Technical Report

NLP Project

TripAdvisor Reviews

Groupe 4

Lucas Duport
Eugenie Beauvillain
Yanis Martin
Arthur Courselle
Flavien Geoffray

Contents

Introduction

As part of our course project, we explored the application of Natural Language Processing (NLP) techniques to real-world textual data. Our initial objective was to work on text classification and prompt injection generation—a growing concern in the context of generative language models. However, the dataset we selected for this task proved inadequate: it was highly imbalanced, partially generated by AI, and the classification problem was overly simplistic, involving only two classes. These limitations led us to reconsider our approach.

We decided to shift our focus to a more structured and data-rich task using a dataset of TripAdvisor reviews. This dataset contains thousands of user-generated reviews, each labeled with a rating from 1 to 5. It offers a more challenging and meaningful context for applying NLP methods. Our revised project is centered around two tasks: (1) review rating prediction based on review text, and (2) conditional review generation given a target rating and a set of relevant keywords. These tasks allow us to explore both supervised learning and text generation techniques, leveraging both traditional and modern NLP models.

This report outlines our methodology, including data preprocessing, model selection, performance evaluation, and an in-depth analysis of the results obtained for each task.

1 Data and pre-processing

1.1 Dataset Presentation

1.1.1 Dataset Choice

We initially selected a different dataset focused on prompt injection classification. However, after running early experiments, we discovered that the labels in that dataset were likely generated using a simple logistic regression model, making the task artificially easy to solve. Additionally, we suspected that some data might be incorrectly labeled or synthetic, reducing its value for serious modeling or evaluation.

To ensure higher data quality and a more meaningful problem, we transitioned to this TripAdvisor dataset. Unlike the previous one, this dataset contains real, user-generated reviews collected via web scraping from a well-known platform, making it a much more reliable and rich resource. The natural structure of the problem—predicting a real-valued sentiment score from nuanced text—also makes it ideal for training and evaluating models in natural language understanding tasks.

1.1.2 Source and Nature of the Dataset

The dataset is sourced from TripAdvisor hotel reviews, originally compiled by Jiwei Li et al. and later filtered and processed by Junichiro Niimi. It contains English-language hotel reviews collected via web scraping, ensuring the data is authentic and based on real user input. The dataset includes multiple fields such as review text, aspect-based ratings (e.g., cleanliness, location), and the overall user rating.

- **Total number of reviews**: 201,295 reviews
- **Classes (Target Ratings)**: 5 (integer ratings from 1 to 5)
- **Languages**: English (filtered using `fastText`)
- **Data Split**: Single 'train' split (360 MB in size)

In our work, we primarily used two fields.

- `review`: The review title and the main content of the review, which contains the detailed opinion of the user.
- `overall`: The overall rating score, which serves as the target variable (from 1 to 5).

Other available features include metadata such as hotel ID, user ID, review title, aspect ratings (cleanliness, location, value, etc.), and timestamps. However, for simplicity and focus, we chose to rely mainly on the review text and the overall rating for our modeling.

1.2 Preprocessing and Exploratory Data Analysis

To prepare the dataset for modeling, standard text normalization steps were applied:

- **Lowercasing**: Ensures consistency across tokens.

- **Punctuation Removal**: Reduces noise and simplifies token patterns.
- **Tokenization**: Primarily used Byte Pair Encoding (BPE) for its ability to handle subword units and scale across vocabularies. This also supports future modular tokenizer integration.

In certain notebooks, NLTK tokenization was used with optional stopword removal and lemmatization. These steps were applied cautiously, as excessive normalization can degrade the coherence of generated text by removing important linguistic structure.

1.2.1 Dataset Statistics

- **Documents**: 201,295 before augmentation
- **Unique Tokens**: 168,577
- **Average review length**: 712 characters (+- 235 tokens)

See the rating distribution before augmentation in appendices. (Figure 3)

This highlights a positive skew, with over 76% of reviews rated 4 or 5 stars. To mitigate prediction bias toward higher ratings, we will apply class weighting and report metrics such as macro-averaged F1 scores to ensure fair performance across all classes.

