UHASSELT
KNOWLEDGE IN ACTION

Maastricht University

Faculty of Sciences
School for Information Technology
Master of Statistics and Data Science

Master's thesis

Are purpose-made models still relevant in languageprocessing?"

AARYAN KAUSHIK

Thesis presented in fulfillment of the requirements for the degree of Master of Statistics and Data Science, specialization Data Science

SUPERVISOR :
dr. Brecht VANDEVOORT

SUPERVISOR :
Dr. Karel KENENS

2023
2024

Transnational University Limburg is a unique collaboration of two universities in two countries: the University of Hasselt and Maastricht University.

UHASSELT
KNOWLEDGE IN ACTION

www.uhasselt.be
Universiteit Hasselt
Campus Hasselt:
Martelarenlaan 42 | 3500 Hasselt
Campus Diepenbeek:
Agoralaan Gebouw D | 3590 Diepenbeek

Maastricht University

Faculty of Sciences
School for Information Technology
Master of Statistics and Data Science

Master's thesis

Are purpose-made models still relevant in languageprocessing?"

AARYAN KAUSHIK

Thesis presented in fulfillment of the requirements for the degree of Master of Statistics and Data Science, specialization Data Science

SUPERVISOR :
dr. Brecht VANDEVOORT

SUPERVISOR :
Dr. Karel KENENS

Acknowledgments

I am deeply grateful to several people and organizations for their support during this thesis.

First, I want to thank my primary supervisor, Dr. Brecht Vandevoort, for his invaluable guidance and encouragement. His advice was essential to the success of this research.

I also appreciate Dr. Karel Kenens from Algorythym Group for his expertise and support throughout this project.

Thank you to the University of Hasselt for providing the resources and support necessary for this work.

I extend my heartfelt gratitude to my family and friends in Belgium and in India for their support and encouragement throughout this journey.

I am also very thankful to Jessa Mae Lastimoso for her continuous support and understanding during this time.

Lastly, I want to express my sincere thanks to my friend Daniel Recep Yildirim for his consistent help and support throughout this master's program.

Thank you all.

Abstract

This thesis explores the relevance and performance of various language processing models in sentiment analysis, with a focus on their application to IMDB movie reviews. It evaluates a range of models, including advanced large language models (LLMs) such as BERT, LLAMA2, and GPT-4, as well as traditional machine learning techniques like XGBoost. The study assesses these models using key performance metrics: accuracy, precision, recall, and F1-score. The results reveal that XGBoost achieves the highest accuracy and F1-score, outperforming other models in these metrics. GPT-4 also demonstrates superior performance with high accuracy and precision, while LLAMA2 and GPT-3.5-turbo show strong results but fall slightly behind XGBoost and GPT-4. BERT, while effective, does not surpass these models in any of the evaluated metrics. These findings underscore the continued relevance of both advanced and traditional models in sentiment analysis, suggesting that model selection should be guided by specific performance requirements and resource considerations.

Keywords: *Large Language Model (LLM), Sentiment Analysis, Embedding, Language Processing, BERT, LLAMA2, GPT-4, XGBoost, Performance Metrics*

Contents

List of Figures

List of Tables

1 Introduction

In recent years, the field of Natural Language Processing (NLP) has undergone transformative advancements driven by increases in computational power, data availability, and algorithmic innovations. At the forefront of this evolution are Large Language Models (LLMs) such as BERT [1] and GPT-4 [2], which have set new benchmarks in understanding and generating human language. These models leverage vast amounts of textual data and sophisticated neural network architectures to perform a variety of language-related tasks with unprecedented accuracy.

Sentiment analysis, a pivotal application of NLP, involves categorizing text into classes that reflect emotional tone, such as positive or negative [3]. This task is crucial across diverse domains, from business and social media to academic research. For instance, businesses use sentiment analysis to gauge customer feedback and refine their strategies, while researchers analyze social media data to understand public opinion on various issues [4].

A popular and influential dataset for sentiment analysis is the IMDB movie review dataset. This dataset consists of thousands of movie reviews labeled with sentiments, making it a valuable resource for evaluating and developing sentiment analysis models. The IMDB dataset is particularly useful due to its large size and diversity of opinions, which provide a robust testbed for assessing the performance of NLP techniques in real-world scenarios.

BERT, introduced by Google in 2018, represents a significant leap in NLP by employing a bidirectional approach to text processing. Unlike traditional models that process text in a unidirectional manner, BERT considers both preceding and following words, allowing for a more nuanced understanding of context [1]. This bidirectional context helps BERT achieve state-of-the-art results across various NLP tasks by capturing complex language nuances.

Similarly, GPT-4, developed by OpenAI, enhances generative modeling with its advanced architecture, offering improved capabilities for understanding and generating human-like text [2]. These advancements have catalyzed new approaches to fine-tuning and optimizing models for specific tasks, including sentiment analysis.

Fine-tuning, a technique that adapts pre-trained models to specific datasets or tasks, plays a crucial role in enhancing model performance [5]. In addition, prompt engineering—a method for crafting specific inputs to guide model responses—has emerged as an important tool for maximizing the effectiveness of LLMs [6].

This research focuses on evaluating and comparing various NLP techniques for binary sentiment analysis of IMDB movie reviews. The primary research question is: How do different NLP techniques, including prompt engineering on state-of-the-art Large Language Models (LLMs) and fine-tuning decoder-only and encoder-only models, compare in terms of accuracy and efficiency for binary sentiment analysis (negative and positive) of IMDB movie reviews? This question seeks to explore the relative performance of advanced NLP methods in accurately classifying the sentiment of text data.

By investigating these techniques, this study aims to provide valuable insights into the optimization of NLP models for sentiment analysis. These insights have implications for both academic research and practical applications, as understanding the strengths and limitations of different methods is essential for addressing real-world text classification challenges.

The structure of the study is as follows: Section 2 describes the datasets used, providing details on their characteristics and preprocessing. Section 3 outlines the methodology, including specific models such as BERT and LLAMA2, techniques like prompt engineering and traditional machine learning, and the evaluation metrics employed. In Section 4, the results of the experiments are presented and analyzed, with separate discussions for each model and technique. Section 5 delves into the discussion, interpreting the findings and their implications. Section 6 addresses the ethical considerations, societal relevance, and stakeholder awareness related to the use of NLP technologies. Finally, Section 7 concludes the thesis with a summary of key findings and Section 8 proposes directions for future research.

2 Description of the Datasets

The IMDB movie review dataset used in this thesis is publicly available and consists of CSV files and parquet files. The dataset contains thousands of reviews. Each review is labeled with sentiment information.

As an example of how the data is structured, Table 1 shows a snapshot of the dataset including sample reviews and their associated sentiments:

review	sentiment
I saw this film (it's English title is "Who's Singing Over There?") at the 1980 Montreal International Film Festival. It won raves then... and disappeared. A terrible shame. It is brilliant. Sublime, ridiculous, sad, and extremely funny. The script is a work of art. It's been 19 years and I've seen only a handful of comedies (or any other genre, for that matter) that can match its originality.	positive
This film could cure sleep disorders, thats how bad it is. The story dragged, and the bad guy is not that scary. You will not even see this one on TBS reruns. This film made me wonder about Chuck film choices. He work on a real dog with this one.	negative
I didn't know what to expect from the film. Well, now I know. This was a truly awful film. The screenplay, directing and acting were equally bad. The story was silly and stupid. The director could have made a smart and thought provoking film, but he didn't. I squirmed in my seat for the last half of the movie because it was so bad. Where was the focus to the film? Where was anything in this film? Christians should boycott this film instead of promoting it. It was shabbily done and a waste of my money. Do not see this film.	negative

Table 1: Sample movie reviews and their associated sentiments

The parquet files are similar to the CSV files but have an extra column that contains embeddings. These embeddings are fixed-size numerical vectors with 3,072 dimensions, capturing the semantic meaning of reviews rather than individual words.

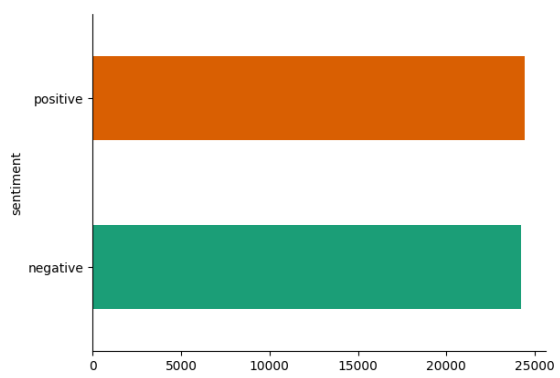
These embeddings were created using a model from OpenAI, which processes the text to generate a summary-like numerical representation. This representation helps the model understand the context and emotion behind the words in the review. Unlike older methods that might have treated each word separately, these embeddings allow the model to grasp the deeper meaning of the entire review, leading to better performance in tasks like sentiment analysis.

