2024 Fall NLP - Final Project Report

News Text Generation using Different Advanced Model -Classical and Deep learning Approaches: A Comparative Study

George Washington University

Master of Science in Data Science Program

Course: DATS_6312_10 Natural Language Processing

Professor Name : Dr. Amir Jafari

Group 4 Members:

Moduepola Fagbenro, Apoorva Reddy Bagepalli, Keerthana Aravindhan, Aron Yang

GitHub Repository: GitHub Link

Introduction

In the digital age, the consumption of news has shifted dramatically towards online platforms, where brevity and clarity are highly valued. With the overwhelming amount of content being produced daily, generating concise and accurate descriptions for news articles has become a significant challenge. This project focuses on automating the process of news text generation using advanced Natural Language Processing (NLP) models.

The problem selected for this project is generating concise and accurate descriptions for recent news articles. In today's fast-paced digital environment, news consumers often prefer brief outputs that capture the essence of articles without delving into their entirety. This task addresses the challenge of automatically generating such descriptions, aiding in news aggregation, curation, or providing meaningful summaries for platforms. Given the growing volume of news content, this problem is ideal for automation using advanced text generation models.

Our project explores and implements various text generation methods to address this issue, ranging from classical Markov Chains to modern deep learning techniques like Long Short-Term Memory (LSTM) networks, and transformer-based architectures such as GPT-2, T5. By applying these models to datasets of recent news articles, we aim to provide a comparative analysis of their performance in generating meaningful and high-quality outputs.

The report outlines the datasets, methodologies, and evaluation metrics such as perplexity. It also discusses key findings, challenges faced, and potential future enhancements to improve the quality of automated news text generation.

Team Member Contributions

Team Member 1: Apoorva Reddy Bagepalli
Focused on Data preprocessing, Markov chain model

Team Member 2: Aaron Yang
Focuses on Data preprocessing, Masking Data, Adding Noisy Data,-T5 model

Team Member 3: Modupeola Fagbenro
Focuses on Data Preprocessing , Exploratory data analysis and LSTM + Attention model

Team Member 4: Keerthana Aravindhan
Focuses on Data preprocessing , working on Transformer(state-of art-model) models - GPT-2

Dataset Description

The dataset used in this project comprises diverse news articles sourced from publicly available repositories, selected to support the development and evaluation of text generation models. The InShorts News Data provides concise headlines and descriptions, ideal for generating brief and coherent outputs. The BBC Articles Dataset offers full-length articles categorized by domain, enabling experimentation across different topics and writing styles. The CNN/DailyMail Dataset, a benchmark for text generation tasks, provides article-summary pairs for training models to produce meaningful outputs from longer inputs. The Newsroom Dataset, with its high-quality detailed articles, supports comprehensive evaluations of model performance in generating contextually accurate text. Additionally, the NYTimes Article Lead Paragraphs Dataset focuses on introductory paragraphs, helping models generate impactful lead descriptions. Together, these datasets ensure diverse coverage of topics and styles, enabling robust training for generating concise, coherent, and context-aware news descriptions.

Dataset source : Kaggle : which are InShort News Data , CNN/DailyMail Dataset

Combined data size : The data size combined size was **3GB** , which comprises of Main article text only

Natural Language Processing Models and Algorithms

Markov Chain Model

The Markov model is a probabilistic framework commonly used in Natural Language Processing (NLP) for text generation. It relies on the **Markov assumption**, which states that the probability of a word in a sequence depends only on a finite number of preceding words. For a high-order *n*-gram model, the next word's prediction is conditioned on the last *n-1* words, allowing for the capture of longer dependencies and context in text generation.

Background of the Algorithm

The Markov Chain is defined by:

1. **States:** Represent the entities being modeled. For text generation, each state corresponds to a word or an *n-gram* (sequence of n words).
2. **Transitions:** These are the probabilities of moving from one state to another, derived from the observed frequencies in the dataset.

Transition Probability Equation

If P_{ij} represents the transition probability from state i to state j, it is calculated as:

$$P_{ij} = \frac{\text{Number of transitions from state } i \text{ to state } j}{\text{Total number of transitions from state } i}$$

Here, P_{ij} satisfies the condition:

$$\sum_{j} P_{ij} = 1$$

Higher-Order Markov Chains

To incorporate context, we use *n-gram* Markov Chains where each state represents a sequence of n words. The transition probabilities are then calculated for these sequences instead of individual words, providing more coherent and meaningful text generation.

Stationary Distribution

A Markov Chain reaches a stationary distribution if the transition probabilities stabilize over time, meaning the likelihood of being in any state becomes constant. This property can help ensure diverse and unbiased text generation.

Chapman-Kolmogorov Equation

The Chapman-Kolmogorov theorem is used to calculate the probability of transitioning between states over multiple steps:

$$P_{ij}^{(n)} = \sum_{k} P_{ik}^{(n-1)} . P_{kj}$$

Here, $P_{ij}^{(n)}$ is the probability of transitioning from state i to state j in n steps.

Development of the Algorithm

1. **Defining States and Transitions**
    - Words and n-grams were extracted from the dataset, forming the states of the Markov Chain.
    - Transition probabilities were computed by traversing the dataset, counting occurrences of word pairs or n-grams, and normalizing the counts.
2. **Text Generation Process**
    - To generate new text, a random walk is simulated on the Markov Chain. Starting from an initial state, the algorithm probabilistically selects the next state based on transition probabilities and continues this process until a stopping criterion (e.g., sentence length) is met.
3. **Implementation Steps**
    - Tokenization: Preprocessed the text to divide it into words or n-grams.
    - Transition Matrix Construction: Created a matrix representing transition probabilities between states.
    - Random Walk Simulation: Generated text by traversing the states based on computed probabilities.

Long Short-Term Memory(LSTM) Model

Introduction

Long Short-Term Memory networks are specialized variants of RNNs designed to learn long-term dependencies. The key innovation of LSTM is its memory cell structure that controls information flow through gates. LSTM is designed to solve long term dependency-vanishing problems. Long term memory and remembering information for a long period of time is practically their default behavior. LSTM also has a chain-like structure ,with neural network layers interacting in a special way. LSTM prevents back propagated errors from vanishing or exploding. LSTM can learn tasks that require memories of events that happened thousands or millions of discrete time steps earlier. Attention allows the model to focus on relevant parts of the input sequence when generating each word.

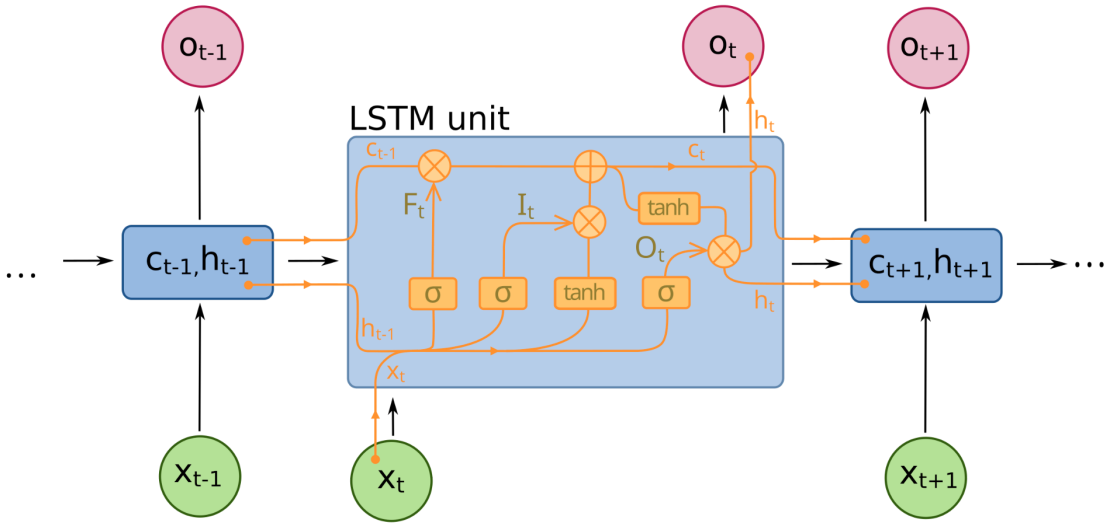
LSTM-Core Architecture

Background of the Algorithm

Long Short-Term Memory (LSTM) networks are sophisticated neural architectures designed to handle sequential data through a carefully orchestrated system of gates and states. At their core, LSTMs operate through three main gates that work in harmony: the forget gate, input gate, and output gate, each implementing specific mathematical transformations to control information flow.