See the wordclouds and the token count histogram in the appendices. (Figure 1 and 2)

2 Deep Exploration of Three Advanced Approaches

2.1 Data Augmentation

Data augmentation is a technique used in Natural Language Processing (NLP) to enrich a dataset by generating new instances from existing data. In the context of our analysis of TripAdvisor reviews, this approach is particularly relevant for addressing class imbalance, where certain categories of reviews are underrepresented. See the class distribution (Figure 3).

2.1.1 Implemented Methodology

Our implementation is based on a `DataAugmentation` class that integrates various text augmentation techniques provided by the `nlpaug` library. The primary objective is to balance the dataset by augmenting samples from minority classes, thereby improving the performance of classification models. By default, the dataset is augmented up to 80% of the size of the majority class rather than applying perfect balancing, in order to avoid introducing an artificial overrepresentation. See the augmented class distribution (Figure 4).

2.1.2 Augmentation Techniques

- **Synonym substitution** (`synonym`): Replaces certain words with their synonyms using WordNet.
- **Contextual substitution** (`contextual`): Uses a language model (DistilBERT) to replace words with others that are semantically appropriate in context.

- **Random word swap** (`random`): Randomly permutes the order of some words in the text.
- **Sentence reordering** (`sentence_shuffle`): Changes the order of sentences in a longer text.
- **Word deletion** (`word_deletion`): Randomly deletes certain words with a predefined probability (10% in our case).

See the data augmentation examples in the appendices (Figure 9).

2.1.3 Conclusion

Data augmentation proved to be a valuable strategy for improving both the quality and balance of our TripAdvisor reviews dataset. By generating plausible and semantically coherent variations of existing reviews, we enriched underrepresented classes and thus enhanced the robustness of our classification models.

We applied three augmentation techniques for our experiments: synonym substitution, sentence reordering, and word deletion. This approach led to a significant performance improvement; for example, the accuracy of our Feed-Forward classifier increased from 0.65 to 0.80.

2.2 Unsupervised Learning

Initial clustering using TF-IDF and KMeans was ineffective, producing noisy groupings based on word overlap rather than meaning. To improve this, we adopted a semantic approach using Sentence-BERT (SBERT) embeddings.

Clustering with SBERT, UMAP, and HDBSCAN

Each review was embedded using the all-MiniLM-L6-v2 SBERT model (384 dimensions), then reduced to 50 dimensions via UMAP to preserve structural relationships. HDBSCAN was applied for clustering, effectively detecting groups of varying densities and filtering out ambiguous reviews as noise (cluster -1).

This approach revealed meaningful clusters:

- One cluster focused on internet/wifi complaints.
- Another grouped reviews about conferences and event facilities.
- A third highlighted issues with noise and sleep quality.

Despite these improvements, a large number of reviews remained unclustered, reflecting the subjectivity and diversity of the data.

Topic Modeling with LDA

LDA was applied to a bag-of-words representation to extract 10 topics based on word co-occurrence. Though limited in semantic depth, topics were interpretable:

- Positive interactions with staff.

- Bathroom/water-related amenities.
- Location and walkability.

Comparison and Insights HDBSCAN (with SBERT embeddings) produced semantically coherent clusters, while LDA offered interpretable but lexically driven topics. A heatmap comparison (See Figure 5) showed overlap between some clusters and topics, suggesting that semantic clustering and statistical topic modeling offer complementary views of the dataset.

2.3 Modular NLP Architecture

We designed a flexible and modular architecture to facilitate experimentation with all models while maintaining a consistent interface. This architecture allows us to easily configure and test different models, vectorizers, and tokenizers through a unified configuration system based on YAML files.

2.3.1 Architecture Design

The core components of our system are organized into a modular structure that promotes code reusability and separation of concerns (See Figure 6).

2.3.2 Configuration System

At the heart of our architecture is a YAML-based configuration system that allows us to specify all aspects of the model training pipeline without changing code. The configuration includes:

- **Dataset configuration**: Source dataset, feature and label columns, and preprocessing parameters
- **Tokenization strategy**: Methods for converting text into sequences of tokens
- **Vectorization approach**: Techniques for transforming tokens into numerical representations
- **Model selection**: Choice between various machine learning models (classification or generative)
- **Training parameters**: Hyperparameters for the selected model
- **Evaluation metrics**: Criteria for assessing model performance
- **Device Selection**: Easily switch in config between cuda, mps for Apple silicon, or cpu if nothing else is available.

