Evolution of Natural Language Processing: A Comprehensive Analysis in Sentiment Classification from Naive to Pre-trained LLMs (BERT & GPT)

## **Lakshmi Devi S**

# **ABSTRACT**

Natural Language Processing (NLP) is a pivotal domain within artificial intelligence that enables machines to understand, interpret, and respond to human language. With applications spanning from language translation to question-answering systems, NLP has become an integral part of numerous modern technologies. Among its various branches, Sentiment Analysis stands out as a significant use case, focusing on extracting emotions, opinions, and sentiments from textual data. This capability is crucial in diverse industries, including e-commerce, healthcare, and social media analytics, for better understanding customer behavior and societal trends.

In this study, we aim to explore the evolution of sentiment analysis models, tracing the journey from traditional machine learning (ML) techniques such as the Naive Bayes model, through deep learning (DL) models like Recurrent Neural Networks (RNNs) and Long Short-Term Memory networks (LSTMs), to the transformative advancements introduced by pre-trained transformer-based architectures. Specifically, we delve into transformer-based models, focusing on BERT (Bidirectional Encoder Representations from Transformers) as an encoder-based model and GPT (Generative Pre-trained Transformer) as a decoder-based model.

The project involves both qualitative and quantitative analyses to evaluate the performance of these models across the same IMDB dataset for sentiment classification. Key metrics such as accuracy, precision, recall, and F1 score will be compared to highlight the advantages and limitations of each approach. Additionally, the qualitative aspect of the analysis explores the long-range dependencies, parallel processing, interpretability and contextual understanding of these models with a comparison of word context understanding from TD-IDF, Word2Vec, Word piece tokenization in BERT and BPE in GPT is relatively studied.

Through this comprehensive study, we aim to provide a detailed comparison of traditional, deep learning, and large language models (LLMs), showcasing their strengths and limitations in sentiment analysis tasks. The findings from this work will underline the progressive advancements in NLP and offer insights into selecting the most suitable approaches for sentiment classification in practical applications.

# **1.INTRODUCTION**

**1.1 EVOLUTION OF NATURAL LANGUAGE PROCESSING**

The history of NLP begins with the early **rule-based and symbolic systems** from the 1950s to the 1980s. Alan Turing's seminal 1950 paper, Computing Machinery and Intelligence (Turing, 1950), introduced the "Turing Test," laying the foundation for machine language understanding. In 1957, Noam Chomsky revolutionized linguistics with his book, Syntactic Structures (Chomsky, 1957), which formalized grammar and syntax in computational terms. In 1966, Joseph Weizenbaum's ELIZA (Weizenbaum, 1966) became one of the earliest chatbots using pattern-matching rules. Terry Winograd's SHRDLU system ([Winograd, 1972](http://hci.stanford.edu/winograd/shrdlu/)) in the 1970s showcased how rule-based systems could handle restricted linguistic contexts in micro-worlds.

The **1980s and 1990s** marked the transition to **statistical NLP**, driven by increasing computational power and data availability. This era introduced probabilistic models like Hidden Markov Models (HMMs) for speech and text. A key paper, A Statistical Approach to Machine Translation (Brown et al., 1990), presented the IBM Models, which were groundbreaking for statistical machine translation. Church's 1988 paper, A Stochastic Parts Program and Noun Phrase Parser (Church, 1988), applied probabilistic parsing to text processing, and Jelinek’s Statistical Methods for Speech Recognition ([Jelinek, 1997](https://www.springer.com/gp/book/9780262100663)) established HMMs for speech tasks.

The **2000s** saw the rise of **feature-based machine learning** approaches, where algorithms like SVMs, Naive Bayes, and Maximum Entropy models became popular. These methods relied on handcrafted features and dominated tasks like Named Entity Recognition (NER) and document classification. Blei et al.’s Latent Dirichlet Allocation (Blei et al., 2001) introduced probabilistic topic modeling. Bengio's Neural Probabilistic Language Model (Bengio et al., 2003) pioneered early neural network-based word embeddings, paving the way for distributed representations of words.

The **deep learning revolution** in the 2010s transformed NLP. Mikolov et al.'s Efficient Estimation of Word Representations in Vector Space ([Mikolov, 2013](https://arxiv.org/abs/1301.3781)) introduced Word2Vec, enabling efficient word embeddings. Bahdanau et al.'s Neural Machine Translation by Jointly Learning to Align and Translate ([Bahdanau et al., 2014](https://arxiv.org/abs/1409.0473)) introduced attention mechanisms, which significantly improved sequence-to-sequence tasks. Vaswani et al.’s Attention is All You Need ([Vaswani et al., 2017](https://arxiv.org/abs/1706.03762)) revolutionized NLP by introducing the Transformer architecture, which forms the backbone of today’s language models. Devlin et al.'s BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding ([Devlin et al., 2018](https://arxiv.org/abs/1810.04805)) set new benchmarks with contextual embeddings.

Finally, the **2020s** introduced **large language models (LLMs)** and **prompt engineering** as dominant paradigms. OpenAI’s GPT-3 paper, Language Models are Few-Shot Learners ([Brown et al., 2020](https://arxiv.org/abs/2005.14165)), demonstrated the power of zero-shot and few-shot learning through prompt-based approaches. Ouyang et al.'s Training Language Models to Follow Instructions with Human Feedback ([Ouyang et al., 2022](https://arxiv.org/abs/2203.02155)) refined GPT with alignment for real-world usability. Raffel et al.’s Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer ([Raffel et al., 2020](https://arxiv.org/abs/1910.10683)) introduced the T5 framework, further expanding NLP’s capabilities. This progression—from symbolic systems to statistical methods, deep learning, and prompt engineering—represents the remarkable journey of NLP as a field.

Sentiment Analysis, often referred to as **opinion mining**, is a vital application of Natural Language Processing (NLP) that focuses on analyzing and categorizing emotions, attitudes, and opinions expressed in text. By identifying whether the sentiment in a text is positive, negative, or neutral, it finds extensive use in areas such as customer feedback analysis, social media monitoring, and product reviews. Beyond sentiment analysis, NLP encompasses diverse applications like machine translation, named entity recognition, text summarization, conversational AI, and speech-to-text systems. These applications collectively enable machines to understand, interpret, and generate human language.

This study focuses on sentiment analysis, exploring its implementation using various traditional and advanced approaches. It is structured into several sessions. **Session 2** covers machine learning and deep learning models, including Naive Bayes, Recurrent Neural Networks (RNNs), Long Short-Term Memory Networks (LSTMs), Transformers, and Large Language Models (LLMs) like GPT and BERT, elaborating on their architectures and applications. **Session 3** takes to the code pipeline elaborating the EDA and data preparation for ML to LLM models. **Session 4** explains the model implementations in sentiment analysis, from data preprocessing to model deployment and evaluation. **Session 5** delves into qualitative and quantitative analyses of model performance along with relative word representation between models, while **Session 6** presents key findings and insights. **Session 7** discusses the future scope of NLP and sentiment analysis, addressing emerging trends and challenges, and **Session 8** concludes the study by summarizing the results and their significance. This comprehensive structure provides a detailed understanding of sentiment analysis, its methodologies, and its growing importance in the NLP domain.

# 2**. MACHINE LEARNING TO TRANSFER LEARNING LLMs: ARCHITECTURAL INSIGHTS FOR NLP IN SENTIMENT ANALYSIS.**

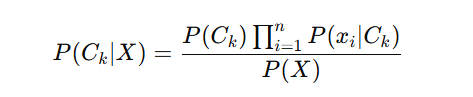
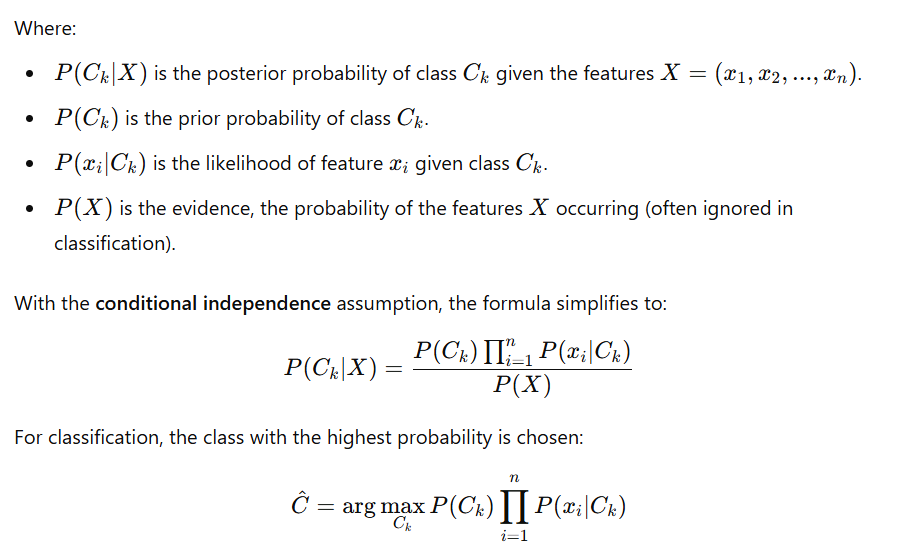
This session focuses on exploring different models and architectures used for sentiment analysis, starting with traditional machine learning techniques and progressing toward advanced deep learning methods. The session covers how these models evolve in terms of handling text-based data, from basic probabilistic models to neural networks and transformers. Additionally, it introduces how transfer learning and pre-trained Large Language Models (LLMs) are transforming sentiment analysis with state-of-the-art accuracy and efficiency.

The session begins with **Naive Bayes**, a foundational algorithm widely used for text classification tasks, which assumes feature independence and relies on probability theory. Next, it delves into **Recurrent Neural Networks (RNNs)**, which are designed to capture sequential patterns in data, making them well-suited for textual analysis. However, RNNs suffer from vanishing gradient problems when dealing with long text sequences. To address this, **Long Short-Term Memory Networks (LSTMs)**, a special kind of RNN, are introduced for their ability to retain long-term dependencies effectively.

Building on this, the session transitions to **Transformer-based models**, which revolutionized NLP by introducing attention mechanisms, enabling models to focus on relevant parts of the input text while processing. Unlike RNNs and LSTMs, transformers process sequences in parallel, drastically improving efficiency and scalability. This leads to the discussion of **Transfer Learning**, a paradigm where models pre-trained on massive corpora, such as **BERT (Bidirectional Encoder Representations from Transformers)** and **GPT (Generative Pre-trained Transformers)**, can be fine-tuned for specific sentiment analysis tasks. These Large Language Models (LLMs) have set new benchmarks in NLP, leveraging the transformer architecture for contextual understanding and generalization.

2.1 Naïve Bayes Architecture in SA

Naive Bayes is one of the simplest yet effective machine learning algorithms for text classification tasks, including sentiment analysis. It operates on the principle of Bayes' Theorem and assumes that the features of the input data are independent of each other. Despite this "naive" assumption of feature independence, the algorithm has proven to be highly effective in practice, particularly for text classification tasks like spam detection, email filtering, and sentiment analysis.

In sentiment analysis, Naive Bayes is used to classify a given text as having a positive, negative, or neutral sentiment. The core idea is to compute the probability of each sentiment category given the input text and assign the category with the highest probability to the text. The algorithm is based on Bayes’ theorem, which is expressed as:

For sentiment analysis, corresponds to the sentiment labels (e.g., positive, negative, neutral), and corresponds to the words or tokens in the input text.

The key assumption of Naive Bayes is that the features (words) in the input are conditionally independent of each other given the class label. In other words, the presence of one word in the input text does not affect the presence of another word. This simplifies the computation of, as:

Where are the individual words (or features) in the input text. This assumption reduces the computational complexity, making the algorithm highly efficient for large datasets. However, it also introduces limitations since real-world text data often contains dependencies between words.

The architecture of Naive Bayes for sentiment analysis can be described in the following steps. Before applying Naive Bayes, the raw text needs to be transformed into a numerical representation. This typically involves tokenization (splitting the text into individual words or tokens), stop word removal (removing common words like "and," "the," "is" which do not contribute much to sentiment), stemming or lemmatization (reducing words to their root forms such as "running" → "run"), and vectorization (converting the text into numerical features using techniques like Bag of Words (BoW) or Term Frequency-Inverse Document Frequency (TF-IDF)).