Including these embeddings in the dataset makes it more useful for NLP tasks. They transform text into a format that reflects their context and meaning, which leads to more accurate sentiment

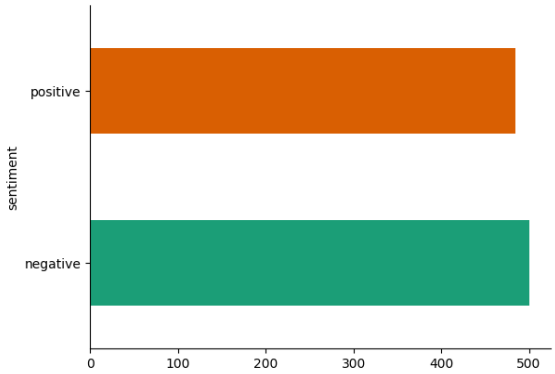
analysis. This is important for training models to detect subtle differences in sentiment and improve their performance on tasks like binary sentiment classification.

The dataset had some duplicate entries, meaning some reviews appeared more than once. To clean the data, duplicates were removed. First, reviews that were present in both the test and training sets were eliminated to ensure each dataset was unique. Then, duplicate reviews within the training set were also removed. After this cleanup, the Training Dataset contained 48,597 reviews, and the Test Dataset had 985 reviews. This process ensured the dataset was accurate and ready for analysis.

After cleaning the dataset, the sentiment labels in both the training and test datasets are nearly equal, with a balanced distribution between negative and positive categories. This balance ensures that the model is exposed to an equal representation of both sentiment classes during training and testing, which is essential for accurately assessing the model's performance across different sentiment categories. It helps prevent biases and ensures that the model learns to distinguish between negative and positive sentiments effectively. Figure 1 provides a visual representation of the balanced distribution of sentiment labels in both datasets, highlighting the representation of negative and positive sentiments.



(a) Distribution of sentiment labels in the training dataset



(b) Distribution of sentiment labels in the test dataset

Figure 1: Analysis of balanced sentiment labels in the training and test datasets

3 Methodology

This section outlines the methodologies employed to analyze sentiment in movie reviews, focusing on both encoder-only and decoder-only models. For encoder-based models, BERT was chosen due to its bidirectional context processing, which improves text understanding by considering surrounding words. BERT was trained and fine-tuned specifically for sentiment analysis tasks. For decoder-only models, LLAMA 2 was selected, leveraging its open-source nature and large-scale language capabilities. LLAMA 2 was adapted for sentiment classification by adding an additional layer that directly predicts sentiment outputs, rather than generating text. Prompt engineering techniques were also utilized with GPT models, such as GPT-4o and GPT-3.5-turbo, where specific prompts were crafted to guide the models in performing sentiment analysis tasks. Traditional machine learning techniques, like XGBoost, were used as a baseline for comparison. Evaluation metrics, including accuracy, precision, recall, and F1-score, were employed to assess model performance. Additionally, this section details the software tools used throughout the study. out the study are detailed in this section.

3.1 BERT

BERT stands for Bidirectional Encoder Representations from Transformers, a transformer-based pre-trained model developed by Google in 2018 that has revolutionized the field of natural language processing [1]. Unlike traditional models that process text data in a unidirectional manner (either left-to-right or right-to-left), BERT employs a bidirectional approach, considering both preceding and following words to understand the context of each word more accurately. Therefore, BERT can correctly understand the meaning of a word based on the words surrounding it in context. Moreover, BERT can learn new meanings of the same word that has a new quantitative meaning. This leads to it giving state-of-the-art results in every NLP task, including question answering, sentiment analysis, etc.

The BERT architecture is based on the Transformer model introduced by Vaswani et al. in 2017 [7]. The Transformer model includes mechanisms to evaluate the importance of each word in a sentence, which helps BERT generate better contextual word embeddings. BERT itself is pre-trained on a corpus of text, specifically from English Wikipedia and BookCorpus. This pre-training involves two main objectives: Masked Language Modeling (MLM), where some words in a sentence are masked and predicted based on the surrounding context, and Next Sentence Prediction (NSP), which involves predicting whether one sentence logically follows another [1].

In MLM (Masked Language Model), some of the words in a particular sentence are masked randomly, and the model is then trained to predict the masked words based on the context of the surrounding words. MLM helps BERT obtain a better understanding of language semantics and syntax. On the other hand, NSP (Next Sentence Prediction) is the task of predicting whether one sentence follows another in a given text pair. NSP helps BERT understand the relationship between different sentences and their context relevance [1].

In addition to its groundbreaking capabilities, BERT has inspired the development of more efficient

variants such as DistilBERT. DistilBERT, introduced by Sanh et al., is a more streamlined version of BERT [8]. This smaller, and faster variant "retains 97% of BERT's language understanding capabilities while being 60% faster and 40% smaller". This implies that this variant would be ideal for real-time processing and deployment on platforms with fewer computational resources. The efficiency is realized using a knowledge distillation process, where a smaller variant, DistilBERT, is trained to replicate the larger BERT. This way, BERT's high-performance language representations are retained and further made more scalable and accessible, thus able to be used in a wide array of NLP applications [8].

3.2 LLAMA 2: Open-Source Language Models

Llama 2, introduced by Meta in July 2023, is a series of advanced open-source language models designed to improve natural language understanding and generation. With sizes ranging from 7 billion to 70 billion parameters, Llama 2 is versatile enough for various applications, including conversational AI and content generation [9].

In this study, Llama 2 was not merely utilized as a generative model but was specifically adapted for sentiment analysis tasks by modifying its architecture. Typically, models like Llama 2 generate text sequences in an auto-regressive manner, predicting one word at a time based on the previous words. However, for the purpose of sentiment classification, an additional layer was introduced to the model. This layer is designed to predict the sentiment of a movie review directly, thereby converting Llama 2 from a generative model into a more efficient classifier that outputs sentiment predictions without generating full text sequences.

This modification allows Llama 2 to bypass the typical text generation process and focus on the sentiment prediction task, making it more suitable for the specific needs of this study. The adaptation leverages Parameter-Efficient Fine-Tuning (PEFT) techniques, which adjust only a subset of the model's parameters or add lightweight layers, thereby ensuring that the model remains computationally efficient while being tailored for sentiment analysis [10].

By combining the strengths of Llama 2's large-scale language capabilities with a tailored architecture for classification, this approach enhances the model's performance in sentiment analysis tasks while reducing computational overhead. This makes Llama 2 a powerful and flexible tool for both research and practical applications, providing targeted enhancements without the need for extensive retraining.

3.3 Prompt Engineering in GPT Models

Prompt engineering plays a critical role in natural language processing (NLP) and machine learning, focusing on the design of effective prompts to guide language models in generating accurate and contextually appropriate responses. This section evaluates the use of prompt engineering with two GPT models, GPT-4o and GPT-3.5-turbo, specifically for sentiment analysis tasks.

The key difference between this approach and the Llama 2 adaptation lies in the methodology:

While Llama 2 was modified to predict sentiment directly through an added classification layer, GPT models are leveraged in their generative form. Here, the focus is on designing effective prompts that instruct the model to perform sentiment analysis by interpreting the sentiment behind the provided text. The process involves carefully constructing and refining prompts to elicit the desired outcomes from the model, making it responsive to specific sentiment-related queries.

Prompt engineering involves crafting prompts that clearly direct the language model to perform specific tasks or generate particular types of responses. For instance, prompts need to be well-defined to elicit the desired outputs from models like GPT-3 and GPT-4, which rely on such prompts to function effectively [6, 11]. In our study, we designed prompts to assess whether movie reviews from IMDb are positive or negative, focusing on how variations in prompt wording influence model accuracy.

The process of designing these prompts includes providing precise instructions and relevant context to ensure that the model understands the task at hand. This requires a deep understanding of the model's capabilities and the specifics of the task. Effective prompts are often the result of iterative refinement, where initial prompts are tested and adjusted based on the model's performance [12]. For example, we refined our prompts by evaluating different phrasings and contexts to optimize the sentiment analysis performance of GPT-4o and GPT-3.5-turbo.