3 Tokenizer

We used Byte-Pair Encoding (BPE) from the Hugging Face Tokenizers library, with a vocabulary size of 20 000 tokens and a minimum pair frequency of 2. We introduced the following special tokens: `[UNK]`, `[PAD]`, `[SOS]`, and `[EOS]`. The `[UNK]` token represents unknown or out-of-vocabulary words, [PAD] is used for padding sequences to equal length, `[SOS]` (Start of Sequence) marks the beginning

of a generated sequence, and `[EOS]` (End of Sequence) indicates its end. These tokens are employed in our generative processes.

4 Vectorization

We explored two approaches to text representation: TF–IDF and Word2Vec. Through our experiments, we saw that TF–IDF outperformed Word2Vec for the classification task, while incorporating an embedding layer at the model's input yielded superior results to pretrained Word2Vec vectors for sequence generation.

4.1 TF–IDF Vectorization

For classification, we employed scikit-learn's `TfidfVectorizer`, restricting the feature space to 1 000 dimensions, removing English stop words, and applying L2 normalization. Weighting was computed with inverse document frequency enabled (`use_idf=true`) and smoothing activated (`smooth_idf=true`), while sublinear term frequency scaling (`sublinear_tf`) was disabled.

4.2 Word2Vec Vectorization

We trained a Word2Vec model using gensim, configured to produce 100-dimensional embeddings with a context window of size 5. To preserve the full vocabulary, the minimum token count was set to 1, and training was parallelized across 4 worker threads. Although this approach offered rich semantic representations, it fell short of TF–IDF in classification and was outperformed by a learned embedding layer in the generation task.

5 Review Classification

Our classification task uses review texts as input features and their corresponding ratings (integers from 1 to 5) as labels, yielding a five-class classification problem.

5.1 Naive Bayes

We used the Multinomial Naive Bayes from scikit-learn. We obtained the following results:

Model	Accuracy	F1 Score	Precision	Recall
MultinomialNB	**0.59**	**0.57**	**0.57**	**0.58**

Table 1: Experimentation results for classification performance of Naive Bayes

These results suggest that our Naive Bayes implementation performs reasonably well for this text classification task, though there is room for improvement. Further optimization of the model parameters, particularly the smoothing parameter α, might yield better performance on this dataset.

5.2 Logistic Regression

We used the Logistic Regression implementation from scikit-learn with a regularization parameter $C = 1.0$, an ℓ_2 penalty, the "lbfgs" solver, and a maximum of 200 iterations.

We obtained the following results:

Model	Accuracy	F1 Score	Precision	Recall
Logistic Regression + Tf-IDf	0.64	0.63	0.63	0.63
Logistic Regression + Word2Vec	0.62	0.61	0.61	0.61
Logistic Regression + BPE + Tf-IDF	**0.66**	**0.65**	**0.65**	**0.65**
Logistic Regression + BPE + Word2Vec	0.63	0.62	0.62	0.62

Table 2: Experimentation results for classification performance of Logistic Regression.

5.3 Feed-Forward Networks

We implemented a Feed-Forward Neural Network using PyTorch with three hidden layers of 512, 256, and 128 units, respectively, each followed by a dropout layer with a dropout rate of 0.2.

We obtained the following results:

Model	Accuracy	F1 Score	Precision	Recall
FNN Tf-IDf	**0.80**	**0.80**	**0.80**	**0.80**
FNN Word2Vec	0.55	0.54	0.54	0.54
FNN BPE + Tf-IDF	0.80	0.80	0.80	0.80
FNN BPE + Word2Vec	0.77	0.77	0.77	0.77

Table 3: Experimentation results for classification performance of Feed-Forward Network.

5.4 Recurrent Neural Network

We used the PyTorch Recurrent Neural Network layer with 2 hidden layers of 256 units and a Linear layer of 5 units for the classification. The RNN layer is followed by a dropout layer with a dropout rate of 0.2. Since RNN is usually used with an embeddings layer, we used here Word2Vec as the embedding layer and then did not use Tf-IDF.

