**Quantifying Sentiment: A Comparative Evaluation of Language Model Performance on the Rotten Tomatoes Corpus**

**Winter 2025 CS 545 Final Project Report  
Alec Rogers, Luis Becerra, Mini Sengupta**

Contents

[Abstract 2](#_Toc191997121)

[Team members and roles 2](#_Toc191997122)

[1.0 Introduction & Research Motivation 3](#_Toc191997123)

[2.0 Literature review 3](#_Toc191997124)

[3.0 Methodology 3](#_Toc191997125)

[3.1 Dataset Evaluation 3](#_Toc191997126)

[3.2 Embedding (Alec) 3](#_Toc191997127)

[3.3 Flan (Luis) 3](#_Toc191997128)

[3.4 RoBERTa (Mini) 3](#_Toc191997129)

[3.5 Llama (Mini) 3](#_Toc191997130)

[4.0 Results and Discussion 3](#_Toc191997131)

[5.0 Conclusions and Future Work 4](#_Toc191997132)

[References 5](#_Toc191997133)

# Abstract

Sentiment analysis, the task of determining the emotional tone behind text, is a crucial application of Natural Language Processing (NLP). With the rapid advancement of Large Language Models (LLMs) and the closed nature of their design, there are many researchers attempting to develop assessments and frameworks for assessment (e.g. bias [1], x-metrics [2]) to characterize their performance. This project conducted a comparative analysis of LLMs (RoBERTa, Llama, Flan and a Custom) on the Rotten Tomatoes movie review dataset, a widely recognized resource for sentiment classification with a diverse range of expressions. By comparing their accuracy, F1-score, precision, or recall, we hope to gain insights into current LLM-driven sentiment analysis.

# Team members and roles

* Alec Rogers: Custom transformer algorithm owner and lead programmer.
* Luis Becerra: Flan algorithm owner and custom prompt engineering.
* Mini Sengupta: Bert and Llama algorithm owner and documentation lead.

# 1.0 Introduction & Research Motivation

To be filled in

# 2.0 Literature review

To be filled in

# 3.0 Methodology

The code was based on exercises found in "Hands-On Large Language Models", by Jay Alammar and Maarten Grootendorst. The transformer codebase was written with the assistance of GPT-4o.

## 3.1 Dataset Evaluation

The dataset that we used is the Rotten Tomatoes dataset (<https://huggingface.co/datasets/cornell-movie-review-data/rotten_tomatoes>). It is a database consisting of move review from the online platform Rotten Tomatoes ([https://www.rottentomatoes.com](https://www.rottentomatoes.com/))

Target Labels = {0,1}

Size of training set = 8530

Size of validation set = 1066

Size of testing set = 1066

## 

## 3.2 Transformer (Alec)

The transformer architecture can be broken down into two main parts, an embedding Layer and a transformer layer:

A diagram of a transformer model

AI-generated content may be incorrect.

To keep things simple, only one transformer layer was used: however, modern GPT networks typically have dozens of transformer layers stacked on top of one another.

## 3.2.1 Embedding

* Word2Vec represents words as vectors in a similarity space, capturing semantic meaning based on context within some corpus.
* The embedding corpus in this case comes from Wikipedia (enwiki\_20180420\_100d.txt)
* The dimensionality of the input space is 100

## 3.2.2 Positional Encoding

* Absolute sinusoidal encoding was chosen.
* As I am not a fan of conflating the data with the positional encoding, the position was used to augment the embedding space.

## 3.2.3 Attention

* In a Transformer model, the first layer weight matrix (the first self-attention layer) transforms the input embeddings (e.g., from Word2Vec) by applying linear projections followed by self-attention. Thus, a word “apple” has an embedding $ x \in \mathbb{R}^{d} $, where d is the embedding dimension.
* Transformers modify embeddings based on sentence context (e.g., “apple” in “I ate an apple” vs. “Apple Inc. is a tech company” will have different contextual embeddings), whereas Word2Vec gives fixed representations (e.g., “apple” always has the same embedding).
* Word2Vec encodes meaning based on co-occurrence statistics, while self-attention refines this by modulating embeddings dynamically based on actual sentence structure.
* Word2Vec embeddings lie in a space optimized for distributional similarity, while the Transformer projects them into a new space.

