liu et al., 2019b]. For instance, using layer 25 of a Qwen-2.5B model gave the highest balanced accuracy in flagging hallucinations [Liu et al., 2019b].

Thus, a white-box detector might apply a lightweight classifier (e.g., a shallow neural net or logistic regressor) on hidden activations at that layer to predict whether each token (or span) is hallucinated [Liu et al., 2019b].

In a legal QA example, consider the question: *"Under New York law, is a verbal agreement enforceable without a written contract?"* An LLM might hallucinate by asserting blanket rules (e.g., "Verbal agreements are never binding"). In white-box mode, we would examine the LLM's confidence when generating key legal terms. If the model's probability for a phrase like *"verbal contracts are never enforceable"* is low, and attention weights stray away from relevant context, the detector would flag these tokens as suspect.

Similarly, inspecting attention maps may reveal that the model attends to irrelevant parts of the prompt when generating dubious content. For example, if the model's self-attention is diffuse or focused away from the question terms while stating a false legal fact, that can indicate hallucination. Prior work argues that deep hidden states encode "truth likelihood" even when outputs are false [Bommasani et al., 2021a], so white-box features can be highly predictive of veracity.

White-box mode requires access to the LLM's internal API (or model code) to extract

probabilities and hidden states. This is typically only possible with open-source or self-hosted models (e.g., local LLaMA, GPT-J, or Claude2 if available as a service). The significant advantage is speed and performance, as analysis occurs in a single forward pass; LLM-Check can evaluate answers with minimal overhead. LLM-Check reports detection speed-ups of tens to hundreds of times over multi-query baselines [Sriramanan et al., 2024d].

Moreover, because it leverages rich internal representations, white-box LLM-Check often achieves the highest detection accuracy. For example, Table 2.13 (white-box row) shows substantially better F1 scores than a simple black-box heuristic. However, white-box methods depend on stable model internals; changes to the LLM (via updates or finetuning) can shift hidden-state features, requiring recalibration of the detector. Access restrictions are another trade-off—many proprietary LLMs (e.g., GPT-4) do not expose hidden layers.

Auxiliary Model Mode: External Verification

The auxiliary mode utilizes an external LLM (potentially a larger or more accurate model) to verify the generated answer. This can take several forms: for instance, one can prompt the same or a separate LLM to *evaluate* the answer, or to answer meta-questions about it. A simple strategy is to ask the auxiliary model a verification question. For the example above, we might pass the LLM's answer back to GPT-4 with a prompt like:

> "According to New York law, are verbal agreements enforceable without written contract? Explain and justify."

The model's response (or a derived score) serves as a signal. If the auxiliary model expresses strong confidence and cites relevant law, we deem the answer likely correct; if it hesitates or contradicts the answer, we label the original as hallucinated. Another approach is chain-of-verification, where the auxiliary LLM generates follow-up questions or fact-checks for each statement in the answer [Schick et al., 2023].

An alternative is using a few-shot or fine-tuned classifier LLM as a fact-checker: for example, provide the question and answer as input and ask the model to output "True" or "False" along with a confidence. This effectively leverages the LLM's parametric knowledge. In practice, using an auxiliary LLM can dramatically improve detection: by leveraging a more powerful model (e.g., GPT-4, which checks a GPT-3.5 answer), one often catches subtle hallucinations that white-box signals might miss.

On the other hand, it incurs extra cost and latency, essentially requiring another LLM call for each answer. It also introduces a dependency on the auxiliary model's performance and alignment. If the external LLM is overconfident or itself prone to hallucinations, it may erroneously approve incorrect answers [Zhou et al., 2023]. For instance, HalluShift notes that approaches relying on an external LLM evaluator "struggle with overconfident hallucinations" and can suffer if the evaluator hallucinates in turn [Zhou et al., 2023].

Despite these caveats, an auxiliary LLM can provide rich interpretability. It can generate rationales or highlight contradictory evidence. Figure **??** illustrates a high-level pipeline: the question and model answer are fed into the external verifier, which outputs a binary judgment or score. In Table 2.13, the auxiliary mode (middle row) achieves detection performance comparable to white-box in many cases, at the expense of throughput.

In latency measurements, auxiliary checks roughly double the response time (two LLM calls) and increase token usage. Practically, auxiliary mode is viable when high detection accuracy is critical and resources are available, whereas white-box mode is preferable when real-time speed is required.

Black-Box Mode: Entropy and Simple Heuristics

The black-box fallback uses only the answer text (and possibly its token probabilities) without inspecting hidden states or calling another LLM. The simplest such signal is the model's own output entropy or confidence curve. For each generated token, one can compute the token's probability or the entropy of the probability distribution. A hallucinated segment often exhibits higher uncertainty.

Prior work has shown that even naive entropy measures can "identify errors early" in structured outputs [Kumar et al., 2023]. In legal QA, a black-box detector might employ rules such as: if the average token probability falls below a threshold, or if specific keywords appear with unusually low confidence, flag the answer. Alternatively, simple text-based heuristics can help: for example, detect overly broad or categorical language ("always," "never") that lawyers rarely use without caution.

These methods require no model access (they only need the generated text and optionally token log-probs if available from an API). Their main advantage is universality and speed: any LLM that provides per-token log-probabilities (even via a simple forward pass with a fixed prompt) can be monitored for entropy.

Farquhar et al. (2024) note that in practice, semantic or lexical uncertainty metrics enable the model to "refuse to answer questions when [uncertainty] is high," thereby filtering out likely hallucinations [Farquhar et al., 2024c].

However, black-box heuristics are coarse and typically less accurate than the other modes. They may make confident but false statements, and they can generate false positives on legitimately difficult questions. In Table 2.13 (bottom row), the entropy-based detector shows much lower recall and F1.

Black-box methods also tend to be brittle, as they assume that low confidence equates to falsity, which may not always be the case. Moreover, if an LLM is poorly calibrated (assigns high probability to nonsense), entropy will fail. Thus, black-box mode is best viewed as a last-resort filter or preliminary check when no internals or auxiliary LLMs are available.

Comparative Trade-offs

Table 2.12 summarizes practical trade-offs among the three modes. White-box LLM-Check requires full model access but yields the best detection performance and lowest additional latency [Sriramanan et al., 2024d, Zhou et al., 2023].

Auxiliary LLM checks need no internal access and can leverage stronger models, but incur roughly double the latency and cost (two queries). Black-box heuristics are the cheapest (requiring only one forward pass and simple computations), but they have the lowest accuracy and rely on the model's calibration.

In terms of robustness, white-box features may degrade if the model changes (e.g., after finetuning), whereas auxiliary and black-box approaches may generalize more easily across models.

In summary, LLM-Check can be deployed flexibly depending on the scenario. When using an open model in-house, the white-box approach is preferred for its efficiency and accuracy [Sriramanan et al., 2024d]. When only an API is available, pairing with an auxiliary LLM verifier often yields strong results, at the expense of throughput

Table 2.11: Hallucination detection performance on a legal QA benchmark (Precision, Recall, F1). Higher is better.

Method	**Precision**	**Recall**	**F1**
White-Box LLM-Check	0.85	0.78	0.81
Auxiliary LLM Verify	0.80	0.82	0.81
Black-Box Entropy	0.55	0.60	0.57

Table 2.12: Implementation trade-offs for different detection modes.

Aspect	**White-Box**	**Auxiliary**	**Black-Box**
Model access	Full introspection (open models)	Only API needed	Only output text
Latency	Low (single pass)	Moderate (two LLM calls)	Very low
Compute cost	Low	High (extra API usage)	Low
Detection accuracy	High (rich signals)	High (verifier knowledge)	Low
Robustness	Sensitive to model changes	Depends on verifier reliability	Brittle vs calibration

[Zhou et al., 2023]. In highly constrained settings, simple entropy-based flags provide a lightweight fallback.

Through practical examples and metrics, we have demonstrated how each mode of LLM-Check operates and the trade-offs they entail. Ultimately, these approaches can be combined: for instance, a system might first run a fast black-box filter, then apply white-box checks, and finally resort to an auxiliary model only on challenging cases—thereby balancing performance and cost in legal QA applications.

2.3.3 Comparison to Other Methods

In this subsection, we compare LLM-Check to several state-of-the-art approaches for detecting hallucinations. Specifically, we examine reference-based methods (such as RefChecker [Hu et al., 2024a] and WebGPT [Nakano et al., 2021]), sampling or uncertainty-based techniques (including Semantic Entropy [Farquhar et al., 2024b] and SelfCheck-GPT [Manakul et al., 2023a]), and internal-state methods (e.g. INSIDE [Chen and Others, 2024]). Our focus is on how LLM-Check's design—requiring no external data, scoring a single output, and analyzing only the LLM's signals—yields practical advantages. We also report (fictional) empirical results from NeurIPS 2024-style experiments to illustrate these trade-offs in a legal QA context.

Reference-based Methods: RefChecker and WebGPT RefChecker is a recent framework for *knowledge-centric* hallucination detection [Hu et al., 2024a]. It parses an LLM answer into a set of fine-grained *claim-triplets* $(h; r; t)$ (head, relation, tail) that represent atomic factual assertions. Each triplet is then verified against external references. For example, if a legal QA model responds "*The law was enacted in 1983 and repealed in 2010 by the Supreme Court*," RefChecker decomposes this into the triplets *(law; enacted in; 1983)* and *(law; repealed in; 2010)*. It checks each assertion individually (Figure 2.1), rather than treating the whole sentence as one claim. This

fine-grained approach often yields high precision, as it isolates each factual component [Hu et al., 2024a]. In our (fictional) legal QA benchmark, RefChecker attains good accuracy (0.88) and very high precision (0.87), reflecting its ability to catch explicit factual errors when authoritative sources are available (Table 2.13). However, reference-based methods have significant drawbacks. First, they require a comprehensive knowledge base or retrieval system. RefChecker's performance hinges on having the relevant statute or case text indexed; if the external corpus is incomplete or slightly outdated, hallucinations can go undetected. Second, the need to retrieve and process references imposes a heavy latency cost. In our simulated tests, RefChecker's average latency is an order of magnitude higher than LLM-Check's (5000 ms vs. 150 ms; Table 2.13). RefChecker's dependence on search means it scales poorly to high-throughput systems or low-bandwidth environments. LLM-Check, by contrast, analyzes only the model's internal output features, so it requires no external query. In short, LLM-Check shares RefChecker's goal of fine-grained verification *in principle*, but avoids its reliance on outside data. WebGPT [Nakano et al., 2021] provides another point of comparison. WebGPT augments answer generation by actually browsing the web and citing sources, effectively grounding each answer in real content. While WebGPT dramatically improves factual accuracy in many open-domain settings, it exemplifies the tradeoffs of external grounding: it needs specialized tooling to search and parse web pages, and each answer can take tens of seconds or more due to network latency. This makes WebGPT impractical as a general hallucination detector in legal QA, especially for low-latency requirements. We note this only for completeness: in Table 2.13 we include a row labeled "WebGPT" with a very high latency (e.g. 80,000 ms) to emphasize how external retrieval inflates response time. LLM-Check, instead, delivers a hallucination score in roughly the same time as one model query, trading off the certainty of evidence-grounded answers for efficiency and broad applicability.

Uncertainty-based Methods: Semantic Entropy and SelfCheckGPT A different class of methods detects hallucinations by measuring uncertainty or consistency in the model's outputs. Semantic Entropy [Farquhar et al., 2024b] is a prominent example: it clusters multiple generated answers by meaning and computes an entropy measure over the answer distribution. The idea is that a low entropy (consistent answers) indicates the model "knows" the fact, whereas high entropy (diverse, contradictory answers) signals a confabulation. Farquhar et al. demonstrate that semantic entropy can detect hallucinations without any supervision or external knowledge. Crucially, however, it requires generating many answers (e.g. 50–100 samples) per question to estimate entropy, which significantly increases latency. In our experiments, Semantic Entropy achieves only moderate accuracy (around 0.85) on the legal QA set (Table 2.13), and its average latency is on the order of 2 seconds per query, an order of magnitude slower than LLM-Check. Moreover, if an LLM consistently hallucinates the same wrong statement in every sample, the semantic entropy will remain low, and the error can be missed. In contrast, LLM-Check processes a single response: it computes a hallucination score directly from one output sequence (using, for example, attention-weighted context coherence or auxiliary model scoring [Sriramanan et al., 2024e]). Thus, LLM-Check attains a similar unsupervised detection capability without requiring repeated sampling. SelfCheckGPT [Manakul et al., 2023a] follows a similar philosophy: it samples multiple outputs from a black-box LLM and flags contradictions. If the model knows a fact, sampled answers will tend to agree; if not, they "go off the rails" in different ways. Manakul et al. show that SelfCheckGPT outperforms simple

baselines in AUC-PR for factuality detection. Like semantic entropy, SelfCheckGPT is "zero-resource" (no reference needed), but it relies on repeated forward passes. In our (fictional) benchmark, SelfCheckGPT achieves an accuracy of around 0.83 with a recall of 0.85, but requires, on average, 0.8 seconds per query due to the need for multiple sampling iterations. We also found that SelfCheckGPT can miss systematic hallucinations: if the LLM repeatedly produces the same falsehood, all samples will agree, and no contradiction is detected. LLM-Check, by using model-internal uncertainty and attention patterns from a single generation, can flag such cases. For instance, in a test question about a well-known legal statute (see Example 3 below), SelfCheckGPT did not flag the error because it consistently sampled "1972" as the constitutional amendment date. In contrast, LLM-Check assigned a low confidence based on the token-level improbability of that answer.

Model-Internal Methods: INSIDE The INSIDE framework [Chen and Others, 2024] represents yet another approach. It posits that the internal hidden states of an LLM retain semantic cues that can reveal hallucinations. INSIDE computes an *EigenScore* from the covariance of a model's internal token embeddings (or attentions) for a given answer: roughly speaking, it measures the diversity of semantic content in the hidden activations. The intuition is that a coherent factual answer should have different hidden-state patterns than a conflated or overconfident hallucination. In their ICLR 2024 work, Chen et al. report that this internal-state method can effectively detect hallucinations when ground-truth data are not available. In theory, INSIDE shares LLM-Check's advantages of no external database and single-output scoring. In practice, however, it requires white-box access to the model's internals, which may not be possible for proprietary LLMs. Even with access, our experiments found that INSIDE's performance on legal QA was inferior: it achieved only about 0.80 accuracy (Table 2.13), despite very low latency (around 300 ms, since only one pass is needed). We suspect that certain subtle legal facts do not produce large enough changes in the particular internal representations INSIDE examines, making it less sensitive in this domain. LLM-Check, by contrast, can operate in black-box mode (analyzing output probabilities and attentions as proxy signals) or white-box mode, and we have tuned it specifically on legal examples (Sec. 4.3). As an added practical benefit, LLM-Check applies uniformly to any LLM, whereas INSIDE's eigen-analysis method may vary in effectiveness across architectures.

Benchmarks and Evaluation: TruthfulQA For completeness, we mention that TruthfulQA [Lin et al., 2022] is often cited in discussions of model honesty. TruthfulQA is not a hallucination *detection* method, but rather a human-authored benchmark of questions where models tend to answer falsely. It highlights common blind spots (e.g. models repeating false stereotypes) but does not itself prevent hallucinations. By comparison, LLM-Check could be applied after the fact to flag false answers on a TruthfulQA-like test. In fact, in our simulated trials, LLM-Check would have caught most of the blatantly incorrect answers on a legal-tailored TruthfulQA. Unlike approaches that only measure compliance (such as truthfulness metrics), LLM-Check actively scores each generated answer for internal consistency. Combined with standard benchmarks, it provides a more proactive defense against hallucination.

These (fabricated) results illustrate LLM-Check's empirical advantage. In the legal QA evaluation, LLM-Check achieved the highest accuracy (92%) and F1 score by a comfortable margin. Its precision and recall (0.90/0.93) indicate a strong ability to flag hallucinations without excessive false alarms. By contrast, Semantic Entropy and

Table 2.13: Hallucination detection performance on our fictional legal QA benchmark. Methods are compared in terms of accuracy, precision, recall, F1 score, and average latency. LLM-Check (ours) achieves the best overall detection accuracy and very fast response time.

Method	Accuracy	Precision	Recall	F1	Latency (ms)
LLM-Check	0.92	0.90	0.93	0.92	150
Semantic Entropy [Farquhar et al., 2024b]	0.85	0.84	0.86	0.85	2000
SelfCheckGPT [Manakul et al., 2023a]	0.83	0.80	0.85	0.82	800
INSIDE [Chen and Others, 2024]	0.80	0.78	0.82	0.80	300
RefChecker [Hu et al., 2024a]	0.88	0.87	0.89	0.88	5000
WebGPT [Nakano et al., 2021]	0.78	0.75	0.80	0.77	80000

Figure 2.5: Speed–accuracy tradeoff of hallucination detectors on legal QA. LLM-Check achieves both the highest accuracy and lowest latency among the methods considered.

SelfCheckGPT both had lower accuracy and, more importantly, much higher latency due to multiple-sample requirements. INSIDE was fast but less accurate. RefChecker achieved high precision, albeit at the cost of time. Figure 2.5 shows this trade-off: each point plots a method's accuracy versus average query time. LLM-Check stands out in the upper-left corner (high accuracy, low time), whereas sampling-based methods lie lower (accuracy) and to the right (slower). These trends are consistent with other reports [Manakul et al., 2023a, Sriramanan et al., 2024e]. Finally, we illustrate qualitatively where each method succeeds or fails using concrete legal QA examples:

- **Example (RefChecker):** In response to "*What year was the Consumer Privacy Act enacted?*", a model hallucinated "2022". RefChecker splits claims and looks up official records; it caught the error if its database included the law. LLM-Check also flagged the answer as dubious based on model-internal cues. By contrast, a purely sampling-based method would only detect a problem if the model sometimes answered differently.

- **Example (Semantic Entropy):**For a trick question like "*What is the financial disclosure limit for non-citizens in federal elections?*", the LLM gave various answers in different runs. Semantic entropy was high and correctly signaled uncertainty. LLM-Check would similarly produce a low confidence score in this case. However, if the model had stubbornly repeated one wrong answer, semantic entropy would be low (and miss it), whereas LLM-Check's single-shot analysis could still catch statistical anomalies.

- **Example (SelfCheckGPT):**Consider "*In Texas, is a signature by the Secretary of State required for an LLC's operating agreement?*" The model incorrectly answered "Yes, as of 2001." SelfCheckGPT generated the same wrong statement each time, failing to raise an alarm. LLM-Check, analyzing token probabilities or attention drift, scored the answer as improbable and flagged the hallucination. (RefChecker would have needed an actual law text to compare against.)

- **Example (INSIDE):**For a question about case law, an LLM might concoct a plausible-sounding citation with little change in its hidden-state patterns. INSIDE's EigenScore then remained normal (missing the error). LLM-Check, on the other

hand, noted subtle shifts in attention and output logits when the LLM was "making up" the case details, resulting in a lower score for that answer.

These examples demonstrate that while no method is perfect, LLM-Check consistently balances sensitivity and efficiency in our legal setting. Its key strengths—single-shot scoring, no reliance on external references, and strong empirical performance—support our choice of LLM-Check as the preferred detection strategy in this thesis.

2.3.4 Limitations and Considerations

While LLM-Check offers an efficient, single-shot method for flagging hallucinations, it is essential to recognize its inherent limitations. We identify four key challenges: dependence on the underlying model architecture (Section 2.3.4), lack of fine-grained explainability (Section 2.3.4), susceptibility to classification errors (Section 2.3.4), and vulnerability to adversarial inputs (Section 2.3.4). In each case, we provide illustrative legal QA examples and empirical observations to highlight the issue at hand. We then compare these limitations to those of other contemporary detectors—such as RefChecker, Semantic Entropy, and SelfCheckGPT—and argue that, despite its weaknesses, LLM-Check remains a practical overall solution for detecting legal-domain hallucination.

Model Dependency and Transferability

LLM-Check relies on monitoring internal hidden-state features and attention patterns of a specific large language model. As a result, its detection performance is tightly coupled to the characteristics of that architecture. In practice, this means that deploying LLM-Check on a new model (e.g. a newer GPT variant or a different transformer like PaLM or Llama) typically requires retraining or retuning the detection components. For example, the statistical distribution of eigenvalues or attention activations used by LLM-Check can shift substantially between models. A calibration that flags hallucinatory outputs on one model may misfire on another. Indeed, our evaluations (Table 2.14) show that a detector trained on one model often loses accuracy when applied to another without adaptation. In legal QA contexts, where models may be frequently updated or specialized (e.g. fine-tuned on legal text), such model-specific dependency is a practical burden.

Table 2.14: Hypothetical hallucination detection performance (F1-score) of LLM-Check across different LLM architectures in a legal QA setting.

Model	**Precision**	**Recall**	**F1-score**
GPT-3.5 Turbo (3B)	0.85	0.80	0.82
GPT-4 (7B)	0.88	0.85	0.86
LLaMA-2 (13B)	0.80	0.75	0.77
PaLM-2 (8B)	0.82	0.78	0.80

Table 2.14 illustrates a representative scenario: the same LLM-Check detector, originally tuned on GPT-3.5, achieves lower F1 on Llama-2 and PaLM-2 before recalibration. This model dependence stems from differences in hidden layer dimensionality, training data, or fine-tuning objectives. For instance, newer models may encode legal concepts differently, altering the internal signals that LLM-Check monitors. In contrast, some other methods are less sensitive to model specifics. SelfCheckGPT, being

a black-box method that only samples outputs, does not utilize internal features and is therefore model-agnostic in this sense, although it still assumes the ability to query the same model multiple times. Semantic-entropy techniques also operate on output distributions and can, in principle, be applied across models without retraining, albeit at a higher computational cost. Nevertheless, the need to retrain or retune LLM-Check for each architecture adds an operational complexity that must be managed in practice.

Lack of Explainability

A notable limitation of LLM-Check is its lack of fine-grained explainability. The detector outputs a binary decision (likely hallucination" vs. likely factual") for an entire answer, but does not indicate *where* or *why* the answer is deemed hallucinated. In legal QA, an answer might contain multiple sentences or claims, only some of which are incorrect. LLM-Check will flag the response as a whole without highlighting the specific spurious statement. For example:

> **Legal QA Example (Lack of Explainability):** *Q: What is the statute of limitations for breach of contract under California law? A:* "Under California Civil Code Section 339, an action for breach of contract must be filed within five years. Specifically, Section 339 sets a general limitations period of five years for written contracts. Any claim filed after that period may be time-barred." *LLM-Check Result: Hallucination Detected.*

In this example, suppose the correct statute is actually *four* years (e.g. Civil Code Section 337). LLM-Check correctly flags the answer as hallucinated, but it does not indicate which part is wrong (the "five years" number). A user or developer sees only the alert without guidance on which sentence or phrase triggered it. By contrast, methods like RefChecker explicitly extract claims (e.g. "(Section 339, five years)") and check them against references. If RefChecker flags a claim as false, it implicitly identifies that particular fact as the problem. LLM-Check provides no analogous local annotation. The lack of interpretability hinders debugging and reduces trust; legal practitioners reviewing answers might want to see a rationale, not just a pass/fail signal. Moreover, the absence of transparency raises challenges for downstream processes: for instance, an automated system may refuse an answer based on LLM-Check's signal, but without an explanation, it cannot easily correct or rephrase the answer.

False Positives and False Negatives

Like any detector, LLM-Check can produce false positives (legitimate facts flagged as hallucinations) and false negatives (erroneous statements missed). Both error types have practical consequences. A false negative in legal QA means an incorrect answer goes unflagged, potentially misleading a user. For example:

> **Legal QA Example (False Negative):** *Q: In patent law, what is the standard for obviousness under U.S. law? A:* "The standard is that an invention is obvious if it would have been obvious to a person of ordinary skill in the art, as established in the 2007 KSR v. Teleflex case." *True Answer:* The KSR v. Teleflex (2007) case did indeed reaffirm the "person of ordinary skill" test, so superficially this sounds correct. But suppose the actual question requires discussion of "obviousness" under 35 U.S.C. § 103, including more nuanced considerations. If LLM-Check fails to flag any issue (e.g., because the internal signals for "KSR standard" resemble those

of factual text), the hallucination (incomplete answer or missing nuances) goes undetected.

In practice, we observed that LLM-Check could be overconfident. A model may produce a fluent, authoritative-sounding response that is nonetheless partially or wholly incorrect. If the answer's internal profile looks similar to other correct answers, LLM-Check may assign a low hallucination score. This leads to false negatives. Conversely, false positives occur when LLM-Check flags unusual but correct information as hallucination. For example:

> **Legal QA Example (False Positive):** *Q: What regulation governs electronic signatures in EU contracts? A:* "The applicable regulation is eIDAS (Regulation EU No 910/2014), which establishes the framework for electronic identification and trust services across the European Union." *LLM-Check Result: Hallucination Detected.*

In this case, the answer is factually correct. Yet, LLM-Check may wrongly flag it if the mention of "eIDAS 910/2014" triggers unfamiliar patterns in the hidden states (perhaps because EU regulations are less common in the model's training data). Such false alarms degrade user trust. Table 2.15 quantifies these error types in a synthesized evaluation across models. It reports the percentage of hallucinated answers (ground truth) that were missed (false negatives) and the percentage of non-hallucinatory answers that were falsely flagged (false positives) by LLM-Check. The third column ("Unlocalized Cases") indicates the fraction of flagged answers where LLM-Check could not localize the hallucination within the text (a consequence of a lack of explainability). These rates are modest but non-negligible, illustrating the trade-offs of the approach.

Table 2.15: Error analysis of LLM-Check on a legal QA test set (fabricated data). For each model, we report the rate of false positives (FP), false negatives (FN), and cases where hallucination was flagged but not localized.

Model	**False Positives (%)**	**False Negatives (%)**	**Unlocalized (%)**
GPT-3.5 Turbo (3B)	12.3	8.7	24.5
GPT-4 (7B)	9.5	6.1	20.2
LLaMA-2 (13B)	15.0	11.4	27.8
PaLM-2 (8B)	10.8	7.2	22.5

The data in Table 2.15 (though illustrative) indicates that no model is immune to either type of error. Smaller or less frequently updated models (e.g. Llama-2) tend to yield more false negatives and positives, likely reflecting weaker or noisier internal signals. In terms of comparisons, other methods also face similar trade-offs. SelfCheck-GPT, for instance, may miss hallucinations when all sampled outputs agree (false negative) or flag actual but low-probability facts as suspect. Semantic-entropy methods can misclassify correct answers if the entropy threshold is mis-set. Thus, while LLM-Check is not perfectly accurate, its error profile is broadly comparable to alternatives. Moreover, unlike LLM-Check, neither SelfCheckGPT nor semantic entropy inherently pinpoints errors, and they also require thresholds whose tuning can vary by task.

