Report:
Text-Summarizer

Enrico Ferraiolo 0001191698

Master's Degree in Computer Science

Course: Natural Language Processing
Academic Year: 2024-2025

Contents

1 Introduction

The main goal of this project is to generate concise and meaningful summaries from longer product reviews, while preserving the original meaning of the text.
The project is structured in the following phases:

- **Data Collection and Preparation:** selection and preprocessing of a dataset of product reviews, with particular attention to text cleaning and normalization.
- **Design and Implementation of Neural Network Architectures:** study and development of models based on attention mechanisms for text summarization.
- **Training and Inference:** implementation of pipelines for model training and for performing summarization operations on new texts.
- **Experimental Evaluation:** comparative analysis of model performance using standardized metrics, in order to identify optimal solutions.

This document aims to illustrate the design choices and methodologies adopted for the realization of the project, as well as the experimental results obtained. *TextSummarizer* is a Python framework for **automatic text summarization**, designed to be *modular and easily extensible*.

1.1 Project Structure

The project is structured into several sections, each addressing a specific aspect of the work carried out.
The main folder contains the files and directories necessary for the execution of the project:

- **architectures**: folder containing the implementations of the architectures of the automatic summarization models;
- **report**: folder containing the final report of the project;
- **results**: folder containing the results obtained during the training and evaluation of the models;
- `requirements.txt`: file containing the libraries required for the execution of the project;
- `text_summarizer_training.ipynb`: Python notebook containing the code for training and evaluating the models;
- `text_summarizer_inference.ipynb`: Python notebook containing the code for model inference.

2 Project Pipeline

The project pipeline is shown in Figure 1, illustrating the various steps and stages involved in the automatic text summarization process. The pipeline is designed to be modular and easily extensible, allowing for the integration of new models and evaluation metrics as needed.

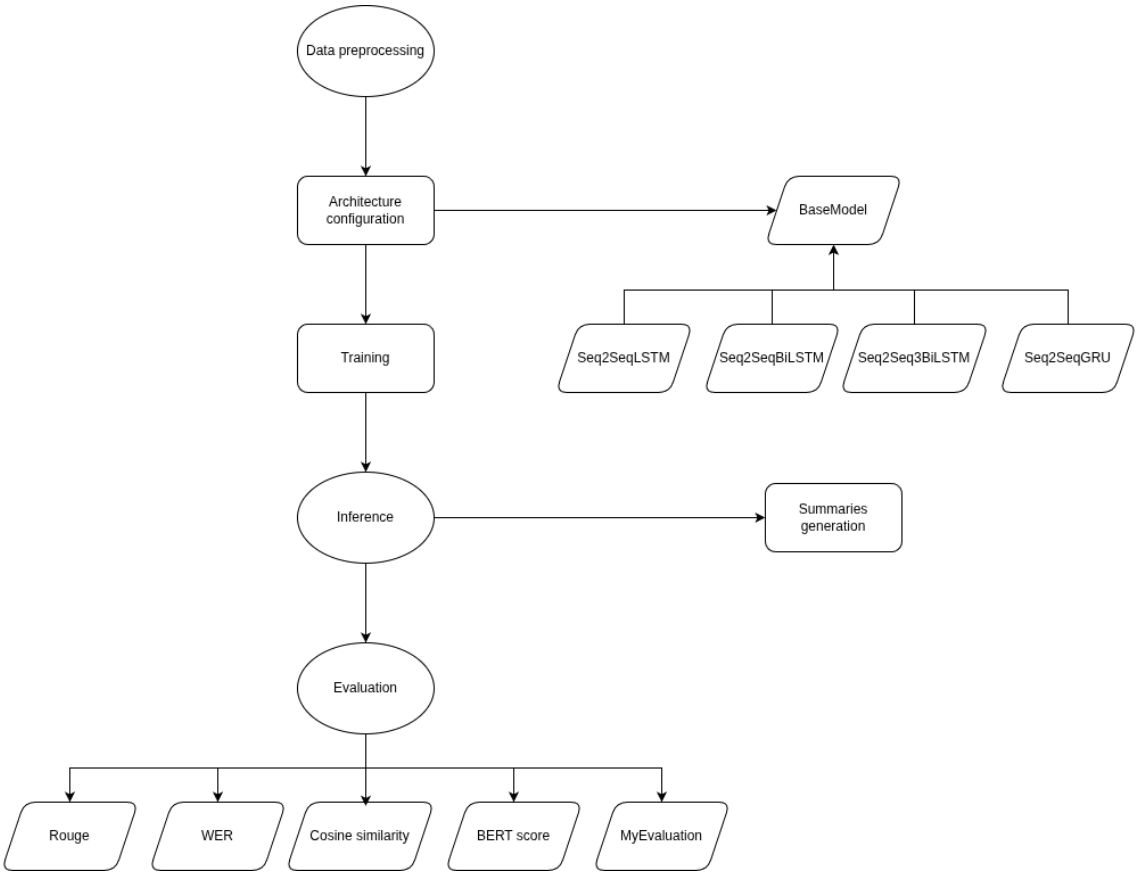


Figure 1: Pipeline del progetto

The pipeline is divided into three main phases:

1. **Preprocessing**: in this phase, data cleaning and preparation operations are performed, such as removing stopwords, tokenization, and text normalization.
2. **Training**: in this phase, the summarization model is trained using the preprocessed data. A split is made between the training set and the validation set, and the model is trained using the training set while monitoring its performance on the validation set.
3. **Evaluation**: in this phase, the summaries generated by the model are evaluated using various metrics, such as ROUGE, BERT score, and the custom metric `myevaluation`.

In the following sections, the details of each phase of the pipeline will be explored.

3 Dataset

For this project, the *SNAP Amazon Fine Food Reviews* dataset[1][3] was used. It contains English-language reviews of food products from Amazon.

Specifically, the dataset includes, for each row, a full review and its corresponding summary.
From the original dataset, which consists of approximately 500,000 rows, a subset of 10,000 rows was selected for analysis and model training.