A key challenge in prompt engineering is achieving the right balance between specificity and flexibility. Prompts must be specific enough to guide the model toward accurate outputs, yet flexible enough to handle a variety of inputs and different types of movie reviews. This balance is crucial for optimizing the model's performance across different scenarios [13]. Our study focuses on how different prompt configurations affect the accuracy of sentiment classification, ensuring that prompts are adaptable to various review styles and contents.

Additionally, prompt engineering is essential for addressing issues of bias and fairness in language model outputs. By carefully crafting prompts and using diverse training data, developers can mitigate potential biases and ensure that the model provides equitable responses across different contexts [14, 15]. In our experiments, we considered how prompt design might influence the model's handling of nuanced sentiments and varied linguistic expressions, aiming to produce fair and balanced sentiment analysis results.

3.4 Traditional Machine Learning

For this study, Extreme Gradient Boosting (XGBoost) was selected as the baseline. XGBoost, introduced by Tianqi Chen and Carlos Guestrin in 2016, has become a highly effective tool for various prediction tasks, including classification and regression. Its exceptional performance, robustness, scalability, and efficiency have led to widespread adoption across both academic and industrial settings [16].

XGBoost operates by sequentially training a series of weak classifiers on subsets of the training data, and then combining them to form a strong predictive model. This process involves minimizing a loss

function, which reduces the discrepancy between the predicted and actual values, using a gradient descent approach to optimize the model parameters. This iterative approach enhances the model's ability to make accurate predictions [16].

A key advantage of XGBoost is its scalability. The algorithm efficiently processes large datasets through parallel processing, utilizing multiple threads or machines to handle big data problems that require rapid and efficient computation [16]. This scalability makes XGBoost particularly well-suited for complex datasets and large-scale machine learning tasks.

In addition to its predictive capabilities, XGBoost provides interpretability through feature importance scores. These scores offer insights into the significance of each feature in the model, helping users understand how predictions are made and the relative importance of different features in the decision-making process [17].

3.5 Evaluation Metrics

The performance of the various NLP techniques for binary sentiment analysis will be assessed using a comprehensive set of evaluation metrics. These metrics will provide insights into the accuracy, efficiency, and overall effectiveness of each method. The following evaluation metrics will be used in this study:

3.5.1 Accuracy

Accuracy measures the proportion of correctly classified reviews out of the total number of reviews. It is a straightforward metric that provides a general sense of how well the model is performing.

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}} \tag{1}$$

High accuracy indicates that the model is reliably distinguishing between positive and negative sentiments.

3.5.2 Precision, Recall, and F1-Score

Precision: Precision measures the proportion of true positive predictions out of all positive predictions made by the model. It indicates how many of the reviews labeled as positive are actually positive.

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \tag{2}$$

Recall: Recall, also known as sensitivity, measures the proportion of true positive predictions out of all actual positive reviews. It shows how well the model captures all positive instances.

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \tag{3}$$

F1-Score: The F1-Score is the harmonic mean of precision and recall, providing a balance between the two metrics. It is especially useful when the class distribution is imbalanced.

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{4}$$

- **True Positives (TP):** The number of positive reviews correctly classified as positive.
- **False Positives (FP):** The number of negative reviews incorrectly classified as positive.
- **True Negatives (TN):** The number of negative reviews correctly classified as negative.
- **False Negatives (FN):** The number of positive reviews incorrectly classified as negative.

These metrics are crucial for understanding the trade-offs between precision and recall, particularly in contexts where false positives or false negatives carry different costs.

3.5.3 Confusion Matrix

A confusion matrix is a table used to evaluate the performance of a classification algorithm. It displays the true positives, true negatives, false positives, and false negatives.

	Predicted Positive	**Predicted Negative**
Actual Positive	True Positives (TP)	False Negatives (FN)
Actual Negative	False Positives (FP)	True Negatives (TN)

Table 2: Confusion Matrix

The confusion matrix provides a detailed breakdown of model performance, allowing for the identification of specific areas where the model may be underperforming.

3.6 Software

The research utilized several key software tools and platforms, each serving a specific role in the development and evaluation of the models. Python was the primary programming language used for developing and running the models. For deep learning tasks, PyTorch was employed, providing robust support for model training and evaluation. The Transformers Library by Hugging Face was essential for accessing pre-trained models and performing various natural language processing (NLP) tasks. Scikit-Learn was used for implementing traditional machine learning algorithms, complementing the modern NLP approaches.

Data manipulation and numerical computations were handled using Pandas and NumPy. To access advanced models like GPT-4, the Azure OpenAI Service was utilized. High-performance computing resources were initially provided by the Flemish Supercomputer Center (VSC), but using VSC

presented several challenges. Training models on VSC proved to be time-consuming and complex, particularly with the integration and management of computational resources. It took almost a month to become proficient with VSC, and despite the effort, only BERT and XGBoost were successfully run there. Attempts to run LLAMA2 on VSC faced significant issues with GPU memory limitations, leading to frequent out-of-memory errors.

Due to these challenges, the decision was made to switch to Google Colab. Google Colab provided a more user-friendly environment and better handled the computational requirements of the models. It offered ease of use and efficient execution of the models, allowing for more effective experimentation and analysis.

Overall, while VSC provided necessary high-performance computing resources, the ease of use and operational efficiency of Google Colab significantly streamlined the research process, allowing for more productive and timely results.

4 Results

This section presents the results of the sentiment analysis using different models. It starts with the DistilBERT model, which was fine-tuned to evaluate its performance. Next, the results for the LLAMA2 model, which used Parameter-Efficient Fine-Tuning (PEFT) and LoRA, are discussed. It is crucial to highlight that LLAMA2 was modified from its original generative model design. Specifically, an additional layer was incorporated into LLAMA2 to enable direct sentiment classification, transforming it from a text-generating model to one that outputs binary sentiment labels. The effectiveness of this modification is a key focus. The effectiveness of prompt engineering is then explored by looking at the GPT-4o and GPT-3.5-turbo models. Finally, the XGBoost model is reviewed as a baseline for comparison. Performance is measured using accuracy, precision, recall, and F1-score to understand how well each model performed.

4.1 BERT

The DistilBERT model was trained and fine-tuned for 20 epochs, with early stopping triggered after 5 epochs due to no improvement in validation loss. The best model was saved and used for evaluation on the test set.

The training process resulted in a decrease in both training and validation loss over the epochs, indicating that the model was learning and improving its performance. However, after the second epoch, the validation loss started to increase, suggesting that the model was beginning to overfit the training data. This behavior is illustrated in **Figure 2**, which plots the training and validation losses over the epochs.

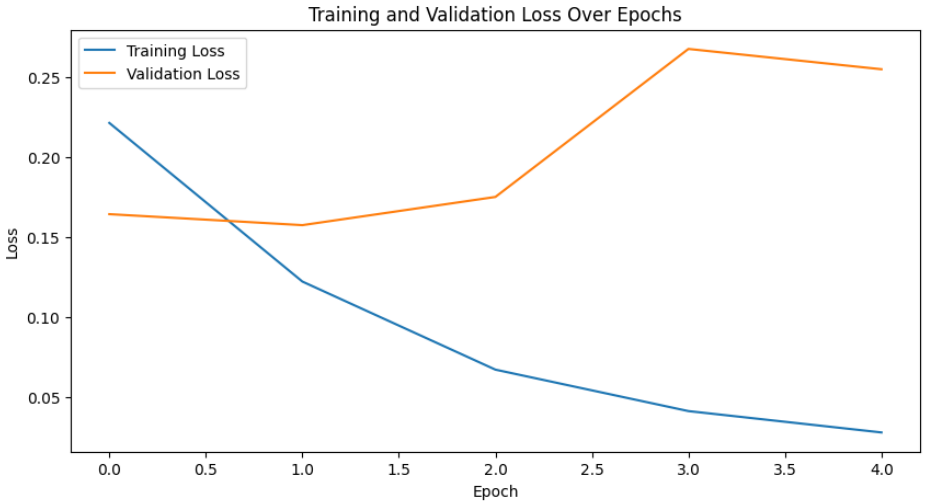


Figure 2: Training and Validation Loss Over Epochs

At the end of the second epoch, the model achieved a validation accuracy of **94.62%**. On the test set, it obtained an accuracy of **94.62%**, precision of **92.19%**, recall of **97.32%**, and an F1-score

of **94.68%**. These metrics are summarized in **Table 3**.