We obtained the following results:

Model	Accuracy	F1 Score	Precision	Recall
RNN Word2Vec	**0.74**	**0.73**	**0.73**	**0.73**
RNN BPE + Word2Vec	0.72	0.72	0.72	0.72

Table 4: Experimentation results for classification performance of Recurrent Neural Network.

5.5 Pre-Trained Transformer

We used the Trainer library from Hugging Face to fine-tune a model on our dataset. We trained through 4 epochs on the prajjwal1/bert-mini model. We achieved the following results:

Model	Accuracy	F1 Score	Precision	Recall
prajjwal1/bert-mini	**0.77**	**0.76**	**0.76**	**0.76**

Table 5: Experimentation results for classification performance of prajjwal1/bert-mini Fine-Tuned.

5.6 Conclusion

From our classification experiments, it emerges that deep learning architectures consistently outperform linear models. However, increasing model complexity does not guarantee better results: a simple feed-forward network with a TF–IDF vectorizer surpassed both recurrent architectures and the small pre-trained Transformer we evaluated. Moreover, the addition of subword tokenization to the feed-forward model provided no measurable benefit. It is worth noting that our pre-trained Transformer comprises only about 22 million parameters; it is entirely possible that larger Transformer variants would outperform the feed-forward baseline.

6 Review Generation

In this section, we evaluate various models for text generation, focusing on their performance, strengths, and limitations. The models evaluated include N-grams, TF-IDF and Word2Vec for generation, Feedforward Neural Networks, Recurrent Neural Networks (RNN), and Transformer-based models (e.g., GPT).

All experiments—with the exception of the n-gram were conducted on a stratified subset of 100 000 samples drawn from the full dataset, each model being trained for 20 epochs. We chose not to employ the Word2Vec vectorizer, since the embedding layers learned end-to-end within the models consistently outperformed any fixed Word2Vec representations.

Our generation experiments employed input–target pairs in which the targets were the full review texts, while the inputs were constructed by concatenating the numeric rating with five to ten keywords summarizing the review content. Given the modest performance observed, we hypothesize that both deriving prompts directly from the target sequences and evaluating outputs via exact-match against reference texts may have constrained the model's generative capacity, thereby contributing to the lower results.

6.1 Ngram

N-gram models are a fundamental approach to text generation, relying on the statistical properties of text. For this project, we implemented an n-gram model of order 15.

6.1.1 Performance

Perplexity: The model achieved a perplexity score of 1098.37, indicating a lack of confidence in its predictions. This high perplexity is expected given the simplicity of the n-gram approach and its inability to capture long-range dependencies.

6.1.2 Generated Samples

The model generated several text samples based on given prompts. While the generated text is somewhat coherent, it often lacks depth and context. For example:

- "the hotel was bad because it was very warm and humid outside. right inside the door was the commode and tub/shower"
- "our stay at the resort has been the most relaxing for all of us. the staff was absolutely fantastic, from the people who brought"

See Ngrams results in the appendices (Figure 10).

6.2 Feedforward Neural Network

The Feedforward neural network is one of the simplest forms of neural networks. It is a good stepping stone to start off our analysis on neural networks for generating. We obtained the following results:

Model	BLEU	ROUGE-1	ROUGE-2	ROUGE-L	BERTScore P	BERTScore R	BERTScore F1
FNN	0.0146	0.2536	0.0303	0.1941	-0.3486	-0.1846	-0.2680

Table 6: Experimentation results for generation performance of out FNN.

6.3 Recurrent Neural Network

We designed a Recurrent Neural Network (RNN) model using PyTorch. The architecture consists of an Embedding layer that transforms token IDs into dense vector representations, followed by an LSTM layer that captures temporal dependencies across the sequence. To mitigate overfitting, a Dropout layer is applied after the LSTM. Finally, a Fully Connected layer maps the LSTM outputs to the vocabulary space, enabling token prediction. This modular design allows for flexibility in adjusting model depth and capacity.

Model	BLEU	ROUGE-1	ROUGE-2	ROUGE-L	BERTScore P	BERTScore R	BERTScore F1
RNN	0.0574	0.4015	0.0921	0.2963	-0.0824	0.0330	-0.0239

Table 7: Experimentation results for generation performance of out RNN.

See a generated review with our RNN model in the appendices (Figure 7).

