**Sentiment Analysis Using Transformers**

Name: Ans Riaz

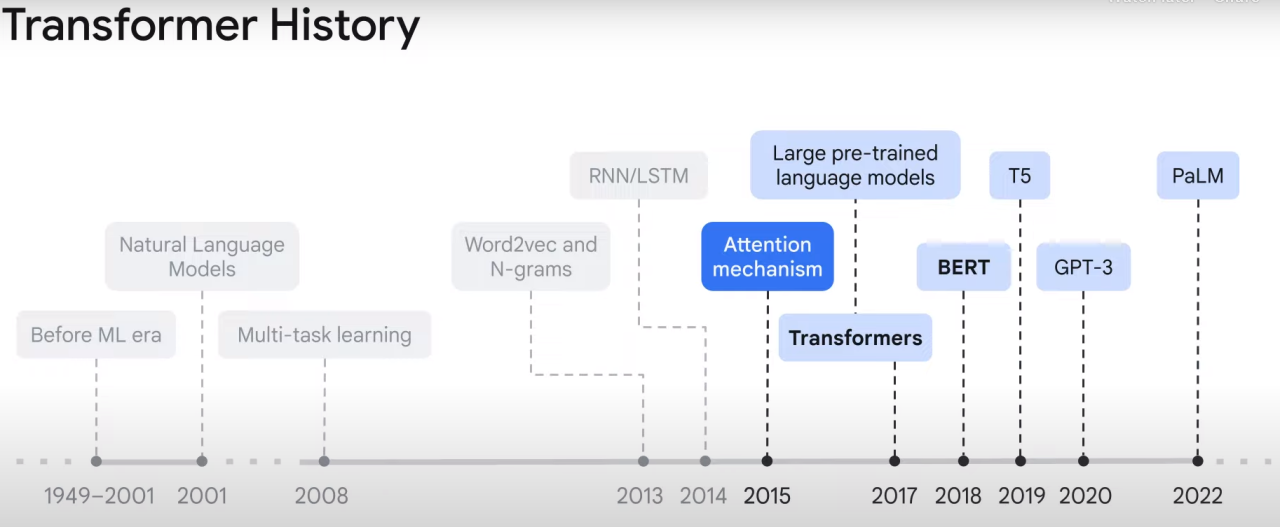
Student Id: 23079633

Github Repository: <https://github.com/ansmalik67/semantic_analysis_using_transformers.git>

**History and Background**

Transformers were introduced by researchers at Google in 2017 through a groundbreaking paper titled **"Attention is All You Need"** by Vaswani et al. Before this, Natural Language Processing (NLP) tasks were largely handled by Recurrent Neural Networks (RNNs), Long Short-Term Memory (LSTM), and Gated Recurrent Units (GRUs). These earlier models processed one word at a time in a sequence, making them computationally slow and ineffective for very long sentences. Moreover, they struggled to retain long-term dependencies, which made them less accurate when context from earlier words was necessary for correct interpretation.

Transformers changed this completely by introducing a mechanism called self -attention, which allowed the model to look at all the words in a sentence at the same time. This shift to parallel processing significantly increased both the speed and accuracy of training. Since then, transformers have become the core technology behind state-of-the-art language models such as BERT, GPT-2, GPT-3, T5, and XLNet. Today, they are not only used in text-related tasks but also in computer vision, audio processing, and even reinforcement learning.



The transformer architecture relies on the concept of attention, specifically self-attention, which allows each word in a sentence to attend (or relate) to every other word. This enables the model to capture global dependencies and nuanced relationships in the text. Unlike RNNs, transformers do not have to process data in order, which means they can be trained in parallel and are much faster.