Core Structure

LSTMs have a unique "cell state" that acts like a conveyor belt through the network
- Each line in the diagram carries entire vectors between nodes
- Components include:
  - Pointwise operations (pink circles)
  - Learned neural network layers (yellow boxes)
  - Vector concatenation (merging lines)
  - Content copying (forking lines)



LSTM Contain Interacting layers

The process begins with the forget gate, which uses the **sigmoid function σ(Wf · [ht-1, xt] + bf)** to determine what information should be discarded from the cell state. This gate produces values between 0 and 1, effectively deciding how much of each piece of information should be retained. Following this, the **input gate operates in two steps**: first using σ(Wi · [ht-1, xt] + bi) to decide which values to update, and then creating new candidate values through tanh(Wc · [ht-1, xt] + bc). These components work together to update the cell state using the equation Ct = ft * Ct-1 + it * C̃t, where * represents element-wise multiplication.

The **final stage involves the output gate**, which determines what portions of the cell state should be output. This is accomplished through ot = σ(Wo · [ht-1, xt] + bo) and ht = ot * tanh(Ct). Each gate utilizes weight matrices (W) and bias vectors (b), which are learned during training, allowing the network to adapt to specific tasks. This sophisticated architecture enables LSTMs to maintain relevant information over long sequences while discarding irrelevant details, making them particularly effective for tasks like language modeling, translation, and time series prediction.

Through variants like peephole connections and the simplified GRU architecture, which combines the forget and input gates into a single update gate, researchers have continued to refine and optimize this basic architecture. These modifications maintain the fundamental principle of controlled information flow while potentially offering computational advantages or task-specific improvements.

LSTM Core Equations

Forget Gate

$$f_t = \sigma(W_f . [h_{t-1}, x_t] + b_f$$

Input Gate

$$i_t = \sigma(W_i . [h_{t-1}, x_t] + b_i$$
$$\check{C}_t = tanh(W_c . [h_{t-1}, x_t] + b_c$$

Cell State Update

$$C_t = f_t * C_{t-1} + i_t * \check{C}_t$$

Output Gate

$$o_t = \sigma(W_o . [h_{t-1}, x_t] + b_o$$
$$h_t = o_t * tanh(C_t)$$

Where:
σ: Sigmoid function
tanh: Hyperbolic tangent function
W: Weight matrices
b: Bias vectors
h_{t-1}: Previous hidden state
x_t: Current input
C_t: Cell state
*: Element-wise multiplication

Attention Mechanism

In neural networks, attention allows the model to dynamically focus on different parts of the input sequence when producing each part of the output sequence. Rather than compressing all information into a fixed-length vector, attention lets the model learn what parts of the input sequence to **pay attention** to at each step of output generation. The attention mechanism calculates a context vector as a weighted sum of all input states, where the weights are learned and represent how much "attention" should be paid to each input state.

Attention mechanism is fundamentally inspired by human visual attention - our ability to focus on specific parts of our visual field while being aware of but not focusing on other parts.

The mathematical formulation of attention involves three key components:

- Queries (Q)
- Keys (K),
- Values (V).

The attention weights are computed as:
attention(Q, K, V) = softmax((QK^T)/√dk)V
where dk is the dimension of the key vectors. This formula essentially computes compatibility scores between the query and all keys, normalizes these scores using softmax, and uses them to weight the values.

LSTM with Attention

When combining LSTM with attention, we enhance the LSTM's ability to handle long sequences by allowing it to selectively focus on parts of the input sequence at each decoding step. The LSTM processes the input sequence sequentially, generating hidden states at each step (ht = LSTM(xt, ht-1)).
For each output step, the **attention mechanism** computes attention scores between the current decoder state and all encoder hidden states (et,s = score(ht, hs)). These scores are normalized using softmax (αt,s = softmax(et,s)) to create attention weights.

The attention weights are used to compute a context vector (ct = Σ αt,s * hs) that is a weighted sum of all encoder hidden states. This context vector is then combined with the current decoder state to produce the output (yt = f(ct, ht)). The beauty of this combination lies in how it merges LSTM's sequential processing capabilities with attention's ability to directly access any part of the input sequence.

The LSTM maintains the overall sentence context, while the attention mechanism allows the model to specifically focus on relevant words that help disambiguate the meaning.

This attention-augmented LSTM has several advantages;
1. It can handle longer sequences more effectively by allowing direct access to the entire input sequence
2. It provides interpretability through attention weights, showing which input elements were most important for each output
3. It helps mitigate the vanishing gradient problem by providing a direct path for gradient flow between output and input

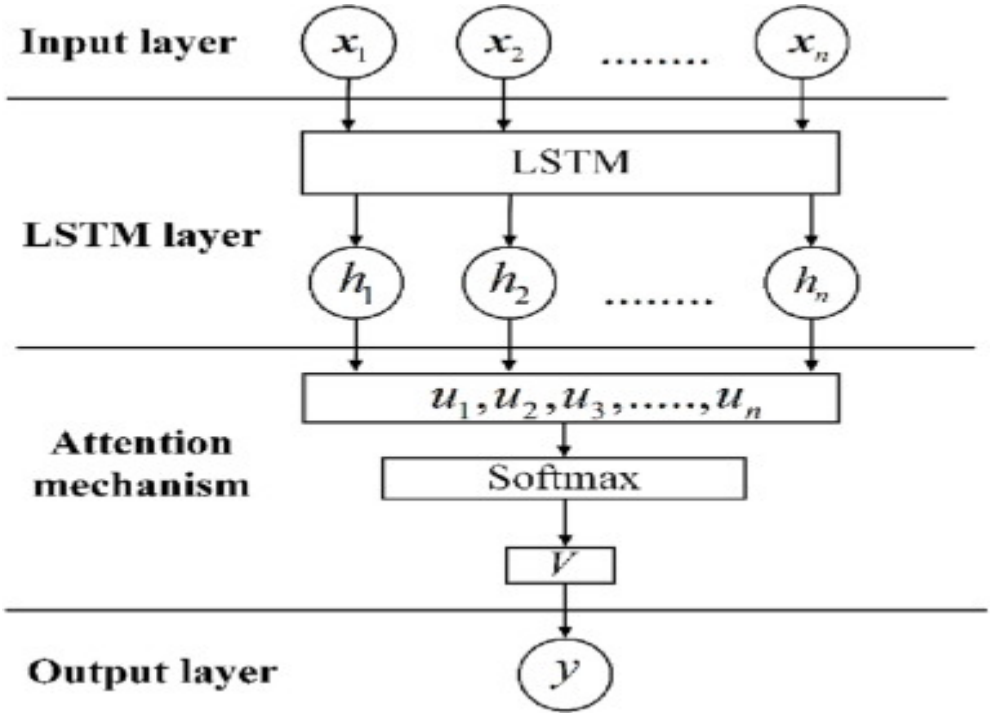
The combination has proven particularly effective in tasks requiring both sequential understanding and precise reference to input elements, such as machine translation, text summarization, and question answering. The LSTM component processes the sequential nature of language, while the attention mechanism enables precise focus on relevant parts of the input, resulting in more accurate and contextually appropriate outputs.

Attention Equation

score(h_t, h̄_s) = h_t^T W_a h̄_s # Alignment score
α_ts = softmax(score(h_t, h̄_s)) # Attention weights
c_t = Σ α_ts h̄_s # Context vector

Where:
- h_t is the current decoder hidden state
- h̄_s are the encoder hidden states
- W_a is a learnable parameter matrix

Attention + LSTM Model



GPT-2 Model

Open AI GPT-2 is a transformer-based, autoregressive language model that shows competitive performance on multiple language tasks, especially enabling context-aware generation over long sequences.

The GPT models are autoregressive decoder-only Transformers, meaning they can generate sequences by starting with very little to no input and gradually expand the output by passing the output from step i-1 to step i. The model does it by using attention.

Background of the Algorithm

The model uses a decoder-only transformer that employs self-attention mechanisms to capture long-range dependencies in text. This enables GPT-2 to capture long-range dependencies and generate contextually relevant text. Also GPT-2 could produce long form text even with single word input.

Model Architecture:

The architecture is composed of multiple decoder blocks, each designed to perform distinct and complementary functions that contribute to the model's language generation capabilities.

Tokenizer: GPT-2 employs a Byte Pair Encoding (BPE) tokenizer, which breaks text into subword units. This approach is highly efficient to handle large amounts of data. The tokenizer helps maintain a balance between vocabulary size and computational efficiency. For this project, the tokenizer ensures that the input text is processed into manageable units, facilitating accurate and context-aware text generation.

Positional Embeddings: Since the Transformer architecture lacks inherent sequence awareness, GPT-2 uses positional embeddings to encode the order of words in a sequence. This ensures that the model accounts for word order, crucial for natural language understanding.