Table 3: Performance Metrics on Test Set

Metric	Value
Accuracy	94.62%
Precision	92.19%
Recall	97.32%
F1-Score	94.68%

The confusion matrix in **Figure 3** provides a detailed view of the model's performance. Out of 500 negative reviews, 460 were correctly classified, while 40 were misclassified as positive. Among 485 positive reviews, 472 were correctly identified, with 13 misclassified as negative. This indicates a slightly better performance in identifying positive reviews compared to negative ones.

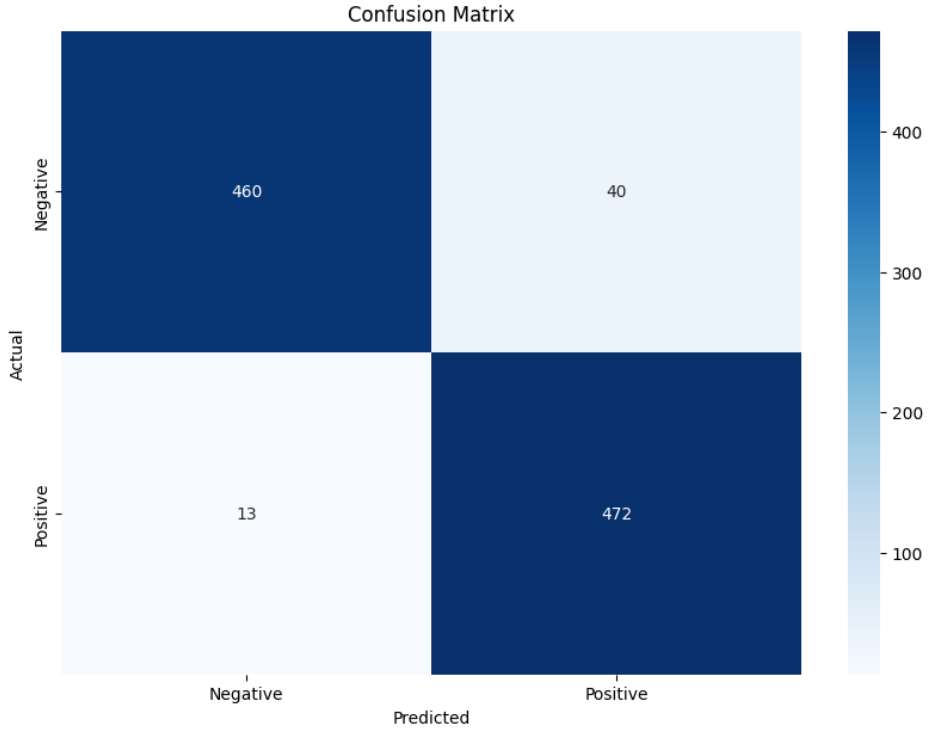


Figure 3: Confusion Matrix on Test Set for BERT

The model was trained with a learning rate of 2e-5, which is commonly used for fine-tuning BERT models as it provides a balance between stability and convergence speed [1]. A batch size of 16 was selected based on typical memory constraints and training stability considerations. Training was set for 20 epochs, but early stopping after 5 epochs without validation loss improvement was employed to prevent overfitting, a common practice in training deep learning models. Details of

the training parameters and their justifications are summarized in **Table 4**.

Table 4: Training Parameters and Their Justification

Parameter	Explanation
Learning Rate	2e-5, effective for fine-tuning BERT models
Batch Size	16, chosen for GPU memory constraints
Number of Epochs	20, with early stopping after 5 epochs
Early Stopping Patience	5 epochs, to prevent overfitting

4.2 LLAMA2

The LLAMA2 model, specifically the 7B variant, was fine-tuned using the Parameter-Efficient Fine-Tuning (PEFT) approach with Low-Rank Adaptation (LoRA). This approach was selected to efficiently adapt a large pre-trained model to the sentiment analysis task while managing computational resources effectively. Given the constraints of typical research environments, PEFT with LoRA is particularly useful for reducing the number of parameters that need to be updated during training.

The model used for this analysis was created using the `AutoModelForSequenceClassification` class from the `transformers` library by Hugging Face. This class helps to load pre-trained models that are ready for sequence classification tasks. For this analysis, the LLAMA2 7B variant was chosen, taking advantage of its extensive pre-training on various types of text. The `num_labels=2` parameter was set to adapt the model for binary classification, which is necessary for distinguishing between positive and negative sentiments. The `AutoModelForSequenceClassification` class adds a classification layer on top of the pre-trained model, enabling it to make predictions for each sentiment label.

The PEFT approach employed LoRA, configured with the following parameters:

- **Rank (r)**: The rank parameter of 8 was chosen to balance model capacity and computational efficiency. It allows the model to adapt effectively without significantly increasing the number of trainable parameters.
- **LoRA Alpha (α)**: Set to 16, this parameter scales the contribution of the low-rank matrices to the model's predictions. A value of 16 was selected to provide a strong adaptation effect without overwhelming the pre-trained model weights.
- **LoRA Dropout**: Set to 0.1, dropout is used to regularize the training process and reduce the risk of overfitting. This level of dropout provides a balance between preventing overfitting and maintaining model performance.

The LLAMA2 model was fine-tuned with an NVIDIA A100 GPU, provided by Google Colab. The training process took approximately 5-6 hours for 1 epoch, with a batch size of 4 chosen to fit within GPU memory constraints. A learning rate of 2e-5 was used, which is a common choice for

fine-tuning large pre-trained models, balancing convergence and stability. The maximum sequence length was set to 512 tokens to effectively handle the length of most input reviews. Padding was applied to all sequences to ensure consistent input sizes, which is crucial for model performance.

The performance of LLAMA2 on the test set is summarized in **Table 5**.

Table 5: Performance Metrics on Test Set for LLAMA2

Metric	**Value**
Accuracy	95.84%
Precision	93.70%
Recall	98.14%
F1-Score	95.87%

The confusion matrix for the test set is illustrated in **Figure 4**. This matrix provides a detailed view of the model's classification performance, highlighting the true and false classifications.

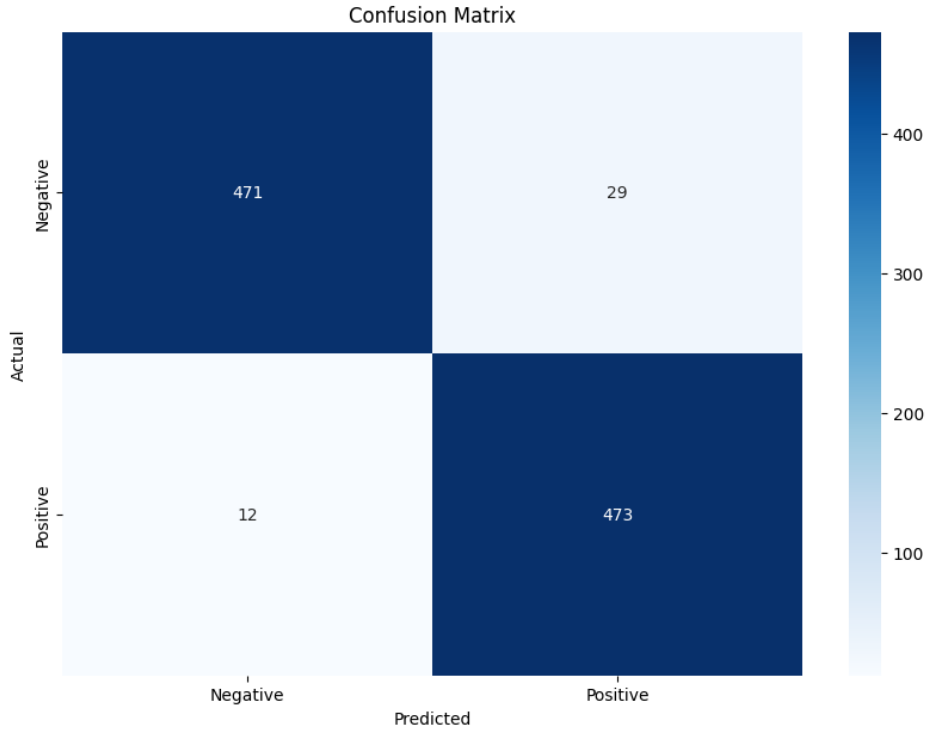


Figure 4: Confusion Matrix on Test Set for LLAMA2

Figure 4 and **Table 5** demonstrates high performance on the test set, with an accuracy of 95.84% and a balanced precision, recall, and F1-score. The confusion matrix indicates that the model effectively distinguishes between positive and negative sentiments, with relatively few misclassifications.

