Hallucination Detection In Large Language Models (LLMs)

Karthikeyan S (3122225001056), Krithika C (3122225001066), and Lavanya Vasudevan (3122225001067)

Department of Computer Science and Engineering Sri Sivasubramaniya Nadar College of Engineering, Kalavakkam, Chennai

Abstract. The surge in applications of Large Language Models (LLMs) [1] has prompted concerns about the generation of misleading or fabricated information, known as hallucinations. Therefore, detecting hallucinations has become critical to maintaining trust in LLM-generated content. Supervised learning [2] offers a robust approach to hallucination detection by training classifiers on labeled datasets to distinguish between factual and hallucinated outputs. This approach enhances the reliability and interpretability of AI systems in domains like healthcare and computer vision. Experimental results demonstrate improved accuracy and robustness, making it a promising solution for real-world applications requiring high precision.

Keywords: Large Language Models (LLMs) · Artificial Intelligence (AI) · Generative Pre-trained Transformer (GPT) · Bidirectional Encoder Representations from Transformers (BERT) · Term Frequency-Inverse Document Frequency (TF-IDF) · Adaptive Synthetic Sampling (ADASYN)

1 Introduction

In today's rapidly evolving landscape of machine learning, large language models (LLMs) have emerged as transformative forces shaping various applications such as question answering, summarization, translation, and content generation. However, a major challenge persists: hallucinations—fabricated or factually incorrect outputs that undermine trust in LLM-generated content. These hallucinations often arise when the desired information is absent from the model's training data or when reasoning within retrieved knowledge fails. The impact of hallucinations can be especially detrimental in critical domains like healthcare, law, and scientific research, where accuracy and reliability are paramount. To address this issue, supervised learning offers a robust approach to hallucination detection. By leveraging labeled datasets, classifiers can be trained to distinguish between factual and hallucinated outputs, enabling the identification of inaccuracies in LLM-generated responses. This approach enhances the reliability and transparency of AI systems while building user trust. Furthermore, it is a vital step toward ensuring that LLMs deliver precise and dependable information. In addition, techniques like Retrieval-Augmented Generation (RAG) [3] attempt to

minimize hallucinations by retrieving relevant information from external knowledge bases. While effective, RAG systems are not immune to hallucinations due to errors in retrieval or reasoning. As a result, detecting hallucinations in LLM outputs remains a crucial area of focus. This project explores the use of supervised learning to develop an effective hallucination detection framework, providing a pathway to creating robust and trustworthy AI systems for real-world applications.

1.1 Problem Statement

This report focuses on detecting hallucinations in Large Language Models (LLMs) using supervised learning, ensuring accurate and trustworthy outputs for real-world applications. Hallucination detection in Large Language Models (LLMs) [4] has profound applications in critical domains where accuracy, reliability, and trust are of utmost importance. In healthcare, it ensures that AI systems provide accurate medical diagnoses, treatment recommendations, and documentation, preventing life-threatening errors. In the legal domain, it safeguards the integrity of AI-generated contracts, legal opinions, and compliance documents, avoiding costly misinterpretations or inaccuracies. In finance, hallucination detection helps ensure the reliability of market predictions, risk assessments, and automated reports, reducing the chances of decisions based on fabricated insights. Similarly, in education, it ensures factual accuracy in AI-generated learning materials, enhancing trust in AI tutors. Beyond these, applications in autonomous systems, such as driverless vehicles and robotic assistants, rely on hallucination detection to prevent misjudgments that could result in harm. As AI continues to evolve, hallucination detection will play a pivotal role in ensuring safe and responsible deployment in real-world, high-stakes scenarios.

2 Literature Survey

This section explores the existing literature surrounding the critical issue of hallucinations in Large Language Models (LLMs) and the advancements made to address this phenomenon.

2.1 Large Language Models (LLMs)

Large Language Models (LLMs) are advanced AI systems that use transformer-based architectures, such as GPT [5] and BERT [6], to process and generate human-like text. Trained on vast datasets, these models encode contextual information through self-attention mechanisms, enabling them to perform tasks like question answering, translation, and summarization with high accuracy. Their pre-trained knowledge allows for minimal fine-tuning on specific applications, making them versatile across domains. For instance, GPT-3 [7] demonstrated exceptional zero-shot and few-shot learning capabilities, setting a benchmark for natural language understanding and generation.Their ability to understand and

generate contextually relevant text has revolutionized various industries, including healthcare, education, and customer service, paving the way for innovative applications and advancements in artificial intelligence.

2.2 Hallucinations in Large Language Models (LLMs)

One of the major downsides of Large Language Models (LLMs) is their tendency to generate hallucinations—outputs that are factually incorrect, fabricated, or irrelevant to the given input. While these issues may seem trivial in low-stakes scenarios, they pose significant risks in critical domains. For instance, in healthcare, hallucinated outputs can result in incorrect medical advice, misdiagnosis, or dangerous treatment suggestions, jeopardizing patient safety. In legal applications, hallucinations could lead to the generation of misleading case summaries or incorrect interpretations of legal statutes, potentially impacting judicial decisions. Similarly, in finance, the generation of inaccurate reports or fabricated data can result in financial losses and damage to institutional credibility. These risks are compounded by the confidence with which LLMs present their outputs, making it difficult for users to discern the validity of the information. As such, the presence of hallucinations underscores the urgent need for robust detection mechanisms, enhanced verification processes, and stringent evaluation frameworks to ensure reliability in high-stakes domains.

2.3 Retrieval-Augmented Generation (RAG)

Retrieval-Augmented Generation (RAG) is a promising framework designed to improve the factual accuracy of Large Language Models (LLMs) by combining them with external knowledge retrieval systems. In RAG, relevant information is fetched from a knowledge base or database using techniques like sparse or dense retrieval and is then fed into the LLM alongside the input query to generate more grounded outputs. While RAG addresses some issues of hallucinations by anchoring responses to external data, it does not entirely eliminate the problem. The effectiveness of RAG depends on the quality of the retrieved data. If the retrieval system provides incomplete, irrelevant, or outdated information, hallucinations can still occur as the LLM extrapolates beyond the given input. Even with accurate retrievals, hallucinations may arise due to misinterpretation or reasoning errors, particularly for tasks requiring complex synthesis. Thus, while RAG helps ground responses, it cannot fully eliminate hallucinations, emphasizing the need for complementary detection of hallucinations.