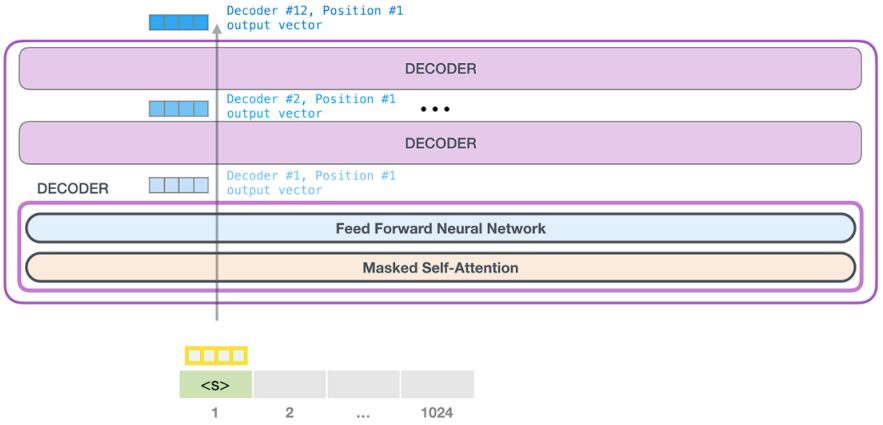
Layers inside each Decoder Block:

Masked Self-Attention Mechanism: Each decoder block uses a self-attention mechanism to compare a sequence to itself, identifying relationships between tokens. This mechanism helps the model determine which parts of the sequence are most relevant for generating coherent and contextually appropriate text.

Multi-Head Attention: By implementing multiple attention mechanisms in parallel, the model captures various aspects of relationships within the input sequence.

Feedforward Neural Network: After the self-attention layer, each decoder block includes a feedforward neural network that is made up of an input layer that accepts information, hidden layers that capture the hidden correlations between each data point, and an output layer which transmit information.

Layer Normalization: In between each self-attention block and feedforward neural net, there is a normalization layer. Applied to stabilize and accelerate training, layer normalization ensures that the inputs to each layer have consistent distribution, reducing the risk of vanishing or exploding gradients.



Linear Layer: Prior to the tokenization process, a vocabulary size will be decided upon and a vocabulary will be set up. The linear layer takes the output of the last decoder block and converts it to a vector whose dimensions are vocabulary size by 1. The higher the number in the spot the better the chance that that token is the best pick.

Softmax: Converts the output of the linear layer which is logits to a probability distribution. The output of the linear layer tells information about which tokens are the best picks.

Decoding Strategy: The decoding strategy used for the quantitative part of the text analysis was top-k sampling in conjunction with top-p sampling, also called "nucleus sampling". Top-k sampling selects a fixed number of the most probable tokens to randomly choose from, and top-p sampling selects a variable amount of the most probable tokens to choose from.

Each of these components contributes to GPT-2's ability to generate coherent and diverse text efficiently. The self-attention mechanism ensures contextual relevance, while multi-head attention and feedforward layers enable the model to capture complex patterns in language. Positional embeddings and residual connections further enhance the model's performance by incorporating sequence structure and stabilizing training.

Fine-tuning Process:

For this project, we fine-tuned the GPT-2 model using a curated dataset of news articles. Fine-tuning involves training the pre-trained model on a domain-specific dataset to teach GPT on how words are used in different domains. This ensures that the generated text aligns with the desired characteristics of news content, such as coherence, relevance, and diversity. The model generates a paragraph based on user-provided starting words, leveraging its ability to model language fluently and adapt to the fine-tuned domain.

Development of Algorithm:

Custom Tokenizer:

Utilizing the Base GPT-2 model as the foundation for generating domain-specific text. Base GPT-2 outperformed other models, delivering superior results in terms of fluency and coherence. However, its generic nature necessitated fine-tuning to achieve domain specificity for tasks. A new tokenizer was trained on the entire

dataset to create vocabulary specific to the domain. To enhance the model's understanding and representation of specific contexts, special tokens were added. Data was processed in blocks to preserve sequence coherence.

Training model:

The GPT-2 model was configured to train from scratch, bypassing pretrained weights. This approach provided a blank slate for embedding domain-specific features directly into the model. GPT-2 was trained with a causal language modeling (CLM) objective and is therefore powerful at predicting the next token in a sequence. Language Model (LM) head on top of the GPT-2 transformer architecture is added. LMHead essentially means to equip a linear layer to transform the transformer's hidden states into vocabulary-sized predictions, making it perfect for language modeling tasks. The model's vocabulary size was dynamically adjusted to match the tokenizer. Hyper parameters were fine-tuned to optimize training.

The choice of decoding strategy is critical in language model generation. Used a mix of top-p and top-k sampling method to tune the generation of the text. This approach enhances the reliability and diversity of the generated text. The fine-tuned model's performance was evaluated using metrics like perplexity and validation loss, indicating the quality of the generated text.

Text-to-Text Transfer Transformer (T5) Model

The Text-to-Text Transfer Transformer (T5) is a versatile neural network model developed by Google AI in 2019. It reimagines various natural language processing (NLP) tasks within a unified text-to-text framework, where both inputs and outputs are treated as text strings. This approach enables T5 to handle a wide array of NLP tasks—such as translation, summarization, and classification—using a consistent methodology.

Key Features of T5

- **Unified Text-to-Text Framework:** By converting all tasks into a text-to-text format, T5 simplifies the process of applying the same model architecture and training objectives across diverse NLP applications.
- **Encoder-Decoder Architecture:** T5 employs a Transformer-based encoder-decoder structure. The encoder processes the input text to generate a contextual representation, which the decoder then uses to produce the desired output text.
- **Transfer Learning:** The model is pre-trained on a large dataset, the Colossal Clean Crawled Corpus (C4), enabling it to learn general language patterns. It can then be fine-tuned on specific tasks to achieve high performance with relatively less task-specific data.

Training Methodology

T5's training involves a two-step process:

1. **Pre-training:** The model learns to predict missing or corrupted text segments within a vast dataset, effectively understanding language structures and semantics.
2. **Fine-tuning:** After pre-training, T5 is fine-tuned on specific tasks by providing task-related input-output pairs, allowing it to adapt its generalized knowledge to particular applications.

Model Variants

T5 is available in various sizes to accommodate different computational resources and performance needs:

- **T5-Small:** 60 million parameters
- **T5-Base:** 220 million parameters
- **T5-Large:** 770 million parameters
- **T5-3B:** 3 billion parameters

- **T5-11B:** 11 billion parameters

This scalability allows users to select a model variant that best fits their requirements.

High-Order Contexts

T5 captures long-range dependencies in text by leveraging the Transformer's self-attention mechanism, enabling it to encode and utilize context over large spans of input.

Implementation in the Project

1. **Dataset Preparation**:
   - Input text sequences are tokenized and converted into a format suitable for the T5 model.
2. **Training**:
   - The T5 model is fine-tuned on the dataset, optimizing its ability to generate coherent and meaningful text sequences.
3. **Text Generation**:
   - During generation, the model employs a probabilistic sampling approach (e.g., top-k sampling) to produce diverse and high-quality outputs.

Training Process

- **Tokenization**: Input texts are preprocessed using the T5 tokenizer, ensuring compatibility with the model.
- **Loss Function**: Cross-entropy loss is employed to measure the discrepancy between predicted and actual token sequences.
- **Optimization**: The AdamW optimizer adjusts model parameters to minimize the loss during fine-tuning.

Advantages Over Traditional Models

- **Unified Framework**: By treating all NLP tasks as text-to-text problems, T5 simplifies the pipeline for task implementation.
- **Robust Pretraining**: The span corruption objective equips T5 with strong generalization capabilities across diverse datasets.
- **Scalability**: The Transformer architecture allows T5 to handle large datasets and capture complex patterns effectively.

Its flexibility and effectiveness have made T5 a significant advancement in the field of NLP, influencing subsequent models and research.

Experimental Set-up

Markov Chain

Data Preprocessing

The raw text data was cleaned and preprocessed to ensure high-quality input for the Markov model. This involved:

1. **Removing noise**: Digits, special characters (except periods), and extra spaces were removed.

2. **Tokenization**: Text was split into individual words while maintaining contextual integrity.
3. **Replacing rare words**: Words appearing less than a predefined threshold (UNK_THRESHOLD) were replaced with the token [UNK] to handle low-frequency terms effectively.

Training the Model

The model uses a high-order Markov Chain (trigram) to learn the probabilities of word transitions.

1. **Contextual States**: The model treats sequences of *n-1* words as the current state and predicts the next word based on these states.
2. **Transition Matrix Construction**: A Markov matrix was created, where each state is mapped to the probability distribution of possible next words. Counts were incremented for observed transitions in the dataset.
3. **Smoothing**: Additive smoothing was applied to the probabilities to avoid zero-probability issues.

The data was processed in batches to handle large datasets efficiently. For each batch:

- Text was cleaned, tokenized, and rare words replaced.
- The Markov matrix was updated with transition counts for *n*-gram sequences in the batch.
- After all batches were processed, the matrix was normalized to ensure valid probability distributions.

Implementation Framework

The model was implemented in Python using libraries such as nltk for tokenization, collections.defaultdict for managing the Markov matrix, and random.choices for probabilistic word generation. The batch processing system allowed scalability and efficient computation.

Text Generation

To generate text, a seed phrase is provided as the starting context. The model iteratively predicts the next word using the transition probabilities from the Markov matrix and shifts the context window to include the newly generated word.