Specifically, the confusion matrix shows 471 true negatives, 473 true positives, 29 false positives, and 12 false negatives.

These results suggest that LLAMA2, when fine-tuned with PEFT and LoRA, is well-suited for sentiment classification tasks. The high recall of 98.14% indicates that the model is particularly good at identifying positive reviews, while the precision of 93.70% shows that it maintains a good balance between positive identifications and avoiding false positives.

Despite these strong results, there are always opportunities for improvement. Future work could focus on enhancing the model's ability to handle more nuanced sentiments and improving performance on edge cases or less common sentiment expressions. This could involve exploring advanced techniques or integrating additional contextual information to further refine the model's accuracy and robustness.

4.3 Prompt Engineering

In this section, the performance of two different GPT models used for sentiment analysis is evaluated: GPT-4o and GPT-3.5-turbo. Predictions were made using the Azure OpenAI service with specific prompt settings. The GPT predictions were conducted by Algorythym Group, Belgium.

To ensure consistent and deterministic responses from the models, specific settings were applied during the sentiment analysis task:

- **Temperature: 0.0** - The temperature setting controls the randomness of the model's predictions. By setting the temperature to 0.0, the model is forced to always choose the most probable answer, eliminating any variability in the responses. This ensures that the model provides the same output every time it encounters the same prompt and input, which is crucial for obtaining reliable and repeatable results in sentiment classification.
- **Max Tokens: 1** - The `max_tokens` parameter limits the number of tokens (words or symbols) that the model can generate in response to the prompt. Setting this to 1 ensures that the model's response is confined to a single word—either "positive" or "negative"—which is exactly what's required for binary sentiment classification. This prevents the model from producing unnecessary or off-topic content.

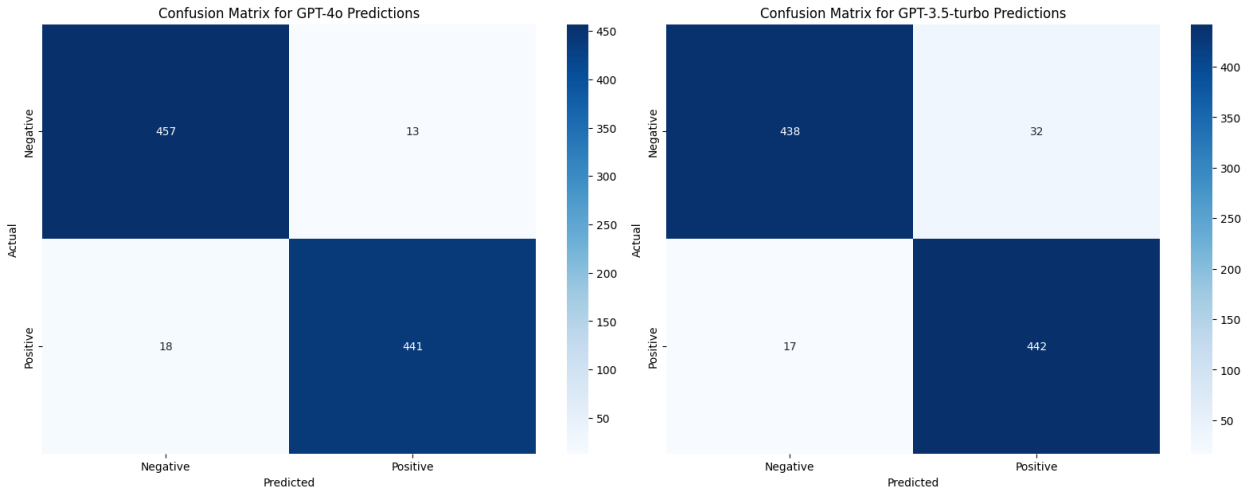
During the analysis, it was observed that some reviews were missing sentiment predictions due to a content filter being triggered. Specifically, 56 reviews were filtered out because of inappropriate text or words in them. The remaining 929 reviews were used for the final analysis, and the reported accuracy is based on this adjusted dataset.

The performance metrics for GPT-4o and GPT-3.5-turbo are summarized in **Table 6**.

Table 6: Performance Metrics for GPT-4o and GPT-3.5-turbo

Metric	GPT-4o	GPT-3.5-turbo
Accuracy	97%	95%
Precision	97%	93%
Recall	96%	96%
F1-Score	97%	95%

The confusion matrices for both models are illustrated side by side in **Figure 5**.



(a) Confusion Matrix for GPT-4o

(b) Confusion Matrix for GPT-3.5-turbo

Figure 5: Confusion Matrices for GPT-4o (a) and GPT-3.5-turbo (b)

The performance metrics and confusion matrices for GPT-4o and GPT-3.5-turbo reveal the following:

- **Accuracy**: GPT-4o achieved a higher accuracy of 97% compared to GPT-3.5-turbo's 95%. This indicates that GPT-4o was slightly more accurate in its overall sentiment classification.
- **Precision**: GPT-4o outperformed GPT-3.5-turbo in terms of precision, with a precision score of 97% compared to 93%. This suggests that GPT-4o was more accurate in its positive sentiment predictions, resulting in fewer false positives.
- **Recall**: Both models had similar recall, with GPT-4o achieving 96% and GPT-3.5-turbo also scoring 96%. This indicates that both models were equally effective at identifying all instances of positive sentiments.
- **F1-Score**: GPT-4o also had a higher F1-score of 97% compared to GPT-3.5-turbo's 95%. The F1-score, which balances precision and recall, further confirms GPT-4o's stronger overall

performance.

The confusion matrices, shown in **Figure 5**, provide additional insight into the models' performance. GPT-4o had fewer misclassifications overall. Specifically, GPT-4o misclassified 13 negative reviews as positive and 18 positive reviews as negative. On the other hand, GPT-3.5-turbo misclassified 32 negative reviews as positive and 17 positive reviews as negative.

In summary, both GPT models performed well in sentiment classification, with GPT-4o demonstrating superior precision and overall accuracy. GPT-3.5-turbo showed strong recall, particularly in identifying positive sentiments, but had a lower precision, indicating a trade-off between these performance metrics.

4.4 XGBoost (Baseline Model)

The XGBoost model was selected as a baseline for benchmarking the performance of more advanced models in the sentiment analysis of IMDb movie reviews. XGBoost, a robust and efficient gradient boosting algorithm, is well-suited for handling large datasets and served as a reliable starting point for this analysis.

For the baseline model, the XGBoost classifier was used with its default hyperparameters. The decision to avoid hyperparameter tuning at this stage was intentional, as the primary goal was to establish a straightforward benchmark against which more complex models could be compared. This approach allowed for a fair evaluation of the model's out-of-the-box performance.

The model was trained on a set of movie review embeddings, each represented as a 3072-dimensional vector, as discussed in the **2.Dataset Description** section. The training process was straightforward, focusing on minimizing the loss function through gradient boosting without any specific tuning. Early stopping or cross-validation was not applied in this baseline setup to maintain simplicity.

The performance of the XGBoost model was evaluated on the test set, with results summarized in **Table 7**. Despite the lack of hyperparameter tuning, the model demonstrated strong performance, achieving an accuracy of **96.24%** on the test set. These metrics indicate that even with default settings, XGBoost is a powerful tool for sentiment classification tasks.

Table 7: Performance Metrics on Test Set

Metric	**Test Set**
Accuracy	96.24%
Precision	94.80%
Recall	97.73%
F1-Score	96.24%

The confusion matrix in **Figure 6** provides further insights into the model's performance on the test set. The model correctly classified 474 out of 500 negative reviews, with 26 misclassified as positive.

Conversely, it accurately identified 474 out of 485 positive reviews, with only 11 being misclassified as negative. This slight imbalance shows that while the model was effective at recognizing positive reviews, it also maintained strong performance with negative reviews.