2.4 Supervised Learning for Hallucination Detection

Supervised learning is a key approach for detecting hallucinations in the outputs of Large Language Models (LLMs), relying on labeled datasets to train classifiers that differentiate between factual and hallucinated content. By analyzing annotated examples, supervised models can identify patterns indicative

of hallucinations, ensuring higher accuracy in distinguishing reliable outputs. Techniques such as semantic similarity analysis and contextual understanding have been integrated to enhance detection capabilities, particularly in tasks like summarization, translation, and question answering. This approach has been instrumental in improving the reliability of AI systems across critical domains, ensuring their outputs are consistent and trustworthy.

3 Proposed System

This section outlines the workflow and components of the proposed system for detecting hallucinations in responses generated by Large Language Models (LLMs).

3.1 System Architecture Diagram

1. **Loading the Dataset**
   - The hallucination dataset is loaded into the environment, consisting of labeled data with columns: `Id`, `Prompt`, `Answer`, and `Target`.
2. **Exploratory Data Analysis (EDA)**
   - Dataset Information: Inspect the dataset structure, columns, and features.
   - Descriptive Statistics: Generate statistical summaries to identify trends, distributions, and anomalies.
   - Word Cloud: Create visual word clouds to identify the most frequently occurring words in hallucinated ("1") and factual ("0") responses.
   - Text Length Analysis: Analyze the length of responses (word or character count), visualizing trends between the two classes ("1" and "0").
   - Class Imbalance: Examine the imbalance, where non-hallucinated responses ("0") dominate the dataset.
3. **Preprocessing**
   - Lowercasing: Convert all text to lowercase.
   - Removing Special Characters: Remove punctuation, numbers, and special symbols, keeping only alphabetic characters and spaces.
   - Tokenization: Split the text into tokens.
   - Stopword Removal: Remove common words (e.g., "the," "is") .
   - Lemmatization: Convert words to their base form (e.g., "running" $\rightarrow$ "run").
   - Joining Tokens: Recombine processed tokens into strings for vectorization.
4. **Vectorization**
   - **TF-IDF:** Term Frequency-Inverse Document Frequency (TF-IDF) is a statistical technique that converts text into numerical features by reflecting the importance of a word in a document relative to the entire corpus.
   - **BERT Vectorization:** BERT embeddings use a pre-trained transformer-based model to generate contextualized vector representations of text, capturing semantic and syntactic nuances.

5. **Dimensionality Reduction Techniques**
   - The feature vectors are reduced dimensionally using techniques like
     - Principal Component Analysis (PCA)
     - Linear Discriminant Analysis (LDA)
     - Truncated Singular Value Decomposition (TSVD)
     - Select KBest
6. **Machine Learning Models**
   - Train and evaluate supervised learning models on both TF-IDF and BERT features:
     - Logistic Regression
     - Naive Bayes
     - Random Forest
     - K Nearest Neighbours Classifier (KNN)
7. **Ensemble approaches**
   - Train and evaluate supervised ensemble models on both TF-IDF and BERT features:
     - Adaptive Boosting (AdaBoost)
     - Extreme Gradient Boosting (XGBoost)
     - Gradient Boosting Classifier
8. **Hyper-parameter Optimization**
   - Hyper-parameter tuning is performed using techniques like Grid Search, Random Search to systematically find the optimal hyper-parameter values.
9. **Evaluation Metrics**
   - Evaluate model performance using:
     - Accuracy
     - Precision, Recall, F1-Score
     - Confusion Matrix
10. **Visualization of Results**
    - Plot ROC curves and confusion matrices to interpret model performance and dataset trends.

Figure [1] shows the architecture diagram of the system.

4 Implementation

4.1 Development Environment

The development environment for this machine learning task is set up using Python as the primary programming language due to its versatility and extensive library support for data science and machine learning. The implementation is carried out on Google Colab, a cloud-based platform that provides free access to GPUs and TPUs, enabling efficient computation and experimentation. Key libraries include pandas and numpy for data handling, matplotlib, seaborn, and wordcloud for visualization, nltk [8] for text preprocessing, torch and transformers for BERT embeddings, and scikit-learn for machine learning models and evaluation.This combination of tools and resources ensures a seamless workflow for data preprocessing, model development, and performance evaluation.

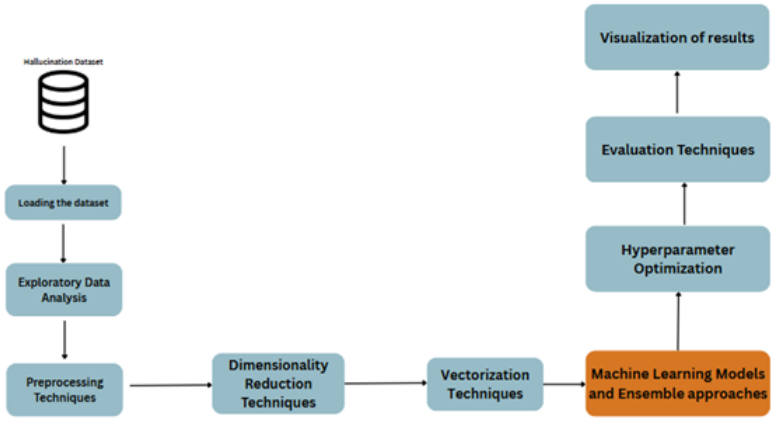

Fig. 1. System Architecture Diagram.