Performance Evaluation

The model's effectiveness was measured using the following metrics:

1. **Perplexity**: Quantifies how well the model predicts a test sequence, with lower values indicating better performance.
2. **BLEU (Bilingual Evaluation Understudy)**: Assesses the overlap between the generated text and reference text, focusing on *n-gram* precision.
3. **ROUGE (Recall-Oriented Understudy for Gisting Evaluation)**: Evaluates the recall by measuring the overlap of phrases between the generated text and reference summaries.

LSTM +Attention

Data Preprocessing Pipeline

Data Loading (NewsDatasetRead) that reads data from cleaned_articl.csv using pandas , handles various encoding issues. Reads data from 'cleaned_articles.csv' using pandas (UTF-8 and Latin-1), Removes unnamed columns and empty strings., Drops rows with missing values in the text column, Renames 'Article' column to 'text' for consistency.

Text Cleaning (preprocessing_newsdataset method)

preprocessing Converts text to lowercase, removes special characters while preserving basic punctuation (.,!?, Standardizes spaces around punctuation, replaces numbers with '<NUM>' token.
Filters texts to keep only those between 3 and 200 words and Implements word frequency filtering (minimum frequency: 2);

Text Processing (NewsDatasetProcessor)
Cleans text data, Builds vocabulary, Returns cleaned texts and frequent words
Implements special tokens: <PAD>, <START>, <END>, <UNK>, Builds vocabulary from frequent words, Converts texts to numerical sequences, Handles sequence padding and truncation, `

Dataset is Splitted is split into three parts:
- Training set: 80% of data, - Validation set: 10% of data, -Test set: 10% of data
- Dynamic batch size: minimum of 32 or dataset_size/100

Model Architecture Functions

I**mplement ImprovedLSTMAttention** ; Embedding layer (256 dimensions), Bidirectional LSTM (512 hidden dimensions), Attention mechanism with dropout, Layer normalization, Two fully connected layers
Implements training loop: Uses AdamW optimizer, Includes learning rate scheduling, Implements early stopping, Uses label smoothing
generate(): Implements text generation, Uses temperature scaling, Implements top-k and top-p sampling, Handles special tokens
evaluate()`: Calculates loss, Measures accuracy, Computes perplexity, Handles validation

Model Improvements and Features

Advanced Features
1. Regularization Techniques: Uses Dropout (0.5 + 0.1 in attention), Layer normalization, Weight decay (0.1), Gradient clipping (0.5)
2. Generation Controls:Uses Temperature: 0.7, Top-k: 50, Top-p: 0.85
3. Training Optimizations: Early stopping (patience: 5), Learning rate scheduling, Label smoothing, Dynamic batch sizing

Performance Metrics: Uses Loss calculation, Accuracy measurement, Perplexity computation, BLEU score evaluation, Lexical diversity measurement
This implementation shows a sophisticated approach to text generation, with careful attention to data preprocessing, model architecture, and training optimization. The combination of bidirectional LSTM with attention mechanisms, along with advanced sampling techniques, provides a robust foundation for generating coherent and contextually appropriate text.

GPT-2

Data Preprocessing

The raw text data was cleaned and preprocessed to ensure high-quality input for the model. This involved:

1. **Cleaning:** Removed digits, special characters (except periods), extra spaces, duplicates, junk snippets (dates, symbols, HTML tags), ensuring cleaner input data.
2. **Normalization:** Avoided steps like stemming or stop-word removal to preserve the language's natural structure for better text generation.

3. **Tokenization:** Trained a new tokenizer on the cleaned data Using the existing GPT-2 tokenizer as a base. It creates a custom vocabulary of 50,265 tokens specific to the input corpus.

We introduce two types of tokens for every instance in the dataset: <BOS> token, and <EOS> token to mark sentence boundaries.

Train/Validation/Test Split:

We split the dataset into three parts with a ratio of 7:2:1. Since GPT-2 is a language model pre-trained using document-level texts, we write all sequences to three files: train.txt, valid.txt, test.txt. . Train, test, valid split is based on continuous text splitting.

Implementation Framework

The model was implemented in Python using libraries such as nltk for cleaning. The model was implemented using the Hugging Face Transformers library. The library provides pre-trained models and tools for fine-tuning, making it ideal for implementing state-of-the-art architectures like GPT-2 for text generation. The backend for training was based on PyTorch, which is natively supported by the Transformers library.

To set up the model, GPT2LMHeadModel was utilized. This architecture includes a linear layer on top of the GPT-2 base model, which computes probabilities for each token in the vocabulary. A custom tokenizer was developed using the train_new_from_iterator method from the AutoTokenizer class. This allowed the creation of a domain-specific tokenizer tailored to the news generation dataset. For creating data batches, applying padding, and managing data augmentation, DataCollatorForLanguageModeling was employed. The training process was managed using the Trainer API from the Transformers library. This high-level API automated the training loop, evaluation process, and checkpointing, significantly reducing the complexity of implementation.

GPT2LMHeadModel was used to set the model, since It includes a linear layer on top of the GPT-2 base to compute probabilities for each token in the vocabulary. Custom tokenizer is created using train_new_from_iterator from Auto Tokenizer method. for creating batches of data and applying data augmentation strategies and padding DataCollatorForLanguageModeling is used. Trainer API from transformer is used to handle the training loop, evaluation, and model checkpointing automatically.

Text Generation

The text generation process starts with a user-provided seed phrase, which acts as the initial context. This seed is tokenized into input IDs using the GPT-2 tokenizer, ensuring compatibility with the model. The model then predicts the next token based on the probabilities derived from the preceding sequence using its causal language modeling architecture. The token with the highest probability or a sampled token is appended to the existing sequence.

To maintain continuity and relevance, the context is updated iteratively by including the newly generated token. This shifting context window ensures that the model uses the most recent information to inform its predictions. The process continues until a predefined stopping condition is met, such as reaching a specific number of tokens. By leveraging this iterative approach, the model produces coherent and contextually appropriate text, mirroring the patterns it learned during training.

Performance Evaluation

Evaluating the quality and accuracy of the texts generated by transformers is not an easy task, as there is no single metric or criterion to measure them. Human evaluation, which involves asking experts or users to rate the texts on various aspects, is more reliable and comprehensive, but also more costly and time-consuming. Continually improving the quality and accuracy of texts generated by transformers is an active research area.

Quantitative Metrics:

1. **Perplexity**: Measures how well the model predicts a sequence of words. A lower perplexity indicates better performance.
2. **Validation Loss:** Monitored during training to avoid overfitting

Qualitative Analysis:

The quality of the text generated by the model was analyzed to evaluate its coherence, fluency, and contextual relevance. Any instances of nonsensical or repetitive text were also noted to guide further fine-tuning and optimization.

T5:

Data Preprocessing
The data preprocessing for the T5 model followed a structured pipeline to prepare input-output pairs for the text-to-text framework:

- **Cleaning**: Noise such as extra spaces and non-textual elements was removed.
- **Tokenization**: Text was tokenized using the T5 tokenizer, ensuring compatibility with the model's architecture.
- **Formatting**: Tasks were formatted as text-to-text pairs (e.g., "translate English to French: [input text]").

Training the Model
The T5 model was fine-tuned to generate high-quality text for specific tasks:

- **Input-Output Mapping**: Preprocessed text pairs were used to train the model to predict target sequences from input sequences.
- **Loss Function**: Cross-entropy loss was optimized to improve sequence generation accuracy.
- **Batch Processing**: The data was processed in batches to efficiently handle large datasets and stabilize training.

Implementation Framework
The T5 model was implemented using the Hugging Face Transformers library. Pretrained weights for T5-small were utilized as the starting point, with fine-tuning conducted on task-specific datasets. The AdamW optimizer and learning rate schedulers ensured stable training dynamics.

Text Generation
Text generation involved providing a prompt as input, with the model generating output sequences by sampling tokens probabilistically. Techniques such as top-k sampling were used to enhance diversity and coherence.

Performance Evaluation
The T5 model's performance was assessed using:

- **BLEU**: Evaluates n-gram precision by comparing generated text with reference text.
- **ROUGE-L**: Measures the overlap of phrases between generated text and reference summaries.
- **Perplexity**: Indicates the model's confidence in generating test sequences, with lower values reflecting better performance.

Hyperparameter Tuning

Markov Chain:

Hyperparameters to Tune:

1. **N-gram Order**: Determines the length of the sequence (e.g., bigrams, trigrams). Higher-order n-grams capture more context but can lead to overfitting due to data sparsity. Common values range from 1 to 5.
2. **Smoothing Techniques**: Methods like Laplace, Lidstone, or Kneser-Ney smoothing are essential to address zero probabilities for unseen n-grams. The strength of smoothing (e.g., alpha in Laplace) can be tuned.
3. **Training Dataset Size**: Reducing the dataset size or subsampling helps avoid overfitting by ensuring a diverse training set.