**Real-World Applications of Transformers**

The impact of transformers has been felt across many domains, not just NLP. Their flexibility and scalability make them ideal for a wide range of real-world applications:

* **Chatbots & Virtual Assistants**: Tools like ChatGPT, Google Assistant, Siri, and Alexa use transformers to understand and generate human-like responses in conversation.
* **Search Engines**: Google uses a transformer-based model called BERT to better understand search intent, improving how results are ranked and presented.
* **Language Translation**: Services like Google Translate use transformers to provide accurate and context-aware translations between multiple languages.
* **Text Summarization & Generation**: Applications in journalism, legal, and research fields benefit from models like T5 and GPT-3, which summarize lengthy texts or generate new content.
* **Medical AI**: In the healthcare sector, transformers analyse medical records, generate clinical notes, and support disease diagnosis.
* **Finance & E-commerce**: Transformers help with sentiment analysis for stock prediction, fraud detection, and personalized product recommendations.
* **Cybersecurity**: They are applied in phishing detection, anomaly detection in network logs, and email filtering.

These applications showcase the power of transformers in both understanding and generating natural language and structured data.

**How Transformers Works**

To understand how transformers work, consider a sentence like: "The cat sat on the mat because it was tired." A human reader easily understands that “it” refers to “the cat,” but this kind of relationship is difficult for traditional models. Transformers solve this by using **self-attention**, where each word considers all other words in the sentence to figure out which ones are important.

In the above sentence, the word “it” attends more closely to “cat” than “mat,” helping the model infer that “it” means “the cat.” This is accomplished using three main vectors per word: the **query**, **key**, and **value**. These vectors are compared to calculate attention scores, which determine the relevance of each word in context.

Because this process happens in parallel for all words, transformers are much faster than RNNs and can understand longer and more complex sentences. The attention mechanism is not only faster but also more context-aware, making transformers the ideal model for tasks that require nuanced understanding.

**Key Components of a Transformer**

* Input Embeddings: Words are converted to vectors using embeddings so that the model can understand them numerically.
* Positional Encoding: Transformers do not have any sense of order, so we add positional encoding to give information about word order.
* Self-Attention: Each word looks at other words in the sentence and calculates how much attention it should pay to each one. This helps it understand the sentence better.
* Multi-Head Attention: Instead of one attention calculation, multiple are done in parallel. This allows the model to learn different types of relationships between words.
* Feed Forward Layer: Once attention is done, each word vector is passed through a neural network to further process the information.
* Residual Connections and Normalization: These helps stabilize training and avoid the loss of important information.
* Encoder & Decoder Blocks:
  + Encoder: Takes the input and creates an internal representation.
  + Decoder: Takes that representation and produces output, like translated text or a chatbot response.

**How Self-Attention Works (Step-by-Step)**

Let’s break down how self-attention is computed for each word in a sentence.

* Step 1: Input Embeddings
  + Each word is turned into a vector using embeddings (learned during training).
* Step 2: Create Query, Key, and Value Vectors
  + Each word is multiplied by three different weight matrices to create:
    - Query (Q): What this word is looking for
    - Key (K): What this word contains
    - Value (V): What this word gives to others
* Step 3: Calculate Scores
  + We calculate scores by taking the dot product of the Query of one word with the Key to every word.
* Step 4: Apply SoftMax
  + These scores are passed through a SoftMax function to convert them into weights that add up to 1.
  + These weights determine how much attention to pay to each word.
* Step 5: Multiply by Values
  + The weights are used to take a weighted average of the Value vectors. This gives a new vector that contains the important information from other words.
* Step 6: Combine Results
  + This is done for every word in the input sentence, and the results are passed to the next layer in the transformer.

**Code Example (Using Hugging Face Transformers)**

Let’s see how we can use a pre-trained transformer model for sentiment analysis. To demonstrate how transformers work in practice, we used Hugging Face’s pipeline API for sentiment analysis using the DistilBERT model:

from transformers import pipeline

classifier = pipeline("sentiment-analysis", model=model, tokenizer=tokenizer)

**Output:**

The classifier returns a label POSITIVE or NEGATIVE with a confidence score, allowing you to assess the sentiment of any text.