4 Data Preprocessing

Data preprocessing is a critical step to ensure the quality and effectiveness of summarization models. It is essential to clean and filter the data accurately.
Several cleaning and filtering operations were performed on the dataset to ensure data quality and consistency during model training.
Below are the steps carried out in this phase:

4.1 Text Cleaning

The following preprocessing steps were applied:

1. **Lowercasing the text**
   - This conversion ensures uniformity in the text, preventing the same word from being considered different solely due to the presence of uppercase letters.
     For example, "Home", "HOME", and "home" are treated as the same word, reducing vocabulary dimensionality and improving training efficiency.
2. **Removing HTML tags**
   - Reviews may contain residual HTML tags from the original web format. These elements do not contribute to the semantic meaning of the text and may interfere with model learning, so they are removed.
3. **Expanding contractions**
   - Contractions in the English language (such as "don't", "I'm", "we're") are expanded to their full forms ("do not", "I am", "we are").
     This process aims to standardize and ensure consistency throughout the text and helps the model better capture semantic relationships by eliminating unnecessary variations of the same expression.
4. **Removing possessive forms ('s)**
   - The possessive form in English does not substantially alter the meaning of the sentence for summarization purposes.
     Its removal simplifies the text and further reduces vocabulary size, allowing the model to focus on key concepts.
5. **Removing text in parentheses**
   - Text within parentheses often contains supplementary information that is not generally essential for the summary.
     Removal helps maintain focus on the main information in the review.

6. **Removing punctuation and special characters**

   - Punctuation and special characters, while important for human readability, can introduce noise into model training.
     Their removal simplifies the text while preserving the essential semantic content needed for summary generation.

7. **Removing stopwords**

   - Stopwords are very common words (such as "the", "is", "at", "which") that appear frequently but carry little semantic meaning.
     Their removal significantly reduces the dimensionality of the problem without losing crucial information for the summary, allowing the model to focus on the most meaningful words.

8. **Removing very short words**

   - Very short words (usually one or two letters) often do not contribute to the meaning of the text.
     Their removal helps further reduce noise in the data, keeping only the most significant terms for analysis.

4.2 Data Filtering

After a statistical analysis of the dataset, the following constraints were applied:

- Maximum length of reviews: 30 words
- Maximum length of summaries: 8 words

These limits were determined through statistical analysis of the length distributions in the dataset, as shown in Figure 2.

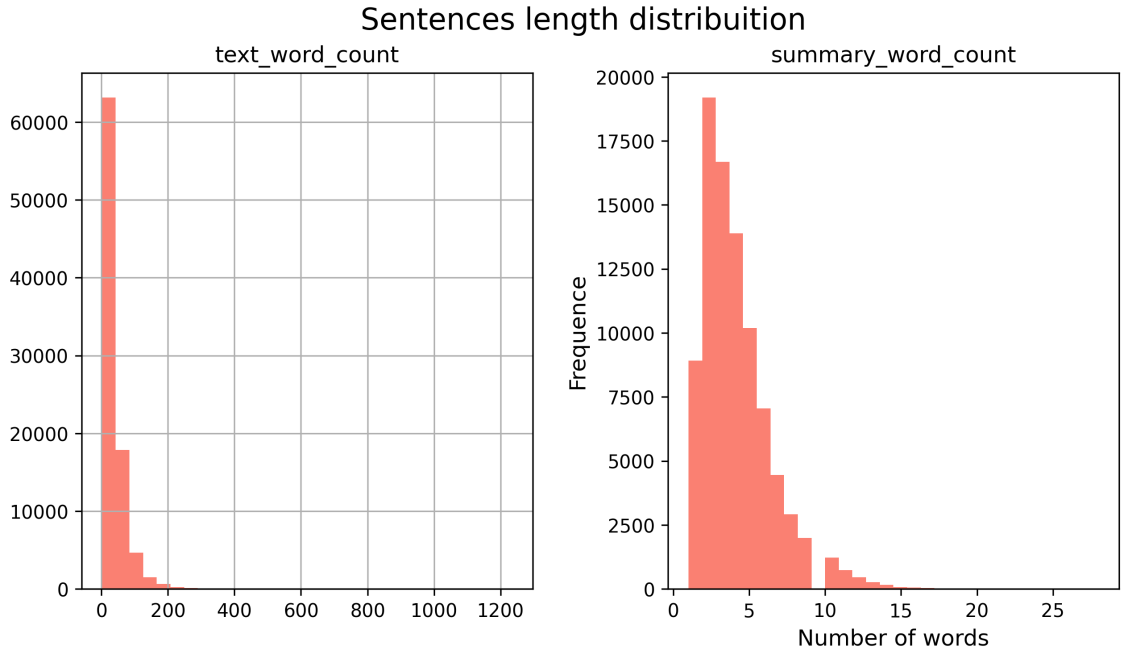


Figure 2: On the left, the distribution of review lengths; on the right, the distribution of summary lengths

Indeed, as can be seen from the two graphs, most reviews and summaries have lengths below the established limits, so these constraints allow for the retention of most of the dataset's data.

4.3 Tokenization and Special Tokens

To prepare the data for the models, I added the special tokens `"sostok"` and `"eostok"` to indicate the beginning and end of a sequence, facilitating tokenization and the training phase.
Additionally, I performed tokenization separately for reviews (input text) and summaries (output text) to ensure that the model can correctly learn the relationship between the two. The two tokenizers are used to create the vocabulary for reviews and for summaries, allowing the conversion of texts into sequences of tokens.

5 Model Architectures

For this project, I chose to compare different models by implementing various **Sequence-to-Sequence** architectures using both **LSTM** and **GRU** layers.
The implementation was carried out using an abstract class called `BaseModel`, from which all specific model classes are derived.
This approach defines a common interface for all summarization models and allows for future architectural extensions with ease.

5.1 Abstract Base Class

The `BaseModel` class provides the base interface for all summarization models:

- Abstract methods for building the encoder and decoder.
- Functionality for saving, loading, and performing inference with the model.
- Token-to-text and text-to-token conversion using the appropriate tokenizers.

5.2 Training

The training of models derived from the `BaseModel` class was carried out using the preprocessed dataset.
Before starting the training, the dataset was split into a training set and a validation set, with a ratio of 90% and 10% respectively.
Then I moved on to the actual training phase of the models, using the loss function `Sparse Categorical Crossentropy`, which is useful in summarization tasks.