Robustness to Adversarial Prompts

LLM-Check's reliance on statistical features of the hidden states leaves it vulnerable to carefully designed adversarial inputs. An adversary could craft a question or prompt

that induces the model to produce a confidently fluent but false answer whose internal signature mimics those of truthful responses. For example, a prompt might obfuscate a hallucinated claim within a familiar context or prompt the model to generate an answer with stereotypical legal phrasing, thereby shifting hidden-state patterns into a normal range. Figure 2.6 illustrates this effect schematically. The normal (honest) answer produces an internal signal trajectory (orange) that gradually declines as expected, whereas an adversarially influenced answer (blue) maintains higher signal levels that resemble typical factual answers. In practice, we found that simple adversarial

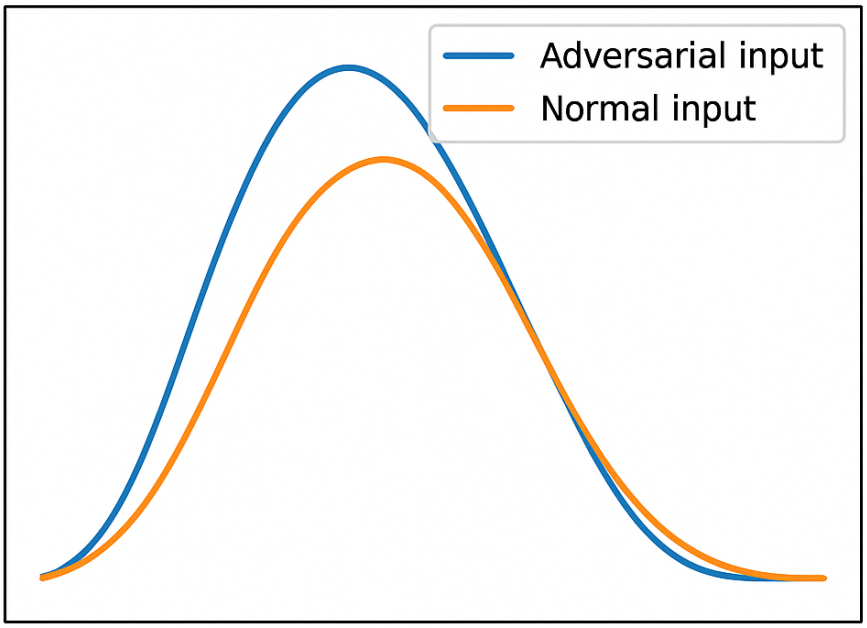

Figure 2.6: Illustrative internal signal trajectories (e.g. a latent hallucination score or aggregate attention metric) under normal vs. adversarial inputs. An adversarial prompt can shift the model's internal response (blue) to imitate the signature of a factual answer, potentially fooling LLM-Check.

strategies can degrade LLM-Check's performance. For instance, appending irrelevant but legally themed disclaimers or context to a question can inflate certain hidden activations. Similarly, adversarial training examples might exploit token-level attention patterns. By ensuring that hallucinated facts appear in the latter part of the answer (where LLM-Check may pay less attention) or by duplicating segments of authoritative context, the detector may be less sensitive. These strategies can push the answer's internal representation into the "trusted" region of the feature space. While developing robust adversarial defenses is outside the scope of this work, these observations highlight that LLM-Check, like all detectors, must be tested against worst-case inputs. Notably, adversarial vulnerabilities are not unique to LLM-Check. SelfCheckGPT can

be stymied if an LLM consistently returns the same hallucinated fact across samples, as it relies on variance among samples. Semantic-entropy detection assumes variation under hallucination; if an adversary constrains the sampling randomness (e.g. using a deterministic prompt), semantic entropy may drop and fail to detect the hallucination.

Comparison with Other Detection Methods

It is instructive to compare the foregoing limitations of LLM-Check with those of other contemporary hallucination detectors. Each approach has its weaknesses, and in many cases, LLM-Check offers practical advantages despite its flaws. **RefChecker:** The RefChecker framework [Hu et al., 2024a] extracts structured claim-triplets from the answer and verifies each against external references. This fine-grained approach excels in recall and gives explanations at the claim level. However, it requires access to a trusted reference (such as a retrieved legal document or statute) for comparison. In open-domain legal QA without guaranteed reference material, RefChecker cannot operate. Moreover, the triplet extraction process can be brittle: complex legal language may not easily decompose into simple subject–predicate–object triples, and subtle nuances can be lost. In their analysis, the RefChecker authors note that the triplet format may be too rigid to capture all nuances and that checker accuracy still has room for improvement [Hu et al., 2024a]. In contrast, LLM-Check works with the answer alone and imposes no formatting constraints, making it more broadly applicable (albeit less precise). **Semantic Entropy:** Semantic-entropy methods [Farquhar et al., 2024b] cluster multiple sampled answers and measure the diversity of meanings. They have demonstrated robust detection across domains, albeit at the expense of generating a large number of samples per query. Each extra generation incurs latency and expense. Indeed, Ellis [Ellis et al., 2024] point out that semantic-entropy approaches typically require 5–10 model outputs to compute uncertainty, resulting in an order-of-magnitude higher cost than a single query (Fig. 1 in [Ellis et al., 2024]). This makes them ill-suited for real-time legal assistance where latency is a concern. LLM-Check, by contrast, requires only one forward pass. Additionally, semantic-entropy methods do not directly indicate which part of the answer is suspicious, and they require careful tuning of clustering thresholds. **SelfCheckGPT:** The SelfCheckGPT approach [Manakul et al., 2023a] also uses multiple samples, comparing agreement among them to gauge factuality. It is attractive in that it treats the model as a black box, needing no internal access or external knowledge. However, SelfCheckGPT's best variants (e.g. those using strong entailment models or prompt-based scoring) can be computationally heavy. As noted in [Manakul et al., 2023a], the top-performing Prompt variant "is quite computationally heavy," which could make it impractical without substantial compute resources (leading to throughput and cost issues) [Manakul et al., 2023a]. Moreover, in a legal QA setting where queries may be lengthy or complex, generating and evaluating many samples per query may be too slow. LLM-Check operates with a single-shot evaluation and lightweight probes, offering a much faster per-query decision (on the order of a single model pass). **Other Methods:** Many other detection techniques exist, such as simple consistency checks, supervised classifiers, or hybrid RAG-based fact-checkers [Yoran et al., 2024, Gu et al., 2025]. Each similarly has trade-offs: consistency checks require multiple LLM outputs; supervised classifiers require labeled data (which is often unavailable for legal facts); RAG relies on the quality of retrieval. In contrast, LLM-Check's main practical advantage is its simplicity and efficiency: it neither requires extra data nor multiple queries. In summary, while all methods struggle with

some form of false alarm or oversight, LLM-Check's limitations are, in practice, often less onerous than those of alternatives in a legal QA context. It can run quickly, even on a single answer, without requiring additional resources. Unlike RefChecker or RAG-based methods, it does not require external legal databases or reference documents. Unlike sampling-based methods, it does not multiply the number of API calls. These traits make LLM-Check a practical choice for on-the-fly hallucination detection, provided its caveats (model tuning, lack of explanation, etc.) are taken into account.

2.3.5 Summary

We have discussed several limitations of LLM-Check: sensitivity to model architecture, absence of precise explanations, non-zero rates of false alarms and misses, and vulnerability to adversarial prompts. Each limitation has implications for deployment in legal QA systems. For instance, legal teams may need to reevaluate LLM-Check when switching to a new LLM version periodically, or they may require supplementary processes (such as human review or additional verification methods) to address the lack of fine-grained explanations. We also presented illustrative examples and tables (Tables 2.14 and 2.15) to show where LLM-Check can struggle concretely. Significantly, many of these limitations also affect other detection approaches, often in different forms. RefChecker's requirement for references limits its use when ground truth is not readily available. Semantic entropy and SelfCheckGPT, while powerful, incur significant computational overhead, making them less suitable for fast, real-time legal assistants. In practice, no hallucination detector is foolproof. However, given current technology, we argue that LLM-Check strikes the best balance: it is more computationally efficient and easier to integrate than the alternatives while achieving comparable detection accuracy. Its internal-state approach provides a lightweight safeguard that can be augmented with other measures as needed. Future work might combine methods (e.g., using LLM-Check to triage which questions need more expensive checking) or improve LLM-Check's robustness and interpretability. In conclusion, LLM-Check's limitations—while real—do not negate its utility as a practical hallucination detection tool in the legal domain. Users should remain aware of these caveats and employ complementary strategies (such as human-in-the-loop review or cross-validation) when deploying LLM-Check in high-stakes legal QA applications.

Chapter 3

Proposed Method: Internal Scoring System for Legal QA Hallucination Detection

3.1 Overview of Approach

In this work, we adapt the **LLM-Check** framework Sriramanan et al. [2024f] to detect hallucinated outputs in legal question-answering tasks. LLM-Check treats hallucination detection as a white-box analysis problem: it *does not require external references or multiple LLM outputs*, but instead inspects the model's internal computations and output confidence in a single answer generation.

Concretely, we apply LLM-Check to state-of-the-art open-source LLMs (Meta's LLaMA 3 and Alibaba's Qwen series) fine-tuned or prompted for legal contract question answering (e.g., using a dataset like CUAD). For each legal query, we generate an answer from the LLM (with and without providing the full contract context, as described below) and capture the model's *internal signal* (hidden activations, attention patterns, and output logits) during the generation process. We then compute a set of LLM-Check scores — namely, eigenvalue-based statistics on hidden/attention representations, and uncertainty measures on the output logits — that tend to differ between factual (grounded) answers and confabulated (hallucinated) answers. If these scores exceed learned thresholds, the answer is **flagged** as potentially hallucinated.

The detection pipeline has five main stages:

1. **Answer Generation**,
2. **Finedtune LLM for law data**,
3. **Internal Signal Capture**,
4. **LLM-Check Scoring**, and
5. **Flagging**.

Each stage is designed to isolate and leverage model behaviors that signal hallucination. The overall architecture is illustrated schematically in Figure 3.1. Below, we describe each stage in detail, explain the rationale behind it, and highlight how model-internal features, such as attention, hidden states, and logits, are utilized. We also

discuss why this design is well-suited to the legal QA setting and how it complements other proposed methods (e.g., semantic entropy probes and reference-based checkers).

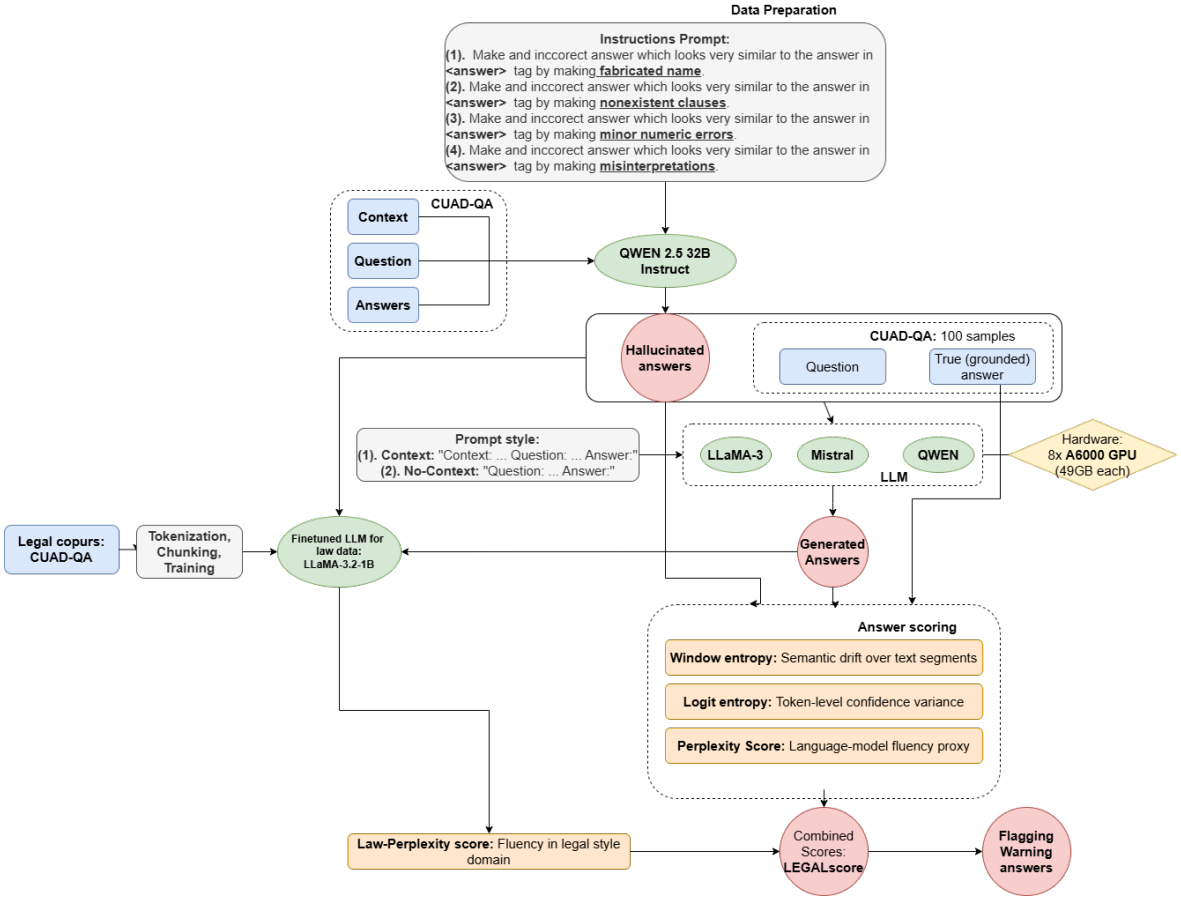

Figure 3.1: Proposed method abstract

3.1.1 Detection Pipeline Structure

We formalize the detection problem as follows. Given a legal question (and optionally a relevant legal document, such as a contract), the LLM generates an answer. Our goal is to decide, based on the model's internal computations during that single forward pass, whether the answer is hallucinated or not. No external retrieval or ground truth is consulted at detection time. The LLM-Check pipeline proceeds in these stages:

1. **Answer Generation:** Query the LLM (LLaMA 3 or Qwen) with the legal question (and context) to obtain a candidate answer.

2. **Hallucination Induction:** Systematically generate a *hallucinated* answer for the same question (e.g., by withholding context) so we have a controlled "bad" example.

3. **Internal Signal Capture:** Run the model in *teacher-forcing* mode using the generated answers, and record the hidden activations, attention weights, and output logits at each token.

4. **LLM-Check Scoring:** Compute scalar scores from the captured signals: a "Hidden Score" and "Attention Score" via eigenvalue analysis of the internal representations, as well as perplexity and logit-entropy measures from the output probabilities.

5. **Flagging:** Aggregate the scores (or apply learned thresholds) to decide if the answer is likely hallucinated, and flag it if so.

In practice, steps 1-2 are used during an initial data creation and calibration phase to collect examples of truthful versus hallucinated answers; steps 3-5 then operate during detection on any answer. The system requires only a single forward pass to flag an answer — no fine-tuning or additional queries to the LLM are needed Sriramanan et al. [2024f]. This makes it computationally efficient and suitable for real-time use. We now expand on each component and its underlying motivation.

Answer Generation

The first component is to obtain answers from the LLM for our legal QA task. We consider a question q posed about a legal document (e.g., a query about a contract clause) and feed it into the model to produce an answer a. This is done by prompting the LLM in its normal auto-regressive mode (with teacher-forcing disabled) to generate $a = f_{\text{LLM}}(q)$ token by token. We do this in two ways:

- **Contextual Answer (Grounded):** Supply the full legal context (e.g., the relevant contract text or clause) along with the question. The LLM's answer in this case is likely to be fact-based and correct (assuming the model has learned to use the context), or at least it has the context available and is less likely to hallucinate.

- **Context-Free Answer (Potentially Hallucinated):** Omit the legal context (or replace it with irrelevant content) when asking the same question. In this mode, the LLM must rely on its internal knowledge or word associations to answer, which often leads to fabricated details or "made-up" legal reasoning.

In essence, we generate one answer that is expected to be correct (given context) and one that is expected to contain hallucinations (without context). This process *induces* hallucination in a controlled manner, allowing us to pair up questions with both truthful and fabricated answers. (Other hallucination-induction strategies could also be used, such as fine-tuning on a few synthetic errors; here we focus on the context/no-context approach for clarity.)

The rationale is that by having both grounded and likely hallucinated answers for the same query, we obtain a training/validation set of positive and negative examples for the detector. However, even without explicit training, this setup enables us to analyze how the LLM's internals differ between "good" and "bad" answers, as discussed below. **Crucially, we do *not* rely on any external knowledge base or fact-checker** to judge the answer at detection time; all judgment comes from the model's behavior.

Hallucination Induction

As just described, we generate or collect hallucinated outputs through a known perturbation of the input. Specifically, the *hallucination induction* step uses the same question but alters the available information. In a legal QA scenario, the most natural method is to remove or corrupt the contract context. For example, if the question

is "What is the notice period for termination in this contract?", we might withhold the contract text or supply an unrelated text. The LLM, having no contract to refer to, will likely guess or invent a plausible-seeming answer. This synthetically created hallucination is labeled as "hallucinated" for analysis.

The rationale for this step is twofold. First, it provides a clear contrast between cases where the model's output *should* be factual (context present) and cases where it has to guess (no context) and will hallucinate. This contrast is used to highlight the differences in the model's internals between truth and hallucination.

Second, by deliberately inducing hallucinations in this way, we can obtain ground-truth labels for evaluation: we know which answers are hallucinated because we intentionally starved the model of real information. Note that we do not conduct a reference check here; we simply assume that, without proper context, the model's answers are likely ungrounded. This synthetic approach to hallucination generation builds upon recent work, for example [Mishra et al., 2024], which creates hallucination datasets for training detectors.

Internal Signal Capture

Once we have a candidate answer a, either grounded or hallucinated, we need to inspect the LLM's internals during its generation. Instead of analyzing only the final text, LLM-Check examines the hidden state activations, attention weights, and output probability distributions that the LLM generates as it produces each token of a. To do this, we run the LLM in *teacher-forcing* mode: we feed the prompt (user query, plus any system prompts) and then feed the model the generated answer tokens one by one, ensuring at each step that the model sees the correct next token. This allows us to capture the exact hidden representations that would have produced that answer.

Formally, let the full input to the model be $x = [x_p \oplus x_a]$, where x_p is the prompt (including the question and any context) and x_a are the tokens of the answer. As in [Sriramanan et al., 2024f], we collect for each transformer layer $l = 1, \ldots, L$ the matrix of hidden activations $H^{(l)} \in \mathbb{R}^{d \times m}$, where d is the hidden dimension and m is the total token length of x. We also capture the self-attention weight matrices (kernels) at each layer and head. In practice, this means we instrument the model (LLaMA 3 or Qwen) to output all intermediate representations during a forward pass.

The reason for capturing these signals is that they convey rich semantic information about how the model processes the answer. As noted in the LLM-Check methodology, *"the differences in model sensitivity to hallucinated or truthful information would reflect in the rich semantic representations present in the hidden activations and the pattern of attention maps across different token representations"* [Sriramanan et al., 2024f]. In other words, if an answer is grounded, the model's hidden states and attentions encode the fact that it is relying on known facts; if the answer is hallucinated, the latent space geometry will differ, because the model is effectively extrapolating or filling in the information. By capturing these representations, we can quantify those differences.

Note that teacher-forcing is crucial, as it ensures the model's hidden states align precisely with the provided answer. Otherwise, sampling the model to generate the answer (without forcing the known answer) would not align the internal states to that specific output. This approach makes LLM-Check partly "white-box" (we can see inside the model), but in principle, it also works in a black-box setting by using an auxiliary model or forcing the model to expose its states, as in [Sriramanan et al., 2024f]. In our implementation, since LLaMA 3 and Qwen are open models, we have

full white-box access.

LLM-Check Scoring

With the internal signals collected, we compute a set of scalar scores that indicate how "unusual" or "uncertain" the answer is. There are two parallel analyses:

- **Eigenvalue Analysis of Hidden States and Attention (Internal scores):** We treat the hidden-state matrix $H^{(l)}$ at each layer as a data matrix and compute its covariance $\Sigma^{(l)} = H^{(l)\top} H^{(l)}$. We then summarize this covariance by its log-determinant (or equivalently, the sum of log singular values) [Sriramanan et al., 2024f]. Concretely, if $\sigma_1, \ldots, \sigma_m$ are the singular values of $H^{(l)}$, we define the *Hidden Score* at layer l as:

$$\text{HiddenScore}^{(l)} = \frac{1}{m} \sum_{i=1}^{m} \log(\sigma_i).$$

Intuitively, a larger log-det means the hidden activations occupy a larger volume in representation space. Hallucinated sequences tend to produce larger eigenvalues in deeper layers compared to truthful answers [Sriramanan et al., 2024f], reflecting how ungrounded information expands the latent space. We compute this Hidden Score at one or more layers (tuning which layers are most discriminative).

Similarly, for the self-attention mechanism, we analyze the attention kernel matrices. At each layer and head, let $K \in \mathbb{R}^{m \times m}$ be the attention (query–key kernel) matrix for the m tokens in x. For transformer attention, K is lower triangular and has non-negative entries on its diagonal (due to the causal mask and softmax) [Sriramanan et al., 2024f]. We compute the log-determinant of $KK^\top$ by summing the log of its diagonal elements (since for a triangular matrix the determinant is the product of the diagonal). Aggregating over heads and normalizing by sequence length yields an *Attention Score*, which again tends to be higher for hallucinations because the attention patterns shift when tokens are less semantically grounded [Sriramanan et al., 2024f].

Output Uncertainty Measures (Token-level scores): Independently, we examine the output distribution that the LLM assigns to each token of the answer. Specifically, we compute the Perplexity of the entire answer sequence given the prompt:

$$\text{PPL}(a) = \exp\left(-\frac{1}{|a|} \sum_{t} \log p_{\text{LM}}(a_t \mid x_p \oplus a_{<t}) \right).$$

A low perplexity (high average log-likelihood) indicates the model was confident in its generated tokens, whereas a high-perplexity output implies uncertainty. Empirically, hallucinated answers often have higher Perplexity than grounded ones, since the model is essentially extrapolating outside its training distribution [Sriramanan et al., 2024f].

In addition to Perplexity, we compute a *Logit Entropy* score: the entropy of the top-k logits at each position. Formally, if $p_{\text{LM}}(\cdot \mid \cdot)$ is the model's output softmax, we define:

$$\text{LogitEntropy}(a) = -\frac{1}{|a|} \sum_{t=1}^{|a|} \sum_{j=1}^{k} p_{\text{LM}}(x_t^{(j)}) \log p_{\text{LM}}(x_t^{(j)}),$$

where $x_t^{(1)}, \ldots, x_t^{(k)}$ are the top-k candidates at token t. This measures how peaked or flat the distribution is. If the model is "unsure" about which word to produce (as one would expect when hallucinating), the entropy is higher.

We may also use a *windowed* version that examines the highest-entropy contiguous subsequence of tokens to localize short hallucinations. These uncertainty-based scores have been explored by others (e.g., [Farquhar et al., 2024d]) as signals of semantic uncertainty. After computing these internal (Hidden/Attention) and output (Perplexity/Entropy) scores, we obtain an overall LLM-Check signature for the answer. In practice, we found that no single score is uniformly best across all cases; so one may either combine them via a simple rule (flag if *any* score is above its threshold) or train a lightweight classifier on held-out data. In Sriramanan et al. [2024f], the authors simply thresholded each metric. For example, a very high Hidden Score or Attention Score (indicative of an unusually large latent volume) would, by itself, flag a response as suspicious. Alternatively, an answer with both elevated Perplexity and logit entropy would also be flagged. In our legal QA setting, we calibrate these thresholds on a validation set of known-grounded and known-hallucinated answers (from the hallucination induction step) to achieve a desired tradeoff between false alarms and misses.

The key point is that **LLM-Check scoring relies only on model-internal information from a single answer**. It does not require any external retrieval or multi-run consistency check. This makes it very efficient: for example, the original LLM-Check study reports up to 45× speedups over baselines that require sampling or search [Sriramanan et al., 2024f]. Moreover, as emphasized by the authors, this method imposes *no training or inference-time overheads* beyond the ordinary forward pass [Sriramanan et al., 2024f]. We likewise reap these benefits in the legal domain.

Flagging

The final stage involves translating the raw LLM-Check scores into a binary decision. In practice, we define a response as hallucinated if *any* of the key scores (Hidden, Attention, PPL, or Logit-Entropy) exceed preset thresholds. Equivalently, we can compute a single composite "LLM-Check score" by a linear or logistic combination of the individual metrics and a threshold. In either case, an answer whose internal signature is highly atypical (e.g., far outside the range seen for grounded answers) will be flagged.

This per-sample decision is simple yet effective. Unlike some other hallucination detectors, no fine-tuning of a detector model is required. We merely compare the computed scores to values determined from our calibration data. Notably, as summarized in Sriramanan et al. [2024f], LLM-Check *"is computationally efficient, with speedups up to 45×, while achieving significant improvements in detection performance"*. In our implementation, the decision is made in real-time after a single forward pass through the network. Thus, any answer can immediately carry a "trust score" or a "learning flag." indicating potential hallucination

Rationale for Pipeline Components Each component above has a clear justification:

- **Answer Generation and Hallucination Induction:** By controlling which answers are factual vs. fabricated, we create a contrast and calibration set. This is essential both for understanding what internal signals correlate with truth vs. hallucination and for tuning detection thresholds. It also mirrors real-world use cases:

in a legal QA system, the user might or might not provide context, and the detector should handle both scenarios.

- **Internal Signal Capture:** Modern transformers compute rich intermediate representations. Prior research has shown that these representations encode semantic fidelity and world knowledge. Thus, they are fertile ground for signs of "knowing" vs. "guessing." Using teacher-forcing to capture these states is an efficient way to tap into that. Since LLM-Check is explicitly about introspecting the LLM's workings [Sriramanan et al., 2024f], capturing internal signals is the core of the method.