6.4 Transformer

Following the architecture introduced by Vaswani *et al.* (2017), we built a sequence-to-sequence Transformer model composed of four identical encoder–decoder layers. Each layer employs multi-head self-attention with eight heads, embedding dimension $d_{\text{model}} = 128$, and a feed-forward sub-layer. To prevent overfitting, we applied a dropout rate of 0.1 throughout the network. Training was conducted for 20 epochs with a batch size of 8. Input sequences were truncated or padded to a maximum length of 32 tokens, while target sequences were limited to 256 tokens. All other hyperparameters were kept consistent with the original "Attention Is All You Need" setup.

We achieved the following results:

Model	BLEU	ROUGE-1	ROUGE-2	ROUGE-L	BERTScore P	BERTScore R	BERTScore F1
Transformer	0.0275	0.3417	0.0770	0.1770	0.1381	0.0073	0.0726

Table 8: Experimentation results for generation performance of out Transformer.

See a generated review in the appendices (Figure 8).

6.5 Pre-Trained Transformer

Our generation pipeline is compatible with large pre-trained language models. However, due to limited computational resources, we were unable to perform the necessary fine-tuning of such models within this work. We anticipate that, given sufficient compute, adapting a full-scale LLM would yield significant gains in output quality, and we look forward to discussing these possibilities during our presentation.

6.6 Conclusion

The generation results highlight the limitations of training models from scratch on relatively small datasets. While n-gram models offer basic local coherence, their high perplexity confirms poor generalization. Neural models (FNN, RNN) show slight improvements but fail to capture semantic depth, as reflected by negative or low BERTScores. The Transformer model demonstrates better contextual understanding, though still constrained by the lack of pre-training.

Our input construction strategy—rating plus keywords—and exact-match evaluations may have further limited the models' ability to generate diverse, high-quality text. Overall, these findings suggest that effective text generation benefits significantly from both richer prompts and pre-trained language models, which we were unable to leverage due to computational constraints.

7 Training

All of our models were trained on CUDA GPUs. The dataset was split into three subsets: training, validation, and test, comprising 64 %, 16 %, and 20 % of the data, respectively.

Training was performed with early stopping, using a patience of five epochs. We optimized with the Adam optimizer at an initial learning rate of 1×10^{-4}, and employed a ReduceLROnPlateau scheduler with a reduction factor of 0.8 and a patience of five epochs. Unless otherwise specified, the batch size was set to 32. The loss function used was cross-entropy.

Appendices

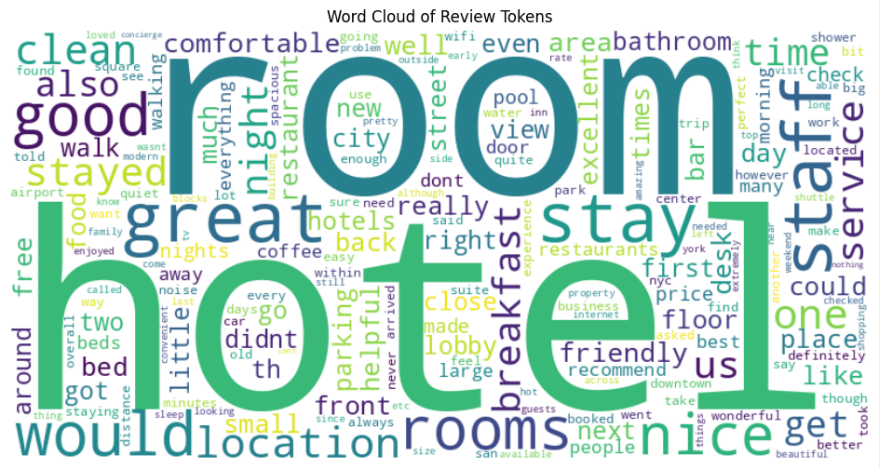

Figure 1: Word Clouds: Highlighted domain-specific lexical patterns, dominated by hospitality-related terms.