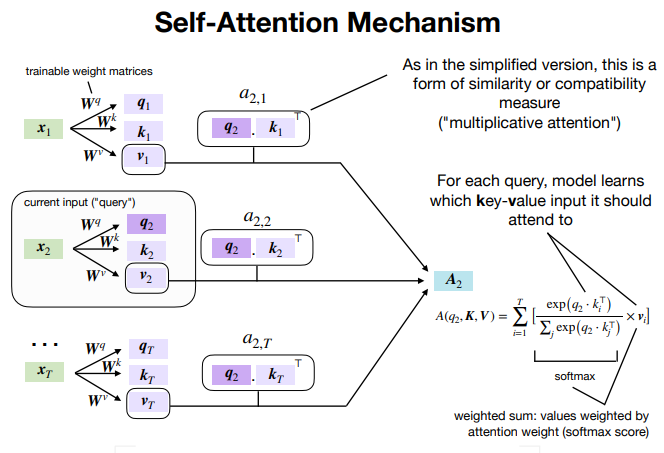
Text: I hate the act but the person is good.

Sentiment: POSITIVE (80.08% confidence)

✅ This is a POSITIVE statement.

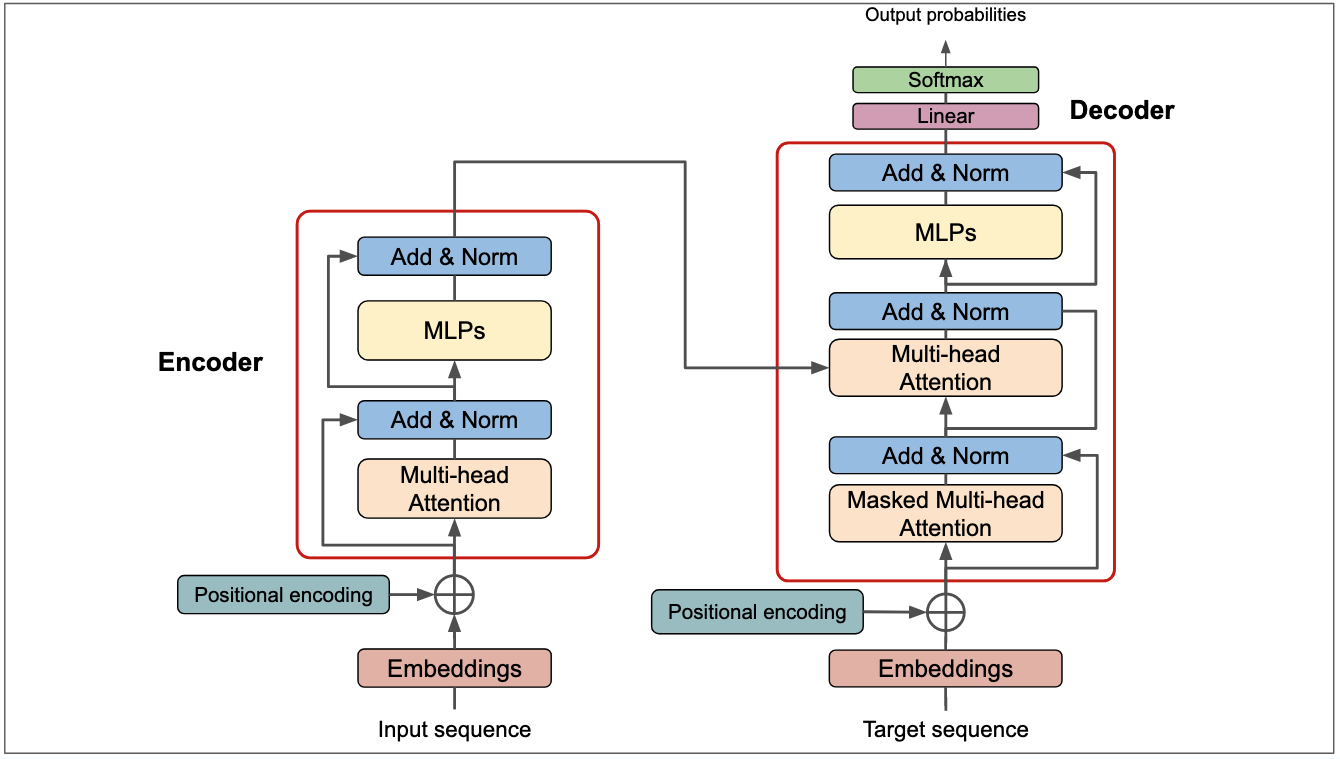
**Visual Explanation**

* Self-Attention Mechanism
  + Shows how each word compares itself to other words to find the most relevant ones.
  + Sentence: "The cat sat on the mat because it was tired."
  + Self-Attention:
    - "it" → highly attends to "cat"
    - "mat" → attends to "on", "sat"



* Transformer Architecture