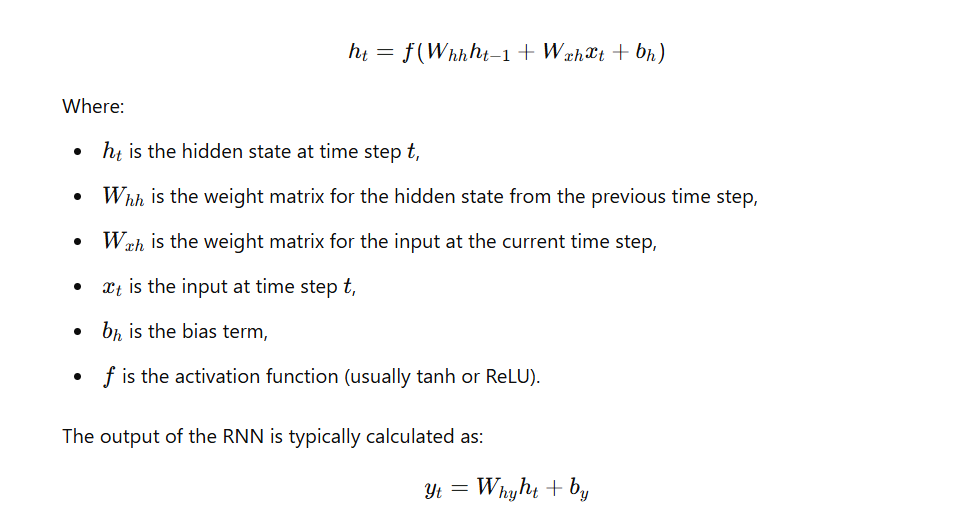
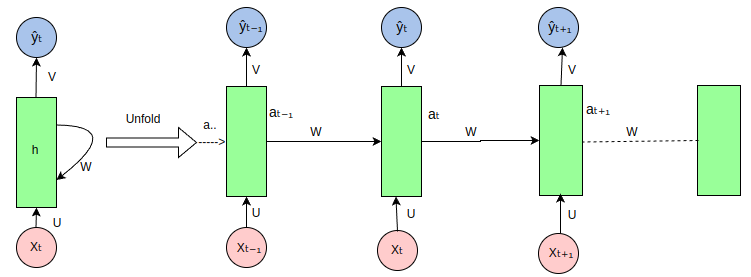
The prior probability for each sentiment class is computed based on its occurrence in the training dataset: For instance, if 60% of the training data is labeled as "positive," then. The likelihood represents the probability of a word appearing in a given class. This is calculated for every word in the vocabulary: Here, the numerator is the count of the word in class, with Laplace smoothing (+1) to handle words that may not appear in the training data for a particular class. The denominator is the total count of all words in class plus the vocabulary size, ensuring normalization. Using the prior probabilities and likelihood, the posterior probability for each class is computed:

The class with the highest posterior probability is assigned as the sentiment label for the input text: There are three common variations of Naive Bayes, based on how the input features are represented. Multinomial Naive Bayes is commonly used for text classification and assumes word frequencies as features, making it suitable for Bag of Words (BoW) or Term Frequency (TF) representations. Bernoulli Naive Bayes assumes binary features (presence or absence of a word) and is suitable for binary text representations. Gaussian Naive Bayes assumes features follow a Gaussian (normal) distribution but is rarely used for text data and is more applicable to continuous numerical features.

Naive Bayes has several advantages in sentiment analysis. It is computationally efficient, works well with large datasets, is easy to understand and implement, and often serves as a strong baseline model in NLP tasks. However, it also has limitations. The assumption that words are independent can lead to inaccurate probability estimates in real-world text, where word dependencies exist. Naive Bayes may also struggle with imbalanced datasets where one class significantly dominates the others, and high-dimensional data (like text) can sometimes degrade performance if the training data is insufficient.

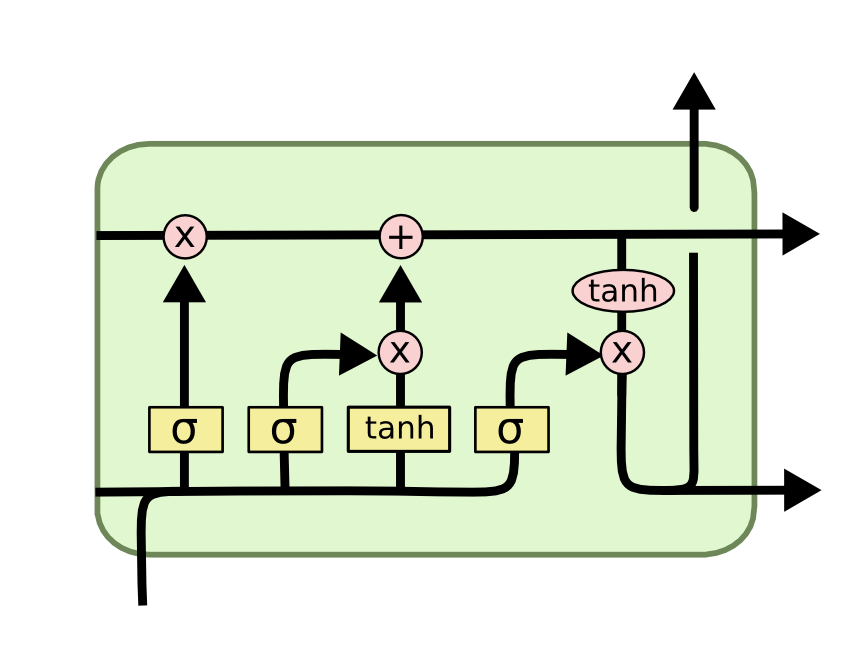
Naive Bayes is a foundational model for sentiment analysis, offering simplicity and efficiency in processing text data. While its assumptions may not fully capture the complexities of natural language, it provides a strong starting point for sentiment classification. In modern NLP, it is often used as a benchmark model before transitioning to more advanced approaches like RNNs, LSTMs, and transformers. Despite its limitations, Naive Bayes remains a vital part of the evolution of sentiment analysis.

**2.2 RNN Architecture in SA**

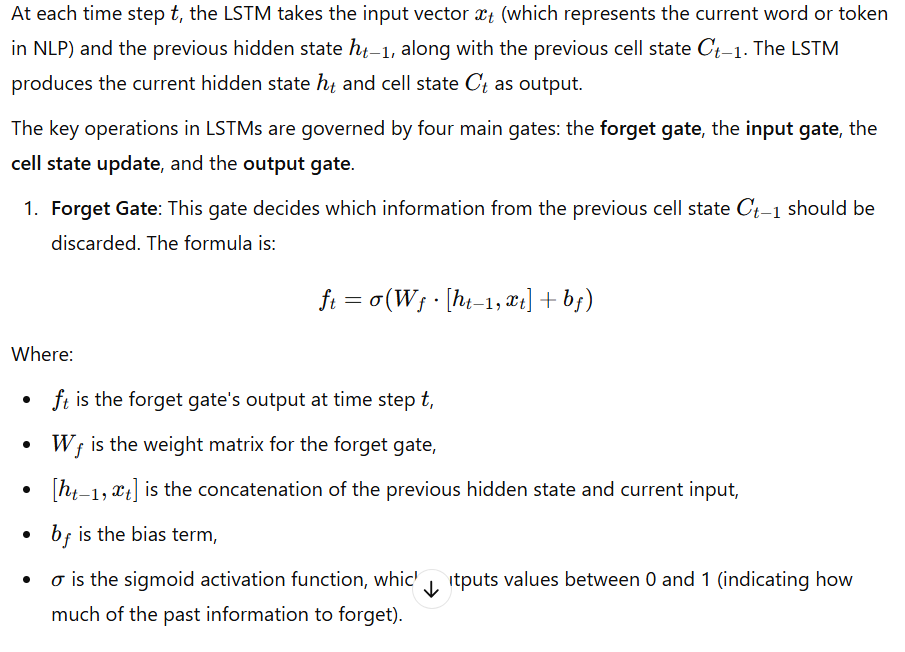
Naive Bayes struggles when dealing with rare or unseen words, as it depends heavily on frequency counts on the training data and missing words are treated as zero frequency which misleads the classification. Recurrent Neural Networks (RNNs) address these limitations by maintaining a hidden state that evolves as they process each word in a sequence, allowing them to capture dependencies and relationships between words based on their context. The formula

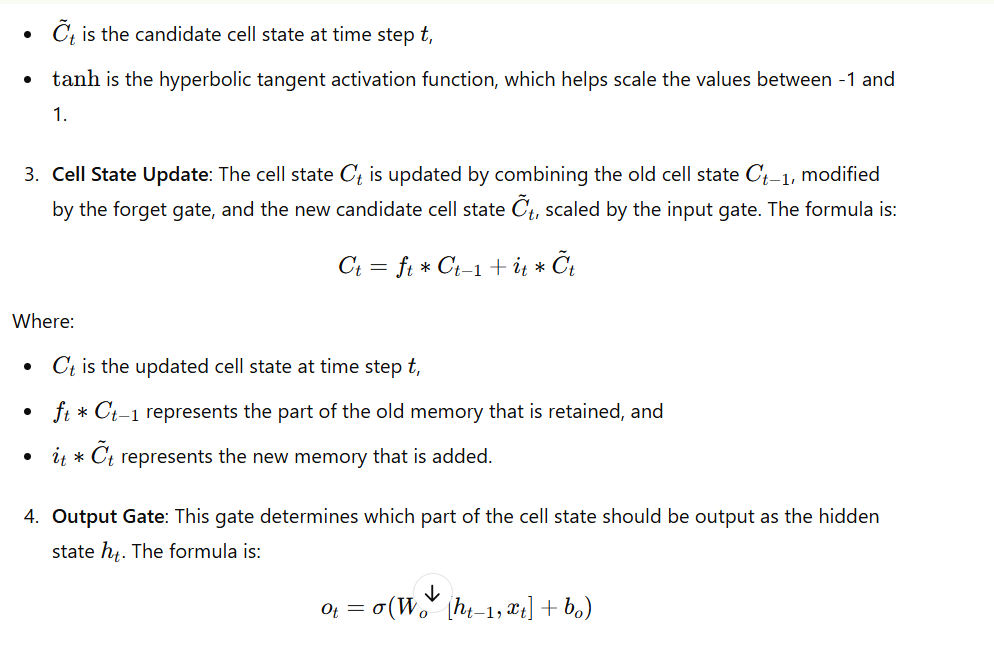
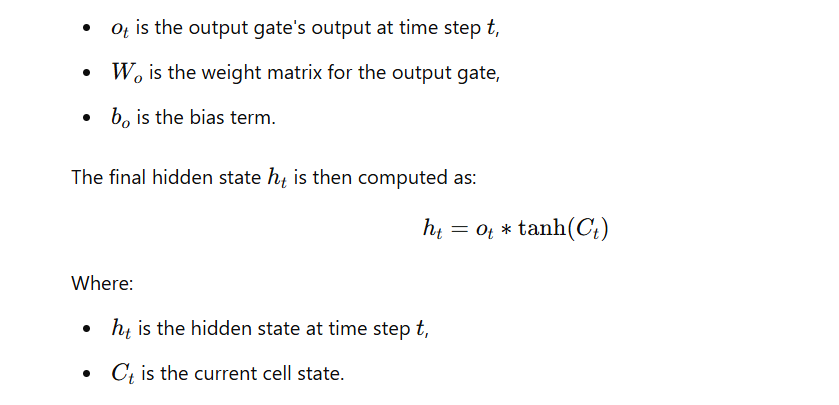
 the output layers. This allows RNNs to effectively model sequential data by capturing the relationships between words and their contexts over time. Unlike Naive Bayes, RNNs understand the order of words and can handle variable-length input, making them more suitable for tasks like sentence classification and machine translation. For word representation, Word2Vec provides a significant improvement over BoW and TF-IDF. BoW creates sparse vectors where word order is ignored, and TF-IDF adjusts word importance but still doesn't capture word context. Word2Vec, on the other hand, learns dense, continuous word vectors that reflect semantic meaning, placing similar words close to each other in the vector space based on their co-occurrence patterns in a corpus. This allows Word2Vec to capture relationships between words, such as analogies (e.g., "king" - "man" + "woman" = "queen"), unlike BoW or TF-IDF which cannot understand these relationships. Therefore, RNNs and Word2Vec together address the limitations of Naive Bayes by providing context-aware learning and representations, enabling more effective modeling of language.