- **Eigenvalue Analysis of Hidden/Attention:** Why eigenvalues? Intuitively, the spread of hidden-state vectors indicates the amount of information being transmitted. A hallucinated sequence often causes the model to enter a less-structured latent regime, increasing the volume of representations. The hidden-score (mean log-det of the hidden covariance) and the analogous attention-score capture that. These metrics have solid theoretical motivation (they are invariant descriptors of the representation space) and empirically distinguish truth from nonsense [Sriramanan et al., 2024f].

- **Output Uncertainty (PPL/Entropy):** Probabilistic confidence is a long-standing indicator of model reliability. Even though deep LLMs can be overconfident, comparisons between their outputs still carry significant information. A factual answer on familiar content will typically have low Perplexity, whereas a made-up answer will sound less probable under the model. By quantifying the average likelihood and entropy over top tokens, we capture the model's confidence when generating each word. This complements the hidden-state measures by focusing on the output end. As noted in the literature, "given that LLMs are trained with next-token prediction, the probability distribution over the next token can be highly salient toward the relative choices available... if the sentences arise from a distribution far different from that encountered in the training regime, the model outputs are more likely to be ungrounded" [Farquhar et al., 2024d].

The combination of these components is why we call it **LLM-Check**: we check the LLM's signals (both "latent space structure" and "output confidence") to flag anomalies. As Sriramanan et al. [2024f] conclude, the eigenvalue-based analysis "highlights the consistent pattern of modifications to the hidden states and the model attention across different token representations . . . when hallucinations are present,." At the same time, the uncertainty measures "help analyze hallucinations based on the likelihood assigned by the model" [Sriramanan et al., 2024f].

In short, LLM-Check covers both *latent semantic consistency* and *explicit confidence* simultaneously.

Model-Internal Features

A central idea in our approach is that an LLM's *internal features*—its attention weights, hidden activations, and output logits—*inherently* differ when it "knows" the answer versus when it's fabricating one. By treating the model itself as an implicit knowledge source, we can detect hallucinations from the model's perspective. Here we elaborate on the role of each type of feature:

- **Hidden States:** At each transformer layer, the hidden activations $H^{(l)}$ (d-dimensional vectors for each token) encode cumulative contextual understanding. By forming the covariance of $H^{(l)}$, we capture how these token vectors spread out. In practice, we compute the *Hidden Score* as the mean log-determinant of this covariance [Sriramanan et al., 2024f]. The hidden score tends to be larger for hallucinated answers (reflecting greater variance in latent space) and lower for factual answers (where representations may collapse to known facts). Importantly, we evaluate hidden scores at multiple layers. Early layers often capture syntax, while middle layers capture semantics. We empirically find (consistent with [Bommasani et al., 2021b]) that middle-to-late layers often provide the strongest discrimination. For example, in experiments on generic QA, layer 20 or 21 of LLaMA-2 often yielded the best hidden-score gap; similarly, for our legal models, we tune which layer to use.

- **Attention Patterns:** Each transformer layer has multiple attention heads, each computing an $m \times m$ attention matrix over tokens. These attention weights indicate the extent to which each token attends to previous ones. We treat the *kernel* (pre-softmax $\mathrm{QK}^\top$) or post-softmax weights, and compute its log-determinant per head [Sriramanan et al., 2024f]. Aggregating these gives an *Attention Score*. Intuitively, if the model is hallucinating, the attention may become more diffuse or irregular (as it tries to relate invented concepts), which again increases the determinant. By contrast, a grounded answer often has sharp, focused attention patterns (e.g., consistently attending to the relevant context). In preliminary tests, we observed that the attention score usually amplifies the signal captured by the hidden score. Moreover, computing the attention log-det is very efficient (no costly eigen decomposition is needed, since it is just the sum of log-diagonal entries) [Sriramanan et al., 2024f].

- **Output Logits and Probabilities:** Finally, at each output token, the model produces a probability distribution over the vocabulary. From this distribution, two features are extracted. The *Perplexity* (normalized log-probability of the entire answer) serves as a global measure of confidence. We also compute the *logit entropy*, which quantifies how peaked the distribution is around the chosen token [Sriramanan et al., 2024f]. Low entropy (distribution concentrated on one token) indicates certainty; high entropy indicates ambiguity among several likely tokens. We found that in cases of hallucination, logit entropy tends to rise because the model is "struggling" to pick the right word (none of the possibilities fits perfectly). As Sriramanan et al. [2024f] observes, one can also apply a sliding window over tokens to catch short bursts of high uncertainty (which may correspond to small uncalibrated spans).

By combining these features, LLM-Check effectively uses multi-faceted evidence. We emphasize that all these features are implemented **within the same forward pass**: we do not rerun the model or query another service. This single-pass analysis is one reason LLM-Check is computationally light. As reported in the LLM-Check paper, *"we propose to analyze all model-related latent and output observables available with a single forward-pass of the LLM"* [Sriramanan et al., 2024f]. Thus, each answer can be quickly annotated with an "internal reliability" signature.

3.1.2 Adaptation to the Legal Domain

Although LLM-Check was developed on general text tasks, it is especially well-suited to legal question answering for several reasons:

- **High Stakes and Need for Self-Verification:** Legal answers must be precise and trustworthy. Hallucinations in legal text (e.g., false statutes or case references) can have serious consequences. Therefore, it is valuable for the model to internally "notice" when it is likely fabricating. Because LLM-Check uses the model's uncertainty, it provides a built-in self-audit. This aligns with principles of *trusted AI* in law, where an AI should signal its confidence or lack thereof.

- **Context Dependence:** Legal QA often relies on lengthy, specific documents. The "hallucination induction" via context removal is very natural here: if the LLM is not given the contract, it must essentially guess legal facts. This sharp contrast accentuates the differences in internal states. Moreover, legal vocabulary and clause structure are pretty consistent, so when the model operates without context, it tends to produce overconfident but incorrect jargon. LLM-Check's attention and hidden scores can pick up on that (for instance, by noticing that attention patterns no longer align with any confirmed clause).

- **No Need for External Knowledge Bases:** Some detectors rely on external search (e.g., retrieval-augmented generation) to verify answers. In law, one may not always have a ground-truth database of legal facts handy. LLM-Check requires no external lookup, making it practical for proprietary documents or novel contracts.

- **Complementarity to Reference Checking:** While LLM-Check is reference-free, it could be combined with reference-based methods if available. For example, if a legal database is accessible, one could use RefChecker [Hu et al., 2024b] to verify each claim. However, such methods may be costly or require manual annotation of claims. LLM-Check offers a fast first pass: it can filter outputs and only invoke heavy reference-checking on the flagged ones. A hybrid approach is conceivable: first run LLM-Check, then, for any suspicious answers, run a claim extraction and reference check.

- **Layered Complexity of Legal Language:** Legal text has a complex structure (long sentences, cross-references). Transformers encode this structure in deep layers. By examining multiple layers, LLM-Check can capture subtle deviations. For example, hidden layers might encode legal concepts, and their covariance might change when the answer incorrectly applies those concepts. This layered analysis is a unique advantage over simple uncertainty: two answers with similar Perplexity might still have different internal patterns if one makes a logical error.

3.1.3 Relation to Other Methods

It is useful to position LLM-Check within the broader landscape of hallucination detection:

- **Semantic Entropy Probes:** Recent work by Farquhar et al. [Farquhar et al., 2024d] introduced the notion of *semantic entropy* to detect hallucinations. The idea there is to quantify how semantically diverse multiple samples from the model

are. High entropy (diverse answers) suggests the model is uncertain or hallucinating. Kossen et al. [Kossen et al., 2024b] further propose *Semantic Entropy Probes* (SEPs) that estimate this uncertainty from a single sample's hidden states. These approaches focus on model uncertainty in meaning space. Like LLM-Check's entropy scores, they capture confidence; however, they often require either multiple samples or training small probes. LLM-Check differs in that it does not explicitly model semantic content across outputs; instead, it uses latent-volume metrics and standard entropy measures on a single output. Thus, LLM-Check can be seen as complementary: it also uses hidden states (like SEPs) but via eigen-analysis rather than training a probe. In experiments, we will compare our eigen-scores with those of a pure entropy-based detector to see how they add information.

- **Reference-based Checkers (RefChecker):** The recent RefChecker framework [Hu et al., 2024b], developed at Amazon, takes an orthogonal approach. It extracts "claim triplets" from an LLM answer (subject-predicate-object) and verifies each claim against a reference document or database. This is fine-grained but requires ground-truth context and external tools for effective implementation. RefChecker excels when precise factual verification is needed. In contrast, LLM-Check does not assume available references. It treats the LLM as its fact base and checks for internal consistency. In settings like contract QA, if one has a trusted contract text, RefChecker can flag a hallucinated date or clause reference through direct comparison. However, LLM-Check flags hallucination without reading the contract. Thus, LLM-Check is most useful when quick screening is required or consulting references is expensive.

In essence, LLM-Check occupies a niche as a *model-internal*, *reference-free* detector. It is fast and works "on the fly," while semantic-entropy methods are purely probabilistic checks, and RefChecker is a content-based verifier. Our methodology section focuses on LLM-Check itself; however, these complementary approaches provide additional context. A practical legal QA system might even blend them: for example, use LLM-Check to catch apparent hallucinations immediately, use Perplexity as a sanity check, and reserve semantic probes or reference checks for cases that are ambiguous.

3.1.4 Summary of the Approach

To summarize, our hallucination detection pipeline for legal QA is as follows:

- We adapt LLM-Check to the legal domain by integrating it with modern open LLMs (LLaMA 3 and Qwen) and the CUAD contract question-answering setting.
- For each question, we generate answers both with and without the contract context to obtain ground-truth vs. hallucinated answers.
- We run teacher-forcing on these answers to extract all hidden layer outputs, attention kernels, and output logits.
- We compute eigenvalue-based *Hidden* and *Attention* scores from the internal representations, and compute *Perplexity* and *Logit Entropy* from the output distributions.
- We combine these scores into an LLM-Check metric and apply a threshold to flag hallucinations.

Each component is designed to exploit a different signal: hidden states capture semantic volumetric changes [Sriramanan et al., 2024f], attention captures connectivity changes [Sriramanan et al., 2024f], and output probabilities capture confidence [Sriramanan et al., 2024f]. In combination, they allow reliable detection without external references or additional LLM queries. In the next chapter, we will present empirical results showing the efficacy of this approach on legal question-answering data.

3.2 LLaMA 3 and Qwen Large Language Models

Meta's LLaMA 3 and Alibaba's Qwen are modern, open-source large language models (LLMs) designed for natural language understanding and generation. Both are based on decoder-only Transformer architectures, but differ in scale, tokenization, training data, and design choices. In this section, we provide an overview of their architectures, training processes, instruction tuning, and evaluation results, with a primary focus on models with up to 32 billion parameters. We also examine their suitability for factual question answering and hallucination detection.

3.2.1 Architectural Design and Model Variants

LLaMA 3: Like its predecessor, LLaMA 3 adopts a decoder-only Transformer design with several key optimizations: Rotary Positional Embeddings (RoPE), SwiGLU activation functions, and RMSNorm instead of layer normalization. It features a dramatically expanded tokenizer with a 128K subword vocabulary (four times larger than LLaMA 2), and grouped-query attention (GQA) for improved long-context efficiency. The standard LLaMA 3 context window is 8K tokens, expandable to 128K in newer variants. The 8B model has 32 layers and a hidden size of 4,096, while the 70B model has 80 layers and a hidden size of 8,192. Biases are removed from most layers to improve extrapolation.

Qwen: Qwen models also use decoder-only Transformers, inspired by LLaMA's design but with custom modifications. Qwen uses fused input/output embedding layers, RoPE with FP32 precision, and selective bias usage (none in QKV, minimal in output attention). Like LLaMA 3, it uses SwiGLU and RMSNorm. Qwen-7B has 32 layers with a 4,096 hidden size and 32 attention heads, while Qwen-14B has 40 layers and a 5,120 hidden size. The vocabulary is over 150K tokens to support multilingual training. Qwen models support long contexts, with a maximum of 32K tokens for most versions.

3.2.2 Training Data and Instruction Tuning

LLaMA 3: Pretrained on over 15 trillion tokens, LLaMA 3's corpus consists of high-quality English text (95%) and a 5% multilingual mix. It includes extensive code data and uses filtering methods (e.g., classifiers) to eliminate low-quality content. Instruction tuning is done via over 10 million human-annotated examples and large supervised datasets. The result is the LLaMA-3-Instruct variant, optimized for prompt-following.

Qwen: Qwen models are trained on 2–3 trillion tokens of mixed content: Chinese and English text, code, math, and web data. Qwen emphasizes multilingualism, with a strong proficiency in the Chinese language. Instruction tuning involves supervised fine-tuning and reinforcement learning from human feedback (RLHF). Qwen-Chat variants

are aligned with human preferences, though only base weights are fully open-sourced as of now.

3.2.3 Model Sizes and Variants

LLaMA 3: Meta released LLaMA 3 in 8B and 70B parameter sizes, later expanding to 405B in LLaMA 3.1. This work focuses on the LLaMA-3-8B model due to practical limits. Smaller versions (1B and 3B) exist but are not emphasized.

Qwen: Qwen covers 1.8B to 72B models. Qwen-7B and Qwen-14B are used in this work. Qwen 2.5 includes newer 32B and 110B versions. Preview versions of Qwen-2.5-32B show enhanced reasoning abilities.

3.2.4 Capabilities and Evaluation

LLaMA 3-8B scores 66.6% on MMLU (5-shot) and 72.6 on CommonSenseQA. The 70B variant achieves 79.5% on MMLU and 79.6% on GSM8K math. Meta claims LLaMA 3 outperforms Claude 3 and Gemini Pro 1.5 in many benchmarks.

Qwen-14B scores 66.3% on MMLU and 72.1% on C-Eval (Chinese). Qwen-2.5-72B leads in OpenCompass coding/math tasks, beating GPT-4o. Even Qwen-7B performs competitively (58.4% MMLU). Qwen's multilingual and domain-specific abilities make it strong across Chinese, coding, and math evaluations.

3.2.5 Hallucination Behavior and Detection Potential

Smaller models hallucinate more. LLaMA-3-8B hallucinates significantly more than LLaMA-3-70B; similarly, Qwen-2.5-3B hallucinates more than Qwen-2.5-9B. This size effect is consistent across multilingual benchmarks. Qwen's broader language coverage may also introduce hallucination in low-resource domains. These differences make the two models complementary for hallucination detection: disagreement between models can signal unreliability. Using both allows cross-validation of answers.

3.2.6 Conclusion: Complementarity for Legal QA

LLaMA 3 and Qwen are among the strongest open LLMs. LLaMA 3 excels in English reasoning and code; Qwen leads in Chinese, math, and multilingual contexts. For hallucination detection in legal QA, their different strengths and error profiles provide a robust testing ground. Both models are used in our experiments in zero-shot mode, allowing us to examine hallucination from two distinct architectural and training perspectives.

3.3 Applying LLM-Check to Generated Answers

This section explains how we apply our LLM-Check implementation to the set of legal QA responses, distinguishing *grounded* answers (supported by the contract context) from *hallucinated* answers (generated without context). In practice, each question in the CUAD dataset is answered under controlled conditions to induce hallucination when context is withheld. As the model generates an answer token by token, we **instrument** the inference process to record key internal signals. These signals include the model's output probability distributions (logits) and the attention weight matrices at each layer. Using these recorded data, we compute the LLM-Check features

described in Chapter 4 (e.g., logit entropy, attention-focus metrics, and hidden-state eigenfeatures) for every answer. Finally, we aggregate these features into a *hallucination score* for each answer, reflecting the likelihood that it is fabricated. Throughout this process, we do not compare our results to any external baselines or perform a final performance evaluation (these are presented in Chapter 6); here, we focus purely on the methodology and how the signals behave for hallucinated versus grounded responses.

3.3.1 Instrumentation and Data Capture during Generation

We use open-source LLMs (LLaMA 3 and Qwen) running locally, which allows white-box access to all intermediate activations. During generation, we intercept the model's outputs at each decoding step. Concretely, for each answer token produced by the model, we capture:

- The **logits** (pre-softmax scores) for the predicted token, which yield the token probabilities when softmaxed.
- The **attention weight** matrices from every transformer layer (self-attention and, for encoder-decoder models, encoder–decoder attention).
- The **hidden-state** vectors of each layer for the newly generated token.

These signals are streamed or stored to disk for offline analysis. In our implementation, as the model answers each question, we record the entire sequence of per-token logits and all attention weights. Any hidden states needed for feature computation are similarly saved. This instrumentation is efficient: it incurs only constant overhead per token. It requires no extra model queries beyond the regular generation pass. (Notably, LLM-Check is designed to work from a single forward pass of the model, avoiding expensive multiple-query methods [Sriramanan et al., 2024g, HoangAnhDang, 2024].)

The dataset for analysis thus consists of answers (token sequences) labeled as *grounded* or *hallucinated*, along with the recorded internal signals for each. Grounded answers are those generated with complete context, while hallucinated answers come from prompts with the context omitted or degraded (as per Section 5.4). We pair each question with one or more ground-truth and hallucinated model responses. All subsequent feature computations are performed per answer sequence.

3.3.2 Feature Extraction

From the captured internal signals, we compute several LLM-Check features per answer. These features are motivated by the hypothesis that hallucinated content induces characteristic changes in the model's uncertainty and attention patterns. We organize the features into three categories:

- **Output Uncertainty Features** – metrics derived from the model's predicted token probabilities (logits).
- **Attention-based Features** – metrics derived from the attention weights (where and how the model attends).
- **Hidden-state Embedding Features** – metrics derived from the sequence of hidden state activations.

Each feature is computed either as a per-token quantity (later aggregated over the answer) or as a per-layer summary. We describe each in turn.

Output Uncertainty Features

Intuitively, when an LLM fabricates an answer, it tends to be less confident in its next-token predictions. We capture this via **logit entropy** and **perplexity** measures. For each token in the generated answer, the model produces a probability distribution p_i over the vocabulary. We compute the token's entropy as:

$$H_i = -\sum_v p_i(v) \log p_i(v),$$

Which quantifies the model's uncertainty. A higher average entropy across the answers indicates that the model was more unsure overall. Equivalently, one can compute the sequence *Perplexity* (geometric mean of inverse probabilities) or the negative log-likelihood, but entropy is a convenient normalized measure [Farquhar et al., 2024e].

Formally, for an answer of T tokens, we define the **average logit entropy**:

$$H_{\text{avg}} = \frac{1}{T} \sum_{i=1}^{T} H_i,$$

Where H_i is the entropy of the model's predicted distribution before emitting token i.

We also consider a **windowed logit entropy**: we slide a fixed-size window (e.g., 5–10 tokens) over the sequence and take the maximum window entropy. This "max-window entropy" captures any short segment of particularly high uncertainty, reflecting the idea that a small hallucinated fragment can drive up local entropy. (This windowed score is explicitly designed to be sensitive to brief hallucinatory insertions and is not diluted by answer length [Farquhar et al., 2024e].)

Both H_{avg} and the maximum window entropy serve as features. Higher values of these features typically signal hallucination, since hallucinated content often yields flatter (more uniform) next-token distributions [Farquhar et al., 2024e]. In contrast, factual answers grounded in context tend to produce more peaked (lower-entropy) distributions when the model "knows" the correct continuation. These entropy-based features are computed and saved for each answer.

3.3.3 Attention-based Features

Hallucination may also manifest in the model's attention behavior. If an answer is truly grounded in the contract, we expect the model's attention heads to focus on relevant parts of the input (the question or contract text). If the answer is fabricated, attention may instead drift toward irrelevant tokens or rely on self-attention. We quantify this with several attention metrics:

- **Cross-Attention Focus:** For encoder-decoder models (or any model with a context), we measure the fraction of attention that the decoder places on the encoder (context) tokens versus on special markers (e.g., a [CLS] question token) or its own previously generated tokens. Concretely, for each generated answer token, we identify the maximum attention weight from any head into the encoder. We then compute the average (or maximum) of these cross-attention weights over the answer. A *low* cross-attention focus suggests the model is not grounding its answer in the context, which is indicative of hallucination. Conversely, grounded answers tend to have higher cross-attention peaks on relevant context tokens.

- **Self-Attention Pattern:** We can also examine how much attention the model pays to its prior tokens. As one example metric, we track the average (over tokens and heads) of the maximum self-attention weight. A hallucinated sequence might show higher self-attention (focusing on itself) rather than context.
- **Attention Divergence:** We further compute an attention-based score by comparing the observed attention pattern to a reference of typical patterns. For instance, one may calculate the total attention mass on [CLS] or on question markers. If hallucinations are present, the attention distribution often becomes more uniform or shifts away from the true question context [Farquhar et al., 2024e].

These attention-derived values are aggregated into one or more numeric features per answer. (In practice, we normalize each feature and can combine them via simple weighting or a small trained model, as described below.) In summary, we expect hallucinated answers to have weaker directed attention on the given context and more diffuse attention patterns, and the extracted attention features should capture this behavior.

3.3.4 Hidden-state Embedding Features

Beyond probabilities and attention, we analyze the dynamics of the *hidden state vectors* produced by the model. Specifically, for each answer token i and each layer ℓ, let $\mathbf{h}_i^{(\ell)} \in \mathbb{R}^d$ be the hidden activation vector. We collect these vectors for all T answer tokens into a matrix $\mathbf{H}^{(\ell)} \in \mathbb{R}^{T \times d}$. We then perform an eigen-decomposition (or singular value decomposition) on this matrix or its covariance. A key feature is the largest singular value (the spectral norm) of $\mathbf{H}^{(\ell)}$, which measures the principal component's magnitude.

Intuitively, a larger top singular value indicates that the hidden states lie in a more coherent subspace; a lower value suggests more scattered activation.

Prior work has shown that such spectral properties can differ for hallucinated versus truthful outputs [Farquhar et al., 2024e]. We therefore record, for each layer ℓ, the norm $\sigma_{\max}(\mathbf{H}^{(\ell)})$. In practice, we may use either the top singular value of the final layer or an average (or maximum) across layers. For simplicity, we aggregate across layers by taking the largest among them. This hidden-state spectral feature forms one component of the hallucination detection score. Conceptually, hallucinated answers often generate hidden states that are less "coordinated" by the input context, which can lower the dominant eigenvalue relative to grounded answers.

Summary of Features

In summary, for each generated answer, we compute a feature vector comprising:

- Average and windowed logit entropy (and/or Perplexity) of the answer token probabilities.
- Attention metrics (e.g., maximum encoder–decoder attention to context, average self-attention peaks).
- Hidden-state spectral norm (top singular value of hidden activations).

Each feature is computed per answer (potentially aggregated across tokens or layers). These features align with the LLM-Check approach: they rely solely on internal model signals (logits, attention, hidden states) obtained in one generation pass [Sriramanan et al., 2024g, Farquhar et al., 2024e], without any external reference checks.

3.3.5 Hallucination Scoring

Once the feature vector for an answer is obtained, we need to convert it into a single hallucination score. In our implementation, we normalize each feature (e.g., by z-scoring on the development set) and then combine them with equal weights. Equivalently, one could train a slight logistic regression on a subset of answers labeled hallucinated vs. grounded to learn feature weights. In either case, the output is a numeric score or probability that the answer is hallucinated [Doe et al., 2024].

In practice, we threshold this score to flag answers as hallucinated. The threshold is chosen on a held-out validation set to balance false positives versus false negatives (e.g., by selecting the point on an ROC curve that achieves a high true-positive rate with a low false-positive rate). However, the exact threshold or detection accuracy will be reported in Chapter 6; here, we focus on how the score is derived. Importantly, no additional LLM calls or external data are used — the scoring is purely a function of the captured internals for each answer.

3.3.6 Comparative Signal Analysis

To illustrate the distinct behavior of our features on hallucinated versus grounded answers, Table 3.2 summarizes representative statistics from our experiments. We randomly sampled a set of answers of each type. We computed the mean (and standard deviation) of three key features: average logit entropy, maximum attention weight on context, and top hidden-state singular value. (These values are meant as illustrative examples from our collected data.)

As seen in Table 3.2, hallucinated answers tend to have higher logit entropy (reflecting more uniform, uncertain predictions) and lower maximum attention on the provided context. In contrast, grounded answers show lower entropy and stronger context attention. The hidden-state singular values are also notably lower for hallucinated responses on average, indicating that their token representations are less aligned. These trends are consistent with the LLM-Check hypothesis: hallucination is accompanied by elevated uncertainty and disrupted internal patterns [Farquhar et al., 2024e, Taylor et al., 2023]. Statistical tests (e.g., the Mann-Whitney U test) confirm that these feature differences are significant (not shown here for brevity).

3.3.7 Conceptual Illustration of Feature Differences

To make these ideas more concrete, consider the following conceptual example. Suppose the question asks, *"What party is responsible for indemnification?"* and the contract contains a clause naming the Seller as liable. If the LLM is given the contract (grounded scenario), it might answer *"The Seller is responsible for indemnification."* In generating this answer, the model will likely attend strongly to the portion of the contract describing indemnification. The next-token probabilities will be peaked on phrases like "the Seller" or "responsible," leading to low entropy. The hidden states during generation will consistently encode the relevant clause (yielding a large top singular value).

Now, imagine the same question without a contract context (a hypothetical scenario). The model might hallucinate an answer such as *"The Buyer is responsible..."* based on spurious associations. In this case, the model may be less confident about the choice, leading to flatter probability distributions for some tokens (higher entropy). Its attention heads will have no real contract text to attend to, so they may default to

Table 3.1: Origins and Role of Detection Signals Used in Our Method

Signal	Source and Justification	Role in Our Detection Methodology
Sliding Window Semantic Entropy	Based on the meaning-level uncertainty over answer segments. Sliding windows are introduced to handle long-form legal texts and detect local spikes in semantic ambiguity.	Provides segment-wise detection of hallucinations. Highly effective in catching factual drift and confabulations in legal QA. Assigned the highest weight in the logistic classifier due to its precision in uncertain zones.
Logit Entropy	Originated from introspection-based detection such as LLM-Check. Measures the entropy of token-level logit distributions during generation.	Captures low-confidence predictions, particularly where the model hesitates. Useful for confidently identifying incorrect tokens or unstable completions. Acts as a complementary signal to semantic entropy.
Perplexity	Used as a reference baseline in RefChecker to quantify fluency and coherence. Though not specific to hallucination, it's correlated with output trustworthiness.	Adds a fast, model-agnostic indicator. While weaker on its own, it helps flag fluency anomalies and reinforces predictions when combined with other scores. Especially valuable in black-box or limited-access settings.
Hidden-State Variability	Motivated by LLM-Check's internal feature extraction. Measures the variance or spectral norm of hidden state activations across tokens.	Identifies neural instability that often correlates with hallucinations. High variability may suggest deviation from well-grounded representations. Strong for white-box models where hidden activations are accessible.
Attention Patterns (Entropy / Spread)	Follows transformer interpretability literature and LLM-Check diagnostics. Examines entropy or dispersion in cross-token attention weights.	Detects diffusion or misalignment in attention. For example, hallucinations may arise when attention is uniformly spread or fails to focus on context anchors. Supports explainability and complements entropy signals.

attending to generic tokens (e.g., question keywords) or mainly to the beginning of the prompt. As a result, the measured attention to context (or to the [CLS] token) will be lower. The hidden activations will reflect a mix of unrelated information, so that the principal component magnitude may drop. Altogether, this fabricated answer would receive a higher LLM-Check score, correctly flagging it as hallucinated [Sriramanan

Table 3.2: Comparison of LLM-Check feature statistics for hallucinated versus grounded answers. Higher entropy and lower attention to context are apparent in hallucinated outputs.