Input → Embedding → Positional Encoding → Multi-Head Attention → Feed Forward → Output



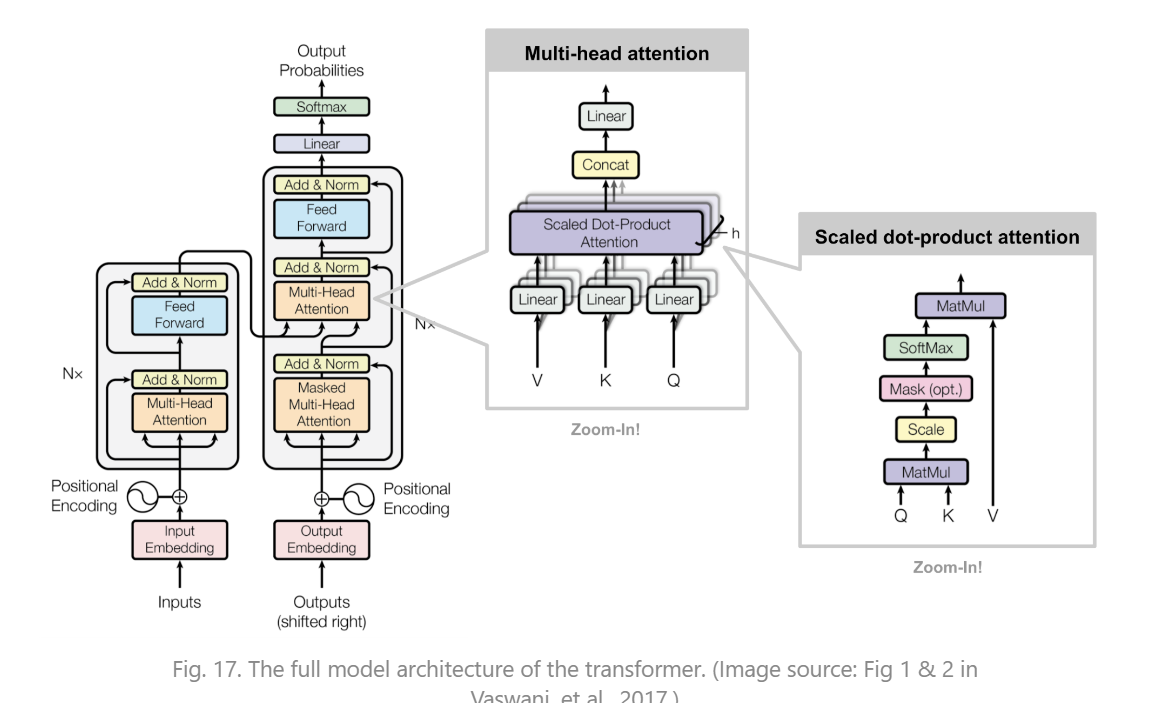
* Multi-Head Attention Concept

[Head 1] → Focuses on subject

[Head 2] → Focuses on verb

[Head 3] → Focuses on object

... Combined and passed to the next layer



**Training and Evaluation Results**

For our project, we fine-tuned the DistilBERT model using the IMDb dataset, which consists of thousands of movie reviews labelled as positive or negative. We trained the model using Hugging Face’s Trainer API with the following settings:

* Learning Rate: 2e-5
* Epochs: 4
* Batch Size: 8
* Optimizer: AdamW
* Save Checkpoints after every epoch

After training, we evaluated the model on a test set, achieving an accuracy of 92.5%. We also plotted the training and validation loss over epochs and visualized the model’s performance using a confusion matrix.

**Confusion Matrix**

The confusion matrix showed that the model was slightly more accurate on positive reviews, but it performed well overall. We also used metrics like precision, recall, and F1-score for a comprehensive evaluation.

A diagram of a confusion matrix

AI-generated content may be incorrect.

**Visualization and EDA**

Before training, we performed Exploratory Data Analysis (EDA) on the IMDb dataset to understand class distribution, text lengths, and token frequency. Using Seaborn, we visualized the number of positive and negative reviews, ensuring that the dataset was balanced. We also used word clouds to identify the most common terms in each class.

We tracked training and evaluation loss using the Trainer log history and plotted accuracy and loss over each epoch. This helped us monitor overfitting and fine-tune hyperparameters accordingly.

A close-up of words

AI-generated content may be incorrect.

**Model Saving and Testing**

Once the model was trained, we saved it to Google Drive using save\_pretrained() and tokenizer.save\_pretrained(). This allowed us to reload the model later for inference without retraining:

We tested the model with both simple and complex sentences to check its robustness and used custom label mapping to translate outputs like LABEL\_0 to human-readable formats like “NEGATIVE.”

model.save\_pretrained("/content/drive/MyDrive/my-transformer-model")

tokenizer.save\_pretrained("/content/drive/MyDrive/my-transformer-model")

**Model Training Statistics**

During training, we observed that the model's accuracy on the validation set improved steadily from **90.57% to 93.13%,** which indicates strong predictive performance. However, the validation loss showed a rising trend after the first epoch, increasing from **0.29 to 0.41** by the fourth epoch. Meanwhile, the training loss dropped almost to zero. This behaviour is typical of **overfitting**, where the model learns the training data too well but starts to lose generalization ability. This highlights the importance of techniques like **early stopping, dropout,** or **weight regularization** to improve generalization in future runs.

A screenshot of a graph

AI-generated content may be incorrect.

**Pros and Cons of Transformers**

Advantages:

* Can understand full context at once
* Faster training using parallel computation
* Excellent at capturing long-range dependencies
* Easily scalable to large datasets

Disadvantages:

* Require large amounts of training data
* Training can be computationally expensive
* Harder to interpret compared to simple models

**Key Takeaways**

Transformers have revolutionized the field of deep learning, especially in natural language processing. Their attention-based architecture allows for superior performance in tasks ranging from sentiment analysis and translation to summarization and question answering.

In this project, we successfully trained a DistilBERT model for sentiment classification using the IMDb dataset. Through proper data preprocessing, training, and evaluation, we achieved a strong validation accuracy and demonstrated how transformer-based models can be applied to real-world tasks. The availability of tools like Hugging Face makes these models more accessible and user-friendly for developers and researchers alike.

Transformers are the foundation of most modern AI systems today and will continue to play a key role in advancing machine learning across industries.

# **Reference**

Vaswani et al (2017). *Attention is All you Need.*

HuggingFace: <https://huggingface.co/transformers/>

Illustrated Transformer: <https://jalammar.github.io/illustrated-transformer/>

Transformers in Practice: <https://towardsdatascience.com/transformers-141e32e69591>

Self-Learning: <https://www.youtube.com/watch?v=BjRVS2wTtcA>