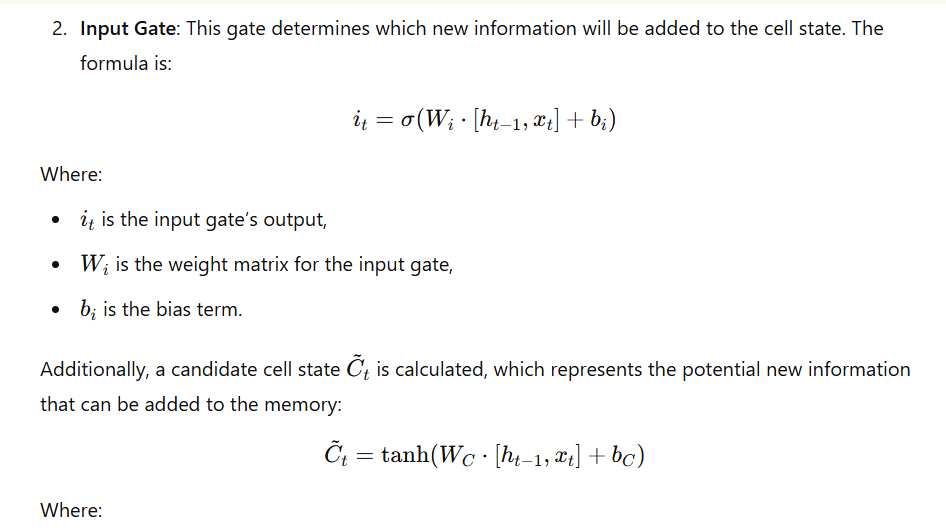
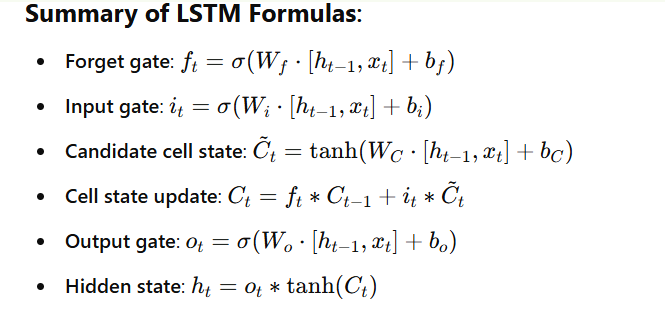
However, RNNs also have their own limitations. One key challenge is the **vanishing gradient problem**, where gradients become very small as they are propagated back through long sequences, making it difficult for the model to learn long-term dependencies. Additionally, **RNNs are computationally expensive** due to their sequential nature, which makes them slower to train compared to models that can process data in parallel. RNNs also struggle with capturing long-range dependencies in sequences, particularly when the relevant information is far apart. This is where **LSTMs (Long Short-Term Memory)** networks become crucial. LSTMs were specifically designed to overcome these limitations by introducing mechanisms to better retain long-term dependencies, allowing them to preserve important information over longer sequences. Their ability to handle vanishing gradients and remember information over long periods has made them a cornerstone of many NLP tasks, especially when dealing with long sequences of text.

**2.3 Long Short-term memory (LSTM) Architecture in SA**LSTMs are a type of Recurrent Neural Network (RNN) designed to address some of the limitations of traditional RNNs, particularly the vanishing gradient problem. They are capable of capturing long-term dependencies and remembering information over long sequences, which is particularly useful for NLP tasks such as machine translation, text generation, and speech recognition.

LSTM units consist of a set of gates that regulate the flow of information into and out of the memory cell, which allows the network to decide what information to retain or forget at each time step. The architecture includes the **forget gate**, **input gate**, **cell state**, and **output gate**. Below is an explanation of each of these components with the corresponding formulas.

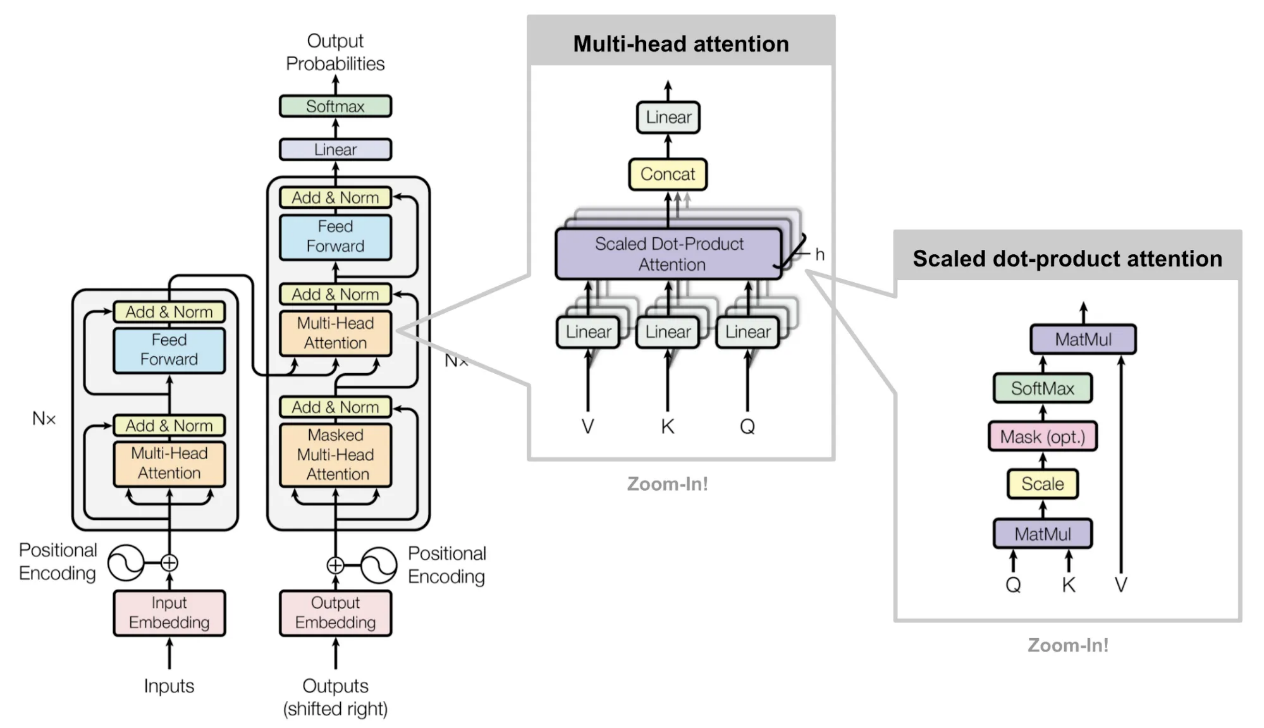




LSTMs are designed to address the vanishing gradient problem in standard RNNs. The gates in an LSTM control how much information from previous time steps should be forgotten, remembered, or updated, making them capable of learning long-term dependencies. This feature is particularly important in NLP, where context from earlier parts of a sentence or document can be crucial for tasks like machine translation or text generation. By incorporating these mechanisms, LSTMs are able to maintain and propagate important information over long sequences without it being lost.

The **Seq2Seq** model, introduced by **Google** in 2014, used an encoder-decoder architecture for sequence-to-sequence tasks like machine translation. Later in 2014, **Bahdanau et al.** introduced the **attention mechanism**, which enhanced Seq2Seq by allowing the model to focus on different parts of the input sequence at each time step, rather than relying on a fixed-size context vector. These innovations laid the foundation for the **Transformer** architecture (2017), which removed recurrence and relied entirely on attention, leading to faster training and better performance in NLP tasks.

**2.3 Transformer Architecture in SA**

The **Transformer architecture**, introduced by Vaswani et al. in 2017, consists of two main parts: the **encoder** and the **decoder**. The encoder uses a self-attention mechanism to process the entire input sequence in parallel, allowing each word to focus on every other word in the sequence, followed by a feed-forward neural network. The decoder similarly uses self-attention and includes an additional encoder-decoder attention layer to focus on the encoder’s output while generating the sequence. Unlike LSTMs, which process input sequentially, Transformers enable parallelization, making them faster and more efficient, while also better at capturing long-range dependencies through attention mechanisms. This shift improves scalability and performance on long sequences. Variations like **BERT**, which uses only the encoder and processes text bidirectionally, **GPT**, which uses only the decoder for autoregressive tasks, and **T5**, which frames all tasks as text-to-text, demonstrate how Transformers can be adapted for a variety of NLP tasks, outperforming LSTMs in both flexibility and effectiveness.

The **encoder** of a Transformer is designed to process the input sequence and generate contextualized representations for each token. It consists of multiple identical layers, each with two main components: a **multi-head self-attention mechanism** and a **position-wise feed-forward network (FFN)**. The self-attention mechanism captures relationships between tokens in the input sequence, regardless of their positions, by computing attention scores. Positional encodings are added to the input embeddings to provide positional information since Transformers lack inherent sequential awareness. After self-attention, the FFN processes the outputs of the attention mechanism for each token independently. Each layer incorporates **residual connections** and **layer normalization** to stabilize training and facilitate gradient flow.

The **decoder**, on the other hand, generates the output sequence token by token in an autoregressive manner. Like the encoder, it is composed of multiple layers but with three key components: **masked multi-head self-attention**, **encoder-decoder attention**, and a **position-wise FFN**. The masked self-attention prevents the decoder from accessing future tokens, ensuring causal generation. The encoder-decoder attention enables the decoder to focus on relevant parts of the encoder's output for each target token. Shifted target embeddings are used as inputs to the decoder, and positional encodings are added to preserve the order of tokens. Each layer of the decoder also uses residual connections and layer normalization, ensuring effective learning and smooth gradient propagation. Together, the encoder and decoder work seamlessly for tasks like machine translation and text generation.

**2.4 Transfer Learning and Large Language Model Era**

Transfer learning is a technique where a model is first pretrained on a large, generic dataset to learn general features, and then fine-tuned on a smaller, task-specific dataset. In NLP, this approach became feasible with the advent of Transformers, which use attention mechanisms to understand relationships between words in a sequence. Transformers, with their ability to process sequences in parallel, allowed large-scale pretraining on massive corpora, enabling models to learn rich language representations that could be reused across tasks like translation, summarization, and sentiment analysis (SA).

BERT (Bidirectional Encoder Representations from Transformers) is a pretrained Transformer model that uses **masked language modeling (MLM)** for pretraining, allowing it to learn the context of words from both directions in a sentence. BERT can be fine-tuned for various tasks like SA by adding a classification layer on top of its encoder architecture. GPT (Generative Pretrained Transformer), on the other hand, uses a **causal language model** for generating text in an autoregressive manner and excels in tasks requiring text generation. T5 (Text-to-Text Transfer Transformer) unifies NLP tasks into a text-to-text format, such as converting input text into a sentiment label for SA. All these models leverage their pretrained knowledge of language to handle tasks like SA efficiently with minimal fine-tuning.

Recent advancements have led to the development of Large Language Models (LLMs) like GPT-3, GPT-4, and BLOOM, which contain billions of parameters and are pretrained on diverse, massive datasets. These LLMs perform remarkably well in tasks like question answering, summarization, and creative writing through **few-shot and zero-shot learning**, requiring little to no task-specific training. Their ability to generalize across tasks without fine-tuning has revolutionized AI applications, powering use cases like virtual assistants, code generation, and real-time translation, making them significant for industries, research, and day-to-day applications. This study focuses on **fine-tuning** the GPT-2 model for a sentiment analysis task using the IMDB movie review dataset. The pre-trained GPT-2 model is adapted to classify movie reviews as either positive or negative by updating its weights through training on a labeled dataset. The dataset is pre-processed by tokenizing the reviews and mapping the sentiment labels to numerical values (positive = 1, negative = 0). The model is then trained using Hugging Face's Trainer API, with the dataset split into training and testing sets, and the model's performance is evaluated after training.

By fine-tuning the GPT-2 model in this manner, the study tailors the pre-trained language model to a specific downstream task, enabling it to perform better on sentiment classification by adjusting the model's internal parameters. This fine-tuning approach allows the model to leverage the knowledge it gained during its pre-training on vast amounts of text, while also learning the specific patterns related to sentiment in the movie reviews dataset.