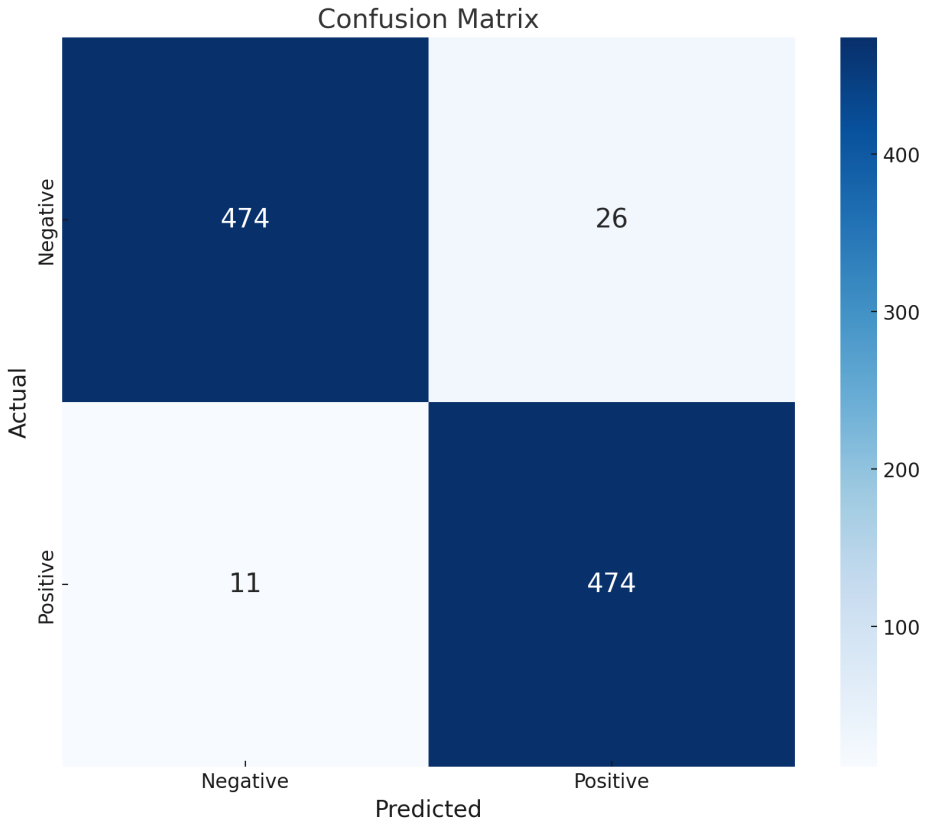


Figure 6: Confusion Matrix on Test Set for XGBoost

5 Discussion

This section provides a comparative analysis of the sentiment analysis models evaluated: DistilBERT, LLAMA2, GPT-4o, GPT-3.5-turbo, and XGBoost. The following table summarizes the performance metrics of each model:

Table 8: Comparison of Performance Metrics Across Models

Model	**Accuracy**	**Precision**	**Recall**	**F1-Score**
DistilBERT	94.62%	92.19%	97.32%	94.68%
LLAMA2	95.84%	93.70%	98.14%	95.87%
GPT-4o	97.00%	97.00%	96.00%	97.00%
GPT-3.5-turbo	95.00%	93.00%	96.00%	95.00%
XGBoost	96.24%	94.80%	97.73%	96.24%

The comparative analysis reveals several key insights and trade-offs among the models:

- **Accuracy and Performance**: GPT-4o achieved the highest accuracy and F1-score among the models evaluated, showcasing its superior overall performance in sentiment analysis. XGBoost also demonstrated strong performance, particularly with its high F1-score, indicating robustness across various metrics. LLAMA2 and GPT-3.5-turbo performed well, with LLAMA2 excelling in recall, which is crucial for tasks that prioritize capturing all instances of positive sentiment.
- **Precision and Recall Trade-off**: GPT-4o excelled in precision, making it particularly effective for applications where minimizing false positives is critical. Although GPT-3.5-turbo had slightly lower precision compared to GPT-4o, it maintained a strong recall rate, making it suitable for scenarios where capturing as many positive sentiments as possible is essential.
- **Model Complexity**: Advanced models like GPT-4o and LLAMA2 offer superior performance but require significant computational resources. In contrast, XGBoost, with its simpler architecture, provides competitive performance while being more computationally efficient. The choice of model should consider the specific requirements of the task, including the availability of computational resources.
- **Application Suitability**: For scenarios requiring high precision, GPT-4o is preferable due to its strong precision metrics. LLAMA2 and GPT-3.5-turbo, with their higher recall rates, are better suited for applications where identifying as many positive sentiments as possible is crucial. XGBoost remains a solid choice for general-purpose sentiment analysis, offering a good balance of performance across all metrics.
- **Generalization**: DistilBERT showed signs of overfitting, as evidenced by the increasing validation loss after the second epoch. GPT-4o and GPT-3.5-turbo, due to their strong per-

formance across metrics, indicate good generalization capabilities, though their performance comes with higher computational costs.

6 Ethical Thinking, Societal Relevance, and Stakeholder Awareness

Ethical considerations, societal impact, and stakeholder awareness are crucial aspects of this thesis, especially in the context of Natural Language Processing (NLP) and sentiment analysis.

Ethical issues are significant when developing NLP systems. These models can unintentionally reflect and reinforce biases present in their training data, potentially leading to unfair or discriminatory outcomes [18]. To address these challenges, it is essential to implement strategies that detect and mitigate biases [19] and ensure compliance with data protection regulations to safeguard user privacy.

The societal relevance of sentiment analysis of IMDB reviews is substantial. By providing insights into public opinion and preferences regarding movies, sentiment analysis enhances audience engagement and contributes to the quality of online discourse and user experiences on review platforms. This understanding aids filmmakers, producers, distributors, and online review platforms like IMDb and Rotten Tomatoes in making informed decisions, influencing viewing choices, and improving content offerings.

Stakeholders including filmmakers, producers, distributors, moviegoers, and review platforms are interested in maximizing audience engagement, influencing viewing choices, and providing valuable content to attract traffic to their websites. Analyzing sentiment in movie reviews enables these stakeholders to better tailor their strategies to meet audience expectations and improve user interactions.

Additionally, measures have been taken to ensure that the data used in the analysis does not contain any personal information, which is crucial for maintaining privacy. Proper citation of all sources and acknowledgment of contributions from other researchers and datasets are also maintained. This practice upholds intellectual property rights and ensures transparency in the research methodology.

7 Conclusion

This thesis explored the relevance of purpose-made models in language processing, specifically in the context of binary sentiment analysis of IMDB movie reviews. By comparing advanced large language models (LLMs) such as BERT, LLAMA2, and GPT-4 with traditional machine learning techniques like XGBoost, this study aimed to assess whether these specialized models offer significant advantages over more conventional methods.

The findings demonstrate that purpose-made models like LLAMA2 and GPT-4o exhibit exceptional performance in sentiment analysis. LLAMA2 achieved the highest accuracy and F1-score, showcasing its advanced capabilities in understanding and classifying sentiment, especially with its high recall of 98.14%. GPT-4o also delivered superior results with the highest accuracy and F1-score, reflecting its effectiveness in precise sentiment classification. BERT, while still robust and performing well, showed slightly lower metrics compared to LLAMA2 and GPT-4o.

Traditional models, particularly XGBoost, were also found to be highly effective, with performance metrics that are competitive with those of the advanced LLMs. XGBoost demonstrated strong performance, with notable accuracy and F1-score, proving its value in sentiment analysis. This suggests that while purpose-made models offer enhanced capabilities, traditional methods remain valuable, especially in terms of computational efficiency and simplicity.

The results affirm that purpose-made models continue to be highly relevant in language processing, particularly when high accuracy and nuanced understanding are required. However, traditional methods like XGBoost provide strong alternatives and should not be overlooked, particularly in scenarios where computational resources or model complexity are constraints.

In summary, both purpose-made and traditional models have their unique strengths and applications. The choice between them should be guided by the specific needs of the task at hand, balancing accuracy, efficiency, and computational resources.

8 Future Research

Future research can build upon this study in several key areas:

Firstly, optimizing the XGBoost model by experimenting with hyperparameters, such as learning rates and tree depths, could further enhance its performance. While XGBoost demonstrated strong performance, fine-tuning these parameters may uncover additional improvements and potentially close the performance gap with advanced LLMs.

Additionally, exploring a broader range of transformer models beyond DistilBERT and LLAMA2 could be valuable. Investigating newer models, such as T5 or advanced GPT variants, could offer further insights into their effectiveness for sentiment analysis tasks. Given the performance of GPT-4o in this study, future work could focus on evaluating other state-of-the-art transformer models to see if they offer similar or improved results.