5.3 Hyperparameters

For each model class, training was performed multiple times, each with a different combination of hyperparameters.
These combinations are generated through a *hyperparameter_grid* implemented in the function `create_hyperparameter_grid`, which returns all the possible permutations of the values provided for the following parameters:

- `embedding_dim`: dimension of the word embedding
- `hidden_dim`: dimension of the hidden state of the encoder and decoder
- `encoder_dropout`: dropout rate of the encoder
- `encoder_recurrent_dropout`: dropout rate for the recurrent states of the encoder
- `decoder_dropout`: dropout rate of the decoder
- `decoder_recurrent_dropout`: dropout rate for the recurrent states of the decoder
- `optimizer`: optimizer to use during training
- `learning_rate`: learning rate
- `batch_size`: batch size during training
- `epochs`: number of epochs for training

For each **hyperparameter permutation**, the following training pipeline was executed:

1. **Data Preparation**: loading and preprocessing of the data.
2. **Model Instantiation**: initializing the current model class with the selected hyperparameters.
3. **Model Compilation**: compiling the model and starting training with various callbacks (discussed in the next section).
4. **Model Training**: executing the training process.
5. **Saving Results**:
    - *Model weights* (`result/{model_class}/weights/`)
    - *Model architecture* (`result/{model_class}/media/architectures/`)
    - *Loss plot* (`result/{model_class}/media/graphs/`)
    - *Training history* (`result/{model_class}/histories/`)
    - *Generated summaries*: at the end of training, summaries were generated from validation reviews and saved in a CSV file (`result/{model_class}/csv/`)

5.3.1 Callback

During training, I also used the following callback functions:

- **Early Stopping**: monitors a metric, in this case the validation loss, and stops training if there are no improvements for a certain number of consecutive epochs. This helps prevent overfitting and saves computation time.
- **Learning Rate Scheduler**: adjusts the learning rate during training according to a strategy, in my case I used `Step Decay`, which reduces the learning rate by a fixed factor every few epochs.

- **Reduce LR on Plateau**: monitors a metric, in this case the validation loss, and reduces the learning rate if there are no improvements for a certain number of consecutive epochs. This helps optimize the training process and find a more effective learning rate.

In this way, I was able to achieve the best results for each model by adjusting its hyperparameters and trying different combinations.

5.4 Tested Architectures

Several summarization model architectures were tested, each with different characteristics and parameter configurations.
The two main categories of implemented models are:

- **LSTM**: models based on LSTM layers for both encoder and decoder.
- **GRU**: models based on GRU layers for both encoder and decoder.

These architectures are based on RNNs (Recurrent Neural Networks) and were chosen for their effectiveness in text summarization tasks, as they handle dependencies between words in text sequences well.

- **LSTM**: Long Short-Term Memory, is a variant of RNNs that solves the vanishing gradient problem, thanks to the presence of a long-term memory mechanism. This gating mechanism allows it to store important information and discard less relevant data.
- **GRU**: Gated Recurrent Unit, is a simpler variant of LSTMs, with fewer parameters and less computational complexity. Again, the gating mechanism allows it to store important information and discard less relevant data.

In order to make the reading smoother, only the best results obtained during training with the best parameters and configurations found (based on the `BERT score`, which will be discussed later) are reported for each class, although numerous attempts and tests were conducted, which are reported in the following sections in a comparative table.

5.5 Seq2SeqLSTM

The `Seq2SeqLSTM` class implements the specific architecture for the Sequence to Sequence summarization model with LSTM layers.

5.5.1 Training

The training of the model was carried out using the following configuration:

We can check the trend of the losses during training in Figure 3.

Parameter	Value
Optimizer	Adam
Learning rate	0.001
Embedding dimension	512
Latent dimension	128
Decoder dropout	0.2
Decoder recurrent dropout	0.2
Encoder dropout	0.2
Encoder recurrent dropout	0.2
Batch size	128
Epochs	50

Table 1: Training configuration

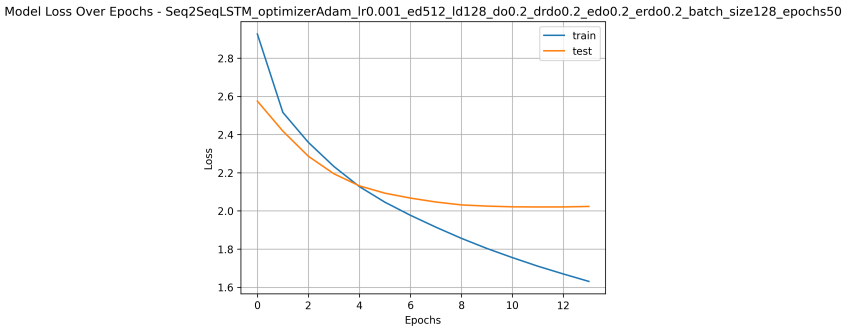


Figure 3: Trend of the *loss* and *validation loss* during training

5.5.2 Results

This model achieved the following results at the end of training:

Metric	Value
Loss	1.63
Validation Loss	2.02
Accuracy	0.67
Validation Accuracy	0.64

Table 2: Results of the model at the end of training

5.5.3 Architecture

The architecture used is as follows:

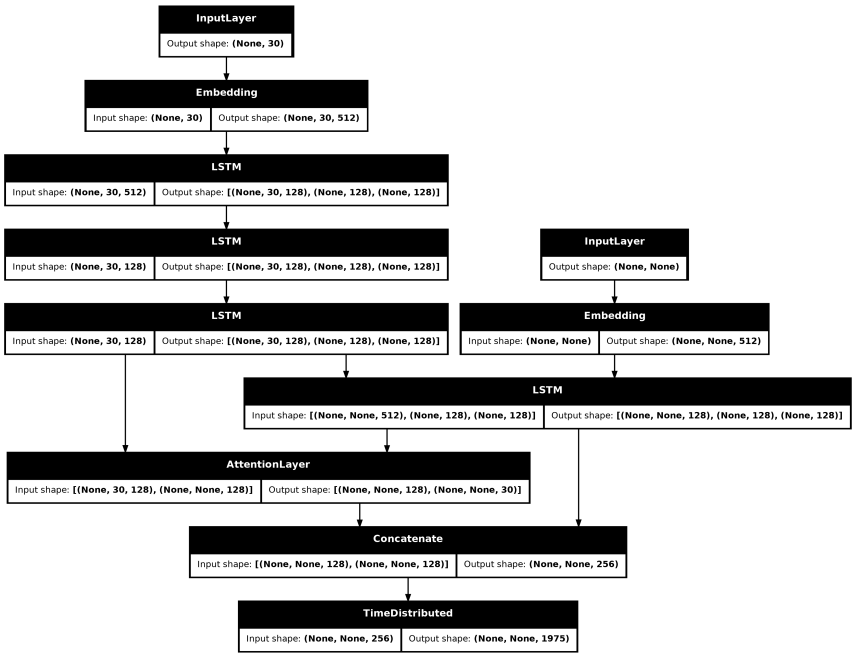


Figure 4: Architecture of the model Seq2SeqLSTM

5.6 Seq2SeqBiLSTM

The `Seq2SeqBiLSTM` class implements an architecture similar to the Seq2SeqLSTM model, but the LSTM layers in the encoder are bidirectional.