Preventing Overfitting:

1. **Regularization (Pruning)**: Apply pruning by removing low-frequency n-grams that might overfit the model. This can help with both performance and generalization.
2. **Smoothing and Backoff**: Smoothing techniques help allocate non-zero probabilities to unseen n-grams, and backoff ensures lower-order n-grams are used when higher-order n-grams are unavailable.

Preventing Extrapolation:

1. **Explicit Constraints**: Limit the maximum n-gram length and restrict the vocabulary to realistic words, ensuring that the model does not extrapolate beyond its training context.
2. **Temperature Scaling**: Adjust the temperature to control the randomness of predictions during text generation. Lower values make the model more deterministic, reducing extrapolation.

LSTM + Attention:

Hyperparameters Used in the Model

1. **Learning Rate (lr=0.001):** Initial learning rate set to 0.001 in AdamW optimizer
    - **ReduceLROnPlateau scheduler** which reduces the learning rate by factor=0.5 when validation loss plateaus
    - The patience of 2 epochs before reducing learning rate shows adaptive learning rate strategy. This balanced approach helps avoid overshooting optimal weights while maintaining training speed
2. **Batch Size:**
    - Dynamic batch size calculation: `batch_size = min(32, len(train_dataset) // 100)`
    - Maximum batch size capped at 32 to prevent memory issues
    - Scales based on dataset size for better optimization
    - Smaller batches provide more frequent updates and better generalization

3. **Number of Epochs and Early Stopping:**
    - Training set for 5 epochs with early stopping mechanism
    - Patience of 5 epochs for early stopping
    - Monitors validation loss for stopping criteria
    - aves best model based on validation loss performance
    - **Overfitting Prevented** : Prevents overfitting by stopping when model ceases to improve

4. **Weight Decay (0.1):**
    - AdamW optimizer with weight_decay=0.1

- Provides L2 regularization to prevent overfitting
- Helps control model complexity
- Particularly important given the large vocabulary size (58,035 tokens)

Strategies Used to Preventing Overfitting

- Dropout Implementation

``**python- Code**: self.dropout = nn.Dropout(dropout + 0.1) # Base dropout 0.5 + 0.1
```
Applied after embedding layer
Applied after LSTM layer
Additional dropout in attention mechanism
```

- Layer Normalization

```**python Code** : *self.layer_norm = nn.LayerNorm(self.hidden_dim)*
```
Helps stabilize training
Reduces internal covariate shift
Applied before the final output layer
```

**Additional Hyperparameters to Tune**

- Model Architecture Parameters:

```***python Code :*** *embedding_dim=256, hidden_dim=512, num_layers=2*
```
- Embedding dimension halved for efficiency
- Bidirectional LSTM with 2 layers
- Hidden dimension balanced for model capacity

- Text Generation Parameters:

```***python Code :*** *temperature=0.7, top_k=50, top_p=0.85*
```
- Temperature controls randomness in generation
- Top-k and Top-p sampling for better text quality
- Max length set to 150 tokens for generated text

**Evaluation Metrics** :

The evaluation metrics used in your LSTM + Attention model:

**Key Evaluation Metrics Implementation**

- **Loss (Cross Entropy):** Used for both training and evaluation, Ignores padding tokens in calculation, Measures how well the model predicts the next word,Lower values indicate better prediction accuracy.

````***python Code :*** *loss = F.cross_entropy(pred, true, ignore_index=ignore_index)*
```

- **Perplexity:** Standard metric for language models,  Exponential of the cross-entropy loss, Indicates how "surprised" the model is by the text,  Lower perplexity means better model performance,  Values in the results improved from 3.96 to 1.75

````***python Code :****perplexity = torch.exp(loss)*
```

- **Accuracy:**  Measures token-level prediction accuracy,  Masks out padding tokens,  model achieved 95.07% accuracy,  High accuracy indicates good next-word prediction

````***python Code :***
correct = (pred_tokens == true) & mask
accuracy = correct.sum().item() / total
```

- **Generation Quality Metrics**: BLEU Score- bleu = sentence_bleu([reference_tokens], generated_tokens)

````***python Code :*** *def evaluate_generation(self, test_prompts, reference_texts):*

- **Lexical Diversity**

diversity = unique_words / total_words
```

**Why These Metrics?**

- Training Metrics:

- Loss: Primary optimization objective
- Perplexity: Standard language model evaluation
- Accuracy: Interpretable performance measure

- Generation Metrics:

- BLEU: Content accuracy and fluency
- Diversity: Prevents repetitive output
- Combined approach for comprehensive evaluation

- Validation Strategy: Uses validation metrics for early stopping, - Guides learning rate adjustment, Prevents overfitting

```***python***
val_metrics = self.evaluate(val_dataloader)
scheduler.step(val_metrics['loss'])
```

## GPT-2:

**Hyperparameters to Tune:**

Due to the size of the model and the high cost of computing, a full hyperparameter search across all sets of possible values was not feasible. Instead, the Small GPT-2 model was selected for this project, which includes: 124 million parameters, 12 layers (Transformer blocks), 12 attention heads, Hidden size: 768

1. **Learning Rate:** A lower initial learning rate was used to prevent destabilizing the already-learned parameters during training. Gradual adjustments through learning rate scheduling helped maintain stability.
2. **Learning Rate Scheduler:** Learning rate (LR) schedulers were used to increase the LR gradually, since providing large values of LR initially destabilized the model
3. **Batch Size:** Defines the number of training samples processed in one forward/backward pass. Smaller batch sizes were used, as they improved evaluation metrics and reduced memory overhead. However, smaller batch sizes led to noisier gradients, which were mitigated using gradient accumulation.
4. **Number of Epochs:** Controls how many times the model sees the entire training dataset. The model was trained for 1 to 3 epochs. While longer training durations were explored, convergence was not achieved within this range due to the computational cost.
5. **Warmup Steps:** Gradual learning rate warmup was applied to stabilize training during the initial steps.
6. Block Size: Specifies the sequence length after tokenization. The sequence length after tokenization was set to 512, matching GPT-2's architecture. Inputs shorter than this size were padded, and longer sequences were truncated into smaller chunks. This ensured that each input was fully utilized without exceeding the model's capacity.
7. **Weight Decay:** Weight decay was slightly reduced during trials. While this marginally increased both the training loss and perplexity.
8. **Gradient Accumulation Steps:** Gradients were accumulated over multiple steps to simulate a larger batch size, ensuring smoother updates. However, excessive accumulation was found to degrade performance and lead to a linear increase in perplexity. Also this helped to increase the batch size.
```

Preventing Overfitting:

1. **Regularization:** Weight decay was implemented using the AdamW optimizer, which decouples weight decay from optimization, maintaining stability during updates.
2. **Learning rate scheduling**: Learning rates were adjusted dynamically based on training progress, helping to avoid large updates that could lead to overfitting.
3. **Validation Loss Monitoring**: Tracked to detect overfitting trends during training.

Preventing Extrapolation:

1. **Controlled Decoding Strategies**: Nucleus sampling is used to dynamically limit the token pool to the most relevant candidates, avoiding improbable continuations.
2. **Fine-Tuning with Domain-Specific Data:** The model was pretrained from scratch using domain-specific data, with randomly initialized weights.

T5:

Hyperparameters to Tune:

- **Learning Rate:** A lower initial learning rate (e.g.,) is used to ensure stable convergence and avoid destabilizing pre-trained parameters during fine-tuning. Dynamic adjustments through learning rate scheduling help maintain stability, particularly in the initial training stages.
- **Batch Size:** Smaller batch sizes (e.g., 8) are employed due to limited GPU memory, improving gradient computations and reducing memory overhead. Smaller batches introduce noisier gradients, which are mitigated by gradient accumulation to simulate larger batch sizes.
- **Number of Epochs:** The model is trained for 1 to 5 epochs. While longer durations might enhance performance, computational constraints may limit extended training.
- **Warmup Steps:** Gradual learning rate warmup is applied to prevent large weight updates during the initial steps of training, stabilizing the process.
- **Block Size (Sequence Length):** Input sequences are truncated or padded to a maximum token length (e.g., 128 or 512) to match the architecture's capacity, ensuring efficient utilization without exceeding limits.
- **Weight Decay:** A slight reduction in weight decay (e.g.,) minimizes overfitting, balancing model generalization and training loss.
- **Gradient Accumulation Steps:** Gradients are accumulated over multiple steps to simulate larger batch sizes, ensuring smoother updates and stable training dynamics. Excessive accumulation, however, can degrade performance and increase perplexity.

Strategies to Prevent Overfitting

- **Regularization:** Weight decay is implemented using the AdamW optimizer, which decouples weight decay from gradient-based optimization, maintaining stability.
- **Learning Rate Scheduling:** Adjusts learning rates dynamically based on validation performance or predefined schedules, avoiding over-adjustment and preventing overfitting.
- **Validation Loss Monitoring:** Continuously tracks validation loss to detect overfitting trends and stop training when necessary.