4.2 Dataset Description

The dataset utilized for hallucination detection, downloaded from a Kaggle repository, consists of 16,687 entries and is organized into four main columns: Id, Prompt, Answer, and Target. The Id column serves as a unique identifier for each record, ensuring traceability and consistency. The Prompt column contains the input queries or instructions provided to the language model, while the Answer column stores the corresponding responses generated by the model. The Target column acts as the ground truth label, indicating whether the generated response is factual (0) or hallucinated (1). This dataset is divided into two distinct classes: “1” (hallucinated responses) and “0” (accurate responses). Out of the total records, approximately 15,000 are labeled as "0" and only around 1,000 are labeled as "1" demonstrating a significant class imbalance [Fig 2].

4.3 Balancing the Dataset for Fair Representation

To address the issue of class imbalance in the hallucination dataset, SMOTE (Synthetic Minority Oversampling Technique) is applied to oversample the minority class. Initially, the textual data from the Prompt and Answer columns is combined into a single feature (X), while the Target column serves as the label (y). SMOTE is usedgenerate synthetic samples for the minority class, ensuring a more balanced class distribution. After resampling, the new distribution of labels is analyzed and a bar graph is created to visualize the improved balance between classes. However, for high-dimensional embeddings, such as BERT embeddings, SMOTE is less effective because of its higher dimensionality, which makes generating meaningful synthetic samples challenging. Therefore, ADASYN (Adaptive Synthetic Sampling) is utilized with BERT embeddings to duplicate existing

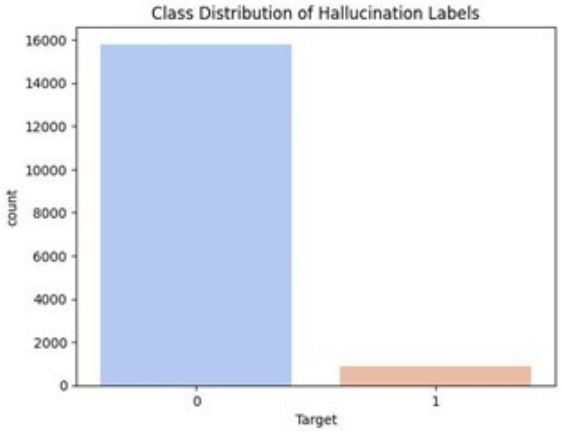

Fig. 2. Class Distribution of Hallucination Labels.

minority-class samples, achieving balance without introducing synthetic noise or complexity.

Figure [3] shows the class distrbution after balancing the dataset.

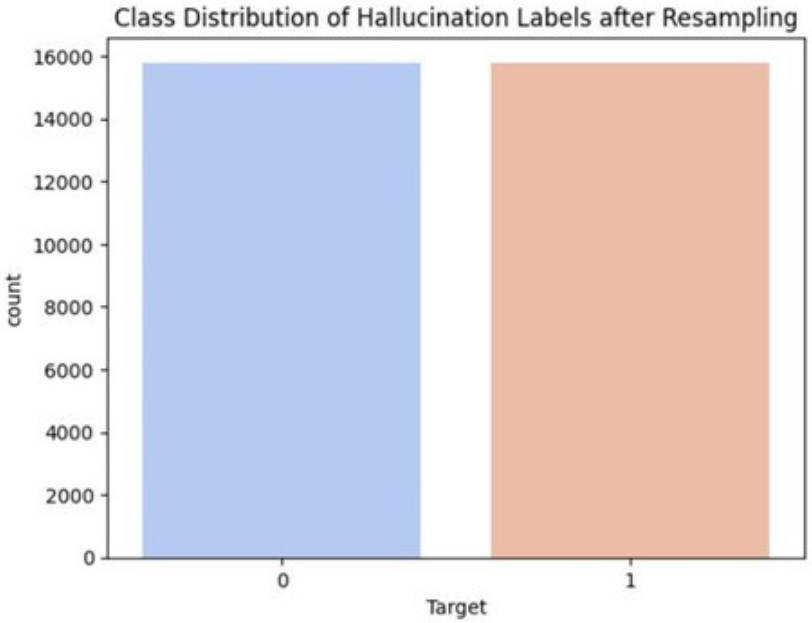

Fig. 3. Class Distribution of Hallucination Labels after balancing.

4.4 Exploratory Data Analysis (EDA)

To understand the characteristics of the textual responses in the dataset, exploratory analysis was conducted on two aspects: word frequency and response length.

Word Clouds: Word clouds were generated to visualize the most frequently occurring words in hallucinated ("No") and factual ("Yes") responses. As shown in Figure 4, there are clear differences in word usage patterns between the two classes.

Fig. 4. Word Cloud for Factual and Hallucinated Responses.

Response Length Distribution: The number of words in each response was calculated to analyze verbosity. Figure 5 shows the distribution of the response lengths.

4.5 Preprocessing

Before feeding the textual data into machine learning models, a series of preprocessing steps are applied to ensure consistency and improve the quality of features extracted from the text. First, all text was converted to lowercase to maintain uniformity and avoid case sensitivity issues during analysis. Special characters, punctuation marks, and numbers were then removed, leaving behind only alphabetic characters and spaces to focus on meaningful textual content. The cleaned text was subsequently tokenized, breaking it down into individual words or tokens for easier manipulation. Common stopwords such as “the,” “is,” and “and” were filtered out predefined stopword list to highlight more significant words. Lemmatization was then employed to reduce words to their base or dictionary form, for example, converting “running” to “run,” which helps minimize redundancy and reduce dimensionality. Finally, the processed tokens were rejoined into coherent strings, preparing them for the vectorization techniques that followed.