5.6.1 Training

The training of the model was carried out using the following configuration:

Parameter	Value
Optimizer	Adam
Learning rate	0.001
Embedding dimension	512
Latent dimension	256
Decoder dropout	0.2
Decoder recurrent dropout	0.2
Encoder dropout	0.2
Encoder recurrent dropout	0.2
Batch size	128
Epochs	50

Table 3: Training configuration

We can check the trend of the losses during training in Figure 5.

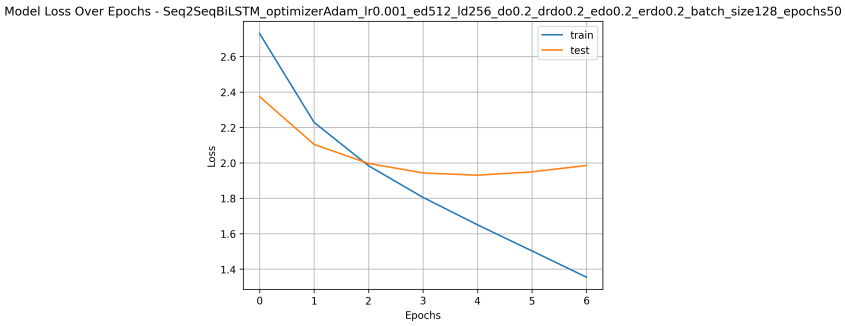


Figure 5: Trend of the *loss* and *validation loss* during training

5.6.2 Results

This model achieved the following results at the end of training:

Metric	Value
Loss	1.35
Validation Loss	1.98
Accuracy	0.70
Validation Accuracy	0.65

Table 4: Results of the model at the end of training

5.6.3 Architecture

The architecture used is as follows:

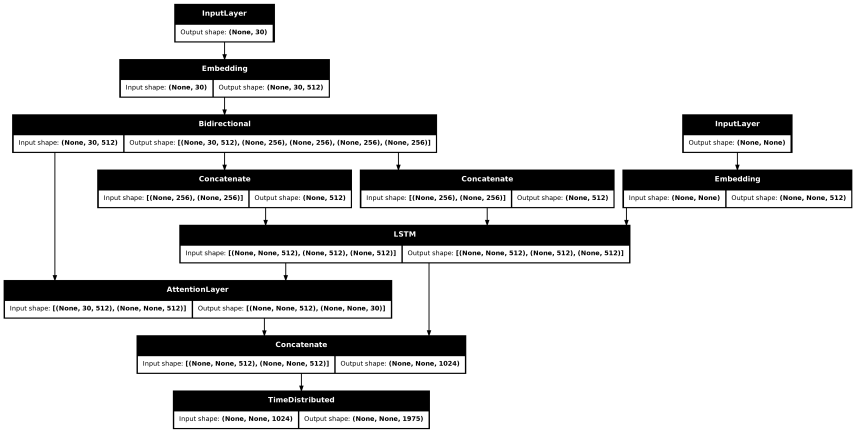


Figure 6: Architecture of the model Seq2SeqBiLSTM

5.7 Seq2Seq3BiLSTM

The `Seq2Seq3BiLSTM` class implements an architecture similar to the Seq2SeqBiLSTM model, but with three bidirectional LSTM layers in the encoder.

5.7.1 Training

The training of the model was carried out using the following configuration:

We can check the trend of the losses during training in Figure 7.

Parameter	Value
Optimizer	Adam
Learning rate	0.001
Embedding dimension	256
Latent dimension	256
Decoder dropout	0.2
Decoder recurrent dropout	0.2
Encoder dropout	0.2
Encoder recurrent dropout	0.2
Batch size	128
Epochs	50

Table 5: Training configuration

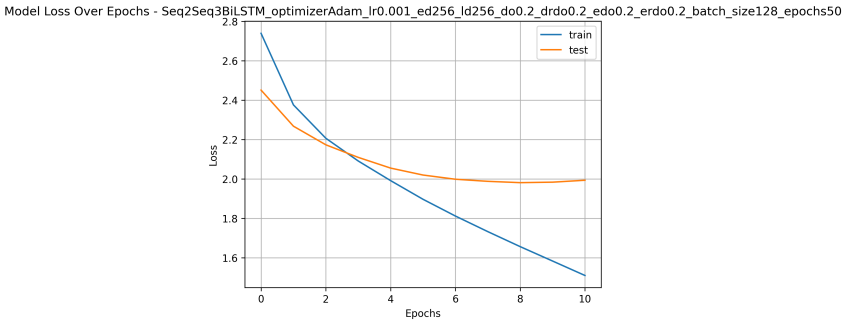


Figure 7: Trend of the *loss* and *validation loss* during training

5.7.2 Results

This model achieved the following results at the end of training:

Metric	Value
Loss	1.51
Validation Loss	1.99
Accuracy	0.68
Validation Accuracy	0.65

Table 6: Results of the model at the end of training

5.7.3 Architecture

The architecture used is as follows:

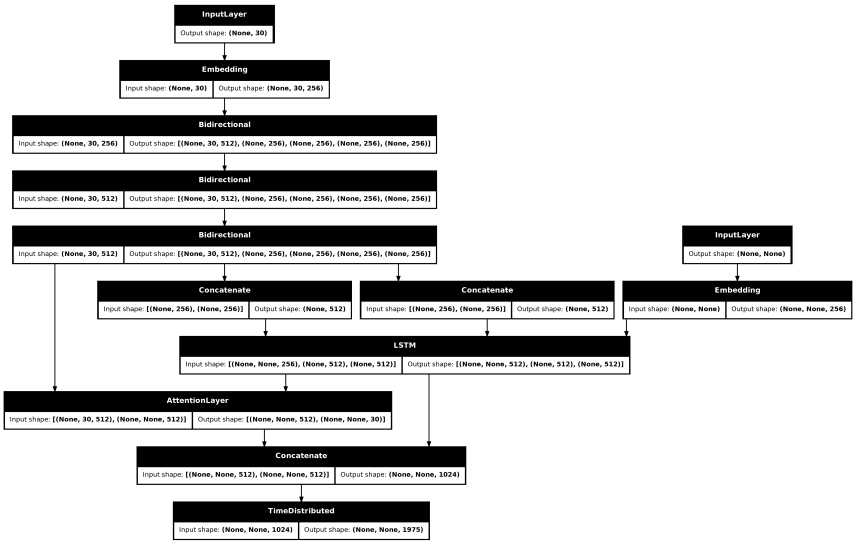


Figure 8: Architecture of the model Seq2Seq3BiLSTM

5.8 Seq2SeqGRU

The `Seq2SeqGRU` class implements a Seq2Seq architecture with GRU, using GRU layers for the encoder and decoder.