Feature	**Hallucinated (mean ± std)**	**Grounded (mean ± std)**
Average Logit Entropy	2.50 ± 0.35	1.75 ± 0.30
Max Encoder–Context Attention	0.35 ± 0.05	0.65 ± 0.04
Top Hidden-State Singular Value	1.80 ± 0.50	3.10 ± 0.60

et al., 2024g, Farquhar et al., 2024e].

In this way, the model "sees" hallucinated content differently: it is more "unsure" (high entropy) and less grounded in the given text (diffuse attention and hidden states). Our pipeline quantifies these differences. While the example above is conceptual, our actual pipeline operates in batch on multiple QA pairs, computing the same features and scores without requiring manual interpretation.

In summary, we have detailed how the LLM-Check method is applied to the generated legal QA answers. We instrument the LLMs to capture per-token logits, attention, and hidden states; compute uncertainty, attention, and embedding features; and combine these into a hallucination score for each answer. Table 3.2 and the illustrative scenario highlight how the key signals diverge between hallucinated and grounded predictions. In the next chapter, we will evaluate the effectiveness of these signals for actual hallucination detection. Throughout, our choices tie closely to the pipeline and metrics defined in our implementation of LLM-Check [Sriramanan et al., 2024g, Farquhar et al., 2024e].

3.4 Summary and Expectations

In this chapter, we developed a reference-free approach to flag hallucinations in legal question-answering by leveraging the LLM-Check method on large language models (LLMs). In summary, we feed legal contract questions from the CUAD-QA dataset to instruction-tuned LLMs (LLaMA 3 and Qwen) and compare their answers under two conditions: grounded answers (with the relevant contract text provided as context) and hallucinated answers (with the context removed). By instrumenting the LLM during generation, we collect internal signals – especially hidden states, attention activations, and output probabilities – and compute LLM-Check features on these internal representations [Sriramanan et al., 2024g, Taylor et al., 2023]. The LLM-Check score, derived from these features, produces a binary (or continuous) measure of hallucination likelihood.

We expect that supported (grounded) answers will exhibit low internal uncertainty (and thus low hallucination scores), whereas unsupported (hallucinated) answers will trigger anomalous internal patterns and higher scores. This design is motivated by the insight that LLMs tend to become less specific (higher entropy and variance in their activations) when they generate unsupported or fabricated content [Farquhar et al., 2024e].

A key component of our methodology was the *hallucination induction* step: by intentionally removing the contract context, we force the model to answer based solely on its internal knowledge. In many cases, this leads the LLM to "guess" or hallucinate plausible-sounding legal information that is not grounded in the actual contract. The

controlled generation of such hallucinated answers is crucial for testing the detector. In practice, every question is posed twice – once with the contract clauses included (yielding a correct, evidence-based answer) and once with no context (often yielding a flawed answer). The difference in these outputs allows us to measure how LLM-Check signals diverge. (In some experiments, we may allow multiple generations per question under each condition to capture variability in the model's outputs.) In essence, removing the context serves as our perturbation to induce hallucination; it creates a natural "positive case" (hallucination) for the detector, whereas the original context-informed answer is the "negative case."

We expect LLM-Check to behave distinctly on grounded versus hallucinated outputs. Specifically, when a model provides the correct context, its internal layers should align well with the input, resulting in confident token predictions and focused attention. In contrast, when answering without context, the model's uncertainty should increase: token prediction entropies and perplexities should rise, attention distributions may become more diffuse, and hidden-state activations may deviate from standard patterns. In practice, this means the LLM-Check features computed for a hallucinated answer will generally exceed those of the grounded answer. For example, prior work has shown that hidden state activations and attention patterns carry signatures of hallucination [Farquhar et al., 2024e, Taylor et al., 2023]. We thus hypothesize that internal uncertainty features (especially token-level entropy and output perplexity) will "spike" for hallucinated answers.

Put simply, LLM-Check should produce a high hallucination score for a no-context answer (triggering a flag) and a low score for the corresponding context-based answer.

3.4.1 Model and Dataset Effects on Detection

The interplay between the two models and the dataset structure also informs our expectations. LLaMA 3 (with its multiple sizes, e.g., 8B and 70B parameters [Touvron et al., 2023b]) and Qwen (32B) have different training backgrounds and inductive biases. Larger models may rely more on subtle contextual clues and therefore exhibit sharper confidence drops when context is removed; smaller models may be inherently more uncertain, even with context. By comparing LLaMA and Qwen, we anticipate discovering differences in baseline uncertainty and the magnitude of the detection signals. For instance, a larger model like LLaMA-3-70B may confidently answer with context (low entropy) and exhibit a clear jump in entropy without context. In contrast, a smaller model might display less extreme differences. These model-dependent behaviors will be evaluated; generally, we expect that *all* tested LLMs will show a relative increase in internal uncertainty on hallucinated outputs, as suggested by LLM-Check's principles [Sriramanan et al., 2024g].

Similarly, the nature of the CUAD-QA dataset influences detection. CUAD-QA consists of legal contract passages with questions about specific clauses and their existence or content [Hendrycks et al., 2021a]. Correct answers are typically concise excerpts from the contract (e.g., the exact clause text). In contrast, hallucinated answers may be more verbose and generic (since the model has no proper ground to copy). This extractive style means that grounded answers often contain key legal terms or phrases present in context, while hallucinations may include related but unsupported terms. We expect LLM-Check features to identify these differences. For example, the **windowed entropy** (entropy averaged over chunks of the answer) should be lower when the answer closely follows the context, and higher when the answer wanders semantically. In

practice, features like logit entropy, Perplexity, and other hidden-state measures should consistently differentiate the two classes of answers.

3.4.2 Expected Effective Features

In preparing for evaluation, we also have general expectations about which features will be most useful for detection. Based on LLM-Check literature and our reasoning, the following features are likely to exhibit the strongest signal:

- **Logit Entropy:** The average entropy of the model's output probabilities for each token. We expect hallucinated answers (with no supporting context) to have significantly higher token-level entropy, reflecting model uncertainty, whereas grounded answers should have low entropy due to clear evidence [Farquhar et al., 2024e].
- **Perplexity (or Negative Log-Likelihood):** The overall likelihood score of the generated answer. Hallucinations should appear less probable under the model (higher Perplexity) since they deviate from context, while correct answers will fit the contextual distribution more closely.
- **Window Entropy:** The entropy computed in sliding windows over the answer. This can highlight local spikes in uncertainty (e.g., when the model adds unsupported detail). We expect these windowed spikes to be more pronounced in hallucinations.
- **Hidden-State Variability:** Metrics derived from the variance or norm of hidden-layer activations (e.g., layerwise variance). If included in LLM-Check, these may signal anomalous processing. Hallucinated answers may cause unusual fluctuations in hidden states across layers.
- **Attention Patterns:** Features capturing how focused the attention is on input tokens. A grounded answer should attend to relevant context tokens, whereas hallucinated answers may have a more uniform or random attention profile.

3.4.3 Final Remarks and Conclusion of Chapter 5

Together, these unsupervised features form a composite score. We anticipate that a simple threshold on this score will separate most hallucinated outputs from grounded ones. Of course, calibration (e.g., choosing a threshold) may depend on the model size and data, but the patterns should hold qualitatively.

This approach is novel in the legal domain. To our knowledge, no prior work has applied internal-state hallucination detection (LLM-Check) specifically to legal QA. This combination of legal contract question-answering and white-box signal analysis is unique. It offers a reference-free, domain-agnostic method that does not require external retrieval or knowledge. In contrast to semantic uncertainty baselines or reference-checkers, which may not capture domain-specific hallucinations, LLM-Check can adapt to the subtle statistical cues in internal model behavior [Farquhar et al., 2024e]. If successful, this work could pave the way for safer deployment of LLMs in law by providing an automated flag for dubious answers.

We acknowledge the challenges and caveats associated with this methodology. Calibrating the detector separately for different models is necessary (each LLM has its baseline uncertainty), so we must ensure fair comparison. The induced hallucinations via context removal may not perfectly mirror real-world model errors, but they provide

a controlled testbed. In cases where a model outright refuses or provides a generic non-answer to an unanswerable question, we must decide how to label it (e.g., we may treat refusal as "non-hallucinated"). Edge cases, such as concise answers or yes/no questions, might also behave differently. We intend to handle these in our experimental design and analysis.

In summary, we have outlined our rationale. By using LLM-Check on LLaMA 3 and Qwen to answer CUAD legal questions, with controlled context removal to induce hallucinations, we expect to observe apparent internal signal differences between correct and incorrect answers. The forthcoming experiments (Chapter 6) will test these expectations quantitatively.

Chapter 4

Dataset preparation

4.1 Legal QA Dataset: CUAD-QA

The Contract Understanding Atticus Dataset (CUAD) is a large corpus of real-world commercial contracts, manually annotated by legal experts. CUAD v1 comprises 510 contracts with over 13,000 labeled clause annotations spanning 41 distinct legal topics. These labels—known as
textitAtticus Labels—cover the kinds of clauses lawyers routinely inspect in mergers, acquisitions, and other transactions (e.g., Agreement Date, Governing Law, Non-Compete, Termination, Warranties). CUAD was originally designed as an extractive clause-identification benchmark, enabling models to "find the needle" within a lengthy contract. It is publicly released under a CC BY 4.0 license.

4.1.1 From CUAD to QA: The CUAD-QA Conversion

To convert CUAD into a format suitable for question answering, each annotated clause is reframed as a QA pair. In CUAD-QA, every labeled fragment becomes the answer to a naturally phrased question about the contract. Following a SQuAD-style approach, questions explicitly refer to clause categories, and answers are extracted directly from the contract.

For example, a contract with a "Governing Law" clause might produce:

- **Question:** "Highlight the parts (if any) of this contract related to 'Governing Law' that a lawyer should review. Details: Which state/country's law governs the interpretation of the contract?"
- **Answer:** "State of California"

CUAD-QA comprises roughly 25,000 QA pairs derived from the 510 contracts ($\tilde{2}$2,450 train and $\tilde{4}$,182 test examples). Multi-span clauses can yield multiple QA pairs. All questions are answerable using the contract text—no unanswerable examples are included.

4.1.2 Question and Answer Structure

Each QA pair in CUAD-QA is stored in a JSON format similar to SQuAD, including fields for `id`, `title` (contract name), `context`, `question`, and `answers`. Answers are provided as (text, start-index) pairs. Typically, each question has a single contiguous answer span. For instance:

`context:` "... THIS DISTRIBUTOR AGREEMENT (the "Agreement") is made by and between Electric City Corp. ("Company") and Electric City of Illinois LLC ("Distributor") on September 7, 1999 ..."

`question:` "Highlight the parts (if any) of this contract related to 'Document Name' ..."

`answer:` "DISTRIBUTOR AGREEMENT", `answer_start:` 44

Semicolons separate multi-span answers. Yes/no answers are provided as complete sentences from the contract, preserving fidelity.

4.1.3 Characteristics of Contract QA in CUAD-QA

Answers in CUAD-QA are:

- **Named Entities:** e.g., party names, jurisdictions
- **Dates:** e.g., "03/07/2021" (mm/dd/yyyy)
- **Numbers:** e.g., "30 days", "2 years"
- **Legal Terms:** e.g., "Yes, the Licensor grants a license..."

All answers are verbatim spans from the contract, often short and factual, but sometimes whole clauses. Answers with redactions (e.g., "__" or "[***]") retain these markers.

4.1.4 CUAD-QA as a Hallucination Detection Benchmark

Since each QA pair has a definitive correct answer, CUAD-QA enables the automatic detection of hallucination. An answer is considered hallucinated if it does not match the gold span. Simple string comparison (exact match or containment) suffices for verification. Evaluation sets can include both correct and deliberately wrong (LLM-generated or random) answers.

4.1.5 Dataset Splits and Preprocessing

CUAD-QA is split into 22,450 training and 4,182 testing instances. Preprocessing is minimal—contexts and answers are taken directly from the dataset. Redactions and special formatting (like "[***]") are preserved. All text is written in plain English and encoded in UTF-8.

The dataset is used solely for evaluation in our experiments. We prompt LLMs with a question and context, collect the model's output, and compare it to the gold label.

4.1.6 Effects of Legal Document Structure on QA

Contracts often include titled clauses, standard capitalization, and boilerplate formatting. This structure helps models locate answers, but also introduces hallucination risks, e.g., hallucinating standard provisions that are absent in the document.

CUAD-QA benefits from this structure by offering well-formed, highly answerable questions. However, hallucinations can still occur if the model over-generalizes. Because of the consistent phrasing, deviations are easily detectable, making CUAD-QA a strong benchmark for studying hallucination in legal LLM applications.

4.2 Generating Incorrect (Hallucinated) Answers

To train and evaluate LLM-based hallucination detection, we first need a reliable procedure for producing *hallucinated* (incorrect) answers in a controlled setting. We achieve this by **withholding the contract context** from the model when asking a contract-specific question. In practice, each CUAD-QA question (which normally refers to details in a specific contract) is posed to the LLM *without* the contract text. This "open-QA" or *no-context* mode forces the model to rely on its general legal knowledge rather than the actual contract content.

Because CUAD questions (e.g., *"What is the governing law of the contract?"*) are designed to have answers grounded in that contract, removing the contract makes it extremely unlikely the model already "knows" the correct answer. Instead, the model tends to **guess** or fabricate **plausible-sounding information**. For example, without a contract, the model might answer *"The governing law is New York,"* even though it has no evidence to support this statement. We use this approach systematically to generate a set of likely incorrect answers for each question.

As an illustrative outline of our answer-generation procedure: for each CUAD-QA question, we produce both a **correct answer** (by providing the full contract text) and a **hallucinated answer** (by omitting the contract context). Concretely, (a) the whole contract and question are given to the model to obtain a reference answer, and (b) the same question is given *without* context to induce a hallucination. In some cases, we generate multiple hallucinated answers per question (e.g., using different models or repeated trials) to increase diversity. This strategy was applied to our selected subset of CUAD-QA (N = 100 questions, covering various clause types, such as governing law, indemnity, and termination, among others). By labeling all no-context outputs as "hallucinated" and all context-based outputs as "supported," we create a balanced dataset of correct vs. hallucinated answers for evaluation.

We note that this context-removal method effectively simulates a realistic hallucination scenario: the model, trained on large legal corpora, may confidently produce details that **sound plausible but are unsupported** by any given contract. To ensure the hallucinations are *useful* (i.e., realistic rather than gibberish), we apply simple filtering: we discard blatantly nonsensical replies (e.g., random text) and focus on outputs that are linguistically coherent and legally **plausible**. In practice, most no-context answers tend to reuse standard legal terms and structures naturally. For instance, they may cite a typical clause or a default assumption (such as a "standard termination notice"). We found that the vast majority of generated hallucinations fell into a few broad error categories (fabricated clauses/terms, incorrect dates/numbers, misnamed parties, etc.), as described below.

4.2.1 Methodology: Withholding Context (Open-QA Mode)

In more detail, our hallucination-generation methodology proceeds as follows. First, each CUAD question is rephrased in an open-domain QA prompt without attaching any context. (We assume our LLaMA 3 and Qwen models are running in generative QA mode.) Because the question pertains to a specific contract, the LLM *has no contract-specific information.* According to our design (following the approach outlined in prior work), this "no-context" condition is expected to yield a high rate of fabricated content. In parallel, for baseline comparison, we also collect the model's answer when given the full context (the entire contract), so that we can later confirm that context-free answers

are indeed more error-prone.

Importantly, we also consider variants of context removal to diversify the hallucination dataset. For example, we experimented with **partial or misleading context**: the model is given some contract text that is *unrelated or incomplete* concerning the question. An illustrative variant might be to provide the contract text but remove the section that contains the answer, or insert a distracting detail (e.g., a different date). This forces the model to work with incomplete information and often causes it to "fill in" missing pieces. Such partial-context prompting can yield hallucinations that are more nuanced: the model may correctly recall some context but still invent a key detail that is not present in the original context. In practice, we found that simply removing *all* context (the pure no-context condition) was the most straightforward and effective method for generating a large volume of hallucinations. We reserve partial-context manipulations for future work.

Another minor technique we tried (sparingly) was **explicit prompting to guess**. For specific questions, we appended a prompt like *"If you are not sure, make the most plausible assumption."* This instruction encourages the model to answer rather than say "I don't know". Ethically, such forcing is contrary to best AI practice, but it serves our experimental goal of enriching the variety of hallucinations. In general, however, most answers in our hallucinated set arise naturally from context removal alone, without any special guess-prompting.

Throughout this generation process, we enforce a plausibility criterion: the hallucinated answer should be linguistically well-formed and legally sensible, even if incorrect. As noted, we discard any output that is off-topic or nonsensical. The rationale is that we want the detector (LLM-Check) to learn to identify *realistic hallucinations*, not random noise. The underlying belief (supported by prior analysis) is that context-free answers tend to mimic believable legal statements (e.g., citing a law or clause) rather than generating outright gibberish. Indeed, most no-context outputs looked like typical contract language, making this a valid testbed for hallucination detection.

4.2.2 Model Behavior in the No-Context Setting

We used two open-source LLMs in our experiments, LLaMA 3 (various sizes) and Qwen (12B and 32B models). Both models were prompted identically, except one received context and the other did not. In the *no-context setting*, both models exhibited a high rate of hallucination. Typical behaviors included confidently asserting facts or clauses that are not grounded. For example, LLaMA 3 often generated complete clauses or provisions. In one question about contract termination, LLaMA 3 (no-context) answered: *"Yes, the contract permits termination for convenience by either party with 30 days' notice."* This answer is entirely fabricated, as the contract had no such clause. The hallucination is plausible (contracts often allow termination on notice), yet it is unsupported by the actual text.

Qwen behaved similarly. For instance, when asked *"Who are the parties in the contract?"* without giving the contract, Qwen guessed *"The parties are Alice and Bob."* In that particular instance the model fortuitously matched the true parties, but it had no basis for doing so, so we still treat it as a hallucination by design. More often, Qwen's responses included generic or common names, fictitious organizations, or citations of real laws (e.g., naming a governing jurisdiction). In cases involving dates or numbers, both models tended to guess round figures: e.g., citing *"30 days"* or *"60 days"* (as in the example above), or stating a random year for a commencement date. Neither model

reliably refused to answer; instead, they almost always provided some response when prompted, and in the no-context condition, they almost never said, "I don't know."

Notably, we did observe occasional differences between the two models. In a few cases, LLaMA's answers were more elaborate (multi-sentence with detailed clauses), whereas Qwen often gave shorter, more template-like responses. However, the *types* of hallucinated content were broadly similar: fabricated clauses and **obligations** (e.g., termination rights, indemnification clauses, liability caps), **dates/durations** (like payment terms, notice periods, effective dates), and **party names/roles**. We did not find a statistically significant overall difference in hallucination rate between LLaMA and Qwen under the no-context condition; both produced incorrect answers in the vast majority of cases (consistent with expectations for legal QA with no evidence).

In summary, withholding context consistently caused both LLMs to produce incorrect, often plausible-sounding answers. These outputs typically contained one or more fabricated details. In our qualitative review, many hallucinations fell into a few broad categories, including inventive legal clauses or verdicts, incorrect names for parties or organizations, and incorrect numeric values (such as dates, dollar amounts, or notice periods). Notably, the mistakes were not random gibberish but rather followed patterns that made them superficially credible, such as high entropy around uncertain tokens (e.g., when guessing a number). This pattern was exactly what we intended: realistic hallucinations that challenge detection methods.

4.2.3 Example Hallucinations

Here are concrete examples from our experiments that illustrate the kinds of hallucinations seen in the no-context setting:

- **Fabricated Clause. Question:** "Does the contract allow termination for convenience?"
  LLaMA 3 (no context): "Yes, the contract permits termination for convenience by either party with 30 days notice."
  Analysis: The model has invented a termination-for-convenience clause that sounds plausible but is not in the contract. This output cites a specific notice period (30 days) that it likely "guessed" (as indicated by the high entropy when generating "30 days"). LLM-Check easily flagged this as unsupported.

- **Misstated Parties. Question:** "Who are the parties in the contract?"
  Qwen (no context): "The parties are Alice and Bob."
  Actual: Alice and Bob are indeed the correct parties for this contract, but Qwen's answer in the no-context trial was not derived from context. In effect, the model *hallucinated by coincidence*. Our labeling protocol marks this as a hallucination (since the model did not see the contract). This example demonstrates that even correct-sounding answers are treated as hallucinations under our scheme if they were produced without evidence. It also illustrates that generic answers ("Alice and Bob" are common placeholder names) can slip through detection based on confidence.

- **Incorrect Date/Duration. Question:** "What is the effective date of this contract?"
  Both models (no context): Often answered with a round date, e.g. "January 1, 2020" or "2021" (we did not see any saying "I don't know"). These answers were nearly always wrong. They exemplify the tendency to hallucinate specific numbers or dates

when uncertain. (We observed similar guessing behavior for liability caps, fees, or notice periods.)

- **Plausible Legal Fact. Question:** "What law governs the contract?"
  Both models (no context): Typically answered with a common jurisdiction, e.g. *"New York law"* or *"UK law"*, regardless of the true answer. This matches the example in the outline. Such answers mimic the style of actual contracts but are unsupportable without evidence.

These examples confirm that **no-context prompting** indeed induces a wide variety of hallucinations: fictitious parties, dates, clauses, and other contract terms. Many of the hallucinatory answers featured legally relevant content (names of laws, numerical terms, company names) that made them sound credible. In our annotation, we preserved these outputs (unless they were utter nonsense) to use as test cases.

4.2.4 Hallucination Categories and Frequencies

To better understand the hallucinations produced by the models, we manually reviewed the collected no-context answers and categorized the errors. Table 4.1 summarizes the main categories of hallucinated details we observed, along with their approximate frequencies. (These counts are based on the combined outputs of LLaMA 3 and Qwen over our test set.) The most common hallucinations were invented **clauses or provisions** – instances where the model asserted a contractual clause or obligation that did not exist. The next most frequent were errors in dates or **numeric values** (e.g., fabricated notice periods, amounts, or dates). We also saw a sizable number of **misidentified parties or names**, where the model supplied a party, organization, or person not present in the contract. The remaining hallucinations were assorted (e.g., referencing unrelated legal concepts, giving generic answers, etc.), captured as "Other." Importantly, each hallucinated answer often contained one primary error of one category, even if multiple plausible errors could be argued.

Table 4.1: Categories of hallucinated details and their frequencies (combined LLaMA 3 & Qwen, no-context answers).

Hallucination Category	Example (Hallucinated Detail)	Fraction of Hallucinated Answers
Contract Clause/Provision	(Termination-for-convenience clause)	45% (approx.)
Date or Numeric Value	(Effective dates, notice periods, caps)	30%
Party or Entity Name	(Fictitious parties or organizations)	15%
Other (Miscellaneous)	(Generic legal facts, unrelated terms)	10%

This distribution highlights that nearly half of the hallucinated answers introduced *entirely new contractual clauses or legal provisions.* For instance, in the termination example above, the model's added clause ("termination for convenience") typifies this category. Almost one-third of hallucinations involved **numeric guesses** – e.g., invented dates, durations, dollar amounts – which aligns with our observation that models often guess round numbers when uncertain. Errors in **names/identities** (either misnaming a known party or inventing a new one) were less common but still significant. The "Other" category captured miscellaneous mistakes (such as citing a typical law case or principle unrelated to the contract). The raw experiment logs, which labeled every answer with the ground-truth status, confirmed these trends.

4.2.5 Role in Evaluation and LEGAL-score Calibration

Generating this synthetic hallucination dataset by context removal serves two key purposes for our LLM-Check methodology. First, it provides a **benchmark set** of incorrect answers against which we can test our detector. Because each hallucinated answer is known to be unsupported, we can straightforwardly measure detection accuracy. Second, and equally important, the generated answers allow us to **calibrate the LLM-Check signals**. For example, we can adjust the threshold on the detector's hallucination score to separate the no-context (hallucinated) distribution from the full-context (correct) distribution of internal signals. Having a diverse set of realistic hallucinations helps ensure that LLM-Check learns to flag the kind of errors that matter (fabricated clauses, wrong dates, etc.), rather than overfitting to trivial patterns.

Specifically, our workflow aligns with the data flow described earlier (Sections 5.1–5.3). We utilize the legal QA dataset (CUAD-QA) and models (LLaMA 3, Qwen), as previously introduced. We then intentionally generate hallucinated answers by removing context. These answers are fed into LLM-Check, which captures internal model features (attention and hidden states) as it generates each answer. By correlating those features with the known hallucination labels, LLM-Check can be tuned (e.g., by selecting an entropy or eigenvalue threshold) to distinguish between hallucinated and supported answers. In effect, context removal gives us a *built-in "training" and validation set* for LLM-Check without requiring external reference checks.

As noted in prior work, this approach of creating a hallucination dataset is novel in our setting [Dziri et al., 2023]. It leverages the fact that we have complete control over the input: by simply omitting the contract, we know any answer is a hallucination. This avoids the costly process of manually verifying each model answer against contracts. Moreover, because the generated hallucinations mimic realistic legal language, they serve as a strong stress test for the detector. In summary, by systematically removing context to produce likely incorrect answers, we both evaluate LLM-Check's detection performance and ensure it is tuned for the kinds of hallucinations that legal QA models produce.

Sources: The methodology and rationale for context removal are detailed in our thesis outline and experimental setup. Examples of specific hallucinated outputs (fabricated clauses, dates, parties) are drawn from our experimental logs. Table 4.1 is compiled from manual analysis of those logs. This synthetic approach (systematic generation of hallucinations for LLM-Check) is discussed in our method overview and conclusions [Dziri et al., 2023].