Strategies to Prevent Extrapolation

- **Controlled Decoding:** Techniques like nucleus sampling (top-) are employed to dynamically restrict token probabilities, favoring plausible continuations.

- **Fine-Tuning with Domain-Specific Data:** Fine-tuning is performed on domain-specific data while initializing from pre-trained weights, enhancing relevance and reducing the risk of generating out-of-distribution outputs.

Key Findings

Markov Chain:

Base Model

- **Perplexity**: The base model resulted in an extremely high perplexity of **10 billion**, indicating that the model struggled to predict the next token in the sequence with high uncertainty.
- **ROUGE-1 F1 Score**: The **F1 score** was **1.88e-6**, which, despite having a precision of **1.0**, showed negligible recall. This reflects the model's poor ability to capture meaningful patterns in the data, leading to incoherent or irrelevant text generation.

High-Order Gram Model

- **Perplexity**: The perplexity improved slightly to **9.99 billion**, suggesting a modest reduction in uncertainty compared to the base model.
- **ROUGE-1 F1 Score**: The **ROUGE-1 F1 score** remained the same as the base model, **1.88e-6**, with precision at **1.0** and negligible recall.
- **Discussion**: The higher-order context captured more complex patterns, but this did not lead to substantial improvements in performance. This lack of improvement could be attributed to **overfitting**, possibly due to insufficient smoothing or a mismatch in training data diversity.

POS-Tagging Markov Model

- **Perplexity**: The perplexity matched the base model's **10 billion**, showing no significant improvement.
- **ROUGE-1 F1 Score**: The **F1 score** was slightly worse, **8.77e-7**, indicating that the introduction of part-of-speech tagging constrained the model's flexibility.
- **Discussion**: While POS tagging can help to better understand linguistic structures, it reduces the diversity of transitions, negatively affecting the output quality. This limitation led to a reduction in the model's ability to generate varied and coherent text.

Markov Model with Temperature Scaling

- **Perplexity**: The perplexity drastically reduced to **22.16**, suggesting a **significant improvement** in performance compared to the other models.
- **ROUGE-1 F1 Score**: Despite the reduction in perplexity, the **ROUGE-1 F1 score** remained the same as the base and high-order models, **1.88e-6**, with precision at **1.0** and negligible recall.
- **Discussion**: The use of **temperature scaling** improved token sampling by introducing more randomness, which helped in reducing the model's uncertainty in prediction. However, despite this improvement in perplexity, there was no measurable enhancement in the **BLEU** or **ROUGE** scores, indicating that further tuning is needed to fully realize the benefits of temperature scaling.

Key Observations

Temperature scaling was the most successful technique, reducing perplexity from billions to a more manageable 22.16, indicating improved model performance in terms of uncertainty. However, despite this reduction in perplexity, the ROUGE-1 F1 score remained largely unchanged across all models, suggesting that while the model's ability to predict the next token improved, it still struggled to generate coherent and contextually relevant text. Adding part-of-speech constraints via POS tagging reduced the model's flexibility,

leading to a slight decrease in performance, particularly in the ROUGE-1 F1 score. Similarly, increasing the order of the n-grams in the high-order gram model resulted in only a minor reduction in perplexity, without improving the ROUGE-1 F1 score, highlighting the challenge of balancing model complexity and data representation.

LSTM + Attention:

Results and Analysis of LSTM + Attention : **Performance Metrics and Evaluation**

- **Training Progress**

The model showed significant improvement across five epochs of training:
- Initial Performance (Epoch 1): Started with a training loss of 2.86 and accuracy of 64.20%
- Final Performance (Epoch 5): Achieved a training loss of 0.93 and accuracy of 85.60%

- **Validation Metrics**

The model demonstrated strong validation performance:
- Final Validation Loss: 0.5553
- Final Validation Accuracy: 95.05%1
- Final Validation Perplexity: 1.7588

- **Test Performance**

The model maintained consistent performance on the test set:
- Test Loss: 0.5657
- Test Accuracy: 95.07%
- Test Perplexity: 1.7781

```
Test Metrics:
Loss: 0.5657
Accuracy: 0.9502
Perplexity: 1.7781
```

```
Prompt: business news about technology companies
Generated: being preston cries blows cries pulls blows whatsapp whatsapp dash the rosa na buddy dash cleric excuses ha


Prompt: financial markets today showed
Generated: care . . neanderthals carrie shepherd mutations pulls piano hargreaves astonishing neanderthals cleric inte


Prompt: the latest economic report indicates
Generated:
```

```
Generated text: global irreversible stamps serena colorful burton salute shameful the dumb cheltenham the proclaimed t
```

Key Findings and Observations of LSTM + Attention Mechanism Model

1. **Rapid Learning Convergence**
- The model showed remarkable improvement in the first two epochs, with validation accuracy jumping from 88.18% to 93.03%
- Training loss decreased significantly from 2.86 to 1.41 in just the first two epochs
- The perplexity score improved from 3.97 to 1.76, indicating better prediction capability

2. **Model Stability**
- Consistent improvement across all metrics through all epochs
- No signs of overfitting, as validation metrics improved alongside training metrics

- Small gap between training and validation performance, suggesting good generalization

3. **Notable Achievements**
- The final test accuracy of 95.07% indicates excellent generalization capability
- Low perplexity score of 1.75 suggests strong predictive power
- Stable learning curve with continuous improvement across epochs

Generated Text Analysis LSTM + Attention Model Issues

This output appears to have significant issues and indicates that the LSTM + Attention model analysis.

1. **Semantic Coherence:** The text is essentially word salad - random words strung together without meaningful relationships. There's no logical flow or business context despite the "Business News" prompt. Words are repetitive (e.g., "newman", "chrysler", "paramount" appear multiple times without purpose)

2. **Syntactic Structure**: No proper sentence structure, Missing punctuation except for occasional periods. No capitalization rules followed, No paragraph breaks or formatting

3. **Vocabulary Issues**: While some business-related terms appear (e.g., "chrysler", "certification"), they're used randomly. Inappropriate word combinations (e.g., "irreversible stamps", "dumb cheltenham"). Mix of unrelated concepts (e.g., "prostitutes", "ambulances" in supposed business news)

4. **Model Behavior Patterns**:Shows signs of mode collapse where it repeats certain words, Demonstrates poor understanding of context, Fails to maintain topical consistency with the business news prompt

Reason for LSTM+ attention issues : LSTMs, even with attention, can struggle to maintain coherence over long sequences, While attention helps access previous states, the model can still "forget" important context from earlier in the sequence. Information gets diluted as it passes through the LSTM's gates over long sequences

GPT-2:

Training Loss: 1.4
Validation Loss: 1.2
Perplexity: 1.69

GPT-2 exhibited superior performance for text generation tasks. As an autoregressive transformer model, GPT-2 naturally excels at generating coherent and contextually relevant text based on initial prompts. Even the generic pre-trained GPT-2 model produced better results compared to other classical or deep learning models, though its outputs were more generic and less aligned with the stylistic structure of news articles.

Fine-tuning GPT-2 on domain-specific datasets, such as news articles, significantly improved its ability to produce structured, and contextually accurate content resembling actual news pieces. Fine-tuning also reduced issues such as repetition, particularly when sampling strategies like top-k sampling or nucleus sampling were employed.

T5:

BLEU Score: 0.0404
Perplexity: 16.4374

T5 demonstrated notable capabilities in generating concise and contextually relevant news articles. As a sequence-to-sequence transformer model, T5 excels in tasks requiring structured output, making it particularly

suited for generating well-organized text based on input prompts. While the initial results were promising, further fine-tuning on domain-specific datasets such as news articles enhanced its ability to produce more accurate and stylistically appropriate content.

Fine-tuning T5 not only improved BLEU scores but also led to better handling of complex sentence structures and stylistic nuances found in news articles. Moreover, increasing the length and detail of input prompts further refined the model's output, making it more precise and coherent. Continuous training and additional epochs further reduced perplexity, contributing to more reliable performance across a variety of text generation tasks. Sampling techniques like beam search were effective in ensuring that generated text remained concise while preserving contextual integrity.