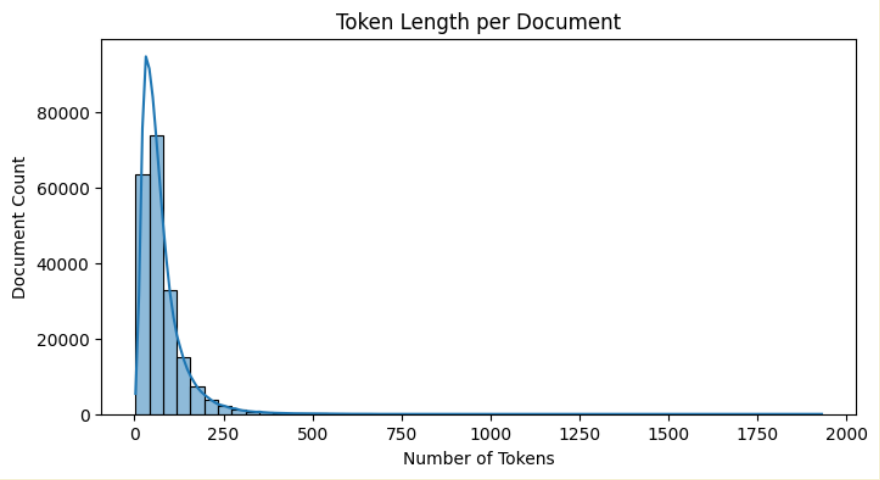

Figure 2: Token Count Histogram: Revealed that most reviews contain 100 tokens, which, while typical, limits context for language models.