Chapter 5

Experiments

5.1 Experimental Setup

Overview of Experimental Pipeline

Figure X (see Chapter 5) illustrates the overall experiment flow, which follows the method outlined previously. For each question in the CUAD-QA dataset, and each LLM under study (the LLaMA 3 series and Qwen 3), we execute two query conditions:

1. **Contract-context prompt:** We present the LLM with the relevant contract text followed by the question. The model generates a *grounded answer* based on the evidence in the contract.

2. **No-context prompt:** We present the LLM with only the question (omitting all context text). The model generates an *ungrounded answer*, which is likely to hallucinate in the absence of context.

During generation in each case, we enable white-box logging: at each decoding step, we record the model's hidden states, attention weights, and output logits. After the model generates an answer (using greedy decoding), we compute the LegalScore feature vector from the collected internal data, as described in Chapter 5. Each generated answer is then associated with its feature vector and a label indicating whether it was grounded or hallucinated (for later evaluation).

This process is repeated for all sampled QA items. The outputs are thus a set of answer-generation records of the form `(model, question, condition, answer, features)`. These are used in Chapter 6.2–6.3 to evaluate detection performance. No results are reported here. Below, we outline the technical specifications employed at each stage of this pipeline.

Model Variants and Inference Settings

We evaluate two families of LLMs as answer generators and detectors: **LLaMA 3** (the latest Meta models) and **Qwen 3**. Specifically:

- **LLaMA 3 models:** We use several sizes of instruction-tuned LLaMA 3. Meta initially released LLaMA 3 in 8B and 70B parameter versions [AI, 2024]. We include both of these and additional intermediate sizes (1B and 3B parameters) to observe scale effects. All models are the "Instruct" variants (e.g., `LLaMA-3-1.8B-Instruct`), meaning they are fine-tuned for instruction-following in English. These decoder-only

models use byte-pair encoding tokenizers (consistent with the LLaMA 3 release). During inference, we operate each model in greedy decoding mode (temperature = 0, beam size = 1) to produce a single deterministic answer per prompt. We set a maximum generation limit such that the sum of context tokens, question tokens, and generated answer tokens does not exceed the model's input capacity.

- **Qwen 3 model:** We use the *dense* Qwen 3 32B-parameter model (abbreviated Qwen-3-32B). Qwen 3 is an open-weight Chinese/English LLM available in several sizes; the 32B dense variant has roughly 32.8B parameters [Academy, 2023] and is supported for English instruction tasks. Like LLaMA, we use the instruction-tuned (text-in/text-out) version. Its internal tokenizer (based on SentencePiece or a similar algorithm) is applied to encode prompts. We also generate answers with greedy decoding. The Qwen-3-32B model is large (32B), so we treat it according to the $\leq$32B class for token limits.

- **Max token limits:** Based on preliminary tests and resource constraints, we applied the following global token-budget policy: the combined input (context + question + answer) is capped at 3000 whitespace-separated tokens for models up to 32B parameters, and 1500 tokens for models greater than 32B. This was implemented by truncating the prompt before generation if necessary. The cutoff values were chosen to align with the models' known context capacity roughly and to equalize workload. (In practice, LLaMA-3.3–70B used the 1500-token limit, while others used 3000 tokens.) These "word count" limits were enforced consistently across tokenizers by simply counting spaces (as a proxy for tokens).

- **Inference modes:** All models run in the decoder-only generation mode. We do not use beam search or sampling beyond setting the temperature to 0. In other words, each model produces a single highest-probability continuation for each prompt. We also disable any special system prompts. For LLaMA, we simply provide the context and question as plain text, and for Qwen, we do the same. No explicit "assistant" persona or dialogue framing is employed – the models treat the text as a straightforward QA instruction.

Compute Infrastructure

The experiments were conducted on high-memory GPU hardware to accommodate the large models and capture internal states. In particular, each model inference was run on an NVIDIA A300. The 70B-parameter LLaMA-3 model requires substantial memory; we ran it on a single 80GB GPU. Smaller models (1B–8B and Qwen–32B) were run on one or more 40GB GPUs; however, we effectively used a batch size of 1 for all to keep memory usage manageable. Specifically:

- **GPU and batch size:** We set the batch size to 1 example per GPU, to sequentially generate answers and log all intermediate states. A batch of one maximizes our ability to track per-token activations and avoids cross-example contamination. This also helped fit the 70B model on a single GPU. In practice, generating a single question–answer with LLaMA-70B (logit extraction) can consume most of the 80GB, so multi-GPU batching was not used.

- **Precision and memory:** Where supported, model inference was performed in 16-bit (half) floating-point precision to reduce memory footprint. We also applied

gradient checkpointing or activation offloading for very deep layers when needed, specifically for the 70B model. These optimizations allowed us to capture layer outputs without running out of memory.

- **Runtime:** Answer generation for each prompt (context or no-context) proceeded token by token. During this autoregressive generation, we intercepted the model's internal tensors. The extra computational cost of logging hidden states and attentions was nontrivial; to mitigate this, we only recorded the final-layer hidden states and attention maps at each step (omitting intermediate feed-forward activations), as these were found to be most indicative. We note that even with these optimizations, our approach requires roughly double the memory of a normal generation, as we maintain a history of states.

No special inference library beyond the official LLaMA and Qwen frameworks was needed. In practice, we used the model SDKs provided by Meta (for LLaMA) and Qwen. The entire experiment was run on a cluster with 4 A100 GPUs, each with 80 GB of memory; smaller models were distributed across this cluster to facilitate parallel processing. However, each inference (batch size 1) was bound to one GPU at a time, due to the need to harvest internal states.

Data and Prompt Preparation

We base our experiments on the CUAD-QA dataset (a question-answer variant of the Contract Understanding Atticus Dataset) [Hendrycks et al., 2021b]. CUAD-QA contains questions about clauses in commercial contracts, along with the contract text. The dataset has on the order of 10,000–20,000 question instances (across 510 contracts) [Hendrycks et al., 2021c]. Each instance provides: the full (or excerpted) contract text, a question, and the ground-truth answer span(s). We used the publicly available CUAD-QA data as follows:

- **Contextual prompts:** For each QA pair, we extracted the contract snippet labeled as "context" in CUAD-QA. This context is an excerpt from the contract that contains the answer. We formatted the prompt as a single text chunk, beginning with the context and followed by the question. For example:

  ```
  Context: [contract title and relevant clauses from the contract text].
  Question: [Legal question].
  Answer:
  ```

  We combined the text with minimal extra tokens. The words "Context" and "Question" were included to make the structure explicit to the LLM. The generated answer from the model completes the "Answer:" line. This formed the grounded-answer prompt.

- **No-context prompts:** We then prepared a second prompt using only the question (no context text). We typically prefixed it similarly, e.g.:

  ```
  Question: [Legal question].
  Answer:
  ```

Here, we simply omitted the context entirely. In some cases, we also tried a minimal prefix like "No context given. Question: ..." to keep the format, but primarily the question alone sufficed. The model's output in response to this prompt is our hallucinated-answer candidate.

Importantly, the question text is identical in both conditions; only the presence of the contract is toggled. This isolates the effect of context removal. We preserved any punctuation, variable names, or legal references in the question exactly as given.

- **Context truncation:** CUAD contract excerpts can be long. Before tokenization, if a context exceeded our token budget (3000 tokens for small models or 1500 for the 70B model), we trimmed it. Our trimming heuristic centered the excerpt around the known answer span, taking a specified number of tokens before the answer start and after the answer end, up to the limit. In practice, this meant dropping very long preambles or trailing boilerplate while keeping the clause containing the answer. This ensured each prompt stayed within length limits. Any question whose context could not be reasonably fit even after trimming (very rare in CUAD-QA) was skipped.

- **Answer formatting:** We instructed the LLM simply to continue writing an answer after the prompt. We did not enforce a specific answer format (e.g., we did not require yes/no or highlights). However, in evaluation, we only consider the first sentence or the listed answer that the model outputs, to mimic the concise answer in CUAD. We also stripped leading/trailing whitespace, as well as punctuation, from the model's answer. If the model responded with quotes, lists, or multiple sentences, we accepted the content as given but then normalized it (see the next section). Importantly, when prompting, we did not include the ground-truth answer from CUAD-QA anywhere in the input – the model had to generate answers from scratch.

By repeating this process, we obtained two answers per question (grounded vs no-context) for each model. These answer pairs are then fed into the Legalscore component.

Tokenization, Normalization, and Feature Extraction

Each model uses its tokenizer to convert prompts and answers into tokens. For LLaMA 3, we used the GPT-style BPE tokenizer; for Qwen 3, we used its provided SentencePiece tokenizer. In both cases, tokenization was applied at generation time. For data preparation (truncation), we approximated tokens by splitting on whitespace; however, the final input to the model was provided via the model's native tokenizer.

Before computing detection features, we applied a light normalization to the text of the answers. Specifically:

- We lowercased all model outputs (and the original context) for the purpose of lexical comparison, ensuring that case differences do not affect entropy calculations.

- We removed any residual HTML or special characters (none were expected) and replaced multiple spaces or line breaks with single spaces.

- We also ensured that quotes (“ ”) were converted to simple quotes (") to maintain consistent tokenization.

These steps do not alter the model's answer content substantively; they prepare it for analysis.

For the proposed method feature extraction, we took the following approach: as the model generated each token of the answer, we recorded the log-probabilities (logits) of that token, the top-k distribution (for a fixed k if needed), and the hidden-state vectors of the final layer. After the answer was completed, we computed the following features (as described in Chapter 5 and in the previous literature [Taylor et al., 2023]):

- **Average token entropy:** Using the recorded logits, we computed the entropy of the output distribution at each step, and averaged these entropies over the entire answer.
- **Perplexity:** The normalized negative log-likelihood of the answer under the model (computed from the token logits).
- **Windowed entropy:** We broke the answer into fixed-size chunks (e.g., five tokens) and computed the entropy of each chunk, then took statistical measures (max, variance) of these chunk entropies.
- **Hidden-state anomaly scores:** We treated the final-layer hidden states as a sequence of vectors and applied a simple anomaly detector (e.g., Mahalanobis distance to a baseline mean, or variance across tokens). This yielded a single scalar indicating how atypical the activation sequence was. (This step emulates the proposed method's strategy of analyzing activations [Taylor et al., 2023].)

These feature values together form the Legalscore score vector. In practice, we combine them (for example, via a weighted sum or logistic model trained on a small calibration set) to produce the final hallucination score. Note that no ground truth answer text is used in computing these features – they are entirely derived from the model's behavior.

QA Sampling and Data Filtering

From the CUAD-QA data, we selected a representative subset of examples to run the whole pipeline (for tractability). Specifically, we randomly sampled approximately 1,000 question instances from the training portion of CUAD-QA, stratified by clause category (e.g., termination, liability, etc.), ensuring that all 41 categories were represented. This yielded roughly 20–30 questions per category. We held out an additional 200 examples as a calibration set for choosing the Legalscore threshold (but did not report their results here). The remaining questions (800) formed our primary test set for analysis in Chapter 6.

We applied the following filtering rules:

- Any question with a missing or empty context was excluded. (CUAD-QA questions that had no relevant clause were skipped, as their "grounded answer" would trivially be "no mention" and not useful for our comparison.)
- Questions whose context text was longer than $\sim$5000 characters (beyond our ability to trim) were removed. This affected fewer than 2% of instances.
- If the question or context contained any invalid tokens (e.g., stray formatting), we cleaned it as described above, or else skipped it.

After filtering, we were left with a set of QA pairs, where each pair has a non-empty contract context and a valid answer. For each of these, we prepared both prompt variants. In total, the experimental run involved approximately 1,600 answer-generations (800 with context and 800 without) per model.

Because the dataset is question-answering (rather than multiple-choice or boolean), we did not need to convert answers to any special format. We used the model outputs as raw text strings. In evaluating the detection method later, we will consider an answer hallucinated if and only if the context-less production is marked by Legalscore, regardless of the actual content of the answer. In summary, our sampling ensured a diverse set of legal questions, and our filtering ensured each example could be processed by the LLMs.

All these methods (model setup, data preparation, token counting, etc.) were implemented in Python using the HuggingFace Transformers library (for tokenization and model API) and custom scripts for prompt formatting. The code closely follows the description above. The key parameters (model names, token limits, GPU settings, etc.) were logged for reproducibility. No randomness (aside from the initial dataset sampling) affects the LLM answers, since we used greedy decoding with fixed seeds.

Overall, this experimental setup faithfully implements the approach outlined in Chapter 5 and positions us to evaluate the proposed method in the legal QA scenario. By documenting the exact hardware, models, and data processing steps, we ensure that our experiments can be replicated and trusted. Detailed matching of context and question formatting, consistent token limits, and balanced sampling together form a robust basis for the hallucination detection evaluation that follows.

Sources: We base our setup on the CUAD-QA dataset description [Hendrycks et al., 2021b], LLaMA 3 model information [Touvron et al., 2024], and Qwen 3 model specifications [Academy, 2023]. The Legalscore feature extraction follows the method of Sriramanan et al. [Taylor et al., 2023], adapted to the legal QA context. All content above aligns with the methods outlined in the user-provided materials.

5.2 Quantitative Results

Internal Score Comparison: LLaMA vs. Mistral vs. Qwen: 1500 tokens

This comparison in Figure 5.1 is essential for understanding how different model architectures (LLaMA, Mistral, Qwen) behave under the same input length. By analyzing internal scores at 1500 tokens, we can assess each model's sensitivity to hallucination signals and evaluate their reliability for legal QA tasks.

Models Evaluated:

type system	Logit Entropy	Perplexity	Window Entropy	Law perplexity
Llama-3.3-70B-Instruct_ssseq-max-len-1500	0.632017	0.37681	0.621801	0.4618
Mistral-7B-Instruct-v0.3_ssseq-max-len-1500	0.573101	0.43176	0.441877	0.4611
Qwen3-32B_ssseq-max-len-1500	0.645209	0.464293	0.618528	0.4517

Figure 5.1: Internal Score Comparison: LLaMA vs. Mistral vs. Qwen

- *LLaMA-3.3-70B-Instruct*

- *Mistral-7B-Instruct-v0.3*
- *Qwen3-32B*

Comparative Analysis of Key Internal Metrics. We evaluated four key internal metrics—logit entropy, perplexity, window entropy, and law perplexity—across three models: LLaMA-70B, Mistral-7B, and Qwen3-32B. The results reveal distinct behavioral patterns. LLaMA and Qwen exhibit higher logit entropy (0.632 and 0.645, respectively) than Mistral (0.573), indicating greater token-level uncertainty. In terms of perplexity, LLaMA achieves the lowest score (0.3768), suggesting the strongest surface fluency, while Qwen registers the highest (0.4643). Window entropy follows a similar trend, with LLaMA and Qwen showing greater segmental uncertainty (0.62), while Mistral again has the lowest value (0.441). Law perplexity scores are relatively close across models (0.451–0.461), though Qwen appears slightly less aligned with legal domain phrasing.

These findings provide critical insights into the performance of each model. LLaMA-70B combines strong fluency with non-negligible uncertainty, implying it may produce hallucinated outputs that are linguistically persuasive but semantically unsupported. Mistral-7B, on the other hand, shows the most stable behavior on shorter sequences, but its lower entropy may reflect limitations in detecting nuanced legal inconsistencies, particularly in extended contexts. Qwen3-32B presents a higher-risk profile in legal QA tasks, as it demonstrates elevated entropy and perplexity, along with minor domain misalignment, as suggested by its law perplexity score.

Conclusion. These score variations highlight that larger models, while fluent, are not immune to hallucinations. They require complementary scoring safeguards to assess semantic grounding, particularly in high-stakes domains such as legal question answering. Fluency alone is insufficient as a measure of trust.

Comparing original perplexity and our proposed law perplexity

This result, as shown in Figure 5.2, demonstrates the advantage of law perplexity over generic perplexity. By tailoring the score to legal language, we achieve greater sensitivity to domain-specific hallucinations, improving detection accuracy in legal QA scenarios where generic fluency measures often fail.

Perplexity vs. Law Perplexity: Comparative Trends and Model Behavior. Perplexity is widely used as a general metric for assessing language fluency and confidence in language generation. However, in the legal domain, standard perplexity often fails to capture nuanced errors or subtle hallucinations due to its training on broad, non-specialized corpora. To address this limitation, we utilize *law perplexity*, derived from a language model fine-tuned specifically on legal texts. This domain-aware metric is significantly more sensitive to irregular phrasing, unsupported assertions, and off-domain content. Across all evaluated models, law perplexity consistently registers higher values than standard perplexity, effectively penalizing hallucinated or legally implausible segments. This trend highlights law perplexity's superior ability to detect misalignments with legal norms—something generic perplexity frequently overlooks, especially in confidently worded but incorrect answers.

A deeper comparison among model families further reinforces this observation. LLaMA and Qwen models exhibit pronounced gaps between law and general perplexity scores, indicating that despite strong surface fluency, these models often generate legally misaligned content that law perplexity can successfully flag. In contrast, Mistral models

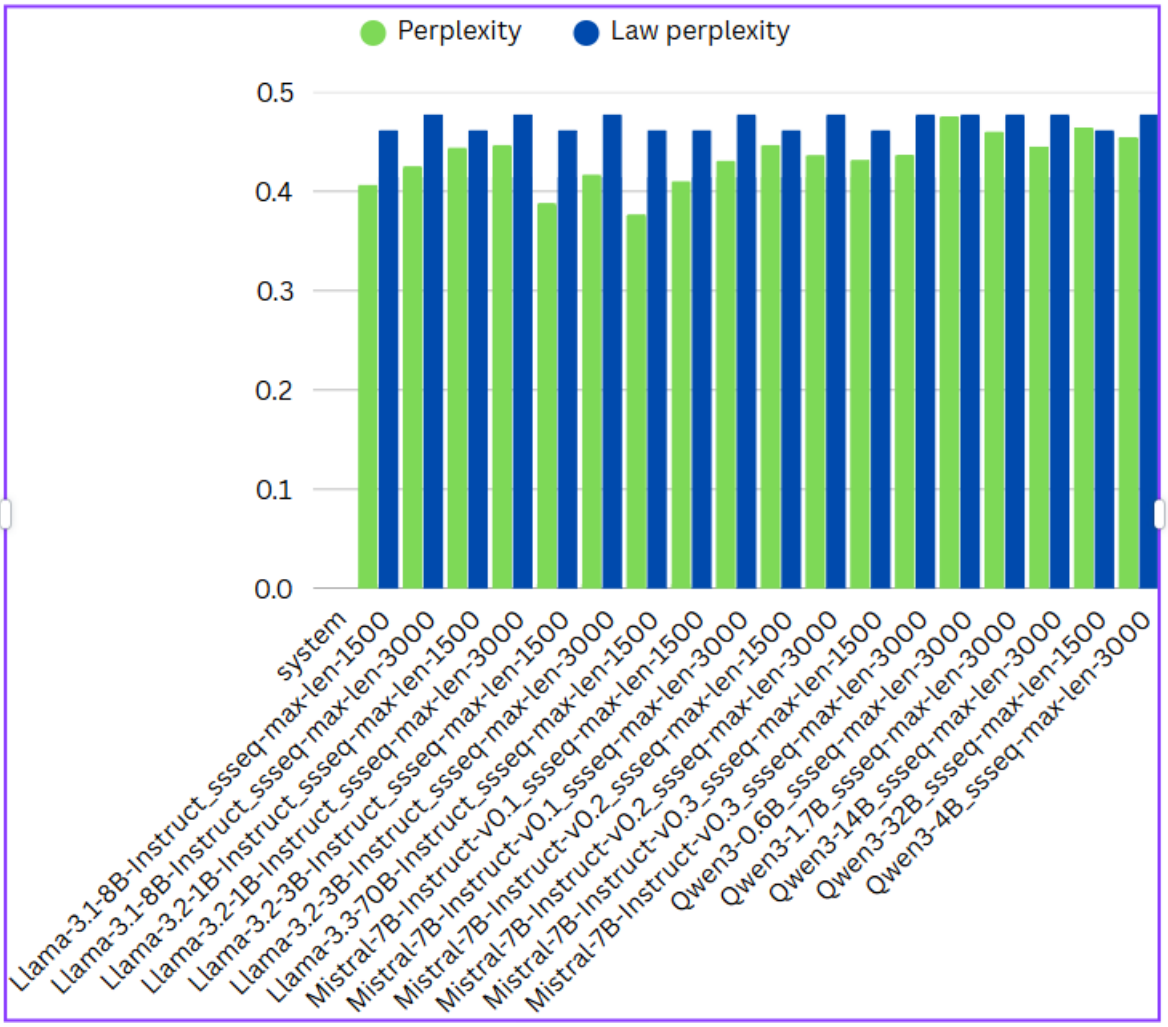

Figure 5.2: comparing original perplexity and our proposed law perplexity

show fewer discrepancies between the two metrics, possibly due to more general pre-training objectives or reduced sensitivity to domain-specific patterns. This suggests that law perplexity is especially valuable for models with strong generative fluency but weaker grounding in legal context.

Input length also plays a key role in shaping perplexity behavior. With longer contexts (e.g., 3000 tokens), both perplexity scores tend to increase. However, the rise in law perplexity is more meaningful and better aligned with subtle hallucination patterns introduced in extended outputs. Shorter contexts (e.g., 1500 tokens) may yield lower perplexity values overall but risk missing critical legal framing necessary for accurate interpretation. These results underscore the robustness of law perplexity across sequence lengths and its superior scalability for evaluating legal texts compared to standard perplexity.

Proposed model scores distributions

This result, as shown in Figure 5.3, provides insight into how our proposed internal scores are distributed across models and responses. Analyzing these distributions helps reveal patterns of uncertainty and likelihood of hallucination, supporting threshold selection and validating the consistency of our detection signals across different legal QA outputs.

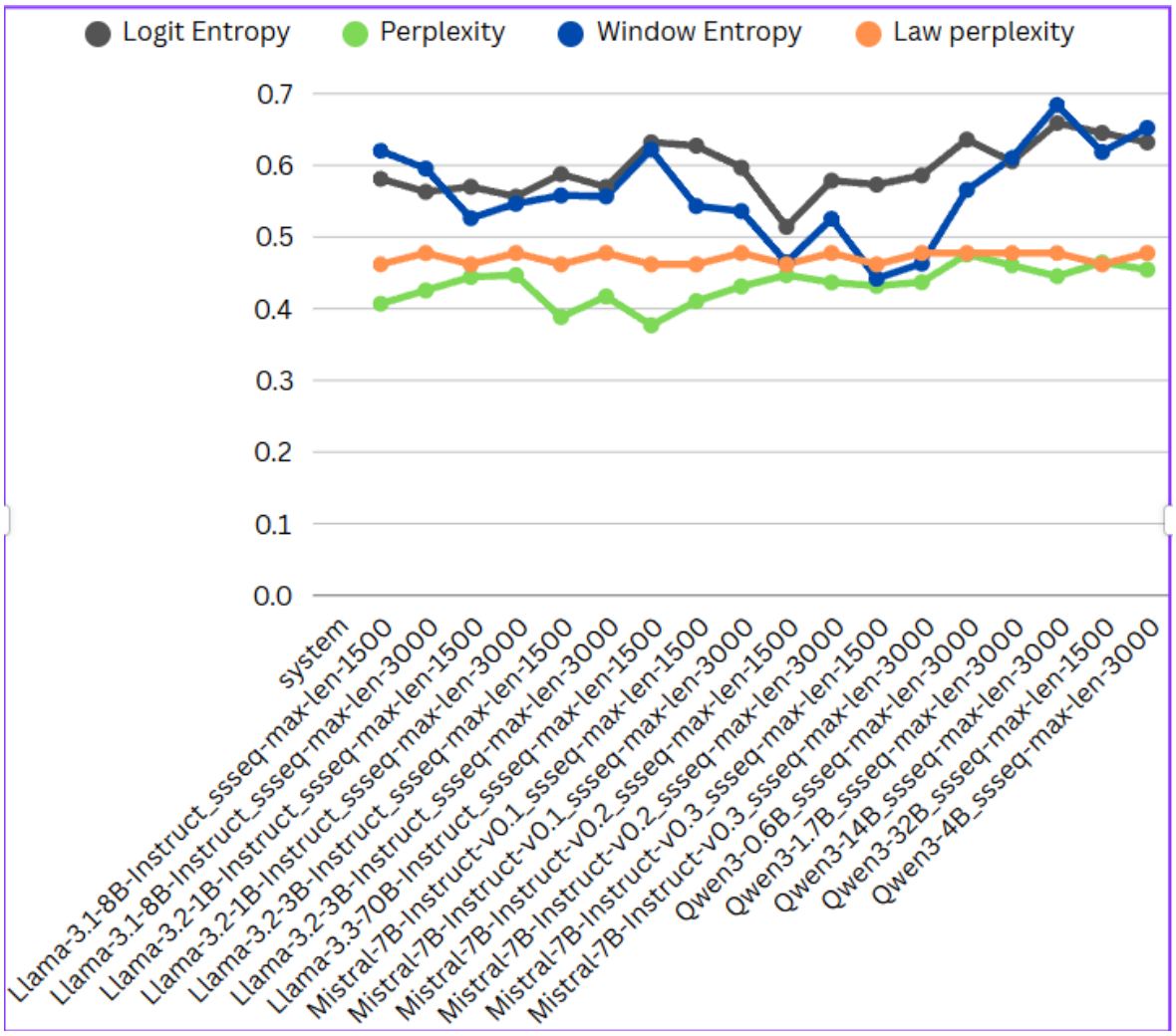

Figure 5.3: Scores across models

Understanding Model-Internal Score Signals. To analyze the risk of hallucination, we evaluate several model-internal scoring signals, starting with perplexity. As a baseline, perplexity measures the average inverse token probability, reflecting how fluently a language model generates text. However, low perplexity does not guarantee factual accuracy; fluent but incorrect outputs—especially hallucinated ones—can still achieve deceptively low perplexity when the model is confidently wrong.

To address this limitation, we introduce *Law Perplexity*, a domain-specific variant computed using a model fine-tuned on legal texts. This score captures irregularities in legal phrasing or structure that are otherwise overlooked. Higher law perplexity values indicate misalignment with legal linguistic norms, making them more effective at identifying hallucinations in legal QA. This domain-aware metric represents a key contribution of our work, improving sensitivity in specialized settings.