5.8.1 Training

The training of the model was carried out using the following configuration:

Parameter	Value
Optimizer	Adam
Learning rate	0.001
Embedding dimension	256
Latent dimension	128
Decoder dropout	0.2
Decoder recurrent dropout	0.2
Encoder dropout	0.2
Encoder recurrent dropout	0.2
Batch size	128
Epochs	50

Table 7: Training configuration

We can check the trend of the losses during training in Figure 9.

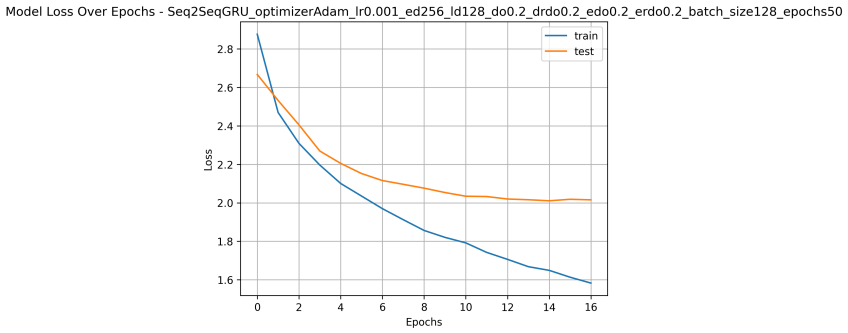


Figure 9: Trend of the *loss* and *validation loss* during training

5.8.2 Results

This model achieved the following results at the end of training:

Metric	Value
Loss	1.58
Validation Loss	2.01
Accuracy	0.67
Validation Accuracy	0.64

Table 8: Results of the model at the end of training

5.8.3 Architecture

The architecture used is as follows:

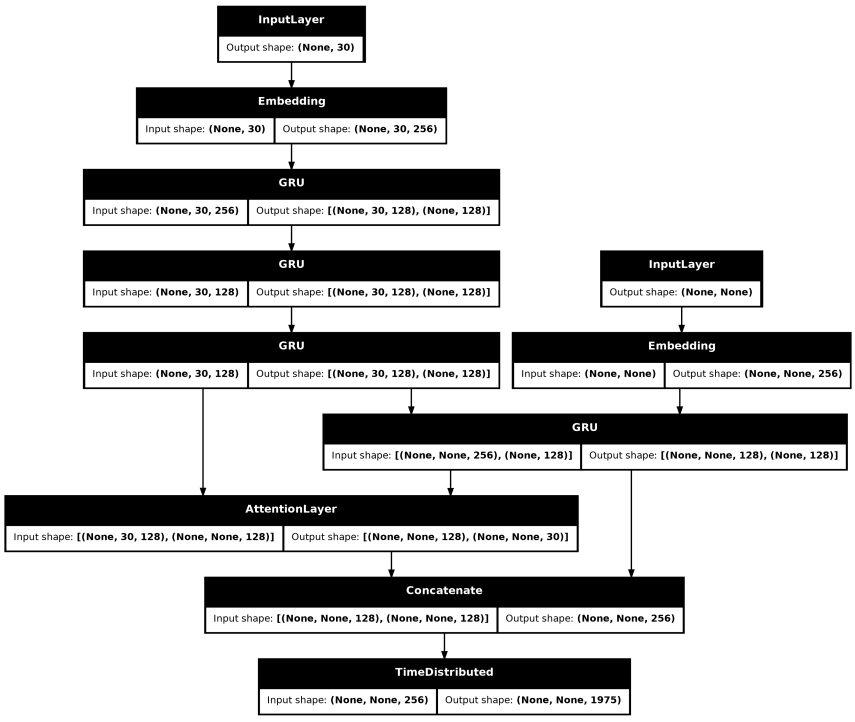


Figure 10: Architecture of the model Seq2SeqGRU

6 Evaluation Metrics

In this section, the performance of the models is analyzed through the graphs of the main evaluation metrics: ROUGE [2], cosine similarity, and BERT score [4].
Additionally, a custom metric `myevaluation` was also used, which will be discussed later.
These metrics were chosen to accurately measure the quality of the generated summaries by comparing them with the reference summaries.
The evaluations were conducted using the test dataset, which was not used during the model training phase.

6.1 Evaluation by Permutation

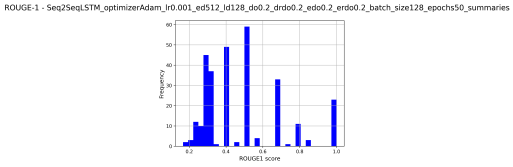
For each permutation of architectures and their respective hyperparameters, a structured evaluation process was followed at the end of the training phase. Below are the evaluation metrics used.

6.2 ROUGE (Recall-Oriented Understudy for Gisting Evaluation)

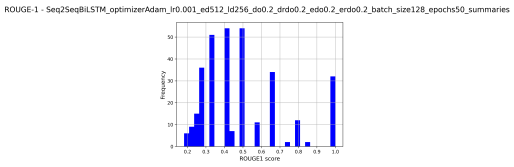
Three variants of ROUGE were calculated:

- ROUGE-1: compares unigrams between the generated summary and the reference summary
- ROUGE-2: considers bigrams to evaluate the similarity between the two texts
- ROUGE-L: compares the longest common subsequence between the two texts

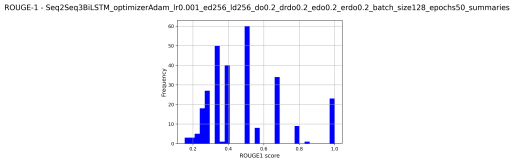
The graphs in Figure 11 compare the performance in terms of ROUGE-1, ROUGE-2, and ROUGE-L for the four models. The Seq2SeqBiLSTM model shows an improvement in ROUGE scores compared to Seq2SeqLSTM, Seq2Seq3BiLSTM, and Seq2SeqGRU, indicating a greater ability to capture lexical similarities.