In the next session, we will explore how these architectures—Naive Bayes, Deep Learning models, Transformer-based models like BERT, GPT, and T5, as well as modern LLMs—can be applied to Sentiment Analysis. This practical approach will demonstrate their strengths, limitations, and effectiveness in understanding and classifying text sentiment, setting the stage for hands-on implementation and experimentation.

# **3. PIPELINE FOR EDA AND DATA PREPARATION FOR ML TO LLMs**

The pipeline for Exploratory Data Analysis (EDA) and data preparation is a critical process for preparing data for downstream tasks in both Machine Learning (ML) models and Large Language Models (LLMs). The IMDB movie review dataset, downloaded from Kaggle, was used as the basis for this preparation. This dataset contains labeled text reviews categorized as positive or negative, offering an excellent opportunity to analyze textual data and build models that can classify sentiments effectively. The process involved collecting and loading the dataset, cleaning it to ensure consistency, performing detailed exploratory analysis to uncover data patterns, and applying preprocessing steps to prepare the data for model training. Each step is elaborated below.

### **Data Collection and Loading**

The first step involved gathering and loading the dataset into a structured format suitable for analysis.

* **Source of Data**The dataset was downloaded from Kaggle, where it had already been prepared and structured for sentiment analysis. It consisted of labeled movie reviews, divided evenly between positive and negative sentiments.
* **Loading the Data**  
  The data was loaded into a Pandas DataFrame to facilitate efficient manipulation and exploration. The main columns included the review text and its corresponding sentiment label.
* **Initial Inspection**  
  After loading, the dataset's structure was examined. The presence of the required columns, such as "review" and "sentiment," was confirmed, and a few rows were displayed to verify the data format.

### **Data Cleaning**

Cleaning the dataset ensured that it was free from inconsistencies and ready for analysis.

* **Handling Missing Values**:  
  Reviews or labels with missing values were identified and removed, as these entries would not contribute meaningfully to model training.
* **Removing Duplicates**:  
  Duplicate entries in the reviews were removed to ensure each review contributed uniquely to the dataset, thereby avoiding biases during model training.
* **Standardizing Sentiment Labels**:  
  The sentiment labels were mapped to numerical values (positive to 1 and negative to 0) to align with the model's requirements for numerical output.
* **Text Review Validation**:  
  Since the data was already prepared on Kaggle, no additional cleaning of the text (such as removing HTML tags or noise) was necessary at this stage.

### **3. Exploratory Data Analysis (EDA)**

EDA provided critical insights into the data's structure, distributions, and patterns.

* **Class Distribution**:  
  The dataset was examined to confirm an even distribution of positive and negative reviews. This balance ensured no inherent bias in the dataset that could affect training.
* **Text Length Analysis**:  
  The length of reviews (in terms of word or character count) was analyzed and visualized to detect outliers (e.g., extremely short or long reviews). This analysis also informed decisions about sequence length for tokenization.
* **Word Frequency Analysis**:  
  The most commonly used words in positive and negative reviews were identified through frequency analysis. Tools like word clouds or bar plots were used to visualize these patterns.
* **Vocabulary Size**:  
  The vocabulary size, or the number of unique words in the dataset, was calculated. This information guided tokenization strategies and informed decisions about vocabulary limits for LLMs.
* **Sentiment Trends**:  
  Patterns and associations between frequently used words and sentiments were explored to uncover insights about language usage in positive and negative reviews.

### **4. Data Preprocessing**

The data was transformed into formats suitable for both ML and LLM models.

* **For ML Models**:
  + **Feature Engineering**:  
    Sentiment labels were encoded as numerical values (1 for positive and 0 for negative).  
    The text reviews were transformed into numerical feature representations using methods like Term Frequency-Inverse Document Frequency (TF-IDF) or Word2Vec embeddings.  
    Additional features, such as review length or specific keywords, could be engineered if necessary.
  + **Train-Test Split**:  
    The dataset was split into training and testing subsets, typically using an 80-20 ratio, to evaluate model performance.
* **For LLMs**:
  + **Tokenization**:  
    Text reviews were tokenized using advanced tokenization techniques like WordPiece (used in BERT) or Byte Pair Encoding (used in GPT). Special tokens (e.g., [CLS], [SEP] for BERT) were added for classification tasks.
  + **Sequence Truncation and Padding**:  
    Longer reviews were truncated to a maximum sequence length, and shorter ones were padded to ensure uniform sequence lengths across the dataset.
  + **Attention Masks**:  
    Attention masks were created to differentiate between real tokens and padding tokens, which are required for transformer-based models.
  + **Balancing Data**:  
    If the dataset had been imbalanced, techniques like oversampling, undersampling, or synthetic data generation would have been applied. However, this step was unnecessary for the balanced IMDB dataset.

In the earlier steps, **Exploratory Data Analysis (EDA)** was performed to gain insights into the dataset and prepare it for model implementation. First, the dataset was collected and loaded from Kaggle, where it was structured for sentiment analysis, with reviews labeled as positive or negative. After loading the data into a Pandas DataFrame, an initial inspection confirmed that the necessary columns, such as "review" and "sentiment," were present. The dataset was then cleaned by removing any missing values, duplicates, and standardizing sentiment labels to numerical values (positive as 1 and negative as 0).

During the EDA process, the class distribution of positive and negative reviews was examined, confirming a balanced dataset. The text length analysis helped identify outliers and informed decisions about sequence lengths during tokenization. Word frequency analysis revealed the most commonly used words, providing insights into sentiment associations, while vocabulary size helped determine the appropriate tokenization strategy.

This EDA was critical for the next step of **Data Preprocessing** in Session 4, as it provided essential information for transforming the data into a format suitable for model training. The insights gained from the class distribution, word frequencies, and text length informed preprocessing decisions such as text tokenization, sequence truncation, and padding. Additionally, the balanced dataset ensured that no data imbalance would bias the model. Thus, EDA guided the preprocessing pipeline, allowing for more efficient and accurate model implementation.

# 4. **MODEL IMPLEMENTATIONS FROM ML TO LLMS IN SENTIMENT ANALYSIS**

The session will focus on performing **Sentiment Analysis** on the **IMDB movie review dataset**, where reviews will be classified as positive, negative, or neutral. To achieve this, various models will be evaluated, starting from traditional Machine Learning techniques like Naive Bayes, progressing through Deep Learning approaches, and advancing to Transformer-based models like BERT, GPT, and T5, as well as Large Language Models (LLMs) such as GPT-4. Each model's performance will be assessed using both **quantitative metrics** (e.g., accuracy, precision, recall, F1-score) and **qualitative analysis** to understand their strengths and limitations. In subsequent sessions, the results of this evaluation will be analyzed in-depth to provide a comparative perspective on how these models handle sentiment classification tasks and how advancements in NLP architectures have impacted their effectiveness.

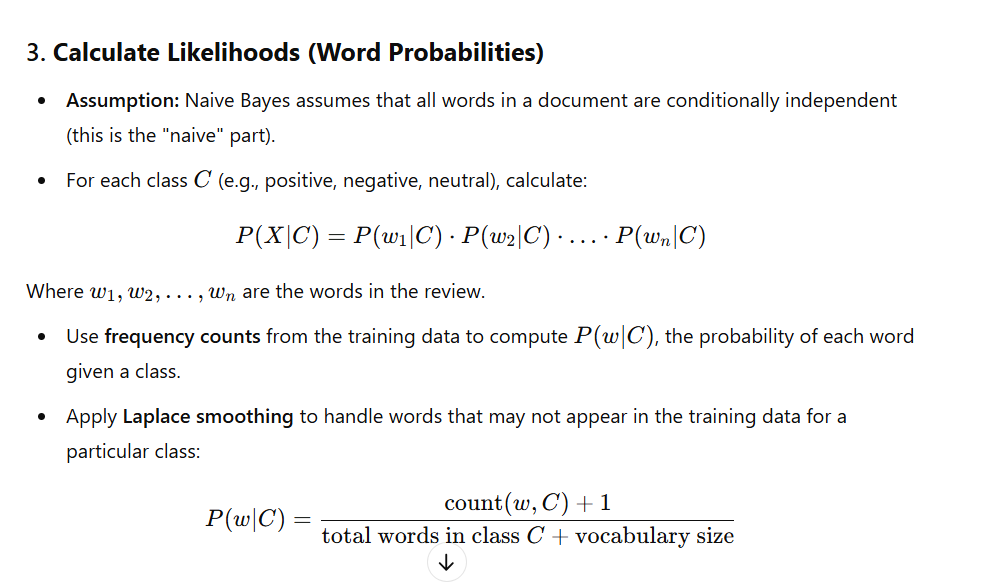
**4.1 Naïve Bayes in Sentiment Analysis**

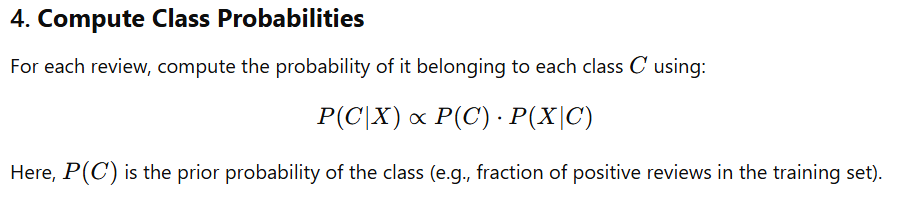
**1.** **Data Preprocessing**

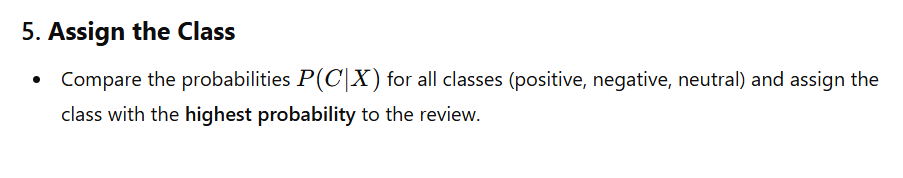
Before applying Naive Bayes, the text data needs to be cleaned and prepared:

* **Tokenization:** Split the review text into individual words or tokens.
* **Lowercasing:** Convert all words to lowercase to ensure uniformity.
* **Stopword Removal:** Remove common words like "the," "is," or "and" that do not contribute to sentiment.
* **Stemming/Lemmatization (Optional):** Reduce words to their root form (e.g., "running" → "run").
* **Vectorization:** Convert text into numerical form using methods like:
  + **Bag of Words (BoW):** Represent each document as a word-frequency vector.
  + **TF-IDF (Term Frequency-Inverse Document Frequency):** Weigh word importance based on its frequency across all documents.

**2.** **Apply Bayes’ Theorem**



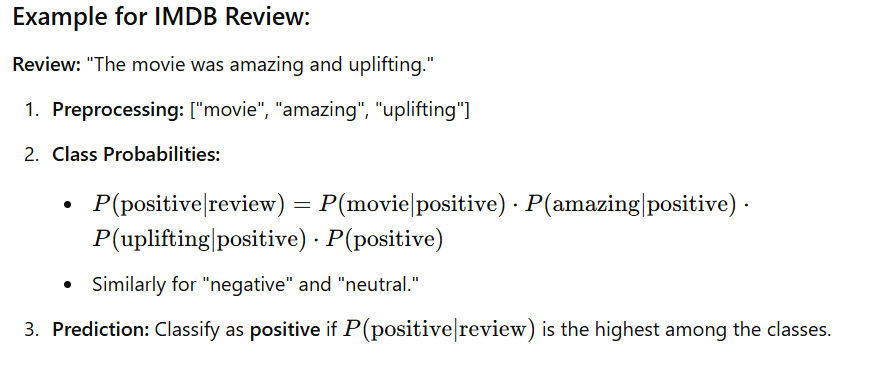




### **6.** **Evaluation**

After training the Naive Bayes model on the labeled dataset, evaluate its performance on a test set using metrics like:

* **Accuracy:** Percentage of correctly classified reviews.
* **Precision, Recall, and F1-Score:** Especially useful if the dataset is imbalanced.