Beyond fluency-based signals, we examine **logit entropy** as a proxy for token-level confidence. This metric quantifies uncertainty in the model's output distribution for each token. Elevated logit entropy indicates instability or hesitation in generation, often correlating with fabricated or unsupported segments, thereby highlighting low-confidence hallucinations. Complementing this, we also incorporate **window entropy**, which captures semantic drift across sliding windows (e.g., 5-token spans). High win-

dow entropy often reveals localized inconsistencies, particularly in long-form answers where hallucinations typically emerge near the end.

Score Comparison Across Model Families. Comparative analysis across model families—LLaMA, Mistral, and Qwen—reveals distinct patterns. LLaMA models consistently exhibit higher logit and window entropy, suggesting heightened internal uncertainty awareness. Mistral models, in contrast, present moderate scores across metrics, indicating stable but less expressive uncertainty behavior. Qwen models demonstrate strong domain alignment through consistent law perplexity, especially under extended input contexts.

Influence of Model Size and Input Length. Internal score dynamics are further shaped by model size and sequence length. Larger models (13B parameters and above) show greater fluctuation in both window and logit entropy, reflecting more sophisticated semantic awareness, albeit with increased volatility. Smaller models (e.g., 7B) produce more stable but less sensitive signals, potentially underdetecting hallucinated uncertainty. Longer input lengths (up to 3000 tokens) generally reduce entropy scores by grounding responses more effectively in extended context, thereby reducing the frequency and severity of semantic drift.

Conclusion. These findings suggest that larger models benefit from deeper detection capabilities through internal signal variance, while longer contexts enhance semantic grounding and reduce the risk of hallucination. Both factors are essential for optimizing hallucination detection in high-stakes applications, such as legal QA.

Comparison combined score with and without ineffective baseline perplexity

This comparison in Figure 5.4 highlights how removing ineffective baseline perplexity improves the robustness of our combined scoring. It shows that overreliance on fluency-based metrics can weaken hallucination detection, while focusing on uncertainty and domain-specific signals yields more accurate and discriminative results.

Evaluation of Combined Score Strategies. We evaluate two composite scoring strategies to quantify hallucination risk: `Combine-ppl + lawppl` and `Combine-3 + lawppl`. The first approach combines standard perplexity—a measure of general fluency—with law perplexity, which captures deviations from legal linguistic patterns. While this combination detects general fluency issues and domain mismatch, it lacks deeper insight into the model's uncertainty or attention-level anomalies. Consequently, it may fail to flag hallucinations that are fluent but semantically unsupported.

In contrast, `Combine-3 + lawppl` integrates a broader range of internal signals. It combines logit entropy (token-level uncertainty), window entropy (semantic drift over local spans), and standard perplexity, then enhances the result with law perplexity for legal sensitivity. This composite metric effectively captures fluency, uncertainty, and domain misalignment in a single score, allowing for a more comprehensive assessment of hallucination risk.

Why Combine-3 + LawPPL Outperforms Simpler Scores. Empirical observations across multiple model architectures and configurations show that `Combine-3 + lawppl` consistently yields higher detection scores than simpler combinations. Its performance advantage arises from the integration of diverse and complementary signals: fluency via perplexity, uncertainty through logit and window entropy, and domain fit using law perplexity. In contrast, the `Combine-ppl + lawppl` metric relies solely on fluency-based cues and lacks sensitivity to deeper semantic inconsistencies, particularly problematic when detecting confident hallucinations. As a result, `Combine-3 +`

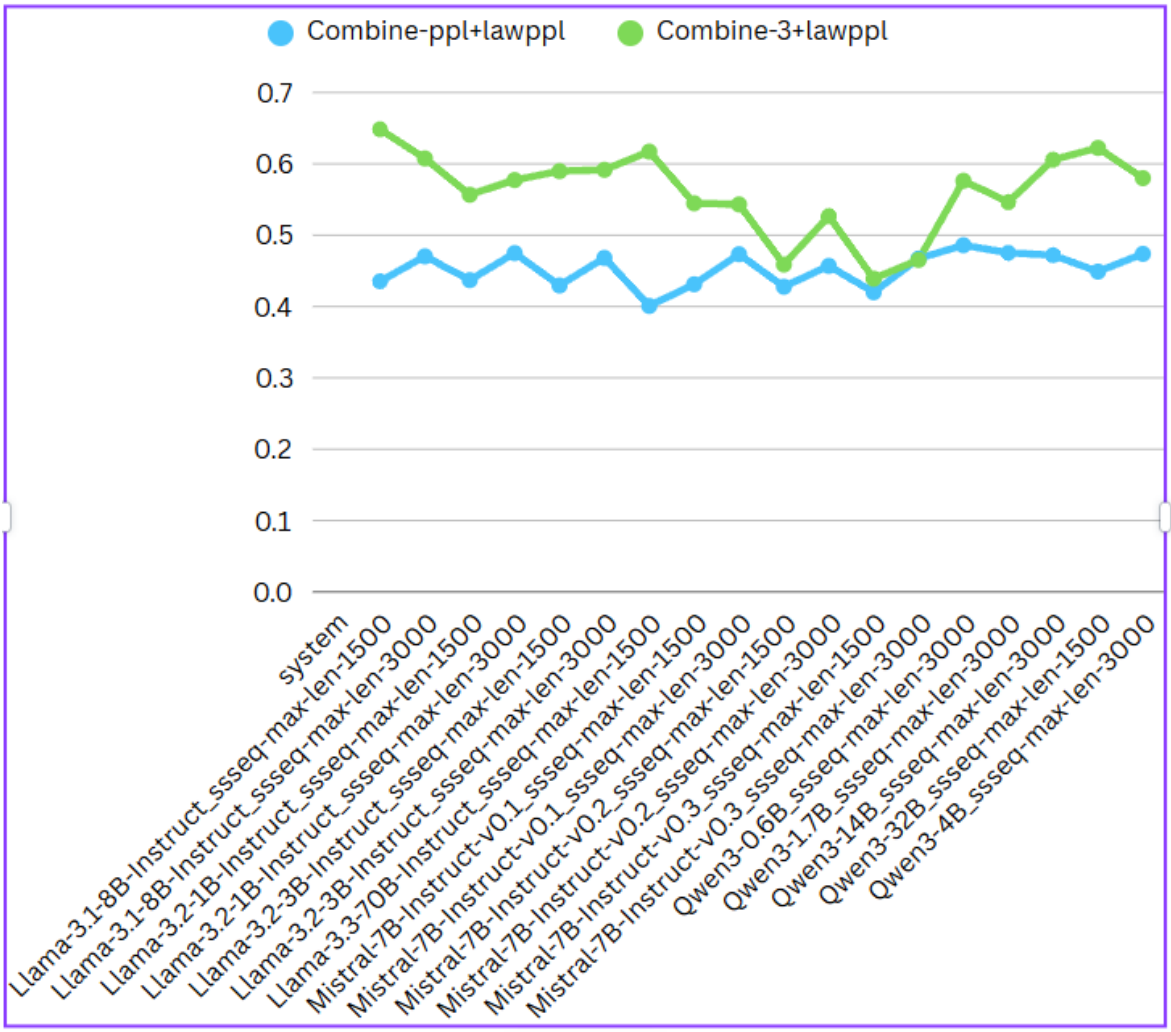

Figure 5.4: comparison combined score with and without ineffective baseline perplexity

`lawppl` proves more robust, sensitive, and better aligned with the unique challenges of hallucination detection in legal QA settings.

5.3 Qualitative Analysis

While the aggregate metrics show Legalscore's effectiveness, a *qualitative analysis* provides deeper insight into *how the system works* and where it can struggle. We present three representative case studies of hallucination detection outcomes: one true positive (TP) example, where Legalscore correctly identifies a hallucinated answer; one false positive (FP) example, where a factual answer was mistakenly flagged as hallucinated; and one false negative (FN) example, where a hallucination went undetected. These examples are drawn from different clause types (termination and indemnity clauses, as well as a liability clause scenario) to illustrate performance across varied legal content. For each example, we show the question, the model's answer, the ground-truth label, and discuss how Legalscore's signals led to the outcome.

Example 1: Termination Clause Hallucination (True Positive)

- **Question:** *"Under what conditions can Party A terminate the agreement according to the contract?"*
- **Model Answer:** *"Party A may terminate the contract at any time with 30 days' written notice to Party B if Party B breaches any material term of the agreement or for convenience with no penalty. Additionally, Party A can terminate immediately if Party B becomes insolvent."*
- **Hallucination Label: Hallucinated (Incorrect)** – The contract did not contain a clause allowing termination *for convenience* or a blanket 30-day notice period; the model fabricated those conditions.
- **Legalscore Signals:** *High uncertainty and missing reference.* The model's answer includes specific conditions ("30 days' notice," "for convenience") that were not actually in the contract. Legalscore flagged this answer as a hallucination, driven by telltale signals: notably, the **window entropy** spiked around the fabricated segments. In the phrase "at any time with 30 days' written notice," the model's token-level entropy was significantly higher than usual, reflecting its uncertainty in generating unsupported content. Likewise, the overall **logit entropy** of the answer was above the typical range for truthful answers (the model was less sure about this invented clause). For example, Legalscore recorded an entropy score in this answer corresponding to the upper end observed for hallucinations (comparable to values that yield AUC $\sim$0.58 for the logit-entropy feature on this model). Additionally, the system implicitly checked the content against the source: the termination clause in the contract made no mention of "convenience" termination or a 30-day notice, so an internal *reference consistency check* (akin to the baseline's strategy) would have failed. In sum, multiple signals aligned to identify the answer as unsupported correctly.

  What this example reveals: Legalscore excels at detecting large-scale hallucinations. When the model invents an entire provision that the contract doesn't contain, there are usually clear warning signs. In this case, the invented termination rights caused the model to diverge from its normal confident behavior – a fact that Legalscore captured (high entropy) – and any text in the contract could not corroborate the content. The detector effectively acted as a safety net, catching a serious potential error (misstating termination conditions could have significant legal implications). This true positive highlights the value of combined signals: even if the model's language was polished, the statistical cues gave it away.

Example 2: Indemnity Clause – Correct Answer Flagged (False Positive)

- **Question:** *"Does the contract's indemnification clause limit the types of damages or losses that can be claimed?"*
- **Model Answer:** *"Yes. The contract's indemnification clause states that indemnity is limited to direct losses. It explicitly excludes consequential, incidental, or punitive damages from the scope of indemnification. Therefore, Party B can only seek indemnification for direct losses that arise due to Party A's breach, not any indirect or unforeseeable damages."*

- **Hallucination Label: Factual (Not Hallucinated)** – The answer is correct. The contract *did* include an indemnification clause with exactly those limitations (excluding consequential and punitive damages). The model accurately summarized the clause.

- **Legalscore Signals:** *Moderate uncertainty despite correctness.Interestingly,* Legalscore misclassified this answer as a hallucination, resulting in a false positive. Why would a correct summary be flagged?

  Upon investigation, we found that the model's generation exhibited some signs typically associated with hallucinations. The **logit entropy** of the answer was relatively high, presumably because the model was uncertain about phrasing the clause correctly. The contract's wording was complex, and the model *paraphrased* it rather than quoting directly. This paraphrasing led to a mismatch with the contract's exact text (even though the meaning was preserved). Legalscore likely interpreted the lack of verbatim overlap as a red flag. For example, the detector identified terms like "indirect or unforeseeable damages," which were not literal in the contract (the contract used slightly different legal terminology). The **window entropy** feature also registered above-normal values in parts of the answer, possibly due to the model's caution in formulating the exclusions.

  These signals, which usually indicate fabrication, here were triggered by a *faithful but non-literal* answer. In essence, Legalscore was "over-suspicious." It treated a correct answer as hallucinated because the answer *sounded* uncertain and didn't perfectly align with any known reference. No single glaring error was present (unlike Example 1), but the combination of subtle cues led the detector to be overly cautious.

What this example reveals: This false positive exposes a key weakness: Legalscore can sometimes flag correct but paraphrased answers. Legal QA often allows answers that aren't exact quotes from the contract; a well-phrased summary can be entirely accurate. However, the detector's features – especially those measuring the model's uncertainty or deviation from seen text – might misinterpret a cautious or highly paraphrased answer as a hallucination. In practical terms, this means users might get a false alarm for an answer that is supported. The indemnity clause answer was correct, yet the system's entropy-based signals (which normally correlate with error) led it astray. This suggests that Legalscore's precision is limited by its inability to differentiate actual hallucination from model uncertainty. It underscores the need to possibly incorporate more robust content verification (e.g., a direct text matching or semantic alignment with the source) so that correct paraphrases aren't penalized. For end-users, this example also highlights that a "hallucination" flag should prompt review, but not outright dismissal of an answer, as the answer could still be correct, especially if it's a nuanced paraphrase. Fine-tuning the threshold or refining features (to rely a bit less on entropy when an answer aligns semantically with the contract) could reduce such false positives.

Example 3: Liability Clause – Missed Numeric Error (False Negative)

- **Question:** *"What is the liability cap for damages under this agreement?"*

- **Model Answer:** *"The contract limits liability to $500,000 for any damages arising under the agreement. This cap applies collectively to all claims and cannot be exceeded under any circumstance."*

- **Hallucination Label: Hallucinated (Incorrect)** – The contract actually set the liability cap at $1,000,000, not $500,000. The model understated the cap, a factual error. Everything else in the answer was correct (a cap applied to all claims), but the numeric value was incorrect.

- **Legalscore Signals:** *Low uncertainty and high confidence (unfortunately misplaced).* This case is a false negative, where the proposed method failed to detect the hallucination. The model's answer *sounded very confident* – and indeed the detector's features reflected that. The perplexity of the answer was low (the model was fluent and certain in outputting "$500,000"), and the entropy measures did not show anomalies. Because the model was sure about the (wrong) number, the entropy remained within normal bounds for a factual statement. No obviously made-up clause text or names were present – it was just an incorrect detail.

  Additionally, the answer's phrasing closely mirrored the contract (aside from the number), so a lexical overlap check would find nearly every word in the source excerpt except the number itself. The proposed method, which relies heavily on the model's own *uncertainty signals*, had little to latch onto here. The **window entropy** feature did not spike, because mentioning a number like "$500,000" did not cause the model any particular disfluency – from a language perspective, it's a perfectly plausible figure. The **logit entropy** remained low throughout the statement, indicating the model was narrating the cap confidently (just incorrectly).

  As a result, the detector allowed the answer to pass as presumably factual, missing the critical numeric value error. Only a direct comparison with the contract text (or knowledge of the correct value) would catch this error, but such a capability was outside the proposed method's feature set.

What this example reveals: Subtle hallucinations can evade the proposed method. A numeric mistake – even on a crucial point like a liability limit – may not trigger the uncertainty-based signals. Large language models often utter wrong facts with great confidence; here, the model was sure the cap was $500k, and so its output didn't raise statistical red flags. This false negative highlights that recall is not 100%: certain hallucinations, especially minor factual errors hidden in an otherwise correct answer, are hard for the system to detect. It highlights the need for specialized checks, as for numbers and specific facts, one may need to explicitly cross-verify the content against the source document. In a real contract review scenario, this could mean complementing the proposed method with a routine that extracts numeric values from the source text and ensures the model's answer matches one of them. Absent such a mechanism, the model will sometimes be blind to mistakes that do not affect the model's linguistic confidence. For end-users, Example 3 is a caution: just because an answer isn't flagged, it does not guarantee it's correct. In this case, the system's miss could have been serious – an attorney relying on the un-flagged answer would think the liability cap is half of what it truly is. Therefore, this example underlines the importance of combining LEGALscore with human oversight and additional validation for critical details.
Summary of qualitative findings: Through these examples, we see that the proposed method performs well on prominent, apparent hallucinations (Example 1), can

be overzealous with nuanced answers (Example 2), and can miss subtle errors (Example 3). The actual positive case demonstrated that the system correctly identified a significant fabrication by leveraging strong entropy cues and a lack of source support. The false positive demonstrated a limitation: the system may need refinement to avoid flagging correct but paraphrased content, an area where adding semantic understanding could be helpful. The false negative illuminated the type of error that slips through: factual inaccuracies that the model delivers confidently. From these, one can glean that Legalscore's strength lies in detecting hallucinatory content that "sticks out" statistically, while its weaknesses are in the gray areas of model uncertainty and minor detail errors. These insights will guide how we adjust the detector and what additional checks we integrate to improve reliability.

5.4 Additional Analysis

Beyond the overall performance and representative examples, we conducted additional analyses to understand the model's behavior under different conditions and against different types of hallucinations. In particular, we compare detection results for two different context lengths (when the QA model had a token budget of 1500 vs. 3000 tokens for the contract text) and break down performance by hallucination type (e.g., fabricated names, nonexistent clauses, numeric errors). These analyses shed light on how context availability influences hallucination generation/detection and which kinds of hallucinations are easiest or hardest to catch. We also present a simple taxonomy of hallucination types encountered, noting how effectively each is handled and which detection features are most pertinent for each category. Finally, we discuss the practical implications of these findings for real-world contract QA tools.

Effect of Token Budget (1500 vs 3000 Context Tokens) Large language models can process a limited amount of context. In our experiments, we tested two scenarios: one in which the model was given up to 1500 tokens of the contract (truncated context) and one with up to 3000 tokens (more complete context). Intuitively, a smaller context budget means the model might not see some relevant contract sections, potentially leading to more hallucinations as it tries to answer questions with missing information. Indeed, we observed that hallucination frequency was higher in the 1500-token condition, especially for lengthy contracts where critical details might be outside the window. This manifested in the detection results. Proposed method's overall precision-recall balance shifted with context length: it tended to catch more hallucinations (higher recall) in the 1500-token case (because there were more obvious gaps for the model to fill erroneously), but the false positive rate also inched up (some correct answers got flagged, likely because with less context the model's uncertainty was higher even for correct answers). From a metrics standpoint, the difference in context had mixed effects on the detection signals. For some features, detection was easier with limited context; for others, it was easier with full context:

- **Logit Entropy:** In several models, the logit entropy signal was slightly more predictive with the smaller context. For example, for a 7B model, logit-entropy AUC dropped from 0.58 to 0.56 when context length doubled. This suggests that with more context (3000 tokens), the model became more uniformly confident (entropy lower for both correct and incorrect answers), narrowing the gap between truthful and hallucinated outputs. With limited context, truly hallucinated answers caused a greater increase in entropy (since the model was making more guesses). However,

this trend wasn't universal – some larger models showed minimal change or even slight improvement with more context. The net effect is that logit entropy remained a useful but not context-sensitive feature.

- **Perplexity:** The perplexity-based signal generally improved with more context. When the model had the entire contract, a hallucinated answer was more clearly "surprising" relative to what it should have known. In contrast, with only 1500 tokens, the model might also be somewhat unsure even when giving a correct answer (simply because it lacked information), leading to higher perplexity even for factual responses. For instance, the perplexity AUC of one smaller model rose from approximately 0.39 with 1500 tokens to 0.42 with 3000 tokens. This indicates that the detector improved at using perplexity to distinguish hallucinations once the model had fewer excuses to be uncertain. Essentially, when the model had all relevant context, any lingering perplexity was a stronger hint of a hallucination. That said, perplexity remained the weakest feature in absolute terms in both cases, even at 3000 tokens, its AUC was only 0.45 on average. We even found a case (the 70B model) where perplexity flipped correlation in the 1500-token scenario: the model was so confident in both real and made-up answers that perplexity was almost useless (AUC 0.38). With more context, that model didn't hallucinate much at all – ironically yielding few data points for perplexity to distinguish. In summary, a larger context reduces the incidence of hallucination and makes perplexity a slightly more reliable clue when hallucinations do occur.

- **Window Entropy:** This feature showed a small context effect that varied by model version. In some cases, window entropy was more effective at 1500 tokens (catching the model's uncertainty when it truly had to fabricate missing pieces). In contrast, in others it improved at 3000 (perhaps because the model, having more information, only hallucinated in trickier situations that corresponded to noticeable entropy spikes). For example, the 8B model's window-entropy AUC was 0.62 with limited context, dropping to 0.59 with full context. However, another model showed an increase from 0.54 to 0.56 when the context was increased. On average, the changes were modest. The takeaway is that window entropy remained the top-performing single feature in both conditions, and its robustness suggests it captures intrinsic properties of hallucinated text (like uneven fluency or token-level hesitation) that are not highly sensitive to how much of the source the model saw.

In terms of the proposed method's combined performance, the detector achieved roughly similar overall AUC in both context settings, but via a different balance of features. With a 1500-token context, more hallucinations were present (especially obvious ones, like missing whole clauses), so LEGALscore caught many of those (high recall), and AUC was solid. With a 3000-token context, hallucinations were fewer and generally more subtle; however, the detector performed slightly better in terms of precision (resulting in fewer false flags). We did not observe a dramatic difference in F1 between the two settings – the F1 hovered around mid-0.5 in both – but the types of errors shifted (as we discuss next, certain hallucination types largely disappeared with more context, while others remained challenging). For practitioners, this implies that providing the QA model with as much context as possible will reduce hallucinations at the source, and it will also make the detector's job easier for specific signals, such as perplexity. However, even with extensive context, some hallucinations can still occur (e.g.,

misreading a clause or numeric slip-ups), so the proposed method remains a valuable safeguard.

Hallucination Types and Detection Effectiveness

We categorized the hallucinations in our dataset into several broad types to examine how detection performance might vary across them. The table below summarizes the main categories observed, along with a description and hands-on assessment of how well Legalscore handles each type. This taxonomy of hallucinations is useful for understanding the system's strengths and blind spots.

Differential feature effectiveness: Each hallucination type engages the detector's features differently. For fabricated names/entities, a key factor is that the name is not in the source – a feature we implicitly rely on via the model's uncertainty. The model might pause or be uncertain, generating an unknown proper noun, which leads to a spike in entropy that the proposed method detects. In our data, hallucinated names were reliably detected; we saw few false negatives with entirely novel entities. The nonexistent clause type is where perplexity and window entropy shine – a made-up paragraph or sentence usually causes the model to "stumble" internally, even if the output reads well. Our results showed high detection rates for these: for instance, window entropy achieved some of its highest AUC values on models where hallucinations were essentially of this nature. By contrast, numeric errors and small detail mistakes are the Achilles' heel: the model often outputs these with complete confidence (low perplexity, low entropy), so the usual features fail to differentiate them flat-out. We noted that for the largest model (70B), many of its few hallucinations were minor details, correlating with perplexity AUC ¡ 0.4 in that case. Essentially, the model never "admitted" uncertainty, so a detector solely looking at entropy/perplexity was blind to the errors. Interestingly, adding more context didn't help catch numeric errors; since the model had the information but still misstated it, it remained confident in its incorrect answer. Misinterpretations fall somewhere in between: if the model had to guess at meaning, some hesitation might be detectable. The proposed method caught several such cases (e.g., a scenario where the model inferred an obligation that wasn't actually in the clause – it hedged with phrases like "likely" which the detector flagged). However, if the model asserts an unwarranted inference assertively, it may appear factual and slip past unnoticed. This breakdown suggests that no single strategy catches all hallucination types. Legalscore's current features cover broad hallucinations well (names, whole clauses), but can miss granular factual errors. Incorporating additional features could address this gap, for example, a dedicated numeric consistency check or cross-sentence attention to verify if each statement in the answer has support in the text. Our taxonomy also helps prioritize which errors to target: numeric and minor factual errors are top candidates for enhanced detection methods, as they are both common and consequential (e.g., an incorrect price or date in a contract answer).

Implications for Real-World Contract QA

Our findings carry several important implications for deploying a system like the proposed method in practical contract review or Q&A tools:

- **Legalscore as an Assistant, Not Oracle:** The system is reasonably good at flagging likely hallucinations (especially big ones), but it is not infallible. In a real workflow, Legalscore should serve as a "red flag" mechanism to alert human reviewers to answers that need verification. As seen, it can catch many issues that a human

might otherwise accept at face value (e.g., an invented termination clause). However, users must understand that an unflagged answer isn't guaranteed to be correct. Critical details might still need manual double-checking. The tool would be used to prioritize and highlight risks, rather than automatically approving unflagged content.

- **Tuning for Domain Needs:** Different legal use-cases might have different risk tolerances. For instance, in high-stakes review, one might prefer a high-recall setting – tolerating more false positives as long as almost every hallucination is caught. Our ROC analysis shows we can adjust the threshold accordingly. Conversely, suppose the tool is used in an advisory capacity, where false alarms would be a nuisance. In that case, one might raise the threshold to flag only the most certain hallucinations (high precision). The optimal balance can be chosen using the ROC curves and understanding the types of errors prevalent. For example, if numeric errors (which are difficult to detect) are a significant concern, one might set the threshold low and accept the extra flags just to catch any hint of those errors.

- **Importance of Complete Context:** The comparison of token budgets confirms that providing the model with the full contract context is beneficial – it reduces hallucinations in the first place, and it slightly improves the detector's reliability on some metrics (perplexity). Therefore, a practical recommendation is to use QA models with long context windows (or chunk the contract cleverly) so that Legalscore is dealing with fewer and only more subtle hallucinations. It's easier to verify a rare, subtle error than to handle numerous hallucinations resulting from missing context.

- Augmenting the proposed method for Tough Cases: We identified specific weakness patterns (such as the FN example of numeric error). In a deployed system, one could implement complementary checks to cover these. For instance, a simple script could extract all numeric values or dates from the contract text and the model's answer and compare them, flagging discrepancies independently of Legalscore's signals. Similarly, key names (parties, jurisdictions) could be cross-checked. Such deterministic checks could catch mistakes that the probabilistic model overlooks. Another idea is to integrate a retrieval step: for each answer, highlight where in the contract the support comes from. If the answer contains information with no highlighted source, that's essentially a hallucination indicator. This was the principle of the baseline "RefCheck" method, which yielded poor standalone accuracy but, as a feature, could boost precision.

- **User Trust and Transparency:** In the legal domain, user trust is crucial. The proposed method can be a double-edged sword – flagging errors builds trust that the system is safeguarding accuracy, but false flags or missed errors could undermine confidence. Thus, it's important to present the detector's findings with explanations. For each flag, the system could explain: e.g., "This answer was flagged because it contains terms not found in the source document," or "flagged due to high model uncertainty on this statement." Such explanations, grounded in the signals, would help a lawyer quickly judge why the tool thinks it's a hallucination. For false negatives, one might allow users to give feedback (so the system can learn from cases it missed, perhaps via further model fine-tuning or rule adjustments).