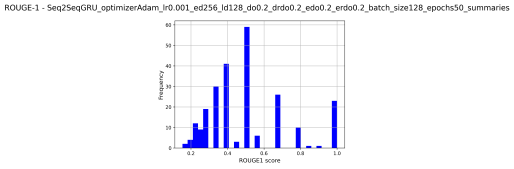
(a) ROUGE-1 Seq2SeqLSTM



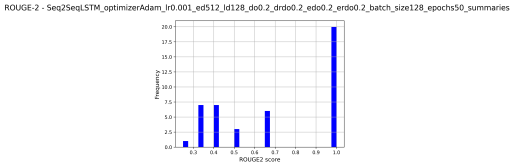
(b) ROUGE-1 Seq2SeqBiLSTM



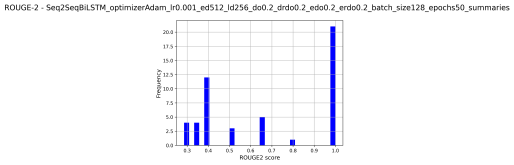
(c) ROUGE-1 Seq2Seq3BiLSTM



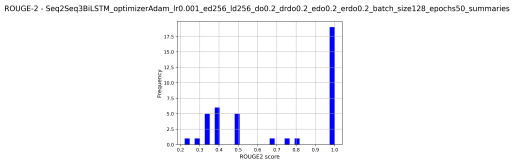
(d) ROUGE-1 Seq2SeqGRU



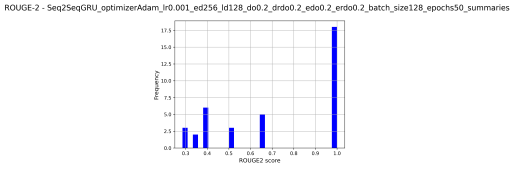
(e) ROUGE-2 Seq2SeqLSTM



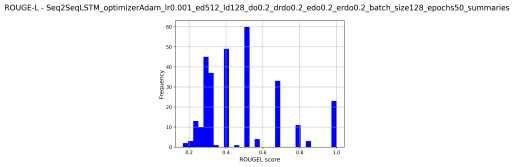
(f) ROUGE-2 Seq2SeqBiLSTM



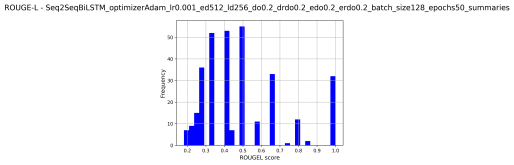
(g) ROUGE-2 Seq2Seq3BiLSTM



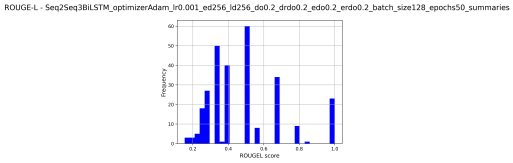
(h) ROUGE-2 Seq2SeqGRU



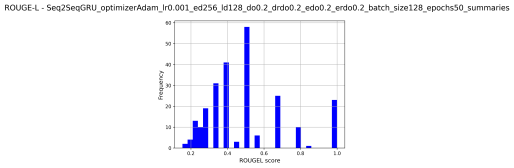
(i) ROUGE-L Seq2SeqLSTM



(j) ROUGE-L Seq2SeqBiLSTM



(k) ROUGE-L Seq2Seq3BiLSTM

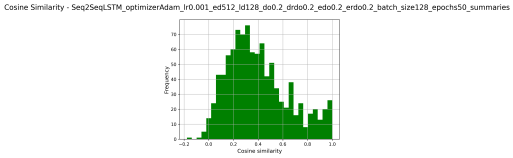


(l) ROUGE-L Seq2SeqGRU

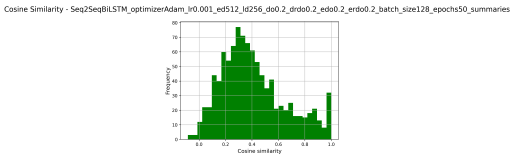
Figure 11: ROUGE scores comparison between Seq2SeqLSTM, Seq2SeqBiLSTM, Seq2Seq3BiLSTM and Seq2SeqGRU architectures.

6.3 Cosine Similarity

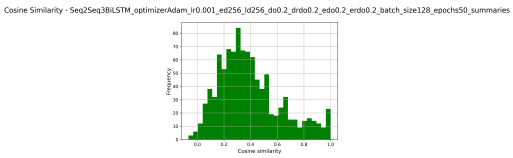
Cosine similarity is a metric that calculates the similarity between two vectors in a multidimensional space.
In the specific case of summary generation, cosine similarity was calculated between the embedding vectors of the words in the generated summaries and those in the reference summaries, in order to evaluate the quality of the generated summaries.
Figure 12 compares the cosine similarity values. Again, the Seq2SeqBiLSTM model achieves higher values, suggesting a greater semantic correlation with the reference summaries.



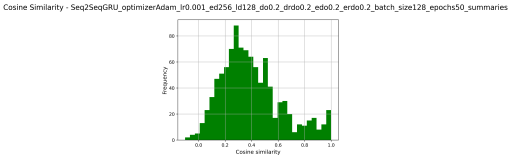
(a) Cosine similarity Seq2SeqLSTM



(b) Cosine similarity Seq2SeqBiLSTM



(c) Cosine similarity Seq2Seq3BiLSTM

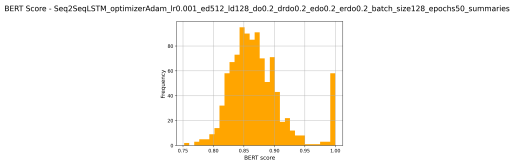


(d) Cosine similarity Seq2SeqGRU

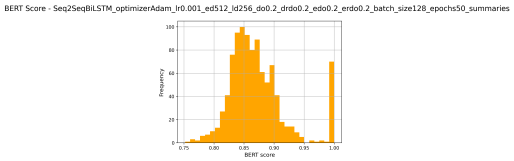
Figure 12: Cosine similarity scores comparison between Seq2SeqLSTM, Seq2SeqBiLSTM, Seq2Seq3BiLSTM and Seq2SeqGRU architectures.

6.4 BERT Score

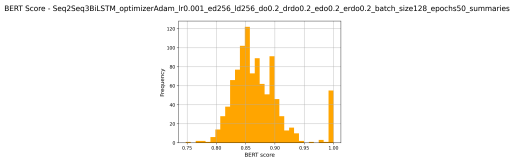
BERT score is a metric that uses the BERT model to calculate the similarity between two texts.
In particular, BERT score calculates the similarity between the embedding vectors of the words in the generated summaries and those in the reference summaries, taking into account the semantic context of the words.
Figure 13 shows the BERT score values for the different models. Again, the Seq2SeqBiLSTM model achieves better results, suggesting a greater ability to capture the semantic similarity between the generated summaries and the reference ones.