The self-attention mechanism used here is called MultiHeadSelfAttention. multiple heads allow attention to pay attention to four different things or locations within weight space. Self-attention relies crucially on the notion of Q, K, and V vectors, which are matrix projections of the input embedding space.

1. Keys Represent Encoded Memory
2. Queries Retrieve from Memory
3. Values are the actual memories

Where:

* A is the transition probability matrix, governing how information flows between tokens.
* QKT captures pairwise attention relationships in the input (or context window).
* Softmax ensures probabilities sum to 1, making attention function as a probabilistic transition process.

Having multiple attention heads allows the model to maintain and retrieve different types of information simultaneously. Each head may store and retrieve different aspects of the sequence (e.g., syntactic relationships, semantic meaning), effectively increasing the short-term memory capacity.

**Short-Term Memory Analogy**

* The keys function like a dynamically updated short-term memory that holds a contextualized representation of tokens in a sequence.
* Since attention mechanisms operate within a fixed-length context window, the stored key representations serve as a temporary memory buffer that helps the model decide which past information is most relevant to the current token.
* Unlike long-term memory (e.g. which is stored in the network weights), this short-term memory is refreshed for each new input.

**Connectionist Analogy**

In a connectionist network, knowledge is stored in the weights of connections between neurons, which dictate how activation flows between units. These weights can be interpreted as transition probabilities, determining how likely it is that activation moves from one state (or neuron) to another. In the multi-head attention mechanism of transformers, the attention matrix (A) plays a similar role by dynamically encoding how strongly different elements (tokens or states) influence each other.

The Attention matrix (A) in a transformer attention mechanism is similar to the connections within a connectionist network. By computing dynamic attention scores via QKT and normalizing them into a short-term attention matrix A, transformers generalize and improve upon connectionist models by enabling asymmetric associations (in virtue of separate Q an K matrices) and allowing context-aware transitions between states in a sequence.

## 3.2.4 Feedforward Network

The output of the embedding is typically transformed with multiple feedforward layers: here, only one layer is used. The operation of the feedforward neural network (FFN) can be written:

where:

* x and y are the input and output, respectively.
* W1, W2 are learned weights.
* b1, b2 are biases.

This network layer is just a MultiLinearPerceptron with an ReLU activation function, with an added linear output layer.

## 3.2.5 Training Parameters

Because overtraining was problematic on this dataset, a stopping criterion was introduced when the performance on the validation data set decreased by more than 0.05

Error = MSE

Embedding dimension = 100

Number of epochs = 100

Learning Rate = 0.001

Batch Size = 100

Stopping Criterion = 0.05

## 3.3 Flan (Luis)

The Flan model is described here: <https://huggingface.co/google/flan-t5-small>

To explore the LLM without having to engage in the difficult process of fine-tuning, we explored the effect of altering the prompt on the model accuracy. Here is a table of sample results using the prompt: "Is the following sentence positive or negative? "

## 3.4 RoBERTa (Mini)

The Bert model is described here: <https://huggingface.co/cardiffnlp/twitter-roberta-base-sentiment-latest>

## 3.5 Llama (Mini)

To be filled in

# 4.0 Results and Discussion

# 4.1 Overall Results

|  |  |  |  |
| --- | --- | --- | --- |
| Algori | foo | bar | Accuracy |
| Transformer | 0 | 0 | 74% |
| Flan | 0 | 0 | 84% |
| Bert | 0 | 0 | 80% |
| Llama | 0 | 0 | 0% |

**Table 1**. Average accuracy for the 4 algorithms discussed in this study. The total number of games played was 2315 and all algorithms used “SLATE” as the starting guess.

# 4.2 Transformer Results

Epoch [99/100]

Train Loss: 0.1613

Validation Loss: 0.2205

Test Loss: 0.1632

Training stopped due to overfitting.