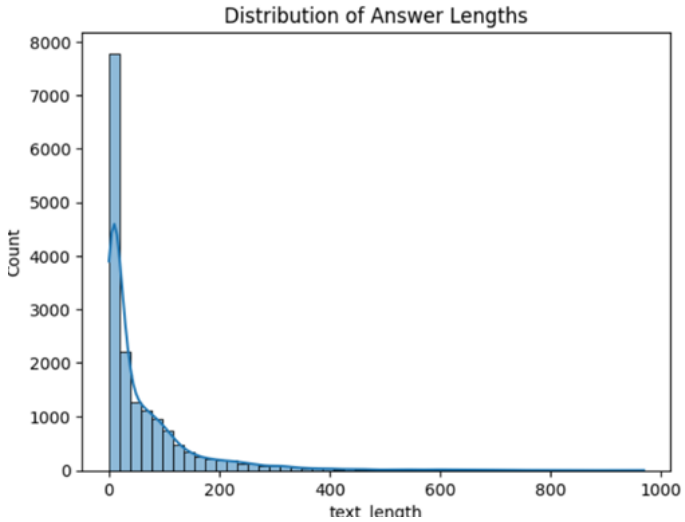

Fig. 5. Distribution of Response Lengths for Hallucinated and Factual Responses.

4.6 Vectorization Techniques

To convert textual data into a format suitable for machine learning models, vectorization techniques are employed. This section explores two prominent methods used in this study: TF-IDF vectorization and BERT embeddings.

TF-IDF Vectorization (Term Frequency-Inverse Document Frequency) is a statistical method used to represent text data by assigning a weight to each term based on its importance in a document relative to a corpus. Term Frequency (TF) measures how frequently a word appears in a specific document, indicating its relevance within that context. However, some words might appear frequently across many documents and may not carry significant meaning—this is where Inverse Document Frequency (IDF) plays a role. IDF reduces the weight of such commonly occurring terms, highlighting words that are more unique to each document. The TF-IDF score, calculated as the product of TF and IDF, strikes a balance between local and global term importance. This results in a sparse matrix where rows represent documents and columns correspond to terms, making it a suitable input format for traditional machine learning algorithms.

BERT Embeddings, on the other hand, leverage a deep learning-based approach using the Bidirectional Encoder Representations from Transformers (BERT) model. BERT is a transformer-based language model pre-trained on large corpora to understand the context of words based on surrounding text. In this approach, the tokenizer first processes the input text into tokens compatible with BERT's vocabulary. The processed text is then passed through the model to obtain contextualized embeddings from the last hidden layer. These embeddings capture rich semantic and syntactic information, allowing models to understand the nuances of language more effectively. The resulting dense vector representa-

tions are applied to each text sample and used as input features for downstream classification tasks.

4.7 Dimensionality Reduction Techniques

This section outlines key dimensionality reduction techniques such as PCA, LDA, Truncated SVD, and SelectKBest, which are employed to simplify high-dimensional text data while preserving relevant information for improved model performance.

Principal Component Analysis (PCA)

Principal Component Analysis (PCA) [9] is a widely used dimensionality reduction technique that transforms high-dimensional data into a lower-dimensional space while preserving as much variance as possible. It works by computing the principal components—orthogonal vectors that capture the directions of maximum variance in the data. These components are obtained by eigenvalue decomposition of the covariance matrix. PCA is primarily used for feature extraction, noise reduction, and visualization but does not consider class labels, making it an unsupervised method.

One limitation of PCA is that it assumes linear relationships in data and may not perform well with complex, non-linear patterns. It is also sensitive to data scaling, so proper normalization is often necessary. Despite these limitations, PCA remains a powerful tool for improving computational efficiency and reducing redundancy in high-dimensional datasets.

Linear Discriminant Analysis (LDA)

Linear Discriminant Analysis (LDA) [10] is a supervised dimensionality reduction technique designed to maximize class separability. Unlike PCA, which focuses on variance, LDA aims to find a linear combination of features that best distinguishes between different classes. It does so by computing the within-class and between-class scatter matrices and selecting directions that maximize the ratio of between-class variance to within-class variance.

LDA is particularly useful for classification tasks where reducing dimensionality while maintaining class discriminability is crucial. However, it assumes normally distributed data and equal class covariances, which may not always hold in real-world scenarios. Despite this, LDA is widely used in applications like face recognition and text classification due to its effectiveness in feature reduction while preserving discriminative information.

Truncated Singular Value Decomposition (TSVD)

Truncated Singular Value Decomposition (TSVD) [11] is a matrix factorization technique that reduces the dimensionality of a dataset by decomposing it into three matrices: U, Σ, and V. Unlike PCA, which centers the data before decomposition, TSVD works directly on the original data matrix, making it suitable for sparse and high-dimensional data such as text or large-scale machine learning applications. By retaining only the top k singular values, TSVD approximates the original matrix while removing less significant components.

TSVD is commonly used in Latent Semantic Analysis (LSA) for natural language processing and document clustering. It is computationally efficient for

large datasets but does not explicitly preserve variance or class structure. As a result, it is often used in conjunction with other techniques to improve interpretability and performance in high-dimensional spaces.

SelectKBest for Feature Selection
SelectKBest is a feature selection technique that selects the top k features based on statistical tests, improving model performance by reducing noise and irrelevant data. It evaluates each feature's relevance to the target variable using methods like chi-square (χ^2) for classification or mutual information and f-regression for regression tasks. Unlike dimensionality reduction techniques like TSVD or PCA, which transform features, `SelectKBest` retains original features, making it interpretable and useful for models sensitive to feature importance. Choosing an appropriate k is crucial, as too few features may lead to underfitting, while too many may introduce unnecessary complexity.

Limitations of PCA, LDA, and SVD in Text Embeddings
PCA and LDA are not suitable for TF-IDF because PCA requires dense, mean-centered data, while TF-IDF is sparse and non-negative, making Truncated SVD a better alternative. LDA, being a supervised technique, is also unsuitable since TF-IDF is often used in unsupervised tasks like document clustering.

Similarly, while SVD can be applied to BERT embeddings, it is not ideal because SVD captures only linear patterns, whereas BERT embeddings are highly contextual and non-linear. Instead, PCA, LDA, or autoencoders are better choices for reducing BERT dimensions while preserving semantic meaning.