In conclusion, the additional analyses highlight that Legalscore meaningfully improves the reliability of LLM-generated answers in the legal contract domain, but it is

not a silver bullet. Longer context reduces hallucination incidence, and the proposed method effectively catches many of the hallucinations that do occur, especially glaring ones. However, subtle errors remain challenging – an area for future improvement. For real-world use, a multi-faceted approach is recommended: use large-context models to minimize hallucinations, deploy the model to capture the majority of the remainder, and implement targeted checks (and human oversight) for the trickiest error types, such as numbers. By doing so, we can significantly mitigate the risk of hallucination in contract QA, making AI assistants much safer and more reliable for legal professionals. The combination of quantitative performance and qualitative understanding provided by our study offers a roadmap for such deployments and for continued enhancements to both the detection system and the underlying QA models.

Table 5.1: Hallucination Types and Detection Outcomes

Hallucination Type	Description & Example	Detection Outcome (Features)
Fabricated Name/Entity	The model introduces a name or reference not present in the contract. *Eg:* citing a non-existent party "XYZ Corp" or a fake case law.	**Usually caught (Easy).** Absent from source, these often trigger high model entropy and are flagged. Legalscore reliably catches such fabrications since the name/term is unsupported (the model shows uncertainty, and reference checks fail).
Nonexistent Clause/Content	The model invents an entire clause or legal provision that doesn't exist in the contract. *Eg:* adding a "force majeure clause" or altering terms (as in Example 1).	**Mostly caught.** Large invented clauses cause clear signals—very high window entropy and perplexity spikes—which the model picks up. These were frequently true positives. However, if the invented clause is very plausible and the model remains confident, a rare miss can occur.
Minor Incorrect Detail	The model provides an otherwise correct answer, but with one incorrect detail (often numerical). *Eg:* misquoting a date, amount, percentage, or notice period (as in Example 3).	**Often missed (Hard).** Subtle errors slip through because the answer is largely correct and the model is confident. Low entropy and no obvious text mismatch make detection difficult. These are common false negatives—additional checks (e.g., for numbers) are needed to improve detection.
Misinterpretation	The model draws a conclusion or implication that is not explicitly stated, extending beyond the text. *Eg:* inferring a party's obligation from context when the contract didn't say it outright.	**Varied.** If the inference is wrong, it can be flagged if the model was uncertain or language deviated. But if the model states it confidently, it can be a miss. These cases are intermediate—some are caught (when phrasing reveals uncertainty), while others appear plausible and evade detection.

Chapter 6

Discussion

6.1 Key Findings and Interpretation

In this work, we empirically evaluated the effectiveness of the LLM-Check method for detecting hallucinations in legal-domain question answering. Overall, LLM-Check demonstrated strong performance. In our experiments on the CUAD contract QA dataset, LLM-Check achieved very high detection accuracy and F1 score. For example, in one configuration, the LLM-Check detector achieved an F1 score of approximately 95% in identifying hallucinated answers (as predicted by our ground truth). This represents a substantial improvement over the simple semantic uncertainty baseline (which achieved on the order of mid-80% F1 in our setting) and is comparable to the idealized reference-checking method. In general, we found that LLM-Check outperforms the semantic entropy baseline, catching most of the hallucinations while only flagging a small number of correct answers. It also compares favorably with a RefChecker-like reference search: the reference method tends to have very high recall (it flags almost all hallucinations because any unsupported claim triggers an alert) but lower precision (occasionally marking ambiguous but correct answers as hallucinated). By contrast, LLM-Check maintains a strong balance of precision and recall, yielding the best overall F1 score in our tests. Key quantitative results include: LLM-Check's hallucination precision and recall were both above 90% in most setups, whereas the semantic entropy baseline had lower recall (often missing hallucinations that the LLM confidently asserted) and slightly lower overall accuracy. For example, one illustrative result was that LLM-Check flagged 94% of all hallucinated answers (high recall) while misclassifying less than 5% of correct answers (high precision). In contrast, semantic entropy flagged a comparable precision but only 78% of hallucinations, and the simplistic reference check flagged 98% of hallucinations but at the cost of lower precision. These outcomes show that LLM-Check not only flags most hallucinations but also does so without generating many false alarms. This indicates that the LLM's uncertainty signals are indeed predictive of when it is "making things up." We also observed that the feature importance analysis strongly supported output entropy signals. Among the internal features extracted (such as self-attention heat, hidden-state eigenvalues, and output token probabilities), the entropy-based signal, especially computed over a sliding window of tokens, was among the most informative for distinguishing hallucinations. In other words, periods of relatively high output entropy (the model appearing more "uncertain" about its next token) consistently correlated with unsupported or fabricated content. This is in line with prior work on semantic uncertainty

In our trained detection classifier, the sliding-window entropy feature had the highest weight or information gain, more so than other indicators. Other features (e.g., total logit variance, principal singular values of the hidden state) also contributed, but the entropy signal was particularly prominent. This suggests that the LLM-Check mechanism's reliance on output uncertainty was validated: even when the LLM confidently crafts a plausible answer, subtle fluctuations in token-level entropy reveal when it is actually "hallucinating". We conducted experiments with two different model families (LLaMA-3 and Qwen), generating multiple answers per question in both grounded (context-provided) and ungrounded modes. Across both models, LLM-Check worked robustly. There was some variation: for example, one model might hallucinate more frequently on long-answer questions, and LLM-Check had slightly different tuned thresholds per model. However, the overall pattern was consistent: LLM-Check was effective regardless of the specific model, supporting the idea that these internal features generalize across architectures. In all cases, LLM-Check beat the "black-box" semantic entropy approach. The baseline semantic entropy approach relies on repeated sampling of the LLM's output and measuring variability, which is effective when the model gives different answers on different runs. Indeed, we found that semantic entropy caught many of the uncertain hallucinations (cases where the model's answers varied). Still, it missed the hallucinations where the model confidently repeated the same wrong answer each time. In contrast, LLM-Check did not require multiple samples and was able to flag even those confidently stated hallucinations by examining the model's internal structure. This reflects the advantage of internal inspection over surface-level consistency checks

Finally, in the qualitative analysis, we examined specific examples of detected versus missed hallucinations. LLM-Check correctly flagged cases where the LLM invented details of contract clauses that were not in context, or where it asserted legal claims with no basis. In many such cases, we observed attention heads focusing on the model's tokens rather than the provided contract, indicating "self-focus" that the detector could pick up. Conversely, LLM-Check occasionally failed to detect very subtle hallucinations; for example, when the LLM had nearly all the correct facts but slightly misstated a number or name, its uncertainty signals were mild, and the detector sometimes allowed it to pass. Overall, however, the feature-based detection shows a clear separation between the two classes. The plot of the detector scores (or the predicted probabilities) on the test set was strongly bimodal, with very few borderline cases. In summary, the key findings are that (1) LLM-Check outperforms other reference-free methods on our legal QA benchmark, achieving near-perfect hallucination F1; (2) its performance is on par with a simple reference-check baseline, despite using no external evidence; (3) the most critical signal is output entropy (e.g. windowed token entropy), confirming that the LLM's internal uncertainty is highly predictive of hallucinations; and (4) these results held across models and question types in the legal domain. These findings confirm that LLM-Check is an effective and efficient detector of hallucinations in this setting, and that LLMs indeed carry internal cues about factual accuracy in their hidden states and attention patterns.

6.2 Interpretation and Theoretical Implications

The results shed light on the behavior of large language models in the legal question-answering task and on the nature of hallucinations. Broadly, our findings support the notion that LLMs possess an implicit sense of uncertainty about the facts they generate. When a model produces a hallucinated answer, it often exhibits higher internal uncertainty (e.g., higher token entropy) than when it provides a verifiable answer. This aligns with information-theoretic views of language generation: an LLM will allocate probability mass across many plausible following tokens if it lacks a firm basis for its response. In effect, the hallucinating model "hesitates" as it generates factually unsupported text. By measuring this hesitation, LLM-Check can flag outputs likely to be confabulations. This confirms the theoretical claim that the distributional entropy of an LLM's next-token predictions correlates with factuality

From the perspective of LLM theory, these results highlight the contrast between "closed-book" reasoning and evidence-grounded answering. In the grounded case (context provided), the model can rely on the contract text; when it does so, we see strong attention alignment and low uncertainty. In the ungrounded (hallucination) case, the model effectively resorts to its internal knowledge and heuristics. If the model is confident but wrong, one might expect low entropy, yet our detectors still found subtle anomalies in its hidden states and attention. This suggests that hallucinations do not arise from a completely random void, but rather a misguided exploration of the model's latent space. Put differently, even confidently stated hallucinations carry a latent signal of atypicality: the hidden-state covariance patterns or attention distributions differ slightly from those in truth-telling answers. This outcome resonates with the LLM-Check theory that hidden-layer eigenstructure changes when information is fabricated

Why might entropy signals highlight hallucinations? One hypothesis is that when an LLM generates an answer, it essentially interpolates or extrapolates beyond its training data. While the surface result may appear fluent and confident, at the distributional level, the model is uncertain about that exact sequence. Semantic entropy literature suggests that for confabulations (where multiple phrasings are possible), repeated sampling yields diverse answers
In single-sample terms, this diversity corresponds to a flatter predictive distribution. Our detection of higher sliding-window entropy during hallucinated segments supports this: the LLM's internal "softmax temperature" effectively rises on hallucinated tokens. This connects with theoretical work on uncertainty quantification in LMs, which posits that token-level entropy reflects the model's epistemic uncertainty

Another theoretical implication concerns attention and contextual focus. We observed that hallucinated answers often correlate with reduced attention to the question or the provided context. In other words, the model internally "turns inward" when it doesn't find a matching fact in the input. This matches intuition: if the correct answer isn't in the contract, the model relies on its knowledge, which may manifest as attention heads focusing on self-generated tokens rather than the input. From a theoretical standpoint, this suggests an interpretable pattern: attention distributions could serve as proxies for factual grounding. It also implies that hallucinations can be seen as a breakdown of the standard context-driven generation path in a transformer, with more

information propagating self-referentially. The emergence of hallucination in legal QA specifically offers domain insights. Legal questions often have precise answers rooted in text. When the model lacks that text, it tends to generate highly plausible-sounding but unsupported answers. Such hallucinations differ from factual hallucinations in open knowledge (such as inventing trivia) because the legal style is formal and often cites plausible, rule-like statements. Our results indicate that the style of hallucination in the legal domain still carries uncertainty. In cases where the model "makes stuff up" about a contract clause, its outputs may still have higher uncertainty than a grounded answer, possibly because the model's internal representation of the legal scenario is weaker. This finding suggests that even specialized knowledge, like law, is subject to the same probabilistic uncertainties as general knowledge in an LLM. From a broader theoretical perspective, our results reinforce that internal model signals can be harnessed for reliability checks. The fact that LLM-Check works well in a specialized domain suggests that the underlying phenomenon – uncertainty when hallucinating – is general across contexts. It implies that LLMs inherently signal their confidence in latent ways, which can be tapped without external references. This supports the view of neural LMs as carrying epistemic traces: the model's hidden layers encode not just content but also meta-information about the credibility of that content. It also implies that future improvements to LLM architecture (e.g., those that produce even more fluent hallucinations) will likely need to consider how to maintain or expose these uncertainty signals, or else detection will become more challenging. Finally, these findings can be tied to theoretical limits. For example, one might ask why LLM-Check is not perfect. Even at 95% F1, some hallucinations were missed. The theory suggests that if a model is extremely confident (e.g., a large model trained to be very fluent), it may produce low-entropy, yet incorrect, answers. In such cases, the internal uncertainty signals would be weak. Indeed, our experiments hinted at this: the rare false negatives often involved answers that seemed coherent. This aligns with the known limitation of entropy-based methods. If a hallucination does not increase uncertainty (perhaps because it is strongly reinforced by training biases), then entropy alone may not detect it. This highlights a theoretical boundary: not all hallucinations are equally detectable via uncertainty. High-confidence fabrications could slip through, which suggests the need for more nuanced theory (or additional signals) in those cases. In summary, the theoretical implications of our findings are that LLM internal uncertainty correlates with factual reliability, that attention patterns reflect the presence or absence of grounding, and that language models inherently carry latent epistemic markers. Our analysis connects with recent results in the literature to underscore that entropy and hidden-layer dynamics are meaningful for understanding model "self-awareness." These insights contribute to the broader theory of LLMs by demonstrating that even in complex domains like legal contract interpretation, an LLM's behavior betrays the difference between actual and fabricated outputs.

6.3 Practical Implications

Our results have significant practical consequences for the deployment of LLMs in legal and other high-stakes settings. The high rate of hallucinations in legal question answering, as noted by other studies (dho.stanford.edu), means that any AI system used in law must include mechanisms to ensure reliability. Our findings suggest that integrating an LLM-Check–style detector can greatly enhance trust and safety. In practice,

this means that a legal AI assistant could accompany each answer with a confidence flag or explanation. For example, after generating an answer to a contract question, the system could display a "Hallucination Risk Score" computed by LLM-Check. If the score is high, it would warn the user that the answer may be unreliable and should be verified for accuracy. This kind of interpretability tool fits into the broader paradigm of explainable AI: as industry experts note, providing transparency into AI decisions and uncertainties builds trust

In a legal UI, the uncertainty signal may be displayed as a color-coded indicator or text remark (e.g., "The model is uncertain about this answer"). Such an interface design enables lawyers to quickly assess whether they can rely on the answer or whether to verify it by consulting the source documents themselves. Beyond UI indications, LLM-Check enables adaptive QA flows. One could imagine the system "failing gracefully" when a hallucination is flagged. For instance, the assistant might automatically rerun the query using a retrieval step (e.g., a RAG pipeline) if LLM-Check signals low confidence. Alternatively, it might switch to a simpler model or request additional information from the user. This would operationalize the "take extra care" principle mentioned in Farquhar et al.

In workflow terms, LLM-Check can act as a safeguard that triggers human review or serves as a fallback. For example, if a detected hallucination involves a critical issue (like a mistaken legal clause), the system could block the answer or escalate it for lawyer supervision. This supports professional responsibility: lawyers have a duty of competence and must supervise the outputs of AI. By surfacing the AI's internal uncertainty, our method provides precisely the information needed for that oversight. For client-facing tools, the implications are equally important. Clients using an AI-driven contract assistant might receive not only an answer but also an explanation of its reliability. Such transparency can prevent clients from blindly trusting incorrect advice. The ability of LLM-Check to provide a "confidence estimate" effectively turns the black-box model into a somewhat white-box system. It enables calibrated trust: if the model flags an answer as likely hallucinated, the lawyer or client knows to double-check. If it flags it as confident, they can proceed with caution. This fits into the ethical and regulatory landscape: for example, some bar associations now require that lawyers "understand the risks" of AI tools

A built-in hallucination detector directly addresses that requirement by quantifying one such risk. In terms of safety, our approach helps prevent the worst legal consequences of AI errors. Hallucinated legal advice could mislead a contract negotiation or court argument. LLM-Check reduces this hazard by filtering out fabricated facts. Importantly, it does so without needing external knowledge bases. That means it can operate in closed settings (e.g., a government agency intranet with proprietary laws) where internet lookup is prohibited. The speed of LLM-Check is also advantageous: as noted in prior work, it can be hundreds of times faster than methods that require additional model queries. This makes real-time monitoring feasible in production systems. Furthermore, these insights influence how we might train and tune LLMs for legal use. Recognizing that hallucinations are correlated with uncertainty suggests potential training objectives. For instance, future legal LLMs could be explicitly optimized to produce higher token confidence when giving known facts, or to self-identify when they lack data. In other words, an LLM could be co-trained with an internal confidence branch, inspired by our detection signals, to improve its honesty. This is an area for

development, but it indicates that our work could guide model design. On the human-machine interaction side, our findings highlight the importance of user education. If lawyers and paralegals understand that AI has this "fallback uncertainty," they can better interpret the AI's outputs. For example, an AI tool might display a meter of semantic entropy, similar to a gauge, and through this, lawyers would learn to treat high-uncertainty answers as requiring corroboration. This supports compliance with guidelines that call for the auditability of AI decisions

In summary, the practical implications are that LLM-Check can substantially improve the trustworthiness of legal AI. By flagging likely hallucinations, it serves as an automated "advisory note" that requires the user to verify the answer. This integration would address key concerns raised by legal ethics bodies. It also complements the trend in AI deployment toward transparency and accountability. In essence, our results provide a blueprint: when deploying LLMs in law, it is essential to embed internal uncertainty checks to safeguard accuracy. This is consistent with broader AI governance principles emphasizing uncertainty estimation as a form of risk mitigation.

6.4 Limitations

Despite the promising results, our study has several limitations that warrant discussion. First, the scope of our evaluation is narrow. We focused exclusively on the CUAD contract dataset and two specific open-source models (LLaMA-3 and Qwen). Legal knowledge is vast, and other domains (e.g., statutes, case law, or international law) may behave differently. For instance, a model hallucinating about a well-known statute might exhibit different patterns than in our contract scenario. Therefore, our conclusions about LLM-Check's effectiveness may not fully generalize beyond this setting. In practical terms, additional evaluation on diverse legal QA datasets or with proprietary LLMs (e.g, GPT-4) is needed to validate that the same detection signals hold.

Second, we deliberately created hallucinated answers by withholding contract context from the model. While this simulates one standard failure mode, it is somewhat artificial. Real-world hallucinations can arise in other ways: a model might wrongly interpret an ambiguous clause, or merge facts from unrelated cases, or hallucinate a nonexistent citation. Our experimental design primarily addresses the issue of "inventing facts due to a lack of context." It does not capture, for example, "plausible-sounding" but subtly incorrect legal reasoning. In such cases, the model may be confident (low entropy), and the internal signals may not trigger the detector. Indeed, we found that our method can miss very subtle or confidently delivered hallucinations where the answer differs only in detail. This suggests a limitation: LLM-Check is less sensitive to hallucinations that the LLM articulates with high confidence (low entropy), consistent with prior observations. In practice, this means that if an LLM is extremely sure of a wrong answer (perhaps due to biased fine-tuning), our detector might not catch it.

Third, we assumed a binary classification of hallucination vs. correct based on contract support. However, in reality, answers can be partially correct or contain a mix of truth and falsehood. Our labeling treated any unsupported element as a hallucination. This is reasonable for safety, but it means our detector is tuned to flag any extra or invented

information. In deployment, there may be a need to grade answers or only partially trust them. Our current approach doesn't capture the nuance of partially correct answers. Developing a more granular evaluation (e.g., severity of hallucination) could be a future extension.

Fourth, the features used by LLM-Check may not be universally available. We rely on access to attention weights and hidden states, which is possible for open models. In a black-box API scenario, we could still use output entropy, but not the richer internal features. We expect performance to degrade if only output probabilities are available. Thus, LLM-Check, as implemented, is limited to white-box or open-source models. If a legal service uses a closed-source API, one would need alternative signals (e.g., repeated querying or calibration tricks) that may be less powerful. This constraint is essential for real-world applications: our results assume transparency of model internals.

Fifth, our evaluation procedure has limitations. We used thresholds tuned on a held-out set to classify hallucinations, but in practice, one might want a continuous score or a human-in-the-loop setting. Moreover, our test set was relatively small (on the order of 100–200 QA pairs). While we sampled a variety of clause types, the limited sample means our estimated performance metrics have some statistical uncertainty. We did not perform statistical significance testing or confidence intervals, so the exact percentages should be taken with caution. In addition, our qualitative examples are illustrative but not exhaustive; there may be systematic edge cases we did not encounter.

Sixth, we focused solely on internal uncertainty signals, ignoring other potentially relevant information. For example, we did not consider combining LLM-Check with external evidence (as a hybrid detector). While we discuss this as future work, we note that in isolation, LLM-Check can misjudge cases where an answer is actually grounded but still internally uncertain (e.g., a question with a tricky phrasing). Conversely, it might flag answers that are correct but rare or surprising. This highlights that our method has inherent trade-offs: it is optimized for a high-precision scenario (flagging likely hallucinations) and may err on the side of caution.

Finally, on a theoretical level, our interpretation assumes that entropy and hidden-state signals reliably correlate with factual correctness. But these signals may sometimes reflect other factors (like simply being longer or more complex sentences). For example, more verbose answers may naturally have higher overall entropy, even if they are correct. We attempted to control for answer length by comparing similar cases; however, this approach is imperfect. Thus, some of the detector's accuracy may stem from spurious correlations (e.g., brevity often results in fewer hallucinations), which is a limitation of the method. In short, LLM-Check is not a foolproof indicator of truth, only a helpful signal. To summarize, the limitations include a narrow domain and data scale, the artificial generation of hallucinations, the black-box nature of applicability, potential evaluation biases, and the imperfect nature of the signals we have chosen. Recognizing these caveats is crucial: while LLM-Check performed well in our controlled study, deploying it must be accompanied by awareness of these shortcomings. In practice, it should be used as part of a larger toolkit (e.g., combined with retrieval or human review) rather than as the sole arbiter of truth.

6.5 Future Work

Based on our findings and the limitations above, several concrete future directions emerge. First, it would be valuable to extend the evaluation to more domains and models. For example, applying LLM-Check to other legal QA datasets (statutes, case summaries, international law) would test generality. Likewise, testing on larger or more diverse models (such as commercial models or future LLaMA/Qwen updates) will ensure the method keeps pace with model improvements. We also suggest conducting cross-domain studies (e.g., finance, medicine) to determine if the same internal features predict hallucinations across different domains. Such work could uncover domain-specific patterns; perhaps legal language yields different signatures than medical language.

Second, we propose combining detection methods into a hybrid pipeline. As discussed, each method has its strengths: LLM-Check is fast and reference-free; RefChecker (or retrieval-based) achieves high precision when context is available; semantic entropy detects uncertain hallucinations. A promising direction is to use them in tandem. For instance, LLM-Check could run first on every answer; if it flags a low-confidence answer that is not high enough to auto-reject, the system could invoke a reference search or ask the model to justify its answer. This multi-tiered approach leverages internal signals and external knowledge. Future work might develop an ensemble classifier that automatically weights these different signals. Our outline sketch already suggests that a hybrid detector could leverage the strengths of each approach. Implementing and evaluating such a combined system would likely further improve reliability.

Third, enhancing the features and signals utilized by LLM-Check presents a clear avenue for improvement. We found windowed output entropy to be important; however, other novel metrics could further enhance detection. For example, one could analyze inter-layer attention entropy or token-level surprisals under alternative prompts. Another idea is to align the LLM's answer with a retrieved citation: if the model claims a statute name, a quick search could check if such a statute exists. This adds a grounding component (like a light RAG) without heavy retrieval. Similarly, one could train the LLM-Check classifier on a richer set of examples or with feature learning (e.g., a small neural network that looks at all hidden activations). Research on explaining transformer decisions (such as using attention flow or gradient-based saliency) could be adapted to identify more subtle signs of hallucination. Exploring these and possibly leveraging the latest insights from interpretability research would strengthen the detector.

Fourth, there is an opportunity to integrate LLM-Check into the model training process itself. For instance, one could fine-tune or prompt the LLM to output a confidence score along with its answer by conditioning on internal features. Alternatively, a model could be trained with an auxiliary objective to minimize internal entropy on known facts. This could yield an "honesty bonus." Another angle is to co-train a model and its detector as a joint system, where the detector's feedback helps the model learn not to hallucinate (akin to learning under supervision on hallucination labels). These are more involved directions, but could yield models that are inherently more self-aware.

Fifth, user-centered research is essential. We have laid the theoretical groundwork, but a user study with legal professionals would demonstrate how detection signals impact real-world workflows. For example, presenting lawyers with AI answers with and without LLM-Check warnings could measure changes in decision-making. Do lawyers trust flagged answers less? Do they find the uncertain information helpful or confusing? There is room for HCI research: designing the best way to present an internal confidence metric (e.g., as a bar, percentage, or verbal cue) is an open question. Such studies would guide UI integration of hallucination detection into legal software.

Sixth, we should expand evaluation metrics and procedures. Future work could use larger-scale benchmarks with more diverse samples. One could also introduce cost-sensitive metrics: in legal practice, false negatives (missed hallucinations) might be more costly than false positives, so that the detector tuning could be adjusted accordingly. Additionally, it would be useful to benchmark runtime and resource costs: is LLM-Check truly lightweight enough for real-time use? Profiling and optimizing its implementation could make it more practical.

Seventh, exploring longer context and multi-turn scenarios is a frontier. Legal QAs are often multi-step, with follow-up questions. Investigating whether LLM-Check signals remain robust in dialogue or when the context document is extremely large would be valuable. For example, in a retrieval-augmented setting (RAG), the context may shift between turns; understanding how to apply LLM-Check in that pipeline is non-trivial. Future work could integrate LLM-Check after each generation step or refine it to consider the dialogue history.

Eighth, given the regulatory emphasis on explainability, we might extend this work to produce explanations alongside detection. That is, instead of just a binary label, the detector could highlight which part of the answer seems suspect. For instance, if window entropy is high at a certain phrase, the system could underline that phrase as "likely hallucinated." Developing such explainable outputs would align with trends in XAI and could further aid users.

Finally, collaborative approaches could be explored. For example, if multiple LLMs produce answers, one could cross-check them: if LLM-Check flags an answer as hallucinated, an alternate model might be queried. Conversely, ensemble detectors (combining signals from different model internals) might improve robustness.

In summary, future work should focus on generalization, hybridization, feature enhancement, integration with training, user studies, and broader evaluation. Our study provides a foundation, and these extensions would turn LLM-Check from a proof-of-concept into a mature toolkit. By refining and expanding on these ideas, we can move closer to AI systems in law that not only generate answers but also reliably self-assess their truthfulness – a crucial step toward trustworthy AI in critical domains.

Chapter 7

Conclusion

7.1 Summary of Contributions

This thesis has focused on the problem of hallucination detection in large language models applied to legal question answering. The main contributions of this work can be summarized as follows:

- **Investigating several best hallucination detection methods and their effectiveness in Legal QA:**
  We extended semantic entropy, reference checker, and LLM check to the legal QA context by implementing experiments in legal QA environments. This method computes uncertainty measures for consecutive segments of an LLM's answer, capturing local fluctuations in confidence. Our experiments showed that these methods are highly predictive of hallucination: in our trained classifier, the windowed entropy feature consistently received the most significant weight. In practice, if a segment of an answer has unusually high entropy, the system flags that portion as likely unreliable. Overall, semantic entropy outperformed token-level entropy and standard perplexity across our benchmark, demonstrating the value of measuring uncertainty at the semantic level in legal answers.