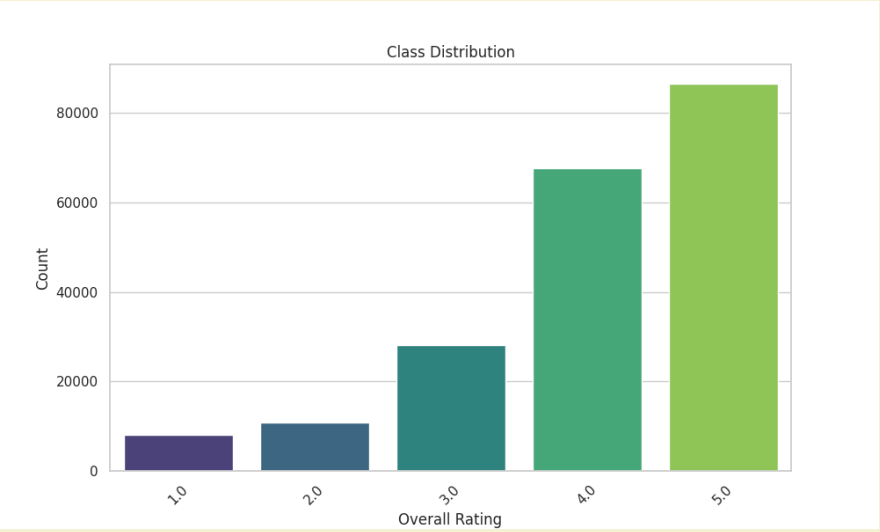

Figure 3: Class distribution before data augmentation

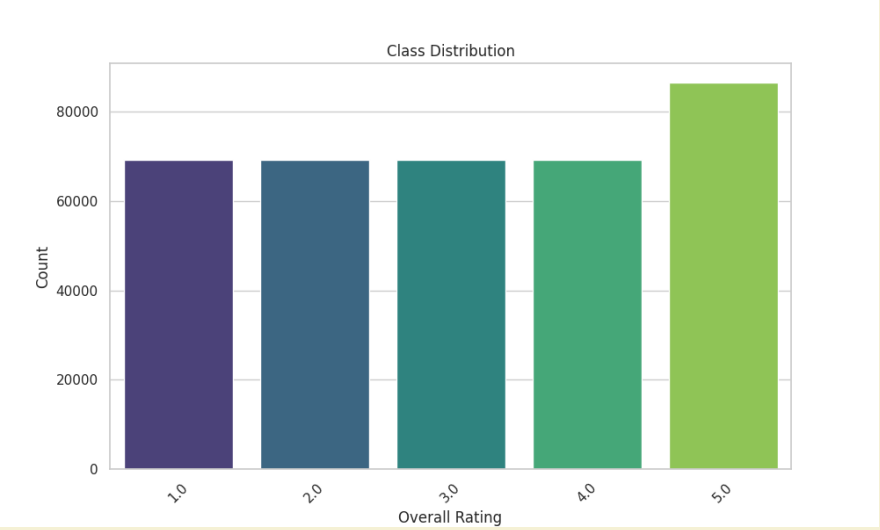

Figure 4: Class distribution after data augmentation

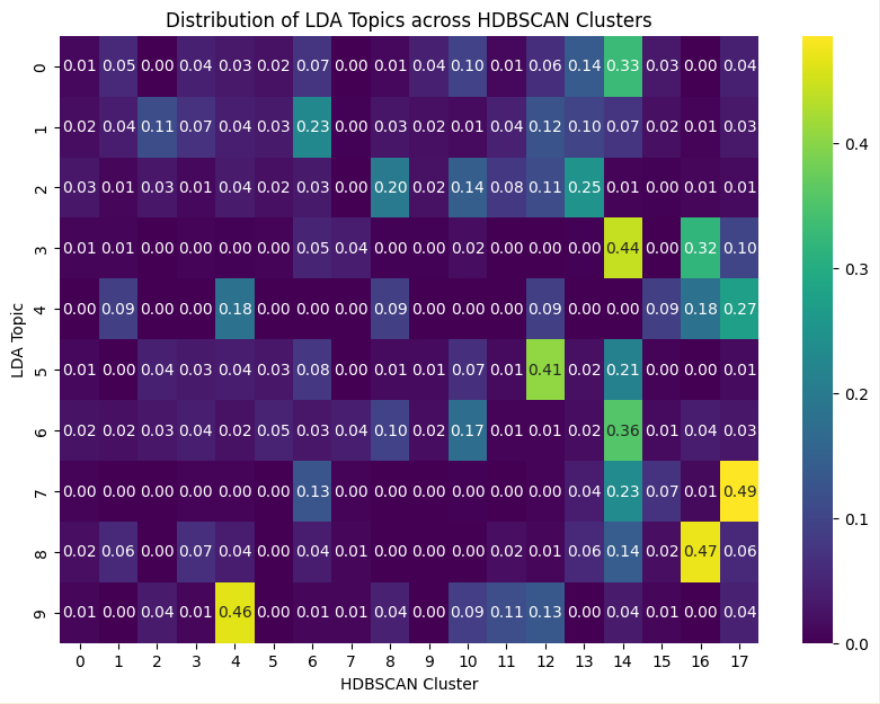

Figure 5: Relationship between HDBSCAN clusters and LDA topics.

Table 9: Examples of data augmentation on TripAdvisor reviews

Technique	**Original review**	**Augmented review**
Synonym substitution	The hotel was beautiful with a breathtaking view of the sea.	The establishment was splendid with a stunning view of the ocean.
Contextual substitution	The staff at the front desk were very attentive and professional.	The reception staff were truly considerate and competent.
Random word swap	The room was clean but quite small for the price.	The small room was clean but quite for the price.
Sentence reordering	We loved the pool. The restaurant served delicious dishes. The room was comfortable.	The room was comfortable. We loved the pool. The restaurant served delicious dishes.
Word deletion	The breakfast was varied and the products were fresh and of good quality.	The breakfast was varied and the products were fresh quality.