4.8 Machine Learning Models

This section outlines the machine learning models employed for the classification of hallucinated and factual responses. A variety of supervised learning algorithms were explored to determine which performs best under different vectorization techniques. Each model was trained using both TF-IDF and BERT embeddings, allowing for a comparative analysis of their effectiveness in identifying hallucinations in text data.

Naive Bayes: Naive Bayes is a probabilistic classifier based on Bayes' Theorem with the assumption of independence between features. It is particularly effective for text classification tasks due to its simplicity and speed. Despite its naive assumption, it often performs surprisingly well in high-dimensional spaces.

Logistic Regression: Logistic Regression is a linear model used for binary classification problems. It estimates the probability of a class label using the logistic (sigmoid) function. This model is interpretable and works well when the classes are linearly separable.

Random Forest: Random Forest is an ensemble learning method that builds multiple decision trees and merges their results. It improves classification performance by reducing overfitting compared to individual decision trees. Random Forests are robust, handle nonlinear data well, and are effective for imbalanced datasets.

K-Nearest Neighbors (KNN): KNN is a non-parametric, instance-based learning algorithm that classifies data points based on the majority label of their

nearest neighbors. It relies on distance metrics like Euclidean distance to measure similarity between data points. KNN is simple, effective for small datasets, and performs well when the decision boundary is not linear.

4.9 Ensemble approaches

This section describes about the feasibility and application of ensemble approaches

Feasibility of Ensemble approaches

Ensemble models present a compelling approach for hallucination detection by combining the strengths of multiple classifiers to enhance robustness and generalization. Methods such as bagging and boosting effectively reduce variance and bias, which is crucial in detecting subtle inaccuracies or fabricated content generated by language models. Techniques like Random Forest (bagging) and Gradient Boosting (boosting) enable the capture of complex patterns in linguistic features, improving detection accuracy over individual models. Additionally, ensemble approaches are well-suited for integrating diverse feature representations—such as TF-IDF and contextual BERT embeddings—allowing the model to better generalize across varying instances of hallucination. Though they require more computational resources, the improved reliability and performance make ensemble methods a valuable addition to hallucination detection pipelines.

AdaBoost (Adaptive Boosting)

AdaBoost is an ensemble technique that combines multiple weak learners sequentially, with each learner focusing more on the instances that were previously misclassified. It effectively reduces bias and is well-suited for clean, structured datasets, progressively improving performance by emphasizing difficult cases. ensemble methods a valuable addition to hallucination detection pipelines.

Gradient Boosting Classifier

Gradient Boosting builds an ensemble of decision trees in a forward, stage-wise fashion by minimizing a chosen loss function. It balances bias and variance, and its flexibility in model tuning makes it ideal for capturing complex relationships in data, often outperforming traditional models in accuracy and generalization. ensemble methods a valuable addition to hallucination detection pipelines.

XGBoost (Extreme Gradient Boosting)

XGBoost is a highly efficient and scalable boosting framework that builds models in a sequential manner using gradient descent optimization. It incorporates regularization to prevent overfitting and is particularly effective for structured data, offering strong performance in a wide range of classification tasks. ensemble methods a valuable addition to hallucination detection pipelines.

Deploying the code in github To ensure effective version control and support collaborative development, the complete project codebase was deployed to a

GitHub repository. This process involved the exploration and application of several essential Git commands, which strengthened familiarity with version control workflows and best practices. The key Git operations performed include:

- **git init**: Initialized a local Git repository for the project.
- **git add .**: Staged all files in the working directory for commit.
- **git commit -m "Initial commit"**: Created a checkpoint to capture the current state of the project with a descriptive message.
- **git remote add origin <repository_url>**: Connected the local repository to the remote GitHub repository.
- **git push -u origin main**: Pushed the local commits to the main branch of the remote repository.
- **git status**, **git log**, and **git diff**: Monitored the repository status, commit history, and file-level changes.
- **git branch** and **git checkout**: Created and switched between branches to manage feature development and experimentation.

4.10 Hyper-parameter Optimization

This section outlines the two search techniques employed for hyper-parameter optimization.

Grid Search

Grid Search [12] is an exhaustive search method where a predefined set of hyperparameter values is specified, and the model is trained and evaluated for every possible combination. This method ensures optimal tuning but can be computationally expensive, especially for large hyperparameter spaces.

Random Search

Random Search [12], instead of evaluating all possible combinations, randomly selects hyperparameter values from a given range and tests a subset of configurations. While it may not guarantee finding the absolute best combination, it is often moreefficient and can discover near-optimal hyperparameters faster, especially in high-dimensional spaces.

In practice, Random Search is preferred when dealing with large hyperparameter spaces due to its efficiency, while Grid Search is useful when computational resources are sufficient, and the search space is relatively small.

4.11 Evaluation Metrics

To assess the performance of the classification models, a variety of evaluation metrics were utilized. These metrics provide a comprehensive understanding of how well each model distinguishes between hallucinated and factual responses.

Confusion Matrix: The confusion matrix is a tabular representation of the actual versus predicted classifications. It consists of True Positives (TP), False Positives (FP), True Negatives (TN) and False Negatives (FN). It provides information on the types of errors made by the model.

Accuracy: Accuracy measures the proportion of correctly predicted instances from the total predictions. It provides a general idea of how well the model is performing. It is most reliable when the dataset is balanced.

Precision: Precision shows the proportion of positive predictions that were actually correct. It is useful when the cost of false positives is high. A higher precision indicates fewer irrelevant results.

Recall: Recall measures the proportion of actual positives that were correctly identified. It is important when the cost of missing positive instances is high. A higher recall indicates fewer missed relevant results.

F1-Score: The F1-Score balances both precision and recall in one metric. It is especially useful when dealing with imbalanced datasets. A higher F1-Score means better overall model performance across classes.