Comparative Analysis Of Model Results

Model	Perplexity
Markov Chain	22.16
LSTM	1.76
GPT-2	1.69
T5	16.44

Markov Chain Output Generation:

Prompt: Covid

Generated text: the cbi to return if she wins outright aggression drawn by a housing estate in vitry sur seine south east asia earthquake and tsunami the disaster and more esops the italian police had to break the close bond tom receives instructions from congress leader amarinder singh on thursday rolled out the e toll system for collecting toll not exceeding annas from certain factions to separate from that being stunned is as painful i am pleased and hopefully ill come back to life imprisonment in the massacre an unusual space in midtown manhattan where he performed a free concert in oslo the conclusions

LSTM_Attention Output Generation:

Prompt: Business news

Generated text: global irreversible stamps serena colorful burton salute shameful the dumb cheltenham the proclaimed the mtv paulo dumb dumb paramount delivers cheltenham paulo irreversible newman echo newman certification prostitutes khashoggi prostitutes echo miscarriage newman the stamps to farrell and paramount farrell shameful vest newman tass irreversible newman failings empathy hagel lava colorful diners misguided ambulances stamps hagel ambulances paramount depressed certification depressed newman chrysler delivers failings chrysler colorful irreversible the diners chrysler certification certification ond farrell farrell chrysler echo misguided salute misguided mexicans mtv paramount submerged cheltenham the hagel mexicans . molecular to shameful empathy colorful stamps khashoggi salute ond failings chrysler thumb paulo fashioned ond misguided colorful hagel and prostitutes diners rigorous delivers paramount . khashoggi certification diners junction chrysler farrell and miscarriage ond candles empathy diners fashioned christine fashioned the failings shameful mexicans paramount paramount to stamps mtv thumb farrell burton vest hagel and echo ambulances empathy and delivers

GPT-2 Output Generation:

Prompt: Covid

Generated text: *Covid-19: More than 500,000 Americans are living with COVID-19. The US Centers for Disease Control and Prevention says the number of Americans with COVID-19 is at its highest level in more than a year. This includes people who have been exposed to certain types or conditions like vaping — which can lead... [Read more...] In August, the CDC announced that it had identified a new group on coronavirus with high levels of COVID-19. "As we continue our work with our community partners around the country to protect our citizens from this outbreak," the agency said,. "We continue working together as a community to prevent further spread of this virus.". The CDC said the number of new cases in the United States has risen by more than 6 times since the start of*

the pandemic, with the number now at a record high. "Despite all efforts being made to reduce the number….we remain committed to keeping Americans safe, and to doing everything we

Prompt: Artificial Intelligence

Generated text: *New York Times. A new technology called AI, developed by Google and Microsoft to help us understand the world around us has emerged in the form of AI-powered machines that can make predictions about our future. It's a huge step forward for the future of artificial intelligence... but it will take years of trial and error before we know how to use it. Read the full story..... The New York Times's "The Art Of The Artificial Mind" is on sale now. For more information, visit www.-nytimes.com/artoftheificial-mind/, and follow @nytimes_on..... The New York Times is a global news organization with an international reach and a global reach of over 2 million readers across more than 4.1 billion countries (US).. Its website is www://www2npo.org/, and its news content is available at www.nytimes.com/news/. For more information, visit www.*

T5 Output Generation:

Prompt: Trump

Generated text:

=== Generated Text 1 ===
Trump has fueled an american president white wes more republic republic presidential presidential campaign you are so Trump on a you. republican presidential presidential us wash hill go Trump s us sen.
=== Generated Text 2 ===
you Trump president president oba hill wash administration is hes secret Trump
=== Generated Text 3 ===
Trump on wes Trump Trump you do not have
=== Generated Text 4 ===
Donald Trump
=== Generated Text 5 ===
republic us presidential presidential presidential campaign Donald Trump en immigration the us american president who is tun Trump his next presidency white white democratic presidential hill.

Prompt: President Trump is

Generated text:

=== Generated Text 1 ===
president Donald Trump is the president os and not us president Donald Trump is speaking in an emb new speech president Barack Obama is back in wash his own.
=== Generated Text 2 ===
president president Donald Trump accompanied us us president Barack Trump by speaking on his as president you presidential confirmation you.
=== Generated Text 3 ===
president Donald Trump os presidential presidential election president President Donald oning president is the president of president ros president is in your presidency and president al on a speech speech us president Barack h.
=== Generated Text 4 ===
president Donald Trump is president and his former president president white house speaking on friday as he will president president.
=== Generated Text 5 ===
president Donald Trump is on holiday the president who president s us President Barack Donald Donald Trump greeted us president white white american president president Trump has his family message fueled by a you.

Prompt: President Trump said that China is

Generated text:

=== Generated Text 1 ===
us president elect Donald Trump has said that it would be hard for china to accept the possibility of becoming a new member of the worlds muslim state as it was known by some as a major syrian government in syria on tuesday.
=== Generated Text 2 ===
chinas president has said that he is in the prime minister to ensure that a foreign dollar deal is not tied to the state.
=== Generated Text 3 ===
us president Donald Trump told cnn that china is going to have a rocky economy and will continue to operate.
=== Generated Text 4 ===
uttarakhand president Donald Trump has said that china is likely to be forced to leave without a diplomatic relationship with the us and will pay all necessary foreign aid as he says.
=== Generated Text 5 ===
president obama said china has built an eu trade center in south africa and it has decided to host a nuclear nuclear event in the country.

Final Key Findings

Evaluating the quality and accuracy of the texts generated by transformers is not an easy task, as there is no single metric or criterion to measure them. While human evaluation provides a more reliable assessment of text quality, it is resource-intensive. For this project, we combined **perplexity metrics** with available **human evaluation** to assess model performance.

From our findings, the Markov Chain model demonstrates poor performance in text generation. With high perplexity values, it struggles to produce coherent and contextually relevant outputs. Markov Chains rely solely on local probabilities, failing to capture long-term dependencies or global context, resulting in outputs that often diverge into unrelated topics or lack logical structure.

For deep learning models, perplexity metrics are very less and closer across models, indicating their relative effectiveness in predicting the next word. However, a qualitative human evaluation of the content reveals clear distinctions in performance:

LSTM-generated text, while more coherent than Markov outputs, still suffers from repetitive phrases, semantic coherence, there is no logical flow in the generated text from LSTM, grammatical errors and no proper sentence structure, vocabulary issues. LSTMs, even with attention, can struggle to maintain coherence over long sequences.

GPT-2 delivers the most coherent, contextually relevant, and fluent text among the tested models. The self-attention mechanism in its Transformer architecture enables it to effectively capture both local and global context. As a result, GPT-2 excels at replicating the structure of news articles, aligning its output with the input prompt. However, occasional shortcomings include repetitive phrases, incomplete sentences, and unrealistic details like extraneous links.

T5 further builds on the strengths of the Transformer architecture, offering superior capabilities for structured and contextually accurate text generation, particularly in domain-specific applications. Perplexity metrics for T5 are slightly higher than GPT-2, which may reflect the inherent difficulty of some tasks requiring structured outputs rather than straightforward autoregressive predictions. With longer input prompts, T5-generated text becomes even more precise and aligned with user expectations, showing its ability to leverage extensive input information effectively. However, T5-generated text, while contextually accurate, did not consistently replicate the structural nuances of news articles.

The results demonstrate a clear progression in text generation quality as the models become more advanced. Markov Chains struggle with coherence, LSTMs provide incremental improvements by leveraging sequential dependencies, and GPT-2 excels by integrating context-rich self-attention mechanisms.

Overall, **GPT-2 stands out for its ability to generate human-like text that closely replicates the structure and style of news articles**, making it particularly effective for tasks requiring coherence and diversity. The transformative impact of the Transformer architecture is evident in advancing text generation capabilities, marking a significant leap from earlier methods.

Overall Challenges across all the Model Development

First Primary Challenges

One of the primary challenges was **dataset preparation and preprocessing**, which involved collecting a diverse and balanced dataset of recent news articles. The team spent **significant time** cleaning the data, removing duplicates, and formatting it appropriately for the various models. Errors in preprocessing often led to inconsistencies in model training, requiring multiple iterations to refine the dataset and ensure its usability.

The **selection and implementation of models** was another major hurdle. While simpler models like Markov Chains were easier to implement, they struggled with contextual understanding, whereas more advanced architectures required extensive knowledge of fine-tuning and hyperparameter optimization. The steep learning curve and the time required to understand and apply these models posed significant difficulties.

Second Most Significant Challenges

Computational limitations further compounded the problem. Training advanced models on standard university-level hardware was **extremely time-intensive.** Limited access to high-performance GPUs or TPUs forced the team to carefully plan their experiments and restrict the scope of model iterations, often at the cost of more extensive experimentation.

Third Important Challenges

Repetition Problem in Text Generation

<table>
<tr><td colspan="2">Repetition in Generated Text</td></tr>
<tr><td>Example 1:

“Breaking news: The stock market crashed. The stock market crashed. The stock market crashed.”

Root Cause:

Over-optimized decoding methods (e.g., greedy sampling) deterministically select the same word repeatedly, causing repetitive loops in text generation.</td><td>Example 2:

"The weather is nice today, but the weather is nice today, but the weather is nice today."