**4.2 Recurrent Neural Networks /LSTM in Sentiment Analysis**

### 1. **Data Preprocessing**

Before feeding text data (like IMDB movie reviews) into an RNN or LSTM, the text needs to be preprocessed. This typically involves:

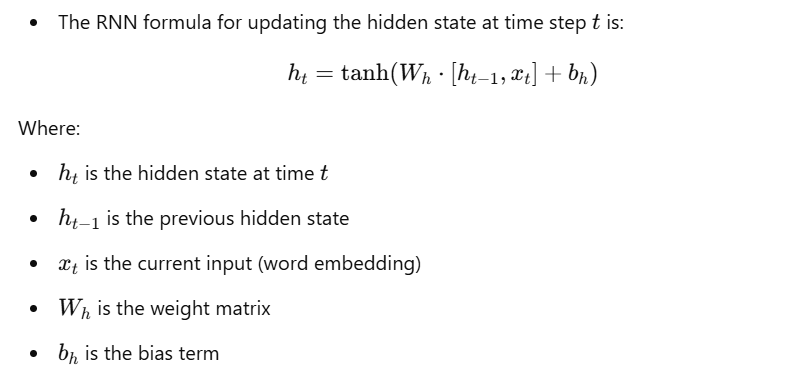
* **Tokenization:** Splitting the reviews into words or subwords.
* **Lowercasing:** Converting all words to lowercase for uniformity.
* **Stopword Removal:** Removing common words that don’t contribute to sentiment (e.g., “is”, “the”).
* **Padding:** Ensuring all input sequences are of the same length, which is important for batching the data.

The most common method for converting text to numerical data is **word embeddings**, such as **Word2Vec** or **GloVe**, which represent words as dense vectors capturing semantic meanings. The models were tested with building an embedding matrix without using pre-trained models, secondly using word2vec and thirdly using glove for study and evaluation purposes the findings are tabulated in section 6

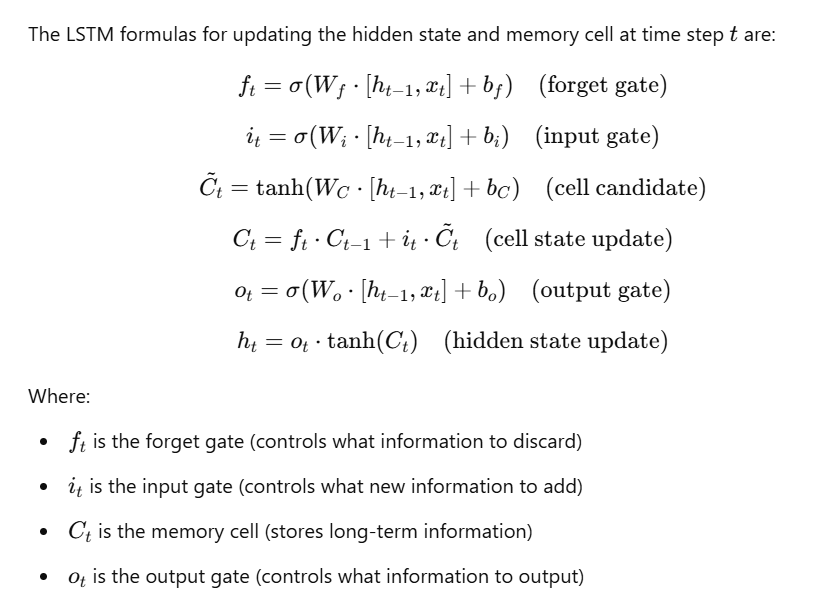
### 2. **Building the RNN/LSTM Model**

Now, let's create an RNN or LSTM model for Sentiment Analysis.

**RNNs:** The RNN takes the input sequence one word at a time, updates its hidden state based on the current word and the previous hidden state, and outputs a prediction for each word. The main limitation of basic RNNs is that they struggle to capture long-range dependencies due to vanishing gradients.



**LSTMs:** LSTMs are a more advanced form of RNN designed to overcome the vanishing gradient problem. They include a memory cell that retains information over long periods, using gates to control the flow of information.



### 3. **Training the Model**

After building the model, it is trained using labeled data (i.e., reviews with corresponding sentiment labels). The loss function used is typically **categorical cross-entropy** for multi-class classification.

### 5. **Prediction**

Once trained, the model can be used to predict the sentiment of new, unseen reviews. This involves passing the preprocessed input through the network and retrieving the class with the highest predicted probability. The output is a probability distribution over the classes (positive, negative, neutral), and the class with the highest probability is the predicted sentiment.

* **LSTMs**, with their more complex architecture that includes memory cells and gates (forget, input, output), are better at retaining long-term dependencies and mitigating the vanishing gradient problem, leading to better performance in tasks like sentiment analysis, especially when handling longer text inputs.

Both models, once trained, can be used for sentiment analysis, but LSTMs are generally preferred for tasks involving long-range dependencies or large datasets.

**4.3 BERT in Sentiment Analysis**

BERT (Bidirectional Encoder Representations from Transformers) is a transformer-based model that has revolutionized many NLP tasks, including sentiment analysis. It differs from traditional RNNs and LSTMs in that it is bidirectional, meaning it takes into account both the left and right context of a word in a sentence, making it more powerful for tasks like sentiment analysis. Below, the same steps are involved as above for using BERT in sentiment analysis, including preprocessing, building the model, training, evaluation, and prediction.

### **1**. **Data Preprocessing**

For BERT, text data needs to be tokenized using the BERT tokenizer. Unlike traditional word embeddings, BERT uses Word Piece tokenization, which breaks words into sub word units. The BERT tokenizer is part of the **Transformers** library from Hugging Face.

Tokenize the text using the BERT tokenizer.

Convert the tokens to BERT's input format (input IDs, attention masks, etc.).

**Padding:** Ensure all sequences are the same length.

**Tokenization**:

BERT uses its own **Word Piece tokenizer**. Breaks text into sub word units (e.g., "playing" → "play", "##ing"). Handles **unknown words** by breaking them into smaller sub words or characters.

Example: Input: "The movie is extremely boring."

Tokenized: ['[CLS]', 'the', 'movie', 'is', 'extremely', 'boring', '.', '[SEP]']

1. **Lowercasing**:

Lowercasing is done automatically for **BERT-base-uncased** models.

For **cased models**, case sensitivity is preserved (useful for tasks where case matters, like named entities).

1. **Special Tokens**:

[CLS]: Added at the beginning of the sequence, used for classification tasks.

[SEP]: Marks the end of a sequence (or separates two sequences in tasks like question-answering).

[PAD]: Padding token to make all input sequences the same length.

1. **No Stopword Removal or Lemmatization**:

**Stopwords** (e.g., "is", "the") are **not removed**, as BERT can extract contextual meaning from them.

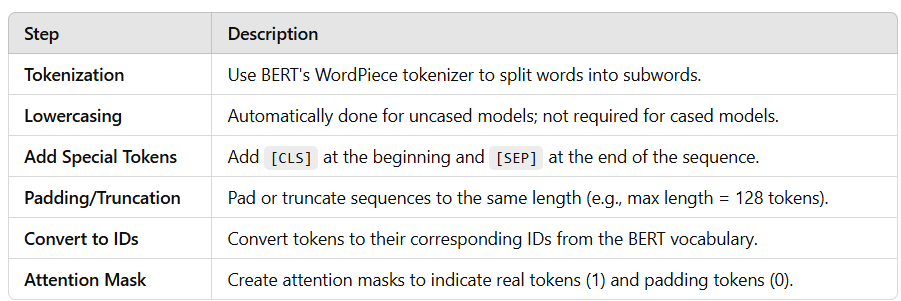
**Lemmatization** and **stemming** are **not required** because BERT handles different forms of the same word (e.g., "run", "running") through its contextual embeddings.

1. **Handling Punctuation**:

Punctuation is preserved because it can change the meaning of sentences.

Example: "Let’s eat, Grandma" vs. "Let’s eat Grandma."

1. **Special Character Mapping**:

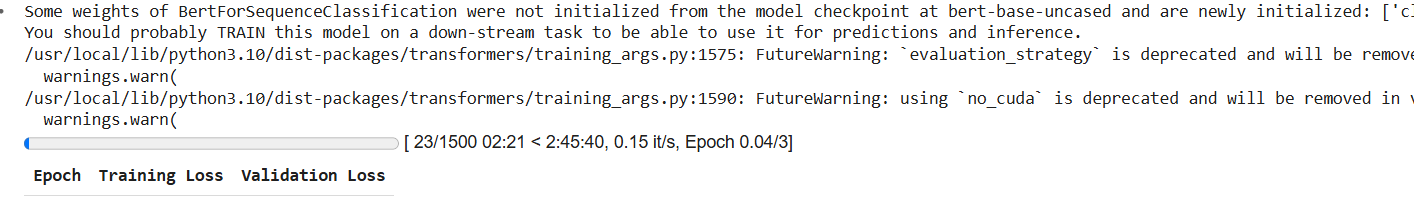
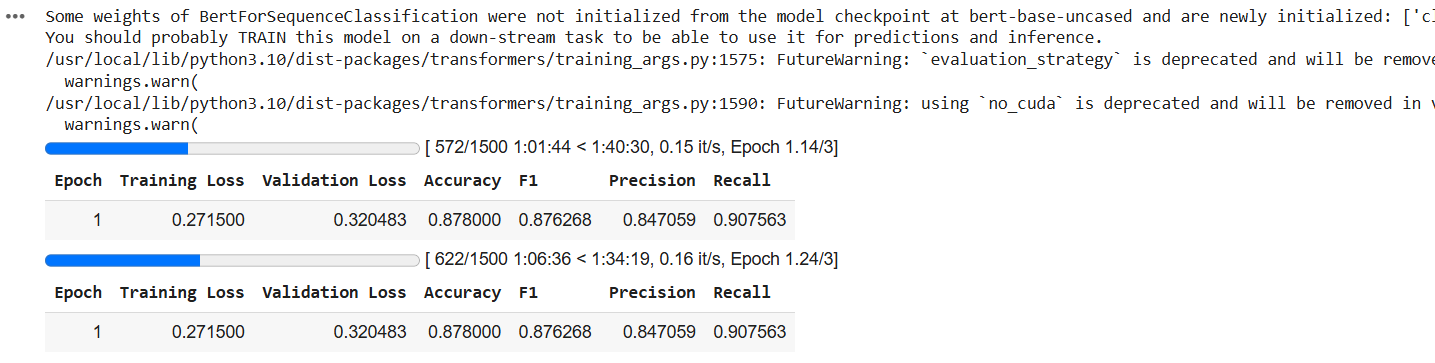
Rare or special characters are handled by the tokenizer by splitting them into subwords or mapping them to [UNK] (unknown) token.

### **2**. **Building the BERT Model**

BERT is a transformer-based model. For sentiment analysis, we typically use the pre-trained BERT model and fine-tune it on our specific task. The **BertForSequenceClassification** class from the Hugging Face transformers library is used for this purpose, which allows for easy integration of the classification head on top of BERT's output. **num\_labels=2** is for a 2-class sentiment classification (positive, neutral).

### **3. Training the Model**

Once the model is built, it can be trained using the preprocessed text data. BERT requires both the **input IDs** and **attention masks** as inputs. **padded\_input\_ids** are the tokenized sequences, and **attention\_masks** are the masks that indicate which tokens are real and which are padding. Training the model is computationally expensive. Below shows the time it takes for three epochs



### **4**. **Evaluating the Model**

The next step after training the model, you can evaluate its performance using a test set. Evaluation is similar to training but done with test data.

### **5. Prediction**

After training, you can use the model to predict sentiment on new, unseen reviews. The model will output probabilities for each class, and you can choose the class with the highest probability. **predictions.logits** contains the raw output of the model, which we convert to class predictions using np.argmax() to pick the class with the highest score.

**3.3 GPT in Sentiment Analysis**

### 1. **Data Preprocessing**

* **Loading the Data**: The model first loads the IMDB dataset from a CSV file using Pandas. It checks that the data contains reviews and their corresponding sentiment labels.
* **Label Encoding**: The sentiment labels ("positive" and "negative") are mapped to numerical values (1 for positive, 0 for negative) to make them compatible with the model's requirements.
* **Balancing the Dataset**: The dataset is balanced by randomly sampling an equal number of positive and negative reviews to avoid class imbalance during training. This ensures that the model doesn't become biased towards one sentiment class.
* **Cleaning**: The column names are cleaned by stripping any unnecessary spaces to avoid errors during further processing.