5 Results and Discussions

This section presents the performance outcomes of the implemented machine learning models on the hallucination detection dataset. The results are analyzed using various evaluation metrics to compare model effectiveness and identify strengths and weaknesses. Visualizations such as confusion matrices and classification performance plots further aid in understanding the models' behavior. Key observations and insights derived from the experiments are discussed in detail.

5.1 Model Performance Evaluation

The performance of different machine learning models was assessed using both training and testing datasets. Each model was evaluated based on various metrics including accuracy, precision, recall, and F1-score. The experiments were conducted using two popular vectorization techniques: TF-IDF and BERT embeddings. Tables 1 and 2 summarize the training and testing results respectively.

Table 1. Training Performance of Models with Different Vectorization Techniques

Model	Vectorization Technique	Accuracy	Precision	Recall	F1-Score
Logistic Regression	TF-IDF	0.77	0.83	0.69	0.75
Logistic Regression	BERT embeddings	0.76	0.75	0.77	0.76
Random Forest	TF-IDF	0.79	0.89	0.67	0.76
Random Forest	BERT embeddings	0.82	0.82	0.81	0.82
KNN	BERT	0.86	0.76	0.79	0.76

Logistic Regression performs decently with both TF-IDF (Accuracy: 0.77) and BERT embeddings (Accuracy: 0.75). The slightly lower performance with BERT embeddings suggests that Logistic Regression is better suited for simpler, sparse representations like TF-IDF, as it relies on linear relationships and performs best when features are independent and less context-driven. **Naive Bayes**,

Table 2. Testing Performance of Models with Different Vectorization Techniques

Model	Vectorization Technique	Accuracy	Precision	Recall	F1-Score
Logistic Regression	TF-IDF	0.77	0.95	0.95	0.95
Logistic Regression	BERT embeddings	0.75	0.83	0.83	0.83
Random Forest	TF-IDF	0.80	0.99	0.99	0.99
Random Forest	BERT embeddings	0.81	0.99	0.99	0.99
KNN	BERT	0.9210	0.9308	0.9224	0.9207
Naive Bayes	TF-IDF	0.94	0.95	0.95	0.95
Naïve Bayes	BERT embeddings	0.69	0.69	0.69	0.69

on the other hand, excels with TF-IDF (Accuracy: 0.94) due to its assumption of feature independence, which aligns well with the bag-of-words nature of TF-IDF. However, it struggles significantly with BERT embeddings (Accuracy: 0.69) as these dense, contextual vectors violate the core assumption of feature independence.

Random Forest performs consistently well with both TF-IDF (Accuracy: 0.80) and BERT embeddings (Accuracy: 0.81), showcasing its versatility. Its ensemble, tree-based structure allows it to model non-linear relationships and handle both sparse and dense feature spaces effectively. This highlights that models like Logistic Regression and Random Forest are better suited for handling complex embeddings like BERT, as they can learn interdependencies between features. In contrast, **Naive Bayes** lacks the capacity to utilize the rich contextual information embedded in BERT. This analysis underscores the importance of choosing models aligned with the complexity of the task and the nature of feature representations—especially for context-heavy tasks like hallucination detection.

5.2 Comparative Analysis of Models across Dimensionality Reduction

This section presents a comparative analysis of various dimensionality reduction techniques applied to different encoding methods and machine learning models. The experiments evaluate the impact of these techniques—such as PCA, LDA, Truncated SVD, and SelectKBest—on the performance of Logistic Regression, Random Forest, and K-Nearest Neighbors using both TF-IDF and BERT embeddings. Train and test accuracies are reported to highlight how dimensionality reduction influences model effectiveness, generalization, and overfitting tendencies.

Table 3. Train Accuracy - Logistic Regression

Model	Encoding Techniques	Dimensionality Reduction	Train Accuracy
Logistic Regression	TF-IDF	None	0.7743
Logistic Regression	TF-IDF	Select KBest	0.6388
Logistic Regression	TF-IDF	TSVD	0.7621
Logistic Regression	BERT	None	0.7565
Logistic Regression	BERT	PCA	0.7491
Logistic Regression	BERT	LDA	0.8062

Table 4. Test Accuracy - Logistic Regression

Model	Encoding Techniques	Dimensionality Reduction	Test Accuracy
Logistic Regression	TF-IDF	None	0.7642
Logistic Regression	TF-IDF	Select KBest	0.6429
Logistic Regression	TF-IDF	TSVD	0.7544
Logistic Regression	BERT	None	0.7455
Logistic Regression	BERT	PCA	0.7403
Logistic Regression	BERT	LDA	0.7946

Table 7. Train Accuracy - KNN

Model	Encoding Techniques	Dimensionality Reduction	Train Accuracy
KNN	TF-IDF	None	0.8700
KNN	TF-IDF	Select KBest	0.8678
KNN	TF-IDF	TSVD	0.9937
KNN	BERT	None	0.9568
KNN	BERT	PCA	0.9600
KNN	BERT	LDA	0.9590

Table 8. Test Accuracy - KNN

Model	Encoding Techniques	Dimensionality Reduction	Test Accuracy
KNN	TF-IDF	None	0.9016
KNN	TF-IDF	Select KBest	0.8251
KNN	TF-IDF	TSVD	0.9872
KNN	BERT	None	0.9210
KNN	BERT	PCA	0.9253
KNN	BERT	LDA	0.9224

Table 5. Train Accuracy - Random Forest

Model	Encoding Techniques	Dimensionality Reduction	Train Accuracy
Random Forest	TF-IDF	None	0.7905
Random Forest	TF-IDF	Select KBest	0.7501
Random Forest	TF-IDF	TSVD	0.7501
Random Forest	BERT	None	0.8126
Random Forest	BERT	PCA	0.8744
Random Forest	BERT	LDA	0.8258

Table 6. Test Accuracy - Random Forest

Model	Encoding Techniques	Dimensionality Reduction	Test Accuracy
Random Forest	TF-IDF	None	0.7972
Random Forest	TF-IDF	Select KBest	0.7506
Random Forest	TF-IDF	TSVD	0.8400
Random Forest	BERT	None	0.7936
Random Forest	BERT	PCA	0.8875
Random Forest	BERT	LDA	0.8113

5.3 Comparative Analysis of Models across hyper-parameter tuning using search techniques

This section presents a comparative analysis of different models after applying hyperparameter tuning techniques such as Grid Search and Random Search.