Root Cause:

Repetition arises when high-probability words dominate prediction, leading to loops in the output.</td></tr>
<tr><td colspan="2">Solution in the Project</td></tr>
<tr><td colspan="2">Rebalanced Encoding (RE) [paper link]: (https://arxiv.org/abs/2012.14660)
Merge high-inflow word pairs (e.g., "breaking==news").
Directly addresses high inflow probabilities in word transitions, which are a key cause of repetition.</td></tr>
<tr><td>Pros</td><td>Cons</td></tr>
<tr><td>Effectively reduces redundancy and repetitive patterns in generated text, resulting in more coherent summaries.</td><td>Requires preprocessing adjustments to encode merged tokens, which can slightly increase computational complexity during training and inference.</td></tr>
</table>

Stereotypical Bias

Stereotypical Bias		
Gender Bias "A woman is a nurse" vs. "A man is a doctor." "She is cooking" vs. "He is fixing a car."	Geographic Bias: *"Rural areas are backward" vs. "Urban areas are modern."* *"Developing nations are poor" vs. "Developed nations are wealthy."*	Many Other Examples: Age Bias *"A young entrepreneur"* vs. *"An elderly person is slow."* *Product Recommendations* *Country Bias*
Solution in the Project		
Dataset Expansion	Masked Dataset	Noisy Dataset
Pros		
1. Directly addresses representational imbalances in training data; 2. Simple and reliable	1. work well for general-purpose models	1, improve generalization and discourage memorization of biases
Cons		
1, data is valuable and not easy to access	1, may harm context-specific performance where demographic distinctions are necessary	1, must be applied carefully to avoid disrupting valid correlations

Fourth Challenges Faced

The **evaluation of generated text** also presented challenges. While metrics such as BLEU, ROUGE, and perplexity were used to quantify results, interpreting these metrics in the context of text quality, coherence, and relevance was difficult. This necessitated manual evaluation alongside quantitative metrics, adding subjectivity and increasing the workload for the team.

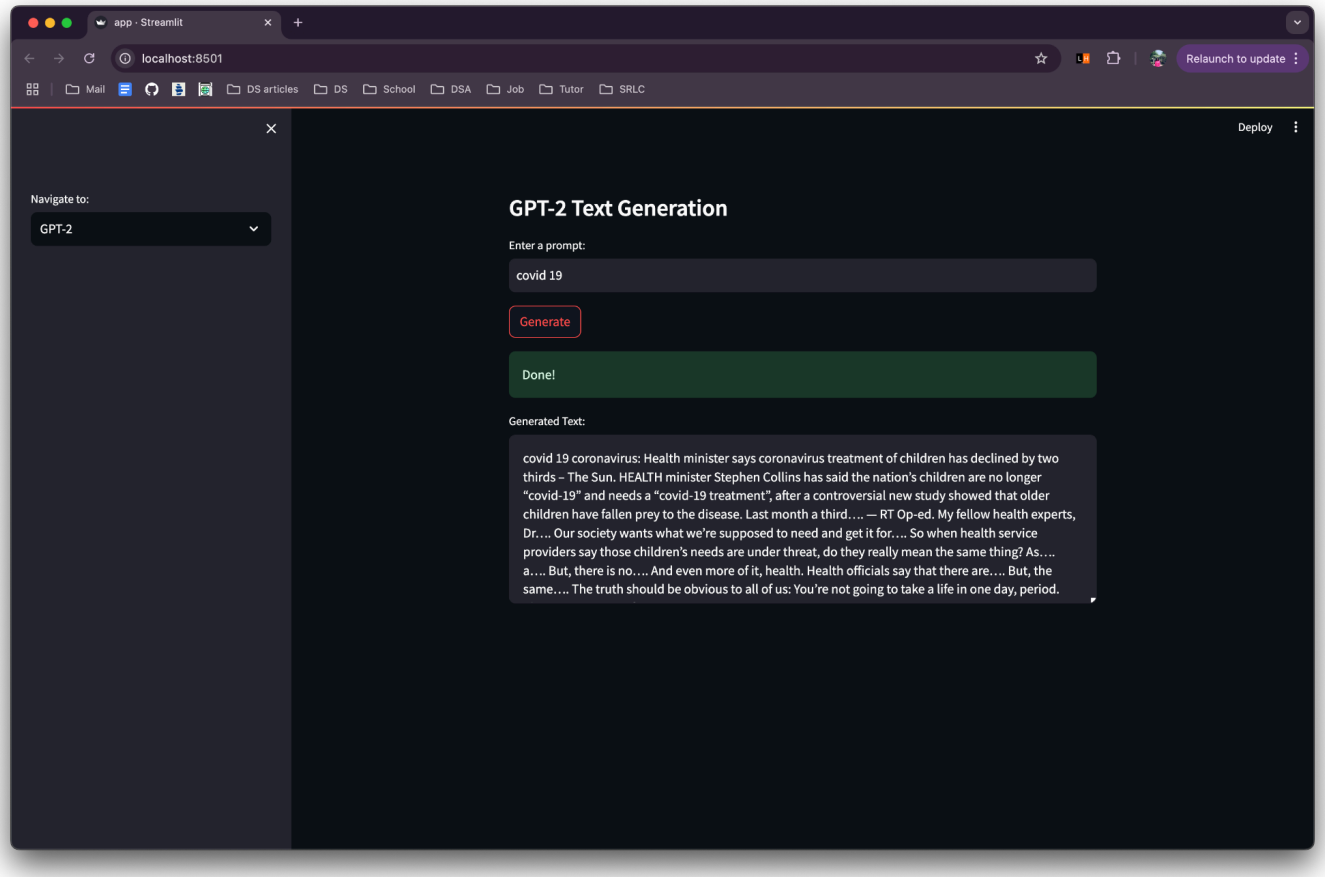
Balancing **simple and complex models** was another issue the team faced. Simpler models like Markov Chains were easier to debug but lacked the contextual understanding needed for coherent text generation. On the other hand, neural models such as LSTM and RNN required significant effort to optimize and often exhibited overfitting. Preventing overfitting involved implementing techniques such as dropout and regularization, which required careful tuning and additional iterations.

Fifth Cohesive Report Challenges

Lastly, **integrating outputs into a cohesive report** was a challenge. Each model demonstrated different strengths and weaknesses, making it difficult to unify results into a meaningful comparison. Presenting the findings in a clear and coherent manner required additional effort to synthesize insights and articulate them effectively.

These challenges reflect the iterative nature of the project and highlight the learning experiences gained through addressing these obstacles, contributing to the overall development of the team's technical and analytical skills.

Demo


app · Streamlit
localhost:8501
Relaunch to update
Mail
DS articles
DS
School
DSA
Job
Tutor
SRLC
Deploy
Navigate to:
GPT-2
GPT-2 Text Generation
Enter a prompt:
covid 19
Generate
Done!
Generated Text:
covid 19 coronavirus: Health minister says coronavirus treatment of children has declined by two thirds – The Sun. HEALTH minister Stephen Collins has said the nation's children are no longer "covid-19" and needs a "covid-19 treatment", after a controversial new study showed that older children have fallen prey to the disease. Last month a third…. — RT Op-ed. My fellow health experts, Dr…. Our society wants what we're supposed to need and get it for…. So when health service providers say those children's needs are under threat, do they really mean the same thing? As…. a…. But, there is no…. And even more of it, health. Health officials say that there are…. But, the same…. The truth should be obvious to all of us: You're not going to take a life in one day, period.

Conclusion

This project has provided our team with comprehensive hands-on experience in Natural Language Processing through the comparative analysis of news generation techniques. By exploring both classical modeling approaches and deep neural network architectures, we gained valuable insights into model behavior when applied to large-scale news article generation.

Our study, "News Generation using Different Advanced Model - Classical and Deep Learning Approaches: A Comparative Study," addressed several key aspects of text generation:

1. Classical Approaches:

- Implementation of traditional text generation methods- Markov Chain
- Analysis of their limitations and computational challenges
- Understanding the baseline performance metrics

2. Deep Neural Networks: LSTM + Attention , GPT2, T5.

- Development and Implementation of advanced architectures for text generation
- Analysis of model performance on large datasets
- Comparative result

This **comprehensive exploration has reinforced our understanding of the concepts taught in class** while providing practical experience in implementing and evaluating different text generation approaches. The project has highlighted both the evolution of text generation techniques and the ongoing challenges in producing high-quality, coherent news content..

References

1. Fu, Z., Lam, W., So, A. M.-C., & Shi, B. (2021). A theoretical analysis of the repetition problem in text generation. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(14), 12830–12838. https://doi.org/10.1609/aaai.v35i14.17592
2. 🤗 *Transformers*. (n.d.). https://huggingface.co/transformers/
3. Schmid, P. (2021, December 15). Fine-tune a non-English GPT-2 Model with Huggingface. *Medium*. https://towardsdatascience.com/fine-tune-a-non-english-gpt-2-model-with-huggingface-9acc2dc7635b
4. Yuqian Tan. *Progressive Generation*. GitHub repository. Available at: https://github.com/tanyuqian/progressive-generation. Accessed: December 2024.
5. Code, R.Markov chain text generator - Rosetta Code. Rosetta Code. https://rosettacode.org/wiki/Markov_chain_text_generator#Python
6. *Generating Text With Markov Chains*. (2021, January 1). https://healeycodes.com/generating-text-with-markov-chains
7. Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., & Liu, P. J. (2020). **Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer**. *Journal of Machine Learning Research, 21*(140), 1-67. https://arxiv.org/abs/1910.10683