### 2. **Tokenization**

* **Converting Text to Tokens**: The GPT-2 tokenizer splits the review text into smaller units (tokens), which are subwords or words, depending on the tokenizer’s method. This allows the model to handle the text more efficiently.
* **Padding and Truncating**: The model ensures all tokenized sequences are of the same length by padding shorter sequences with a special token (like eos\_token in GPT-2) and truncating longer ones. This is necessary because transformer models like GPT-2 require consistent input lengths to process the data effectively.
* **Returning Tokenized Format**: The tokenized text is converted into a format that can be understood by the model, with inputs such as input IDs and attention masks.

### 3. **Splitting the Dataset**

* **Train-Test Split**: The dataset is split into training and testing sets using the train\_test\_split function. Typically, a larger portion (80-90%) of the data is used for training, while the remaining 10-20% is set aside for testing.
* **Training Data**: The training set is used to train the model, allowing it to learn patterns in the text that correspond to the sentiment labels.
* **Testing Data**: The testing set is used to evaluate the model’s performance after training. It helps assess how well the model generalizes to unseen data.

### 4. **Model Initialization**

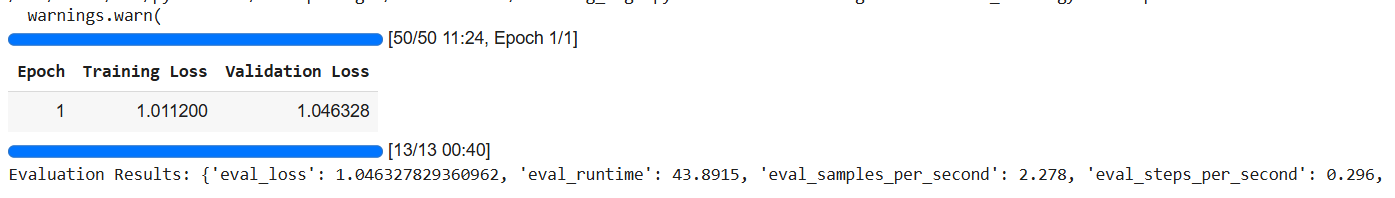
* **Loading Pre-trained GPT-2**: The GPT-2 model is loaded from Hugging Face’s Transformers library, along with its tokenizer. This model has already been pre-trained on a large corpus of text, which gives it an understanding of language.
* **Modifying the Model for Classification**: The GPT-2 model is modified by adding a classification head, which enables the model to perform sentiment analysis (classifying reviews as positive or negative).
* **Handling Padding Token**: If the GPT-2 model doesn’t already have a padding token, it assigns the eos\_token (End Of Sequence token) to handle padding. This ensures that the padding is treated properly during training.

### 5. **Training**

* **Training Arguments**: The model’s training process is controlled using the TrainingArguments class, where hyperparameters such as the number of epochs, batch size, and evaluation strategy are defined.
* **Model Training**: The Trainer class from Hugging Face is used to train the model. It takes in the training dataset and the model, and performs forward and backward passes to minimize the loss (the difference between predicted and actual sentiment labels). During training, the model’s weights are adjusted to improve its predictions.
* **Evaluation**: After each epoch, the model is evaluated on the testing dataset to monitor its performance. Evaluation results like accuracy are printed to track how well the model is learning.

### 6. **Model Prediction**

* After fine-tuning, the model can be used to predict the sentiment of new, unseen reviews. When a new review is input, the model tokenizes the text, processes it, and outputs a prediction (either 0 for negative or 1 for positive sentiment).
* The fine-tuned model applies what it has learned from the training data to classify the sentiment of new reviews, making it capable of handling sentiment analysis tasks on real-world text.

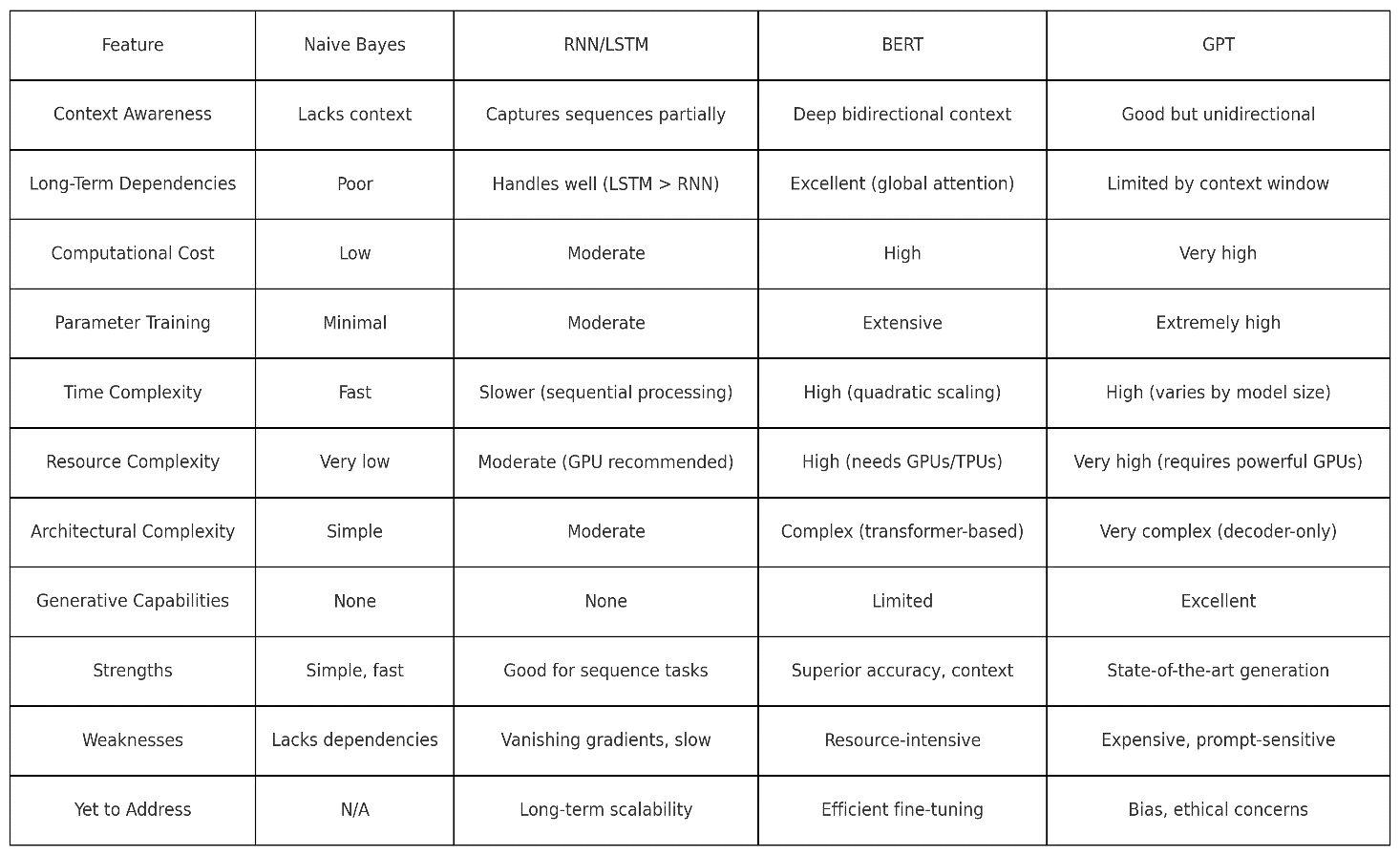


In essence, this model uses a pre-trained GPT-2 to classify sentiments in text by fine-tuning it on a labeled dataset of movie reviews. It transforms the text into tokenized format, trains the model to classify the sentiments, and evaluates its performance before making predictions on unseen data.

## **5. QUANTITATIVE/QUALITATIVE ANALYSIS OF SENTIMENT ANALYSIS MODELS – NAIVE BAYES, RNN/LSTM, BERT, AND GPT**

In this session, we will perform both **qualitative** and **quantitative analysis** to evaluate the performance of sentiment analysis models, ranging from traditional algorithms like Naive Bayes to advanced large language models (LLMs) such as GPT. Both types of analysis are essential to obtain a comprehensive understanding of the models' effectiveness, strengths, and weaknesses. Quantitative analysis focuses on numerical metrics such as accuracy, precision, recall, and F1-score, providing an objective measure of performance. On the other hand, qualitative analysis explores how well models handle nuanced, real-world challenges like sarcasm, ambiguity, or domain-specific language, which metrics alone cannot capture.

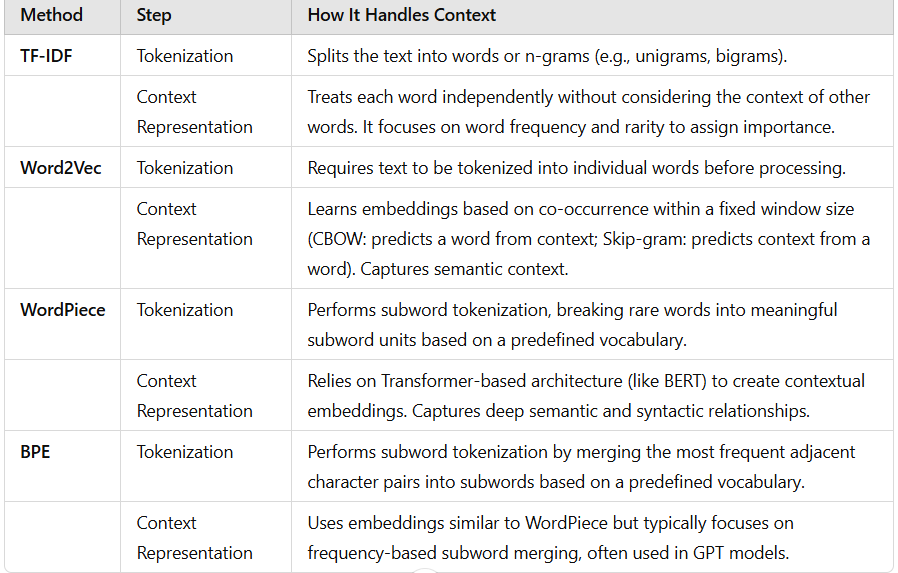
### **5.1 Qualitative Analysis**

Qualitative analysis examines the **behavior and interpretability** of the models by analyzing their outputs in specific scenarios. It helps identify patterns in errors, assess how well a model captures context, and observe how it performs on edge cases such as sarcastic, ambiguous, or domain-specific reviews. It also evaluates **explainability**, checking if the model provides insights into why a certain sentiment is assigned. For generative models like GPT, qualitative analysis evaluates the relevance and coherence of explanations or outputs.

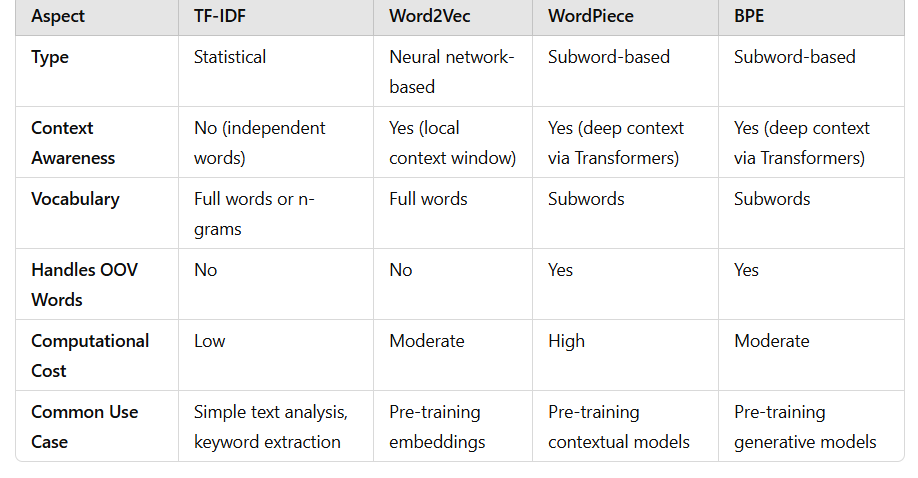
**5.1.1 Comparison of Word Context in TF-IDF, Word2Vec, Word Piece, and BPE**

This section compares **TF-IDF**, **Word2Vec**, **Word Piece**, and **BPE** in handling word context in NLP. It outlines their tokenization processes, context representation approaches, and key features like context awareness, vocabulary handling, computational cost, and use cases. A comparative table highlights similarities and differences, providing insights into their strengths and weaknesses. Recommendations are included to guide the choice of the best method based on task requirements, from simple keyword analysis to advanced contextual and generative modeling.