Table 9. Train Accuracy after Hyperparameter Tuning

Model	Encoding	Dimensionality Reduction	Search Technique	Train Accuracy
Logistic Regression	BERT	LDA	Grid Search	0.8108
Logistic Regression	BERT	LDA	Random Search	0.8111
Random Forest	BERT	PCA	Grid Search	0.9990
Random Forest	BERT	PCA	Random Search	0.9990
KNN	TF-IDF	TSVD	Grid Search	0.9990
KNN	TF-IDF	TSVD	Random Search	0.9990

Table 10. Test Accuracy after Hyperparameter Tuning

Model	Encoding	Dimensionality Reduction	Search Technique	Test Accuracy
Logistic Regression	BERT	LDA	Grid Search	0.7945
Logistic Regression	BERT	LDA	Random Search	0.7950
Random Forest	BERT	PCA	Grid Search	0.9990
Random Forest	BERT	PCA	Random Search	0.9988
KNN	TF-IDF	TSVD	Grid Search	0.8641
KNN	TF-IDF	TSVD	Random Search	0.8641

The accuracy variations across different models, encoding techniques, and dimensionality reduction methods can be attributed to how well each combination preserves relevant features for classification. **Logistic Regression (LR)** generally performs well with TF-IDF but suffers a drop in accuracy when using Select KBest, likely due to feature loss. TSVD, however, retains more variance, leading to only a slight accuracy reduction. Interestingly, BERT with LDA for LR achieves the highest accuracy (0.7946), possibly because LDA extracts class-discriminative features, making it more suitable than PCA, which primarily captures variance without focusing on class separability. For **Random Forest (RF)**, accuracy improves significantly with PCA on BERT embeddings (0.8875) and TSVD on TF-IDF (0.8400), indicating that reducing noise and redundant features enhances decision tree-based learning. **K-Nearest Neighbours** (KNN) benefits the most from dimensionality reduction, with TSVD on TF-IDF achieving 0.9872, likely because KNN is sensitive to high-dimensional data, and TSVD effectively reduces dimensionality while preserving essential structures. The impact of hyperparameter tuning is minor, with Random Search and Grid Search yielding similar results, except for RF+BERT+PCA, where Grid Search achieves near-perfect accuracy (0.9990), suggesting an optimal selection of hyperparameters significantly refines the model's performance.

5.4 Comparative Analysis of Ensemble Models across Dimensionality Reduction

Table 11. Train Accuracy of AdaBoost across Dimensionality Reduction Techniques

Model	Encoding Technique	Dimensionality Reduction Technique	Train Accuracy
AdaBoost	TF-IDF	None	0.8812
AdaBoost	TF-IDF	SelectKBest	0.7762
AdaBoost	TF-IDF	TSVD	0.8850
AdaBoost	BERT	None	0.8881
AdaBoost	BERT	PCA	0.8438
AdaBoost	BERT	LDA	0.8306

Table 12. Test Accuracy of AdaBoost across Dimensionality Reduction Techniques

Model	Encoding Technique	Dimensionality Reduction Technique	Test Accuracy
AdaBoost	TF-IDF	None	0.8748
AdaBoost	TF-IDF	SelectKBest	0.7896
AdaBoost	TF-IDF	TSVD	0.8760
AdaBoost	BERT	None	0.7619
AdaBoost	BERT	PCA	0.8398
AdaBoost	BERT	LDA	0.8108

Table 13. Train Accuracy of XGBoost across Dimensionality Reduction Techniques

Model	Encoding Technique	Dimensionality Reduction Technique	Test Accuracy
XGBoost	TF-IDF	None	0.9888
XGBoost	TF-IDF	SelectKBest	0.8621
XGBoost	TF-IDF	TSVD	0.9990
XGBoost	BERT	None	0.9998
XGBoost	BERT	PCA	0.9998
XGBoost	BERT	LDA	0.8313

Table 14. Test Accuracy of XGBoost across Dimensionality Reduction Techniques

Model	Encoding Technique	Dimensionality Reduction Technique	Test Accuracy
XGBoost	TF-IDF	None	0.9711
XGBoost	TF-IDF	SelectKBest	0.8452
XGBoost	TF-IDF	TSVD	0.9808
XGBoost	BERT	None	0.9825
XGBoost	BERT	PCA	0.9934
XGBoost	BERT	LDA	0.8198

Table 15. Train Accuracy of Gradient Boosting across Dimensionality Reduction Techniques

Model	Encoding Technique	Dimensionality Reduction Technique	Train Accuracy
Gradient Boosting	TF-IDF	None	0.8968
Gradient Boosting	TF-IDF	SelectKBest	0.8006
Gradient Boosting	TF-IDF	TSVD	0.9080
Gradient Boosting	BERT	None	0.8353
Gradient Boosting	BERT	PCA	0.9018
Gradient Boosting	BERT	LDA	0.8311

Table 16. Test Accuracy of Gradient Boosting across Dimensionality Reduction Techniques

Model	Encoding Technique	Dimensionality Reduction Technique	Test Accuracy
Gradient Boosting	TF-IDF	None	0.8922
Gradient Boosting	TF-IDF	SelectKBest	0.8018
Gradient Boosting	TF-IDF	TSVD	0.8847
Gradient Boosting	BERT	None	0.8169
Gradient Boosting	BERT	PCA	0.8884
Gradient Boosting	BERT	LDA	0.8195

5.5 Comparative Analysis of Ensemble Models across Hyper-parameter optimization

Table 17. Train Accuracy of ensemble models across hyper-parameter optimization Techniques