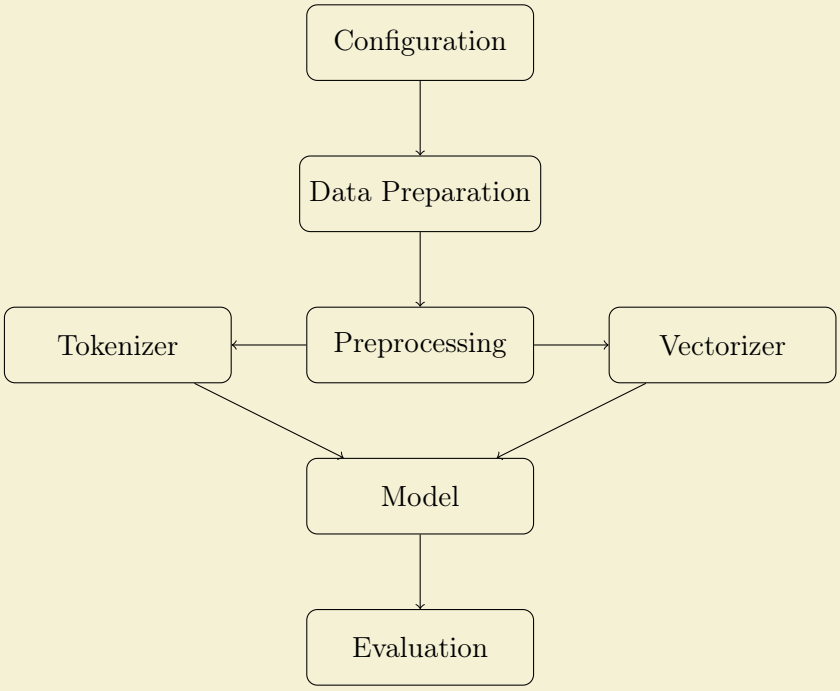

Figure 6: High-level architecture diagram

Prompt	**Generated Text**
The hotel was bad because	the hotel was bad because it was very warm and humid outside. right inside the door was the commode and tub/shower
Our stay at the resort has been	our stay at the resort has been the most relaxing for all of us. the staff was absolutely fantastic, from the people who brought
The room was clean and	the room was clean and the bed was comfortable. the half glass wall in the bathroom (rather than a shower curtain)
Breakfast was included and delicious,	breakfast was included and delicious, and the price was decent. plenty of space, and nice amenities in our room. the bed was
I would not recommend	i would not recommend this hotel the hotel may be nice for weddings but it is not for guests. the "w
The service was excellent and	the service was excellent and the food delicious. we were there during the gala opening after the refurbishment – the t

Table 10: Results of the Ngram generation: Preplexity = 1098.3735334086969

Input	Generated Review
Rating: 5 Keywords: price, value, expensive	"The hotel is a solid hotel if you're going to get into San Francisco so this is where it is. A great choice for everything. The staff are very helpful and friendly. The room was nicely decorated and the staff very friendly. The only complaint I have was that the room had a musty smell and the bed was very uncomfortable (I think there was no place to hang clothes). I could see everything else in the hall, and a desk to ceiling windows on the small side. I thought the bed and pillows were very nice and we enjoyed the layout. The bed was super comfy but was very nice to have a room with a rollaway bed. The room was extremely clean and comfortable, the bed was comfortable, and the bed comfortable and a wonderful touch. The bed was wonderful - I slept like a baby. The only downside is the noise. You may get on your room as you're staying there, but no matter what it comes. I can see the city around the corner of"

Figure 7: Examples of review generated by the RNN model

Input	Generated Review
Rating: 2 Keywords: hotel, small, room, sink, toilet, put, time, would	"While I understand NYC hotel rooms need to be small, I do believe hotels can provide thoughtful ways to make small areas more user-friendly to travelers. I don't think this hotel does that at all. The room was pretty small with just enough room on either side of the bed to fit nightstands. A dresser, a good sized closet with mirrors. The room itself wasn't terrible, but the bathroom took the (small) cake. It had a pedestal sink, a stock shower/bath combo, and a toilet. The pedestal sink had no counterspace around it to put things out, there was no storage underneath, virtually nowhere to put your toiletries. There was a very narrow shelf that went across over the sink and toilet (I kept waiting for my curling iron or blowdryer to fall into the toilet). We stayed here two nights and it was two too many. If they put up a few shelves or areas to store stuff, it would be a little different. However, there were many other negatives as well. The lobby was huge but unbelievably crowded every time we walked through it. This hotel obviously has a ton of rooms - and yet all the elevators are only accessible through a narrow corridor off the lobby, which makes everyone pack into a mosh pit mess trying to get to or out of the elevators. The bar area was VERY expensive for drinks - I expect a higher price but $20 for a glass of (cheap) Chardonnay was just ridiculous. The location was just ok. Overall, I've stayed in places where I don't feel like a number or a sardine - I'd recommend the Omni over this any day. Service is so much better with the boutique Omni chain that this giant corporate chain. By the way, New York was sweltering in July - to the point that meeting our family here to do some touristy things was just not pleasant. No one wanted to be outside during the day. I wouldn't plan a walking trip this time of year even if prices might be best at this time."

Figure 8: Examples of review generated by the Transformer model