Table showing the methodologies of word context handling



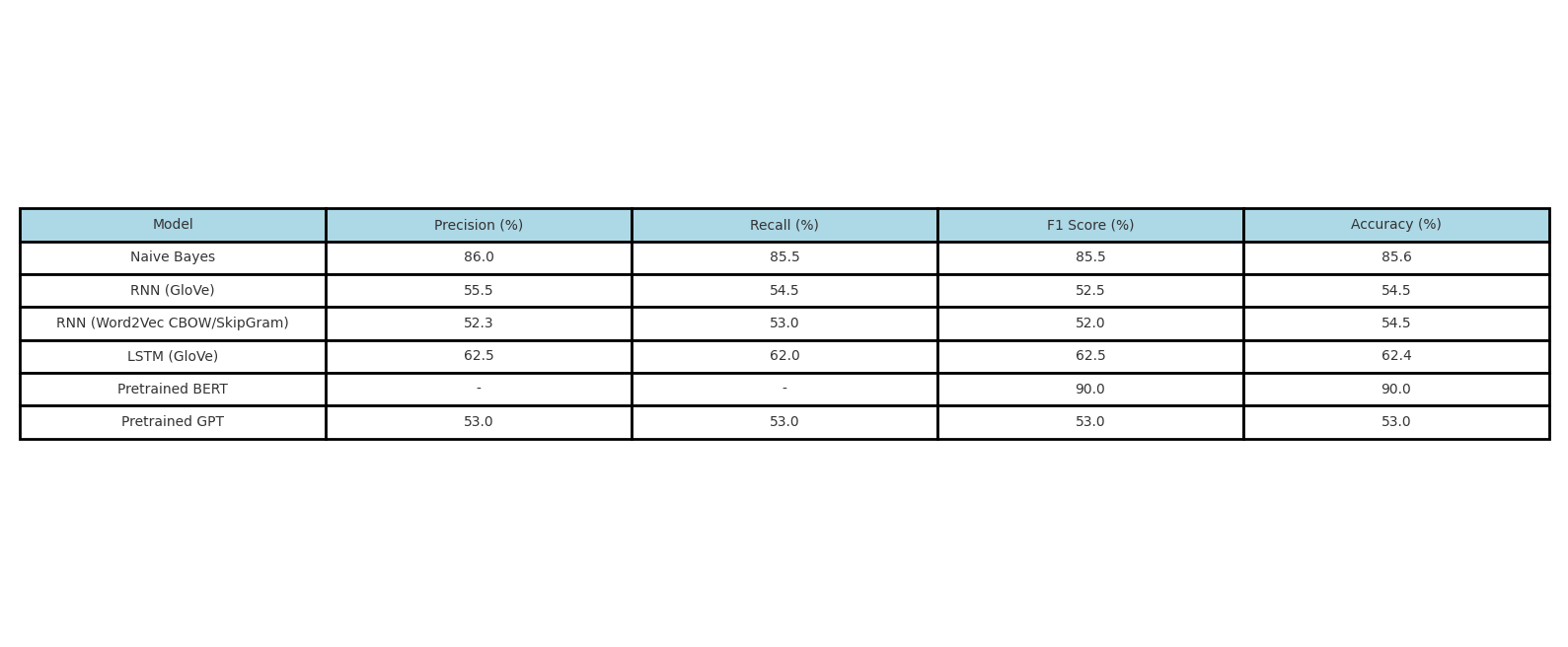
The table below lists the similarities and differences in each methodology

 **For most NLP tasks today, Word Piece is often preferred**, especially in tasks requiring deep contextual understanding (e.g., sentiment analysis, classification).

**BPE is better for generative tasks**, where the focus is on efficient vocabulary handling.

**TF-IDF** is a good starting point for simpler, resource-limited projects, while **Word2Vec** strikes a balance for moderate complexity tasks without requiring Transformers.

**5.2 Quantitative Analysis**

Quantitative analysis is essential for objectively evaluating sentiment analysis models like Naive Bayes, RNN/LSTM, BERT, and GPT. It provides measurable insights using metrics such as **accuracy**, **precision**, **recall**, and **F1-score** to assess model performance on actual data. This analysis helps validate strengths, identify weaknesses (e.g., handling ambiguity or sarcasm), and assess computational efficiency through metrics like **inference time** and **memory usage**. Additionally, it ensures comparability between models and highlights their generalization power across datasets. By combining these metrics, quantitative analysis ensures the selection of the most effective and efficient model for sentiment analysis tasks.

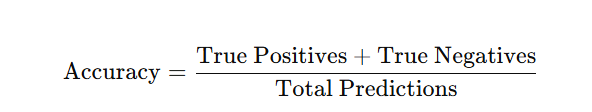
The table shows a clear progression in performance across models. Naive Bayes provides a strong baseline with high accuracy (85.6%) despite its simplicity. RNNs, especially with GloVe embeddings, show moderate performance with accuracy reaching 54.5%, while LSTM (GloVe) improves contextual handling, achieving 62.4% accuracy. Pretrained BERT outperforms all with impressive accuracy and F1 score of around 90%, benefiting from advanced attention mechanisms. Pretrained GPT, while consistent across metrics (53%), underperforms compared to BERT, highlighting its strength in generative tasks over classification. Overall, model performance improves from traditional methods to transformer-based models, with BERT excelling in sentiment analysis.

### **5.2.1 Evaluation Metrics for Quantitative Analysis in Sentiment Analysis**

When performing quantitative analysis in sentiment analysis, metrics like **accuracy**, **precision**, **recall**, and **F1-score** are crucial for objectively assessing the performance of models. These metrics provide insights into how well a model classifies reviews into categories such as positive, negative, or neutral. Here's how each metric applies to sentiment analysis:

1. **Accuracy**:  
   Accuracy measures the percentage of correct predictions out of the total number of predictions. It is the most straightforward metric but can be misleading in imbalanced datasets.

**Formula:**



**In Sentiment Analysis**: Accuracy evaluates how many reviews were correctly classified

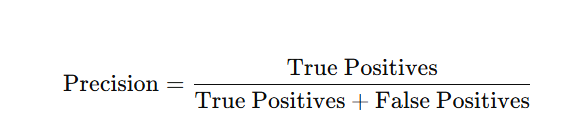
as positive, negative, or neutral. However, if one sentiment category dominates (e.g.,

mostly positive reviews), accuracy alone may not reflect the true performance of the

model.

1. **Precision**:  
   Precision measures the proportion of correctly predicted positive instances out of all predicted positives. It focuses on the model's ability to avoid false positives.

**Formula**:

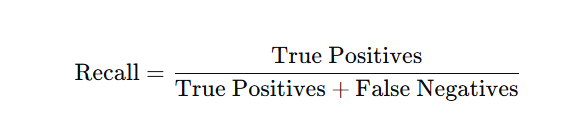


**In Sentiment Analysis**: For example, if the model predicts many reviews as positive but

misclassifies neutral or negative reviews as positive, precision will drop. A high precision

indicates the model is confident about its positive predictions.

1. **Recall**:  
   Recall (or sensitivity) measures the proportion of correctly predicted positive instances out of all actual positives. It focuses on the model's ability to avoid false negatives.

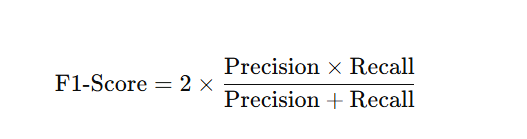
 **Formula**

**In Sentiment Analysis**: If the model misses many positive reviews (false negatives),

recall will be low. High recall ensures the model captures all relevant positive reviews.

1. **F1-Score**:   
   The F1-score is the harmonic mean of precision and recall, providing a balanced metric when both are important. It is especially useful for imbalanced datasets.

**Formula**

: 

**In Sentiment Analysis**: F1-score gives a single performance measure, balancing the

trade-off between precision and recall. It is crucial when the costs of false positives

and false negatives are high or unequal.

### Example in Sentiment Analysis:

Suppose we use a dataset of movie reviews where the true labels are **positive, negative, and neutral**. After applying the model, we analyze its predictions:

* If the model classifies most positive reviews correctly but frequently confuses neutral reviews as negative, precision for the positive category will be high, but recall might suffer.
* Similarly, if the model misses many negative reviews but confidently classifies some correctly, it will have a low F1-score for that category.

By using these metrics, we gain a deeper understanding of the model's strengths and weaknesses, ensuring a robust evaluation of its performance across all sentiment categories.

Firstly, let us compare the DL methods of RNN VS LSTM using pretrained Glove Embedding vector on “glove.6B.50d.txt” the below gives interesting observation on the same dataset on IMDB review samples



LSTM outperforms RNN in every metric, proving it to be a better choice for tasks involving sequential data like sentiment analysis.

## **FINDINGS AND INFERENCE: A DETAILED LEARNINGS**

This session focuses on presenting the findings and insights derived from the comparative evaluation of various models used for sentiment analysis, ranging from traditional machine learning approaches to advanced deep learning models and large language models (LLMs). The session delves into the **architecture and mathematical formulation** of each model, aiming to understand their core working principles and their application in natural language processing (NLP). It traces how these architectures are designed to handle linguistic challenges and contextual information in NLP tasks, eventually narrowing down to their specific use in sentiment analysis. This structured analysis helps highlight how the models evolve in terms of capability and complexity while addressing the limitations of their predecessors. BERT brought bidirectional context understanding and excelled at handling nuanced sentiments but required significant computational resources. Lastly, GPT, with its generative capabilities and few-shot learning, enables sentiment analysis with minimal labeled data, though it lacks bidirectional context understanding. These findings highlight the gradual advancements in handling linguistic complexities and computational efficiency in sentiment analysis.

### 1. **Naive Bayes and Early Methods**

Naive Bayes relies on frequency-based methods like Bag-of-Words and TF-IDF for tokenization. While simple and computationally efficient, it lacks the ability to account for word order and context, which are essential for understanding nuanced sentiments in text. For instance, phrases with negations (e.g., "not good") are often misclassified due to its independence assumption. It performs well on straightforward tasks but struggles with complex linguistic patterns (Zhang et al., 2022).

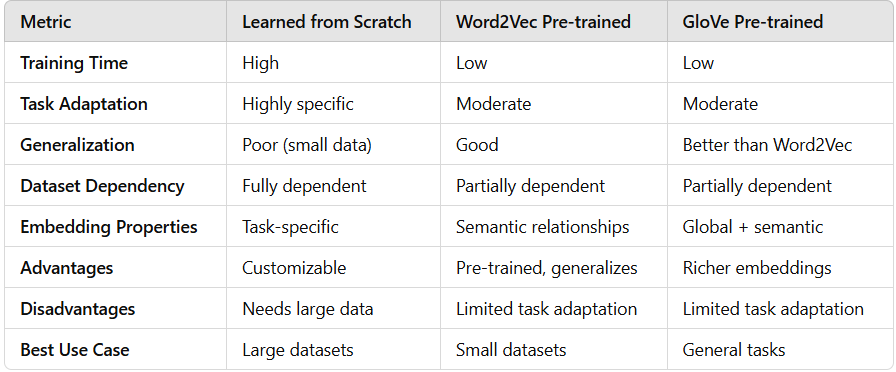
### 2. **RNNs**

RNNs introduced sequential processing of text by using word embeddings, such as Word2Vec, to capture semantic similarity beyond word frequency. Trainable parameters like input-to-hidden and hidden-to-hidden weights allow the network to process word sequences and learn patterns over time. However, RNNs face the vanishing gradient problem, which limits their ability to learn long-term dependencies. For example, in sentiment analysis, RNNs may lose critical information from earlier parts of a review, affecting accuracy (Cho et al., 2014). The model use two pretrained models such as word2vec and Glove and build an embedding layer for context representation from scratch and the findings were observed below:

**No Pre-trained Embeddings** offer better results for positive sentiment but require more computational resources for training embeddings.