Model	Encoding	Dimensionality Reduction	Search Technique	Train Accuracy
AdaBoost	TF-IDF	TSVD	Grid Search	0.7466
AdaBoost	TF-IDF	TDVS	Random Search	0.7762
XGBoost	BERT	PCA	Grid Search	0.8999
XGBoost	BERT	PCA	Random Search	0.8999

Table 18. Test Accuracy of ensemble models across hyper-parameter optimization Techniques

Model	Encoding	Dimensionality Reduction	Search Technique	Train Accuracy
AdaBoost	TF-IDF	TSVD	Grid Search	0.7352
AdaBoost	TF-IDF	TDVS	Random Search	0.7352
XGBoost	BERT	PCA	Grid Search	0.8443
XGBoost	BERT	PCA	Random Search	0.8443

6 Impact of the project on human, societal, ethical and sustainable development

6.1 Impact on Humans

This project improves the reliability of AI-driven systems, ensuring that users receive factual and trustworthy information. This reduces the risk of misinformation, aiding individuals in making better-informed decisions in areas like education, healthcare, and everyday problem-solving.

6.2 Impact on Society

By minimizing hallucinations in language models, the project contributes to fostering societal trust in AI technologies. Reliable AI systems can support critical societal functions, such as governance, public policy, and disaster response, enabling more effective solutions for collective challenges.

6.3 Ethical Impact

The project promotes ethical AI practices by addressing the challenge of hallucinations, which can lead to harmful consequences when unchecked. By ensuring accurate outputs, it aligns with principles of transparency, accountability, and fairness, essential for the responsible use of AI.

6.4 Impact on Sustainable Development

Enhancing the accuracy of AI systems contributes to sustainable development by fostering innovation and enabling data-driven solutions to global challenges. Reliable AI applications can be leveraged in achieving goals like quality education, good health, and reduced inequalities, driving sustainable progress across sectors.

7 Conclusion and Future Work

7.1 Conclusion

This study provides a comprehensive framework for detecting hallucination in textual responses, addressing a critical challenge in ensuring the reliability of Large Language Models (LLMs). By employing preprocessing, resampling techniques, and rigorous evaluation metrics, the project emphasizes the importance of building balanced and robust models capable of distinguishing between factual and hallucinated content. The integration of vectorization techniques and systematic exploration of the dataset has illuminated the complexities involved in textual analysis, particularly in identifying patterns that differentiate hallucinated responses. This work has broader implications in ethical and societal contexts, as hallucination detection is vital for fostering trust in AI systems—especially in sensitive domains such as healthcare, law, and education. By reducing inaccuracies and enhancing accountability, this project contributes to developing more reliable and sustainable AI-driven solutions.

7.2 Future Work

Future advancements could focus on adapting hallucination detection models for domain-specific applications, particularly in high-stakes areas like healthcare and the legal field. In healthcare, ensuring the correctness of AI-generated content is critical to prevent the spread of incorrect medical information, which could have serious consequences. Fine-tuning the model using medical datasets could enhance its ability to identify hallucinations in clinical texts, thereby safeguarding patient safety. Likewise, in legal contexts, the model could be optimized to detect fabricated case references, misrepresented statutes, or misleading arguments, improving the reliability of AI-generated legal documents. Such targeted specialization would promote the ethical and responsible use of AI, aligning its capabilities with societal expectations and professional standards.

References

1. Chang, Y., Wang, X., Wang, J., Wu, Y., Yang, L., Zhu, K., Chen, H., Yi, X., Wang, C., Wang, Y. and Ye, W.: A survey on evaluation of large language models. *ACM Transactions on Intelligent Systems and Technology*, **15**(3), 1–45 (2024)
2. Cunningham, P., Cord, M. and Delany, S.J.: Supervised learning. In *Machine learning techniques for multimedia: case studies on organization and retrieval*, 21–49. Springer Berlin Heidelberg (2008)
3. Gupta, S., Ranjan, R. and Singh, S.N.: A Comprehensive Survey of Retrieval-Augmented Generation (RAG): Evolution, Current Landscape and Future Directions. *arXiv preprint arXiv:2410.12837* (2024)
4. Huang, L., Yu, W., Ma, W., Zhong, W., Feng, Z., Wang, H., Chen, Q., Peng, W., Feng, X., Qin, B. and Liu, T.: A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *ACM Transactions on Information Systems* (2024)
5. Yenduri, G., Ramalingam, M., Selvi, G.C., Supriya, Y., Srivastava, G., Maddikunta, P.K.R., Raj, G.D., Jhaveri, R.H., Prabadevi, B., Wang, W. and Vasilakos, A.V.: GPT (Generative Pre-trained Transformer) – a comprehensive review on enabling technologies, potential applications, emerging challenges, and future directions. *IEEE Access* (2024)
6. Ravichandiran, S.: Getting Started with Google BERT: Build and train state-of-the-art natural language processing models using BERT. *Packt Publishing Ltd.* (2021)
7. Kublik, S. and Saboo, S.: GPT-3. *O'Reilly Media, Inc.* (2022)
8. Hardeniya, N., Perkins, J., Chopra, D., Joshi, N. and Mathur, I.: Natural Language Processing: Python and NLTK. *Packt Publishing Ltd.* (2016)
9. Maćkiewicz, A. and Ratajczak, W.: Principal components analysis (PCA). *Computers & Geosciences*, **19**(3), 303–342 (1993)
10. Tharwat, A., Gaber, T., Ibrahim, A. and Hassanien, A.E.: Linear discriminant analysis: A detailed tutorial. *AI Communications*, **30**(2), 169–190 (2017)
11. Hansen, P.C.: Truncated singular value decomposition solutions to discrete ill-posed problems with ill-determined numerical rank. *SIAM Journal on Scientific and Statistical Computing*, **11**(3), 503–518 (1990)
12. Liashchynskyi, P. and Liashchynskyi, P.: Grid search, random search, genetic algorithm: a big comparison for NAS. *arXiv preprint* arXiv:1912.06059 (2019)