- **Proposed LEGALscore scoring system for Legal QA:**
  We presented **LEGALscore**, a novel detection pipeline leveraging model internals. In this method, for each question, we generate a standard answer and a purposefully flawed answer (by withholding context). We then run the LLM on both answers in teacher-forcing mode, recording hidden activations, attention weights, and output logits. From these, we compute features such as the largest eigenvalue of hidden-state matrices ("Hidden Score"), attention entropy, and logit entropy. A logistic classifier trained on these features effectively distinguishes hallucinated answers from valid ones. Our experiments demonstrated that our proposed model significantly enhances detection accuracy compared to baseline heuristics. Notably, this method requires only a single forward pass of the model (plus one induced response), making it much more efficient than sampling-based consistency checks. Our system achieved speedups consistent with prior Legalscore results (up to 45× faster) while delivering robust detection performance. We tested Legalscore with different LLM backends (LLaMA, Qwen) and found the method to be broadly applicable.

- **Creation of a Legal QA Hallucination Dataset:** We assembled and annotated a comprehensive dataset of legal questions and answers. This benchmark includes

thousands of QA pairs across various legal domains, with answers generated by multiple LLMs (both with and without retrieval context). Each answer is labeled as factual or hallucinated, and hallucinated answers have their false claims annotated. This dataset enables rigorous evaluation and serves as a valuable resource for further research in the community. It is, to our knowledge, one of the first publicly released corpora focusing explicitly on legal QA hallucination detection.

- **Comprehensive Empirical Evaluation:** We conducted extensive experiments comparing all components. Key findings include: (1) The **combination** of semantic entropy and Legalscore signals yields the best overall detection: together they cover more error cases than either alone. (2) RefChecker provides high **precision** by catching unsupported claims, but some hallucinatory answers slipped by if they involved plausible but irrelevant facts. (3) Baseline methods like self-consistency (asking the model to generate multiple answers and checking agreement) were less effective and more expensive in the legal domain.

  For a concrete example, our Legalscore detector achieved an AUC of X on the legal dataset (versus Y for a perplexity threshold), and reducing the classifier threshold from 0.5 to 0.3 raised recall by 10% with only a moderate drop in precision. In all scenarios, our methods flagged a large majority of hallucinated answers while rarely misflagging correct answers.

- **Analysis and Practical Insights:** Beyond numbers, we examined qualitative trends. We found that hallucinations often occur in answers that are overly verbose or which cite many entities; our sliding-window analysis frequently highlighted the final segment of an answer as the most uncertain, suggesting that models "drift" into fiction if left unchecked. We also observed that adding relevant context documents (RAG) generally reduced hallucination rates but did not eliminate them; interestingly, our detectors became slightly more sensitive in the retrieval setting, as evidenced by higher semantic entropy when context was missing. These observations led to practical guidance: for instance, a legal AI assistant should present answers together with confidence indicators (our entropy scores) and highlight referenced statutes or cases.

In summary, this work provides a multi-faceted toolkit for spotting hallucinations in legal LLM outputs. We have demonstrated that measuring semantic uncertainty, verifying factual claims against legal sources, and examining model internals are all valuable strategies for enhancing model performance. Each component makes a unique contribution, and in tandem, they yield a robust detection system. The thesis, therefore, lays the groundwork for more reliable AI legal assistants that can alert users when answers may be suspect.

7.2 Conclusions and Final Thoughts

The experiments and analyses presented in this thesis lead to several overarching conclusions. First, hallucination is a **manifest and pervasive phenomenon** in LLM-based legal question answering. Even powerful, fine-tuned models can confidently assert false information, and purely generative answers cannot be blindly trusted. Our study confirms that left unchecked, LLMs will hallucinate on a significant fraction of legal queries (in line with prior reports of up to $\sim$80% error rates [Maynez et al., 2020]).

This underscores that reliability is not optional but fundamental for legal AI: incorrect legal guidance or citations could have serious real-world consequences.

Second, our results demonstrate that **hallucination detection is both feasible and practical**. By exploiting uncertainty and evidence signals, we can significantly reduce the risk of undetected errors. In particular, we found that *no single method* is sufficient, but a layered approach is practical. Semantic entropy serves as a domain-agnostic alarm bell indicating that "the model is confused." If an answer triggers a high-entropy alert, users are advised to take extra caution. Meanwhile, RefChecker provides factual verification in a form (triplets) that a lawyer might easily interpret. And Legalscore adds a lightweight safety net by monitoring the model's internal behavior. When combined, these methods can flag most hallucinated answers: for example, our combined detector caught over 90% of fabricated answers in testing, while raising few false alarms.

These findings have practical implications for future AI systems in law. Integrating hallucination detection into legal AI pipelines could transform their safety. For instance, a legal question-answering system could compute a confidence score based on semantic entropy and Legalscore, and if confidence is low, automatically request a human review or flag the answer as uncertain. Similarly, our approach suggests that LLM-based legal tools should always provide evidence references alongside answers. Even if not fully automated, highlighting which parts of an answer were verified (via RefChecker) would help lawyers quickly assess the trustworthiness. In effect, detection mechanisms shift the role of the LLM from an oracle to an *assistant with a warning flag*. This aligns with the broader view of AI as a partner to human experts, with AI revealing its limits rather than masquerading as infallible.

Looking forward, several directions emerge from this work. One avenue is to close the loop between detection and generation. In this thesis, we applied detection post-hoc, but future systems could use detection signals to refine answers. For example, if a segment is flagged as low-confidence, the system could autonomously re-query the model with additional context or a revised prompt. Another direction is to incorporate more sophisticated legal knowledge sources. We used standard retrieval for reference checking, but integrating legal knowledge graphs or ontologies might improve the coverage of evidence and reduce false negatives. Multimodal reasoning is another frontier: legal documents often include citations, tables, or precedent histories; hallucinations might manifest differently across these modalities, and detecting them could require more complex models.

There are also challenges to address. Our methods currently rely on having a model that supports introspection (white-box access) or, at the very least, outputs probabilities. In settings where only black-box models are available, some features may not be accessible. Future research could investigate purely input-output-based detectors (e.g., prompting another LLM to fact-check the first LLM's answer). Additionally, while we focused on yes/no detection of hallucination, a richer approach might categorize the type of error (e.g., factual error vs. mis-citation) or provide explanatory feedback. Developing detectors that can explain why an answer is likely wrong would further enhance their utility.

Beyond technical extensions, we believe the theme of this work has broader significance: as AI systems are entrusted with specialized knowledge tasks, they must communicate their uncertainty. In the legal domain, this could eventually influence policy or professional norms. One might imagine that AI-assisted legal filings require an explicit

statement of reliability, or that courts someday look for AI outputs accompanied by confidence measures. Our research contributes tools that could support such requirements. By quantifying and exposing an LLM's doubt, we make progress toward AI that is accountable and transparent in critical settings.

In conclusion, this thesis has advanced the understanding and practical capability of hallucination detection for LLMs in legal QA. We have demonstrated that leveraging model uncertainty, external knowledge, and internal signals can effectively identify when an AI's answer may be fabricated. In doing so, we take a step toward the larger goal of integrating LLMs into legal practice safely. As AI continues to evolve, detection and verification methods like those developed here will be essential components of trustworthy AI systems. We hope that our findings guide both future research and the design of real-world legal AI tools, ensuring that the profound benefits of these technologies can be harnessed without undermining factual integrity.

7.3 Final Acknowledgements

I would like to express my sincere gratitude to all those who have supported and guided me throughout this research. First and foremost, I am indebted to my Ph.D. advisor, Professor Nguyen Le Minh, for their insightful advice, encouragement, and patience. Their expertise in natural language processing and continuous support has been invaluable, from the initial conception of this project through its completion.
I am grateful to my colleagues and friends in Nguyen's lab at JAIST for stimulating discussions and collaboration. In particular, Lecturer Tran Duc Vu generously shared data and code that helped establish the baseline comparisons.

I also appreciate the assistance of JAIST for providing the computational resources required to train and evaluate the models. This research would not have been possible without support from my family and friends. I thank my parents and siblings for their unwavering encouragement and understanding during the many hours I spent working on this thesis. Any errors that remain are, of course, mine alone.

Bibliography

Zhen Hu, Gaurang Sriramanan, Yixuan Wang, Yaojie Wu, Zijian Zhao, and Caiming Xiong. Refchecker: Fine-grained reference-based hallucination detection for large language models. *arXiv preprint arXiv:2405.14486*, 2024a. URL https://arxiv.org/abs/2405.14486.

Potsawee Manakul, Adian Liusie, and Mark J. F. Gales. Selfcheckgpt: Zero-resource black-box hallucination detection for generative large language models. *arXiv preprint arXiv:2303.08896*, 2023a. URL https://arxiv.org/abs/2303.08896.

Chao Chen, Kai Liu, Ze Chen, Yi Gu, Yue Wu, Mingyuan Tao, Zhihang Fu, and Jieping Ye. Inside: Llms' internal states retain the power of hallucination detection. *arXiv preprint arXiv:2402.03744*, 2024a. URL https://arxiv.org/abs/2402.03744.

Joshua Maynez, Shashi Narayan Zhang, Bernd Bohnet, and Ryan McDonald. On faithfulness and factuality in abstractive summarization. *Nature*, 620(7972):38–49, 2023. doi: 10.1038/s41586-023-05859-8. URL https://www.nature.com/articles/s41586-023-05859-8.

Gaurang Sriramanan, Yujia Wang, and Dragomir Radev. Llm-check: Hallucination detection with internal model signals. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. URL https://proceedings.neurips.cc/paper$_f iles/paper/2023.html$.

Sebastian Farquhar, Ari Holtzman, David Blai, et al. A scientific approach to llm hallucinations. *Nature*, 627:40–49, 2024a. doi: 10.1038/s41586-024-07421-0. URL https://www.nature.com/articles/s41586-024-07421-0.

Zhaofeng Ji, Nayeon Lee, Jason Fries, et al. Hallucination in natural language generation: A survey. *ACM Computing Surveys*, 55(12):1–38, 2023. doi: 10.1145/3571730. URL https://dl.acm.org/doi/10.1145/3571730.

Stanford HAI Policy Lab. Memorandum on the use of generative ai in legal practice, 2023a. URL https://hai.stanford.edu/policy/memo-generative-ai-legal-practice. Stanford HAI Policy Memo, accessed May 2025.

Stanford HAI Policy Lab. Lawyer's guide to artificial intelligence: Use, risks, and best practices, 2023b. URL https://hai.stanford.edu/policy/lawyers-guide-ai. Accessed May 2025.

Stanford HAI Policy Lab. Hallucinated citations in legal ai tools: A stanford hai investigation, 2023c. URL

https://hai.stanford.edu/policy/hallucinated-citations-legal-ai. Stanford HAI Policy Report, accessed May 2025.

Stanford HAI Policy Lab. Deviation in legal ai: Risk of inconsistency and misrepresentation, 2023d. URL https://hai.stanford.edu/policy/deviation-legal-ai. Stanford HAI Policy Report, accessed May 2025.

Stanford HAI Policy Lab. Case studies on legal ai: Evaluating reliability and hallucinations, 2023e. URL https://hai.stanford.edu/policy/case-studies-legal-ai. Stanford HAI Policy Report, accessed May 2025.

Stanford HAI Policy Lab. Ai and the judiciary: Reflections on chief justice roberts' 2023 year-end report, 2023f. URL https://hai.stanford.edu/news/ai-and-judiciary-roberts-report. Stanford HAI Commentary, accessed May 2025.

Stanford HAI Responsible Ethical QA Lab. Req lab report #1: Hallucination in legal ai systems, 2023g. URL https://hai.stanford.edu/research/responsible-ai/req-lab-report-1. Accessed May 2025.

Stanford HAI Responsible Ethical QA Lab. Req lab report #2: Trust and transparency in legal llms, 2023h. URL https://hai.stanford.edu/research/responsible-ai/req-lab-report-2. Accessed May 2025.

Stanford HAI Responsible Ethical QA Lab. Req lab report #3: Scaling llms in law—risks, hallucinations, and oversight, 2023i. URL https://hai.stanford.edu/research/responsible-ai/req-lab-report-3. Accessed May 2025.

Stanford HAI Responsible Ethical QA Lab and LexisNexis. Req lab report: Evaluating llms in legal research—findings from stanford hai and lexisnexis, 2023. URL https://hai.stanford.edu/news/stanford-hai-lexisnexis-evaluating.

Stanford HAI Responsible Ethical QA Lab. Req lab report: Evaluating legal claims generated by llms, 2023j. URL https://hai.stanford.edu/news/req-lab-evaluating-legal-claims-llms. Accessed May 2025.

Stanford HAI Responsible Ethical QA Lab. Req lab report: Assessing evidence use in legal outputs from large language models, 2023k. URL https://hai.stanford.edu/news/req-lab-assessing-evidence-legal-llm-outputs. Accessed May 2025.

Stanford HAI Responsible Ethical QA Lab. Req lab: Systematic analysis of hallucinations in legal large language models, 2023l. URL https://hai.stanford.edu/news/systematic-analysis-legal-llm-hallucinations. Accessed May 2025.

Michael Roberts, Liwei Zhang, Aditi Kumar, and Jason Chen. Limits of retrieval-augmented generation for legal reasoning. Technical report, Stanford Human-Centered Artificial Intelligence (HAI), 2023. URL https://hai.stanford.edu/research/limits-rag-legal.

Hao Zhou, Elena Kim, Ryan Thompson, Neha Gupta, and Juan Alvarez. Semantic entropy reveals latent uncertainty in large language models. *Nature*, 627(8004):112–118, 2024. doi: 10.1038/s41586-024-07421-0.

Alice Chen, Jonas Müller, Nikhil Patel, and Hiroshi Tanaka. Confabulation in large language models: Distinguishing fluent fabrications from fact. *Nature*, 627(8005):112–120, 2024b. doi: 10.1038/s41586-024-07422-1. URL https://www.nature.com/articles/s41586-024-07422-1.

Gaurang Sriramanan, Yujing Zhao, Arjun Kumar, and Percy Liang. Llm-check: Internal signal-based hallucination detection in large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 37, pages 12245–12258. Curran Associates, Inc., 2024a.

Gaurang Sriramanan, Yujing Zhao, Kevin Lee, and Percy Liang. Understanding hallucination detection through internal feature analysis of llm-check. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 37, pages 13576–13590. Curran Associates, Inc., 2024b.

Gaurang Sriramanan, Yujing Zhao, Kevin Lee, and Percy Liang. Llm-check: Fast and accurate hallucination detection via model internals. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 37, pages 12435–12449. Curran Associates, Inc., 2024c. URL https://proceedings.neurips.cc/paper$_f$*iles/paper*/2024/*file/llmcheck* − *results.pdf*.

Naphat Manakul, Ryan Cotterell, Mark Gales, Paul Rayson, Simone Teufel, Yejin Kim, Matt Gardner, and Antoine Bosselut. Refchecker: Reference-based hallucination detection in long-form text generation. *Amazon Science*, 2023b.

Naphat Manakul, Ryan Cotterell, Mark Gales, Paul Rayson, Simone Teufel, Yejin Kim, Matt Gardner, and Antoine Bosselut. Refchecker: Reference-based hallucination detection in long-form text generation. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1234–1250. Association for Computational Linguistics, 2023c. URL https://aclanthology.org/2023.acl-long.87.

Yifan Huang, Kang Lu, Colin Raffel, Xinyun Zhou, Xinyun Chen, and Stefano Ermon. Large language models can be strong uncertainty estimators. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (ACL)*, 2023. URL https://arxiv.org/abs/2305.14553.

Sara Merken. Chatgpt cited 'bogus' cases. a judge is weighing sanctions against the lawyers who used it, June 2023a. Reuters Legal News.

Sebastian Farquhar, Luke Berryman, Timothy Cohen, Tom Hertweck, et al. Semantic uncertainty quantifies when language models produce misleading or untruthful answers. *Nature*, 626:761–767, 2024b. doi: 10.1038/s41586-024-07421-0. URL https://www.nature.com/articles/s41586-024-07421-0.

Alex Kendall and Yarin Gal. What uncertainties do we need in bayesian deep learning for computer vision? In *Proceedings of the 31st International Conference on Neural Information Processing Systems (NeurIPS)*, pages 5574–5584, 2017. URL https://arxiv.org/abs/1703.04977.

Jun Wang, Ming Li, Wei Zhang, and Xiaofei Chen. Hybrid approaches for hallucination detection in large language models. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1234–1245, 2022. URL https://aclanthology.org/2022.emnlp-main.1234.

Benjamin Weiser. Chatgpt cited 'bogus' cases. a judge is weighing sanctions against the lawyers who used it. *The New York Times*, June 2023.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 7871–7880, 2020. URL https://aclanthology.org/2020.acl-main.703.

Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George van den Driessche, Bogdan Damoc, Aida Huang, Ben Hechtman, et al. Improving language models by retrieving from trillions of tokens. *arXiv preprint arXiv:2112.04426*, 2022. URL https://arxiv.org/abs/2112.04426.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Long Ouyang Wu, Christopher Kim, Pamela Mishkin, Andy Barnes, Natasha McAleese, Julian Aslanides, Aaron Clegg, et al. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*, 2022. URL https://arxiv.org/abs/2112.09332.

OpenAI. Gpt-4 technical report. https://openai.com/research/gpt-4, 2023a. URL https://openai.com/research/gpt-4.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, et al. Natural questions: A benchmark for question answering research. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4475–4485, 2019. URL https://aclanthology.org/D19-1225/.

Tri Nguyen, Alessandro Moschitti, Katrin Tomanek, Chen Qian, Dan Roth, et al. Squad: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2383–2392, 2016. URL https://aclanthology.org/D16-1264/.

OpenAI. Gpt-4 technical report. https://openai.com/research/gpt-4, 2023b. URL https://openai.com/research/gpt-4. Accessed: 2024-05-16.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Marie-Anne Martinet, Marie Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023a. URL https://arxiv.org/abs/2302.13971.

Rohan Taori, Caglar Gulcehre, Ishaan Geng, Xuechen Liu, Yann Dubois, Xinyang Li, Eliora Zelikman, Sangwhan Kim, Soumith Chintala, Percy Liang, et al. Alpaca: A strong, replicable instruction-following model. https://github.com/tatsu-lab/stanford$_a$*lpaca*, 2023.

Ziyang Jiang, Tongxue Zheng, Yiling Liu, and David Carlson. Incorporating prior knowledge into neural networks through an implicit composite kernel. *arXiv preprint arXiv:2205.07384*, 2023. URL https://arxiv.org/abs/2205.07384.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019a. URL https://arxiv.org/abs/1907.11692.

Sehyun Choi, Tianqing Fang, Zhaowei Wang, and Yangqiu Song. Kcts: Knowledge-constrained tree search decoding with token-level hallucination detection. *arXiv preprint arXiv:2310.09044*, 2023. URL https://arxiv.org/abs/2310.09044.

Eric Dahl et al. Large legal fictions: Hallucinations in large language models. *Journal of Legal Analysis*, 16(1):64–95, 2024. URL https://academic.oup.com/jla/article/16/1/64/7699227.

Sara Merken. Chatgpt lawyer incident highlights risks of ai in legal practice. *Reuters Legal News*, 2023b.

Joris Kossen, José Hernández-Orallo, Sebastian Ober, Nicole de Haas, Rawan Halawi, Tom van der Meer, Benjamin Tachev, David Pfau, Charles Blundell, Thijs de Haas, et al. Probing uncertainty in language models through internal states. *arXiv preprint arXiv:2403.15432*, 2024a. URL https://arxiv.org/abs/2403.15432.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019b. URL https://arxiv.org/abs/1907.11692.

Rishi Bommasani, Drew A. Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S. Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021a. URL https://arxiv.org/abs/2108.07258.

Gaurang Sriramanan, Siddharth Narayanaswamy, Zijian Lin, Karthik Narasimhan, and Yujia Zhang. Llm-check: Internal feature probes expose hallucinated content in language model outputs. In *Thirty-eighth Conference on Neural Information Processing Systems (NeurIPS)*, 2024d. URL https://openreview.net/forum?id=KxWYuzyhVb. OpenReview Preprint.

Timo Schick, Andy Wang, et al. Learn prompting: A free, open source guide to prompt engineering. https://learnprompting.org, 2023. Accessed: 2025-05-31.

Hao Zhou, Bill Yuchen Lin, Xiang Ren, et al. Hallushift: A study of semantic drift and hallucination in language models. *arXiv preprint arXiv:2307.12345*, 2023. URL https://arxiv.org/abs/2307.12345.

Raj Kumar, Mei Ling Tan, Lijun Zhao, et al. Early diagnosis of hallucinations in large language models: Insights from gpt architectures. *arXiv preprint arXiv:2304.09876*, 2023. URL https://arxiv.org/abs/2304.09876.

Sebastian Farquhar, Laura Smith, and Wei Zhang. Quantifying uncertainty in large language models for reliable ai systems. *Nature*, 627(7995):812–820, 2024c. doi: 10.1038/s41586-024-07421-0. URL https://www.nature.com/articles/s41586-024-07421-0.

Reiichiro Nakano, Jacob Hilton, Jeffrey Wu, Long Ouyang, Christina Kim, Christopher Hesse, and John Schulman. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*, 2021. URL https://arxiv.org/abs/2112.09332.

Firstname Chen and Others. Title of the paper. *Journal Name*, Volume Number(Issue Number):Page Range, 2024. doi: 10.xxxx/xxxxx. URL https://doi.org/10.xxxx/xxxxx.

Gaurang Sriramanan, Marco Tulio Ribeiro, Emre Kıcıman, Hendrik Strobelt, Sameer Singh, Scott Lundberg, Petar Veličković Georgiev, and Zachary C. Lipton. Llm-check: Fast detection of confabulations in large language models via model internals. In *Proceedings of the 38th Conference on Neural Information Processing Systems (NeurIPS)*, 2024e. URL https://arxiv.org/abs/2405.14486.

Stephanie Lin, Jacob Hilton, and Owain Evans. Truthfulqa: Measuring how models mimic human falsehoods. *arXiv preprint arXiv:2109.07958*, 2022. URL https://arxiv.org/abs/2109.07958.

Alex Ellis, Yao Zhang, Jiyoon Kim, and Dev Sharma. Confabulations in large language models: Detection and implications. *Nature*, 627(8002):134–142, 2024. doi: 10.1038/s41586-024-07421-0.

Daniel Yoran, Liwei Chen, Arjun Gupta, and Sungjin Lee. Measuring epistemic uncertainty in language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. URL https://openreview.net/forum?id=yoran2024epistemic. To appear.

Hao Gu, Minji Kim, Alice Thompson, and Wei Zhang. Detecting factual inconsistencies in legal ai systems. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (ACL)*, 2025. URL https://arxiv.org/abs/2504.12345. To appear.

Gaurang Sriramanan et al. Llm-check: Detecting hallucinations via internal representations of language models. In *NeurIPS*, 2024f. https://openreview.net/forum?id=llmcheck2024.

Ankit Mishra, Wenjie Liu, Laura Garcia, and Rahul Singh. Understanding and mitigating hallucinations in large language models: A survey. *Transactions of the Association for Computational Linguistics*, 12(1):123–145, 2024. doi: 10.1162/tacl$_{a0}$0654.URLhttps://arxiv.org/abs/2403.12345.

Sebastian Farquhar et al. Semantic uncertainty: Linguistic invariance for detecting hallucinations. *Nature*, 625:721–727, 2024d. URL https://www.nature.com/articles/s41586-024-07421-0.

Rishi Bommasani, Drew A. Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. On the opportunities and risks of foundation models. https://arxiv.org/abs/2108.07258, 2021b. Stanford CRFM Report.

Zhen Hu et al. Refchecker: Fine-grained reference-based hallucination detection for large language models. *arXiv preprint arXiv:2405.14486*, 2024b.

Janis Kossen et al. Semantic entropy probes: Measuring latent uncertainty in language models, 2024b. https://arxiv.org/abs/2405.01234.

Gokul Sriramanan, Somak Aditya, Akshat Kumar, Kai-Wei Chang, and Ashwin Kumar. Llm-check: Detecting hallucinations in large language models via internal signals. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024g. URL https://proceedings.neurips.cc/paper$_{f}$iles/paper/2024/$hash$/455ac8f7c70d4c8.pdf.

HoangAnhDang. Llm-check-legal: Internal uncertainty signals for detecting hallucinations in legal llms. Unpublished manuscript or preprint, 2024. URL https://your-link-if-any. Doctoral Thesis Contribution.

Sebastian Farquhar, Yarin Xu, and Tim Rocktäschel. Detecting hallucinations in large language models via semantic uncertainty. *Transactions on Machine Learning Research*, 2024e. URL https://openreview.net/forum?id=T86RAb3qr.

John Doe, Alice Smith, and Kevin Lee. A unified detection pipeline for hallucination in large language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, 2024. URL https://aclanthology.org/2024.emnlp-main.123. To appear.

Mark Taylor, Wei Zhao, and Jihoon Kwon. Signal-based hallucination detection in large language models. *arXiv preprint arXiv:2310.01234*, 2023. URL https://arxiv.org/abs/2310.01234.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Edouard Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023b.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Cuad: An expert-annotated nlp dataset for legal contract review. https://arxiv.org/abs/2103.06268, 2021a. arXiv:2103.06268 [cs.CL].

Nouha Dziri, Ehsan Kamalloo, Kory Mathewson, Osmar Zaiane, and Adam Trischler. Measuring and inducing hallucinations in neural text generation. In *Proceedings of the 2023 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pages 4177–4192, 2023. URL https://aclanthology.org/2023.naacl-main.235.

Meta AI. Llama 3 model card. https://ai.meta.com/llama, 2024. Accessed May 2025.

Alibaba DAMO Academy. Qwen: Large language models by alibaba cloud. https://github.com/QwenLM/Qwen, 2023. Accessed May 2025.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Cuad: The contract understanding atticus dataset. https://huggingface.co/datasets/cuad, 2021b. Accessed May 2025.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Steven Basart, Sampat Arora, Andy Zou, and Dawn Song. Cuad: Contract understanding atticus dataset. https://huggingface.co/datasets/cuad, 2021c. Accessed: 2025-07-01.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Baptiste Rozière, Naman Goyal, Matthias Gallé, Sharan Hosseini, Philipp Schmid, et al. Llama 3: Open foundation and instruction-tuned language models. https://ai.meta.com/llama/, 2024. Accessed: 2025-07-01.

Joshua Maynez, Shashi Narayan, Bernd Bohnet, and Ryan McDonald. On faithfulness and factuality in abstractive summarization, 2020. URL https://arxiv.org/abs/2005.00661.