precision recall f1-score support

Negative Review 0.76 0.70 0.73 533

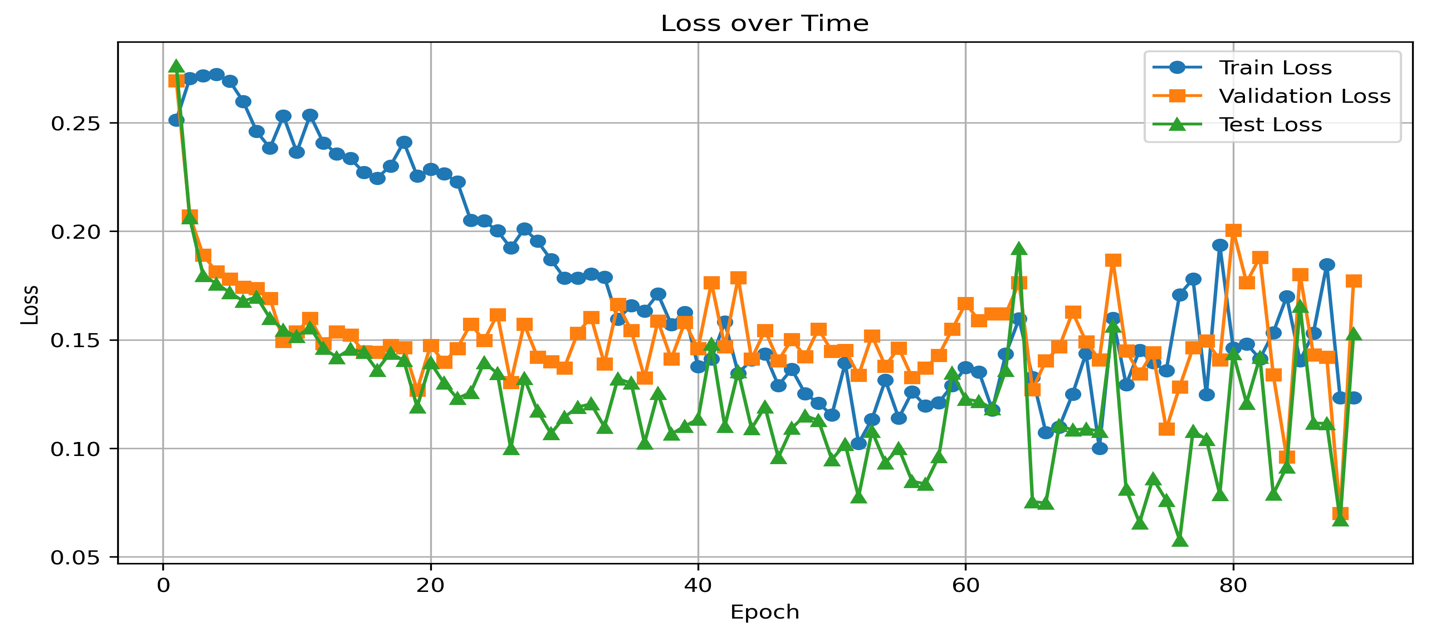
Positive Review 0.72 0.78 0.75 533

accuracy 0.74 1066

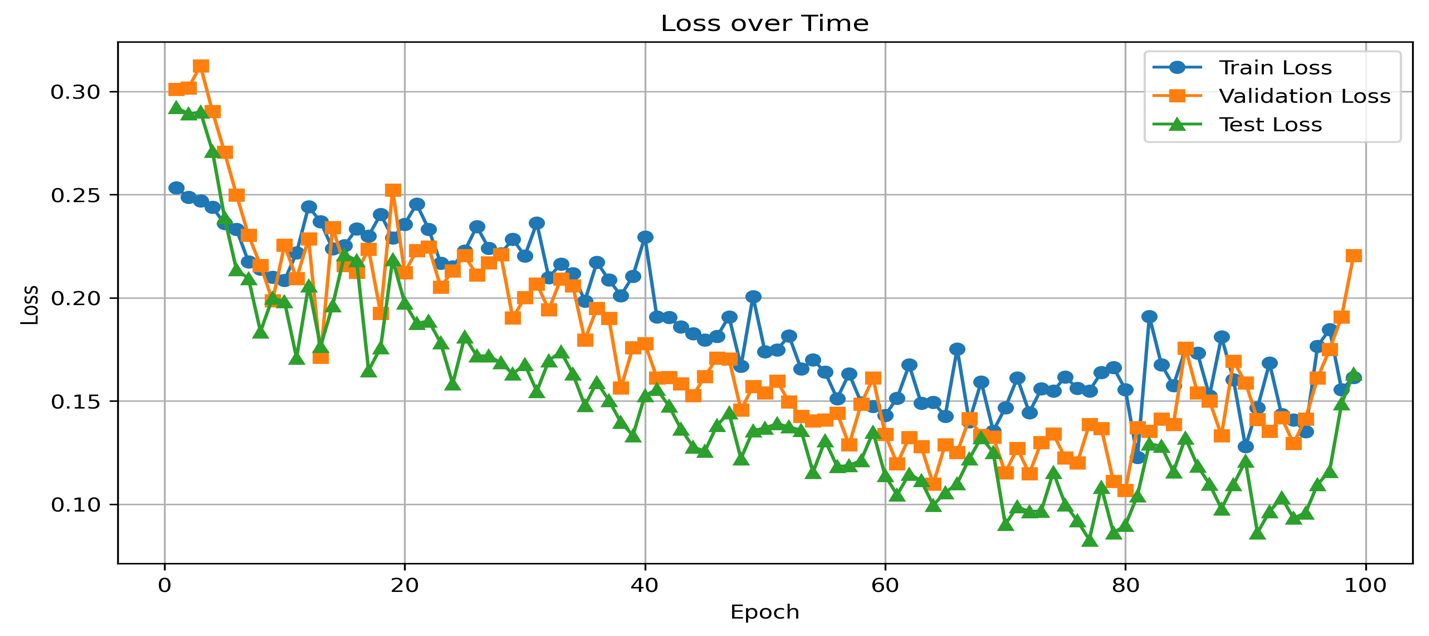
macro avg 0.74 0.74 0.74 1066

weighted avg 0.74 0.74 0.74 1066

# Figure 4.2.1 **256 hidden layer neurons**



# Figure 4.2.2 **64 hidden layer neurons**



Preventing overfitting was a main concern when tuning the algorithm parameters. The use of a validation set as a stopping criterion helped somewhat, but it also prevents training from happening for very long. Reducing the complexity of the network by using a smaller number of hidden layer neurons helped somewhat (see figures 4.2.1 and 4.2.2), although that did not prevent the network from triggering the validation stopping criterion. It also seemed to help reduce the inter-epoch variation in the data, which might also be addressed by a more intelligent schedule of learning rate decrease.

# 4.3 Flan Results

precision recall f1-score support

Negative Review 0.83 0.85 0.84 533

Positive Review 0.85 0.83 0.84 533

accuracy 0.84 1066

macro avg 0.84 0.84 0.84 1066

weighted avg 0.84 0.84 0.84 1066

# 4.4 BERT Results

precision recall f1-score support

Negative Review 0.76 0.88 0.81 533

Positive Review 0.86 0.72 0.78 533

accuracy 0.80 1066

macro avg 0.81 0.80 0.80 1066

weighted avg 0.81 0.80 0.80 1066