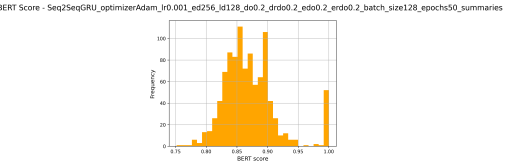
(a) BERT score Seq2SeqLSTM



(b) BERT score Seq2SeqBiLSTM



(c) BERT score Seq2Seq3BiLSTM



(d) BERT score Seq2SeqGRU

Figure 13: BERT score comparison between Seq2SeqLSTM, Seq2SeqBiLSTM, Seq2Seq3BiLSTM and Seq2SeqGRU architectures.

6.5 My Evaluation

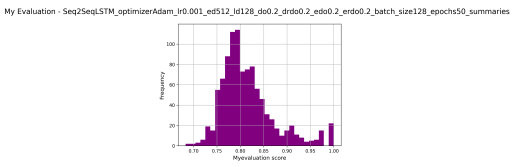
The `myevaluation` metric was designed to evaluate the quality of the generated summaries in a more detailed way.
It takes into account several factors, each of which has a specific weight based on its importance.
The formula used to calculate the `myevaluation` metric is as follows:

$$\texttt{MyEval} = cs\_PS\_OS \cdot W_{cs\_PS\_OS} + keyword\_overlap \cdot W_{keyword\_overlap} + bert\_score \cdot W_{bert\_score}$$

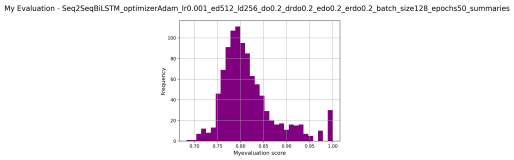
Where:

- cs_PS_OS is the cosine similarity between the generated summary and the original one
- $keyword\_overlap$ is the percentage of keywords present in the generated summary compared to those in the original summary
- $bert\_score$ is the BERT score between the generated summary and the original one
- $W_{cs\_PS\_OS}$, $W_{keyword\_overlap}$ and W_{bert_score} are the weights associated with each factor
  - $W_{cs\_PS\_OS} = 0.07$
  - $W_{keyword\_overlap} = 0.03$
  - $W_{bert\_score} = 0.9$

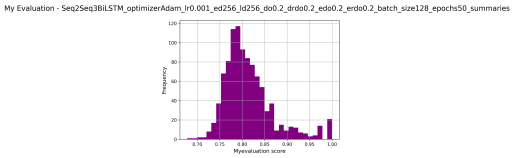
Figure 14 shows the results of the `myevaluation` metric for the different models. The Seq2SeqBiLSTM model achieves the highest scores, suggesting a superior quality of the generated summaries compared to the other models.



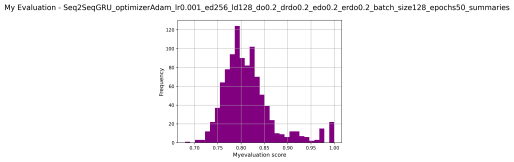
(a) My Evaluation Seq2SeqLSTM



(b) My Evaluation Seq2SeqBiLSTM



(c) My Evaluation Seq2Seq3BiLSTM



(d) My Evaluation Seq2SeqGRU

Figure 14: Comparison of the `myevaluation` metric between Seq2SeqLSTM, Seq2SeqBiLSTM, Seq2Seq3BiLSTM and Seq2SeqGRU architectures.

6.6 Architectures Comparison

Below is a comparative table (Figure 15) between the various implemented models, with their respective results obtained at the end of training.
For each model class, numerous attempts and tests were conducted, which are reported later in a comparative table of the various configuration instances and the results obtained.
We can immediately see that the Seq2SeqBiLSTM model achieved the best results in almost all metrics.

Model - Instance	mean_cosine	mean_myevaluation	mean_BERTScore	mean_rouge1	mean_rouge2	mean_rougeL
Seq2Seq3BiLSTM - Seq2Seq3BiLSTM_optimizerAdam_lr0.001_ed256_ld128_do0.2_drdo0.2_edo0.2_erdo0.2_batch_size128_epochs50_summaries	0.3917	0.8117	0.8680	0.1405	0.0264	0.1394
Seq2Seq3BiLSTM - Seq2Seq3BiLSTM_optimizerAdam_lr0.001_ed256_ld256_do0.2_drdo0.2_edo0.2_erdo0.2_batch_size128_epochs50_summaries	0.3935	0.8136	0.8702	0.1358	0.0283	0.1358
Seq2Seq3BiLSTM - Seq2Seq3BiLSTM_optimizerAdam_lr0.001_ed512_ld128_do0.2_drdo0.2_edo0.2_erdo0.2_batch_size128_epochs50_summaries	0.3944	0.8114	0.8677	0.1360	0.0308	0.1357
Seq2Seq3BiLSTM - Seq2Seq3BiLSTM_optimizerAdam_lr0.001_ed512_ld256_do0.2_drdo0.2_edo0.2_erdo0.2_batch_size128_epochs50_summaries	0.3961	0.8130	0.8694	0.1326	0.0313	0.1321
Seq2SeqBiLSTM - Seq2SeqBiLSTM_optimizerAdam_lr0.001_ed256_ld128_do0.2_drdo0.2_edo0.2_erdo0.2_batch_size128_epochs50_summaries	0.4073	0.8147	0.8695	0.1538	0.0334	0.1516
Seq2SeqBiLSTM - Seq2SeqBiLSTM_optimizerAdam_lr0.001_ed256_ld256_do0.2_drdo0.2_edo0.2_erdo0.2_batch_size128_epochs50_summaries	0.4017	0.8146	0.8701	0.1538	**0.0350**	0.1534
Seq2SeqBiLSTM - Seq2SeqBiLSTM_optimizerAdam_lr0.001_ed512_ld128_do0.2_drdo0.2_edo0.2_erdo0.2_batch_size128_epochs50_summaries	0.4014	0.8135	0.8685	0.1548	0.0335	0.1536
Seq2SeqBiLSTM - Seq2SeqBiLSTM_optimizerAdam_lr0.001_ed512_ld256_do0.2_drdo0.2_edo0.2_erdo0.2_batch_size128_epochs50_summaries	0.4107	**0.8165**	**0.8711**	**0.1596**	0.0339	**0.1589**
Seq2SeqGRU - Seq2SeqGRU_optimizerAdam_lr0.001_ed256_ld128_do0.2_drdo0.2_edo0.2_erdo0.2_batch_size128_epochs50_summaries	0.3972	0.8142	0.8707	0.1222	0.0268	0.1210
Seq2SeqGRU - Seq2SeqGRU_optimizerAdam_lr0.001_ed256_ld256_do0.2_drdo0.2_edo0.2_erdo0.2_batch_size128_epochs50_summaries	0.4049	0.8143	0.8696	0.1474	0.0310	0.1468
Seq2SeqGRU - Seq2SeqGRU_optimizerAdam_lr0.001_ed512_ld128_do0.2_drdo0.2_edo0.2_erdo0.2_batch_size128_epochs50_summaries	**0.4108**	0.8153	0.8703	0.1416	0.0310	0.1406
Seq2SeqGRU - Seq2SeqGRU_optimizerAdam_lr0.001_ed512_ld256_do0.2_drdo0.2_edo0.2_erdo0.2_batch_size128_epochs50_summaries	0.4054	0.8147	0.8698	0.1541	0.0313	0.1531
Seq2SeqLSTM - Seq2SeqLSTM_optimizerAdam_lr0.001_ed256_ld128_do0.2_drdo0.2_edo0.2_erdo0.2_batch_size128_epochs50_summaries	0.3906	0.8134	0.8703	0.1272	0.0293	0.1262
Seq2SeqLSTM - Seq2SeqLSTM_optimizerAdam_lr0.001_ed256_ld256_do0.2_drdo0.2_edo0.2_erdo0.2_batch_size128_epochs50_summaries	0.3979	0.8132	0.8693	0.1344	0.0281	0.1340
Seq2SeqLSTM - Seq2SeqLSTM_optimizerAdam_lr0.001_ed512_ld128_do0.2_drdo0.2_edo0.2_erdo0.2_batch_size128_epochs50_summaries	0.4013	0.8152	0.8710	0.1410	0.0309	0.1406
Seq2SeqLSTM - Seq2SeqLSTM_optimizerAdam_lr0.001_ed512_ld256_do0.2_drdo0.2_edo0.2_erdo0.2_batch_size128_epochs50_summaries	0.4018	0.8122	0.8678	0.1405	0.0287	0.1397