Combining traditional machine learning techniques with transformer models could also be beneficial. Hybrid approaches that integrate XGBoost with transformer embeddings might leverage the strengths of both methodologies, potentially resulting in improved performance. Exploring ensemble methods that combine predictions from both XGBoost and transformer models could be a promising direction.

Expanding the application of these models to diverse datasets and domains, including different review platforms, languages, and contexts, could validate their robustness and generalizability. This would help assess the models' effectiveness across various scenarios and enhance their practical applicability.

Overall, these research directions present promising opportunities to advance sentiment analysis techniques and explore innovative approaches to improve model performance, ensuring that both purpose-made and traditional models are utilized to their fullest potential.

References

[1] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT 2018)*. 2018.

[2] Alec Radford, J. W. Kim, C. Hallacy, et al. "GPT-4 Technical Report". In: *OpenAI* (2023).

[3] Bo Pang and Lillian Lee. "Opinion Mining and Sentiment Analysis". In: *Foundations and Trends® in Information Retrieval* 2.1-2 (2008), pp. 1–135.

[4] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. MIT Press, 2008.

[5] Jeremy Howard and Sebastian Ruder. "Universal Language Model Fine-Tuning for Text Classification". In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2018, pp. 328–339.

[6] Tom B. Brown, Benjamin Mann, Nick Ryder, et al. "Language Models are Few-Shot Learners". In: *Proceedings of the 34th Conference on Neural Information Processing Systems (NeurIPS 2020)*. 2020.

[7] Ashish Vaswani et al. "Attention Is All You Need". In: *Proceedings of the 31st Conference on Neural Information Processing Systems (NeurIPS 2017)*. 2017.

[8] Victor Sanh, Thomas Wolf, and Sebastian Ruder. "DistilBERT, a Distilled Version of BERT: Smaller, Faster, Cheaper and Lighter". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP 2019)*. 2019.

[9] Hugo Touvron, Thibaut Lavril, and Armand Joulin. "LLaMA: Open and Efficient Foundation Language Models". In: *Proceedings of the 2023 Conference on Neural Information Processing Systems (NeurIPS 2023)*. 2023.

[10] Neil Houlsby, Andrei Giurgiu, and Srini Mishra. "Parameter-Efficient Transfer Learning for NLP". In: *Proceedings of the 36th International Conference on Machine Learning (ICML 2019)*. 2019.

[11] Alec Radford et al. "Language Models are Unsupervised Multitask Learners". In: *OpenAI Blog* (2019).

[12] Taylor Shin, Yasaman Razeghi Swanson, and Eric Liang. "Autoprompt: Eliciting Knowledge from Language Models with Automatically Generated Prompts". In: *Proceedings of the 37th International Conference on Machine Learning (ICML 2020)*. 2020.

[13] Jieyu Zhao, Eduard H. Hovy, and Marvin D. Shum. "Calibrate and Evaluate: Learning Stable Classifiers from Weakly-Labeled Data". In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP 2021)*. 2021.

[14] Su Lin Blodgett et al. "Language (Technology) is Power: A Critical Survey of "Bias" in NLP". In: *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency (FAccT 2021)*. 2021.

[15] Lucas Dixon et al. “Measuring and Mitigating Unintended Bias in Text Classification”. In: *Proceedings of the 2018 ACM Conference on Fairness, Accountability, and Transparency (FAccT 2018)*. 2018.

[16] Tianqi Chen and Carlos Guestrin. “XGBoost: A Scalable Tree Boosting System”. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2016, pp. 785–794.

[17] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. MIT Press, 2012.

[18] Tolga Bolukbasi et al. “Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings”. In: *Proceedings of the 30th International Conference on Neural Information Processing Systems (NeurIPS 2016)*. 2016.

[19] Jieyu Zhao, Tianlu Wang, and Liqun Li. “Gender Bias in Coreference Resolution: Evaluation and Debiasing”. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP 2019)*. 2019.

Appendix

Codes

Listing 1: Distilbert Model

```
from google.colab import drive
drive.mount('/content/drive')

import torch
import pandas as pd
from transformers import DistilBertTokenizer,
    DistilBertForSequenceClassification
from torch.utils.data import DataLoader, TensorDataset
from sklearn.metrics import accuracy_score,
    precision_recall_fscore_support, confusion_matrix,
    classification_report
import matplotlib.pyplot as plt
import seaborn as sns

# Check if GPU is available
print("GPU␣available:", torch.cuda.is_available())

# Install required libraries
!pip install datasets transformers huggingface_hub
!apt-get install git-lfs

# Load datasets
train_data = pd.read_csv('/content/drive/MyDrive/thesis_data/imdb_train␣-␣
    Karel␣Kenens.csv')
test_data = pd.read_csv('/content/drive/MyDrive/thesis_data/imdb_test␣-␣
    Karel␣Kenens.csv')

# Preprocess data
tokenizer = DistilBertTokenizer.from_pretrained('distilbert-base-uncased')
train_encodings = tokenizer(train_data['review'].tolist(), max_length=512,
     padding='max_length', truncation=True, return_tensors='pt')
test_encodings = tokenizer(test_data['review'].tolist(), max_length=512,
    padding='max_length', truncation=True, return_tensors='pt')

# Sentiments to labels (0 for negative, 1 for positive)
train_labels = torch.tensor([1 if sentiment == 'positive' else 0 for
    sentiment in train_data['sentiment']])
test_labels = torch.tensor([1 if sentiment == 'positive' else 0 for
    sentiment in test_data['sentiment']])
```

```
# Create TensorDatasets & DataLoaders
train_dataset = TensorDataset(train_encodings['input_ids'],
    train_encodings['attention_mask'], train_labels)
test_dataset = TensorDataset(test_encodings['input_ids'], test_encodings['
    attention_mask'], test_labels)

train_dataloader = DataLoader(train_dataset, batch_size=16, shuffle=True)
test_dataloader = DataLoader(test_dataset, batch_size=16)

# Load pre-trained DistilBERT model
model = DistilBertForSequenceClassification.from_pretrained('distilbert-
    base-uncased', num_labels=2)

# Fine-tune DistilBERT
optimizer = torch.optim.AdamW(model.parameters(), lr=2e-5)
criterion = torch.nn.CrossEntropyLoss()

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model.to(device)

# Early stopping parameters
num_epochs = 20
patience = 3
best_val_loss = float('inf')
epochs_without_improvement = 0

# Initialize lists to store metrics
train_losses = []
val_losses = []
val_accuracies = []

for epoch in range(num_epochs):
    model.train()
    running_loss = 0.0
    for batch in train_dataloader:
        input_ids, attention_mask, labels = batch
        input_ids, attention_mask, labels = input_ids[:, :512].to(device),
            attention_mask[:, :512].to(device), labels.to(device)

        optimizer.zero_grad()
        outputs = model(input_ids, attention_mask=attention_mask)
        loss = criterion(outputs.logits, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
```

```
    avg_train_loss = running_loss / len(train_dataloader)
    train_losses.append(avg_train_loss)

    model.eval()
    val_predictions = []
    val_targets = []
    val_loss = 0.0
    with torch.no_grad():
        for batch in test_dataloader:
            input_ids, attention_mask, labels = batch
            input_ids, attention_mask, labels = input_ids[:, :512].to(
               device), attention_mask[:, :512].to(device), labels.to(
               device)
            outputs = model(input_ids, attention_mask=attention_mask)
            loss = criterion(outputs.logits, labels)
            val_loss += loss.item()
            _, predicted = torch.max(outputs.logits, 1)
            val_predictions.extend(predicted.cpu().tolist())
            val_targets.extend(labels.cpu().tolist())

    avg_val_loss = val_loss / len(test_dataloader)
    val_losses.append(avg_val_loss)
    accuracy_val = accuracy_score(val_targets, val_predictions)
    val_accuracies.append(accuracy_val)

    print(f'Epoch␣{epoch+1}/{num_epochs},␣Training␣Loss:␣{avg_train_loss},
       ␣Validation␣Loss:␣{avg_val_loss},␣Accuracy␣on␣validation␣set:␣{
       accuracy_val}')

    if avg_val_loss < best_val_loss:
        best_val_loss = avg_val_loss
        epochs_without_improvement = 0
        torch.save(model.state_dict(), 'best_model.pt')
    else:
        epochs_without_improvement += 1
        if epochs_without_improvement >= patience:
            print("Early␣stopping␣triggered")
            break

# Load the best model
model.load_state_dict(torch.load('best_model.pt'))

# Predict on test set and calculate evaluation metrics
test_predictions = []
test_targets = []
with torch.no_grad():
    for batch in test_dataloader:
```

```
        input_ids, attention_mask, labels = batch
        input_ids, attention_mask, labels = input_ids[:, :512].to(device),
            attention_mask[:, :512].to(device), labels.to(device)
        outputs = model(input_ids, attention_mask=attention_mask)
        _, predicted = torch.max(outputs.logits, 1)
        test_predictions.extend(predicted.cpu().tolist())
        test_targets.extend(labels.cpu().tolist())

# Calculate evaluation metrics
accuracy_test = accuracy_score(test_targets, test_predictions)
precision, recall, f1, _ = precision_recall_fscore_support(test_targets,
   test_predictions, average='binary')
conf_matrix = confusion_matrix(test_targets, test_predictions)

print("Accuracy␣on␣test␣set:", accuracy_test)
print("Precision␣on␣test␣set:", precision)
print("Recall␣on␣test␣set:", recall)
print("F1-Score␣on␣test␣set:", f1)
print("Confusion␣Matrix␣on␣test␣set:\n", conf_matrix)
print("\nClassification␣Report:\n", classification_report(test_targets,
   test_predictions, target_names=['negative', 'positive']))

# Plot the training and validation loss
plt.figure(figsize=(10, 5))
plt.plot(train_losses, label='Training␣Loss')
plt.plot(val_losses, label='Validation␣Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.title('Training␣and␣Validation␣Loss␣Over␣Epochs')
plt.show()


# Plot confusion matrix
plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['
   Negative', 'Positive'], yticklabels=['Negative', 'Positive'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion␣Matrix')
plt.show()
```