# 5.0 Conclusions and Future Work

Much of the work of the analysis of the review is done by the embedding itself. Anecdotally, predicting good or bad reviews simply by doing a cosine similarity and using that vector as a predictor achieves a best-in-class accuracy of 84%, which suggests that we were not able to take much advantage of any training above and beyond that embedding (which is the first step in the use of any transformer architecture).

Transformers need lots of data; training them on limited datasets adds to the difficulty of an already difficult problem. Training a transformer from scratch, as opposed to using a GPT (Generative Pre-trained Transformer), is probably not a great solution to this prediction problem since transformers generally rely on vast amounts of data, and the rotten tomatoes dataset is not large. That said, the simple transformer developed here does serve as a good reference implementation to understand the cognition of LLMs in general.

Using existing Generative Pretrained Transformers seems adequate, with careful prompt engineering.

# References

|  |  |
| --- | --- |
| [1] | J. Echterhoff, Y. Liu, A. Alessa, J. McAuley and Z. He, "Cognitive Bias in Decision-Making with LLMs," 2024. [Online]. Available: https://arxiv.org/abs/2403.00811v3. |
| [2] | M. Kahng, I. Tenney, M. Pushkarna, M. X. Liu, J. Wexler, E. Reif, K. Kallarackal, M. Chang, M. Terry and L. Dixon, "LLM Comparator: Visual Analytics for Side-by-Side Evaluation of Large Language Models," 2024. [Online]. Available: https://arxiv.org/html/2402.10524v1. |
| [3] | J. Alammar and M. Grootendorst, Hands-On Large Language Models: Language Understanding and Generation, O'Reilly Media, 2024. |