Figure 15: Comparative analysis of model instances

7 Conclusions

7.1 Final Considerations

The work carried out led to the development of an automatic summarization system capable of generating coherent summaries from food reviews. The system was designed to ensure ease of extension and adaptability to different types of tasks, thus increasing its versatility and potential applicability in varied contexts.
During the project, several automatic summarization models were implemented, including Seq2SeqLSTM, Seq2SeqBiLSTM, Seq2Seq3BiLSTM, and Seq2SeqGRU. The results were evaluated through numerous metrics, such as ROUGE, Cosine Similarity, BERT score, and `MyEvaluation`. In particular, BERT score proved to be the most representative metric of summary quality, as it considers the semantic context and similarity of words.

Furthermore, the `MyEvaluation` metric provided a more detailed assessment of summary quality, integrating various factors and assigning each a weight based on its relevance. Experimental results indicate that the **Seq2SeqBiLSTM** model achieved the best performance on almost all metrics, suggesting a superior ability to capture semantic similarities between the generated summaries and the reference ones. The **Seq2SeqBiLSTM** architecture achieved excellent results, obtaining on 1000 summaries:

Metric	Value
ROUGE-1	16%
ROUGE-2	3%
ROUGE-L	16%
Cosine similarity	41%
BERT score	87%
MyEvaluation	82%

Table 9: Evaluation metrics

It is important to underline that the results were influenced by the choice of dataset and the specific characteristics of the data, including aspects related to preprocessing. These elements highlight the importance of careful selection and

preparation of data to optimize the performance of automatic summarization models.

7.2 Testing the Best Model

The **Seq2SeqBiLSTM** model showed superior performance compared to the other implemented models.
Below are some examples of summaries generated by the **Seq2SeqBiLSTM** model and their respective texts and reference summaries.

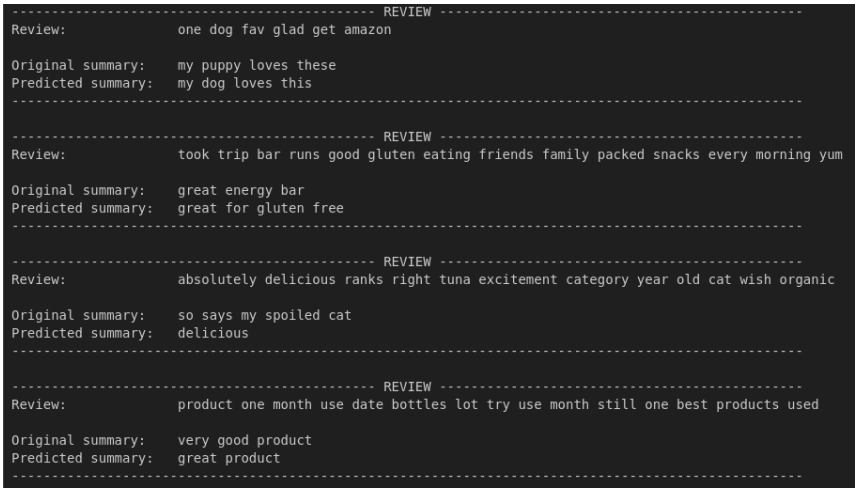


Figure 16: Example of summary generated by the Seq2SeqBiLSTM model

References

[1] *Amazon Fine Food Reviews*. https://www.kaggle.com/datasets/snap/amazon-fine-food-reviews. Visitato il: 21/05/2025. 2025.

[2] Chin-Yew Lin. "ROUGE: A Package for Automatic Evaluation of Summaries". In: *Text Summarization Branches Out*. Barcelona, Spain: Association for Computational Linguistics, July 2004, pp. 74–81. URL: https://aclanthology.org/W04-1013/.

[3] Julian McAuley and Jure Leskovec. *From Amateurs to Connoisseurs: Modeling the Evolution of User Expertise through Online Reviews*. 2013. arXiv: 1303.4402 [cs.SI]. URL: https://arxiv.org/abs/1303.4402.

[4] Tianyi Zhang et al. *BERTScore: Evaluating Text Generation with BERT*. 2020. arXiv: 1904.09675 [cs.CL]. URL: https://arxiv.org/abs/1904.09675.