 **GloVe Embeddings** are more effective for recognizing negative sentiment and are quick to integrate but may underperform for positive sentiment.

**Word2Vec Embeddings** strike a balance between the two, providing decent performance across both classes with consistent metrics, making it a more generalized choice.



Comparison on Embeddings build from scratch vs pretrained Word2vec and Glove

3. **LSTMs**

LSTMs overcome RNN limitations by introducing gating mechanisms: input, forget, and output gates. These gates allow selective memory retention, enabling the model to capture long-term dependencies. In sentiment analysis, LSTMs excel at handling sequences with complex structures, such as sentences with multiple clauses or negations (e.g., "The movie was interesting, but the ending was disappointing"). Despite their success, LSTMs are computationally intensive due to their larger number of trainable parameters (Hochreiter and Schmidhuber, 1997).

### 4. **BERT**

BERT’s bidirectional transformer architecture allows it to capture context from both directions in a sentence, making it particularly effective for sentiment analysis. Its wordpiece tokenization handles rare and compound words by breaking them into subwords, improving generalization. Pre-trained using Masked Language Modeling (MLM) and fine-tuned on sentiment datasets, BERT achieves state-of-the-art performance, understanding nuanced sentiment, negations, and sarcasm. However, its high resource demand and computational complexity pose challenges (Devlin et al., 2019). BERT was pre-trained on a corpus of 3.3 billion words (Books Corpus + English Wikipedia).

### 5. **GPT**

GPT, with its decoder-only architecture, excels in generative tasks and few-shot or zero-shot learning, reducing the need for extensive labeled data. For sentiment analysis, GPT relies on prompts to classify sentiments and generate explanations. While its unidirectional context limits nuanced understanding compared to BERT, GPT’s generative capabilities make it valuable for applications requiring both classification and textual insights (Brown et al., 2020).

### Summary of Findings:

* **Naive Bayes**: Simple and fast but lacks context-awareness and struggles with negation.
* **RNNs**: Sequentially processes text but fails to retain long-term dependencies due to vanishing gradients.
* **LSTMs**: Addresses long-term dependencies effectively but is computationally heavy.
* **BERT**: Captures bidirectional context, excelling in sentiment nuances but resource-intensive.
* **GPT**: Strong in generative tasks and low-data scenarios but weaker in bidirectional context.

These findings reflect the evolution of sentiment analysis models, addressing challenges like context awareness, long-term dependencies, and computational efficiency. The shift from frequency-based models to advanced transformer architectures highlights the growing sophistication in handling complex linguistic patterns (Zhang et al., 2022; Hochreiter and Schmidhuber, 1997; Devlin et al., 2019; Brown et al., 2020).

Large Language Models (LLMs), such as GPT and BERT, have significantly advanced text analysis by excelling in diverse tasks like sentiment analysis, summarization, and translation. However, their limitations necessitate further research and innovation. The next section discusses these challenges and potential future directions of NLP and its applications such as sentiment analysis

## **LIMITATIONS AND FUTURE SCOPE**

#### This section focuses on analyzing the limitations of large language models (LLMs), such as resource intensity, biases, and lack of explain ability, while exploring future advancements like modular architectures, energy-efficient designs, knowledge graph integration, and human-AI collaboration to enhance their efficiency and applicability in text analysis tasks.

#### **Limitations of LLMs in text analysis**:

#### **Resource-Intensive Nature:** Training and deploying LLMs require massive computational resources, data, and energy, making them expensive and environmentally unsustainable (Dai et al., 2024; Zhang et al., 2023).

#### **Bias and Fairness Issues:** These models inherit biases from their training data, which can lead to perpetuation of stereotypes and unfair predictions. Addressing this remains a critical challenge (Singh et al., 2023).

#### **Lack of Explain ability:** Despite their effectiveness, LLMs operate as black-box models, which complicates their adoption in high-stakes domains where interpretability is crucial (Singh et al., 2023).

#### **Limited Contextual Understanding:** While powerful, LLMs struggle with long-range dependencies and nuanced tasks, especially in specialized domains where general pretraining does not suffice (Wu & Chen, 2024).

#### **Future Scope of text analysis (NLP)**

**Future scope** is essential to address the limitations of current technologies and guide further innovation. For LLMs, it helps overcome challenges like inefficiency, biases, and lack of interpretability while driving advancements like modular architectures and energy-efficient designs. This ensures continuous improvement, making models more ethical, sustainable, and adaptable to real-world needs.

**Integration with Knowledge Graphs:** Merging LLMs with knowledge graphs can enhance their reasoning and factual accuracy, enabling them to better understand and process domain-specific information (Wu & Chen, 2024).

**Energy-Efficient Models:** Innovations such as sparse architectures and adaptive computation strategies are being explored to make LLMs more environmentally sustainable and accessible (Zhang et al., 2023).

**Federated Learning:** Federated learning architectures could enable decentralized training, preserving user privacy while reducing the need for large centralized datasets (Wu & Chen, 2024).

**Task-Specific Modular Architectures:** Creating smaller, modular LLMs tailored for specific tasks could enhance interpretability and reduce computational costs while maintaining high performance (Dai et al., 2024).

**Human-AI Collaboration:** Future systems may emphasize collaboration between humans and LLMs, leveraging the strengths of both for tasks requiring judgment, domain knowledge, and creativity (Wu & Chen, 2024).

By addressing these limitations, future LLMs could become more ethical, efficient, and effective for various text analysis tasks. Research combining modularity, sustainability, and domain expertise will likely drive the next wave of innovation in the field.

1. **CONCLUSION**

This study was structured across six key sections, each addressing a critical aspect of sentiment analysis and its advancements. The first section introduced the history and evolution of natural language processing (NLP) and text analysis, setting the foundation for understanding how sentiment analysis has progressed over time. The second section delved into the architectural frameworks used for sentiment analysis, ranging from traditional machine learning models like Naive Bayes to advanced deep learning and transformer-based architectures such as RNNs, LSTMs, BERT, and GPT. The third section outlined the step-by-step implementation of these models, providing a detailed account of how sentiment analysis has been approached, from simpler models to more sophisticated LLMs. In the fourth section, a comparative analysis was conducted using both quantitative metrics (accuracy, precision, recall, F1-score) and qualitative features (context-awareness, long-term dependencies, computational cost, and generative capabilities). The fifth section focused on the major findings and inferences drawn from the comparative study, identifying each model's strengths, limitations, and performance in sentiment classification. Finally, the sixth section examined the limitations and future scope of NLP and sentiment analysis, exploring how innovations like knowledge graph integration, energy-efficient models, and modular architectures could shape the field's progression.

The primary objective of this study was to analyze and evaluate various sentiment analysis models, understand their underlying architectures, and identify their effectiveness in processing text data. This goal has been accomplished through a systematic investigation of each model's performance and an exploration of their evolution, from frequency-based methods to advanced transformer architectures. By conducting both qualitative and quantitative analyses, the project provided a comprehensive understanding of how these models address challenges such as context-awareness, long-term dependency resolution, and computational efficiency.

This work is particularly useful for researchers, data scientists, and NLP practitioners seeking insights into the strengths, limitations, and applicability of various models for sentiment analysis. In the broader context, this study serves as a valuable resource for applications in domains like customer feedback analysis, social media monitoring, market research, and more, where sentiment analysis is critical. The findings and insights contribute to the ongoing efforts in advancing text analysis techniques, helping stakeholders choose the most appropriate tools and methodologies for their specific use cases.

REFERENCES

1. Turing, A.: Computing Machinery and Intelligence. Mind 59(236), 433–460 (1950). Available: https://www.csee.umbc.edu/courses/471/papers/turing.pdf
2. Chomsky, N.: Syntactic Structures. Walter de Gruyter, Berlin (1957). Available: https://philpapers.org/rec/CHOSS-2
3. Weizenbaum, J.: ELIZA – A Computer Program for the Study of Natural Language Communication Between Man and Machine. Communications of the ACM 9(1), 36–45 (1966). Available: http://www.cse.chalmers.se/~coquand/AUTOMATA/eliza.pdf
4. Winograd, T.: Understanding Natural Language. Academic Press, New York (1972). Available: <http://hci.stanford.edu/winograd/shrdlu/>
5. Church, K.W.: A Stochastic Parts Program and Noun Phrase Parser for Unrestricted Text. In: Proceedings of the Second Conference on Applied Natural Language Processing (ANLP), vol. 1, pp. 136–143 (1988). Available: https://aclanthology.org/J88-2004.pdf
6. Brown, P.F., Cocke, J., Della Pietra, S.A., Della Pietra, V.J., Jelinek, F., Lafferty, J.D., Mercer, R.L., Roossin, P.S.: A Statistical Approach to Machine Translation. Computational Linguistics 16(2), 79–85 (1990). Available: https://aclanthology.org/J90-2002.pdf
7. Jelinek, F.: Statistical Methods for Speech Recognition. MIT Press, Cambridge (1997). Available: <https://www.springer.com/gp/book/9780262100663>
8. Blei, D.M., Ng, A.Y., Jordan, M.I.: Latent Dirichlet Allocation. Journal of Machine Learning Research (JMLR) 3, 993–1022 (2003). Available: https://www.jmlr.org/papers/volume3/blei03a/blei03a.pdf
9. Bengio, Y., Ducharme, R., Vincent, P., Janvin, C.: A Neural Probabilistic Language Model. Journal of Machine Learning Research (JMLR) 3, 1137–1155 (2003). Available: https://www.jmlr.org/papers/volume3/bengio03a/bengio03a.pdf
10. McCallum, A., Li, W.: Early Feature-Rich Approaches for Named Entity Recognition. In: Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003, pp. 188–191 (2003). Available: https://aclanthology.org/W03-0424.pdf
11. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient Estimation of Word Representations in Vector Space. arXiv preprint arXiv:1301.3781 (2013). Available: <https://arxiv.org/abs/1301.3781>
12. Bahdanau, D., Cho, K., Bengio, Y.: Neural Machine Translation by Jointly Learning to Align and Translate. arXiv preprint arXiv:1409.0473 (2014). Available: <https://arxiv.org/abs/1409.0473>
13. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gómez, A.N., Kaiser, Ł., Polosukhin, I.: Attention Is All You Need. Advances in Neural Information Processing Systems (NeurIPS) 30, 5998–6008 (2017). Available: <https://arxiv.org/abs/1706.03762>
14. Devlin, J., Chang, M.-W., Lee, K., Toutanova, K.: BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv preprint arXiv:1810.04805 (2018). Available: <https://arxiv.org/abs/1810.04805>
15. Brown, T.B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J.D., Dhariwal, P., Neelakantan, A., et al.: Language Models Are Few-Shot Learners. arXiv preprint arXiv:2005.14165 (2020). Available: <https://arxiv.org/abs/2005.14165>
16. Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., et al.: Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. Journal of Machine Learning Research (JMLR) 21(140), 1–67 (2020). Available: <https://arxiv.org/abs/1910.10683>
17. Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., Zhang, C., et al.: Training Language Models to Follow Instructions with Human Feedback. arXiv preprint arXiv:2203.02155 (2022). Available: <https://arxiv.org/abs/2203.02155>
18. Dai, Z., Sun, P., Wu, J., et al. (2024). Limitations of Pre-trained Language Models in Text Processing. Springer. [Link to Springer]
19. Zhang, T., Zhao, X., & Li, Y. (2023). Greener AI for Large Language Models: Challenges and Solutions. IEEE Xplore. [Link to IEEE Xplore]
20. Wu, H., & Chen, Y. (2024). Unifying Large Language Models and Knowledge Graphs: A Roadmap. IEEE Journals & Magazine. [Link to IEEE Xplore]
21. Singh, A., Kumar, P., & Das, S. (2023). Addressing Bias and Explainability in NLP Systems. Springer.
22. Zhang, Y., Zhao, L., & Li, M. (2022). A Comparative Study of Naive Bayes Classifier for Sentiment Analysis. Springer.
23. Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. IEEE Transactions on Neural Networks and Learning Systems.
24. Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. Neural Computation, 9(8), 1735–1780. Link
25. Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. NAACL-HLT 2019.
26. Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., ... & Amodei, D. (2020). Language Models are Few-Shot Learners. NeurIPS 2020