Listing 2: LLAMA Model

```
# Install necessary libraries
!pip install transformers peft datasets huggingface_hub
!apt-get install git-lfs
```

```
# Import required libraries
import pandas as pd
import numpy as np
import torch
from transformers import AutoTokenizer, AutoModelForSequenceClassification
from datasets import Dataset
from peft import LoraConfig, get_peft_model
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score,
   precision_recall_fscore_support, confusion_matrix
from transformers import Trainer, TrainingArguments
from huggingface_hub import login

# Log in to Hugging Face
login(token='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')

# Load your data
train_data = pd.read_csv('/content/drive/MyDrive/thesis_data/
   cleaned_imdb_train.csv')
test_data = pd.read_csv('/content/drive/MyDrive/thesis_data/
   cleaned_imdb_test.csv')

# Load and configure tokenizer
model_name = "meta-llama/Llama-2-7b-chat-hf"
tokenizer = AutoTokenizer.from_pretrained(model_name)

# Add padding token if it does not exist
if tokenizer.pad_token is None:
    tokenizer.add_special_tokens({'pad_token': '[PAD]'})

print("Pad␣Token:", tokenizer.pad_token)
print("Pad␣Token␣ID:", tokenizer.pad_token_id)

# Preprocessing function
def preprocess_function(examples):
    return tokenizer(
        examples['review'],
        padding='max_length',
        truncation=True,
        max_length=512
    )

# Convert to Hugging Face datasets
train_dataset = Dataset.from_pandas(train_data)
test_dataset = Dataset.from_pandas(test_data)
```

```
# Convert sentiment labels
label_encoder = LabelEncoder()
label_encoder.fit(train_data['sentiment'])
train_data['sentiment'] = label_encoder.transform(train_data['sentiment'])
test_data['sentiment'] = label_encoder.transform(test_data['sentiment'])

# Update datasets with new labels
train_dataset = Dataset.from_pandas(train_data)
test_dataset = Dataset.from_pandas(test_data)

# Map preprocessing function
train_dataset = train_dataset.map(preprocess_function, batched=True)
test_dataset = test_dataset.map(preprocess_function, batched=True)

# Set format for PyTorch
train_dataset.set_format(type='torch', columns=['input_ids', '
    attention_mask', 'sentiment'])
test_dataset.set_format(type='torch', columns=['input_ids', '
    attention_mask', 'sentiment'])

# Load and apply PEFT
model = AutoModelForSequenceClassification.from_pretrained(model_name,
    num_labels=2)
lora_config = LoraConfig(
    r=8,
    lora_alpha=16,
    lora_dropout=0.1
)
model = get_peft_model(model, lora_config)

# Move model to GPU
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model.to(device)

# Training setup with specified learning rate
def compute_metrics(pred):
    logits = pred.predictions
    predictions = np.argmax(logits, axis=-1)
    labels = pred.label_ids
    accuracy = accuracy_score(labels, predictions)
    precision, recall, f1, _ = precision_recall_fscore_support(labels,
        predictions, average='binary')
    conf_matrix = confusion_matrix(labels, predictions)
    return {
        'accuracy': accuracy,
        'precision': precision,
        'recall': recall,
```

```
        'f1': f1,
        'confusion_matrix': conf_matrix.tolist()
    }

training_args = TrainingArguments(
    output_dir='./results',
    evaluation_strategy="epoch",
    per_device_train_batch_size=4,
    per_device_eval_batch_size=4,
    num_train_epochs=1,  # Set to 1 epoch due to memory constraints
    logging_dir='./logs',
    learning_rate=2e-5
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=test_dataset,
    compute_metrics=compute_metrics,
)

# Train and save the model
trainer.train()
model.save_pretrained('./fine_tuned_model')

# Evaluate model
eval_results = trainer.evaluate()
```

Listing 3: XGBoost Baseline Model

```
from google.colab import drive
drive.mount('/content/drive')

import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,
   precision_recall_fscore_support, confusion_matrix,
   classification_report
from xgboost import XGBClassifier
import pandas as pd
import numpy as np

# Load the training and testing dataset
train_data = pd.read_parquet('/content/drive/MyDrive/thesis_data/
```

```
    cleaned_imdb_train.parquet')
test_data = pd.read_parquet('/content/drive/MyDrive/thesis_data/
    cleaned_imdb_test.parquet')

# Separate features and labels for training and test data
X_train = np.array(train_data['embedding'].tolist())
y_train = train_data['sentiment'].map({'negative': 0, 'positive': 1}).
    values

X_test = np.array(test_data['embedding'].tolist())
y_test = test_data['sentiment'].map({'negative': 0, 'positive': 1}).values

# Split data into training and validation sets
X_train_split, X_val, y_train_split, y_val = train_test_split(X_train,
    y_train, test_size=0.2, random_state=42)

# Initialize XGBoost classifier with binary objective
xgb = XGBClassifier(objective='binary:logistic', random_state=42)

# Train XGBoost classifier on the training data
xgb.fit(X_train_split, y_train_split)

# Predict on training set
y_train_pred = xgb.predict(X_train_split)

# Predict on validation set
y_val_pred = xgb.predict(X_val)

# Predict on test data
y_test_pred = xgb.predict(X_test)

# Calculate evaluation metrics on training set
accuracy_train = accuracy_score(y_train_split, y_train_pred)
precision_train, recall_train, f1_train, _ =
    precision_recall_fscore_support(y_train_split, y_train_pred, average='
    binary')

# Calculate evaluation metrics on validation set
accuracy_val = accuracy_score(y_val, y_val_pred)
precision_val, recall_val, f1_val, _ = precision_recall_fscore_support(
    y_val, y_val_pred, average='binary')

# Calculate evaluation metrics on test set
accuracy_test = accuracy_score(y_test, y_test_pred)
precision, recall, f1, _ = precision_recall_fscore_support(y_test,
    y_test_pred, average='binary')
conf_matrix = confusion_matrix(y_test, y_test_pred)
```

```
# Print evaluation metrics
print("Training␣Set␣Metrics:")
print("Accuracy␣on␣training␣set:", accuracy_train)
print("Precision␣on␣training␣set:", precision_train)
print("Recall␣on␣training␣set:", recall_train)
print("F1-Score␣on␣training␣set:", f1_train)

print("\nTest␣Set␣Metrics:")
print("Accuracy␣on␣test␣set:", accuracy_test)
print("Precision␣on␣test␣set:", precision)
print("Recall␣on␣test␣set:", recall)
print("F1-Score␣on␣test␣set:", f1)
print("Confusion␣Matrix␣on␣test␣set:\n", conf_matrix)
print("\nClassification␣Report:\n", classification_report(y_test,
    y_test_pred, target_names=['negative', 'positive']))

# Plot confusion matrix
plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['
    Negative', 'Positive'], yticklabels=['Negative', 'Positive'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion␣Matrix')
plt.show()
```