Abstractive Text Summarization for Tweets with Custom Transformer Architecture

Project Report

https://github.com/dyinghorizon/NLP_Project-2025

Group No. 20

Nishad Bagade (MT2024102)
Rishabh Kumar Singh (MT2024125)
Kuldeep Chamoli (MT2024081)
Abhishek Kumar Singh (MT2024006)

AIM-829 Natural Language Processing

April 22, 2025

Abstract

This project aims to develop an abstractive summarization system specifically for tweets using a custom transformer architecture implemented from scratch. Due to the lack of large-scale, high-quality tweet summarization datasets, we first built and trained our models on the CNN/DailyMail dataset, with the intention of adapting the trained models for tweet summarization. Our complete pipeline includes custom tokenization, an optimized transformer model, and efficient decoding strategies, followed by application to Twitter data. The project demonstrates the challenges in building transformer-based summarization systems for social media content and provides insights into cross-domain adaptation for summarization tasks.

Contents

1 Introduction

Text summarization is the task of creating concise and fluent summaries while preserving key information from the source text. This project focuses specifically on abstractive summarization for tweets, where the model generates new, condensed text rather than simply extracting portions from the original tweets.

Our primary goal was to develop a transformer model capable of generating meaningful summaries from Twitter data. However, a significant challenge in this domain is the lack of large-scale, high-quality datasets specifically for tweet summarization. To address this, we adopted a two-phase approach:

- First, develop and train base transformer models on the extensive CNN/DailyMail dataset
- Then, adapt these pre-trained models for tweet summarization

This approach allowed us to leverage the strengths of a well-established dataset for initial model development before tackling the more specific challenge of tweet summarization.

2 Dataset

2.1 CNN/DailyMail Dataset

Due to the lack of large-scale tweet summarization datasets, we used the CNN/DailyMail dataset as our primary training corpus. This dataset consists of news articles from CNN and Daily Mail websites paired with human-written summaries (referred to as "highlights"). These summaries are bullet-point style abstracts created by professional editors.

Key statistics of the dataset include:

- Training set: 287,113 article-summary pairs
- Validation set: 13,368 article-summary pairs
- Test set: 11,490 article-summary pairs

While different from Twitter content, this dataset provided several advantages for initial model development:

- Large-scale, high-quality supervised examples necessary for training transformer models
- Professionally written summaries with consistent patterns
- Diverse topics, helping models learn more generalizable text compression skills
- Well-documented benchmark for establishing baseline performance

2.2 Twitter Data

For the tweet summarization phase, we collected data from various political hashtags in India, including:

- #AYODHYAVERDICT
- #cancelallBlueTicksinIndia
- #HistoryOfAyodhya
- #jnuprotest
- #Kashmir
- #KejriwalMustResign
- #ShivSenaCheatsMaharashtra
- #ShutDownJNU
- #WhereIsAmitShah

For each hashtag, we processed tweets to:

- Remove URLs, user mentions, and extra spaces
- Filter for English-language tweets
- Group tweets by hashtag to create coherent documents for summarization

2.3 Data Preprocessing

Our preprocessing pipeline for the CNN/DailyMail dataset included:

- Truncating articles to 512 tokens and summaries to 128 tokens
- Converting padded tokens in target sequences to -100, which PyTorch's loss functions automatically ignore
- Processing the training data in chunks to prevent memory overflow
- Saving processed datasets to disk to avoid repeated preprocessing

For Twitter data, we employed additional preprocessing steps:

- Language detection to filter non-English tweets
- Removal of Twitter-specific elements (hashtags, mentions, URLs)
- Grouping tweets by hashtag to create coherent documents

3 Model Architecture

3.1 Tokenization Strategy

We implemented a custom `OptimizedTokenizer` class using Hugging Face's Rust-based tokenizers library:

- Used Byte-Level Byte-Pair Encoding (BPE) algorithm with a vocabulary size of 32,000
- Defined special tokens: `<pad>`, `<sos>`, `<eos>`, and `<unk>`
- Trained on approximately 100,000 texts including both articles and summaries

This approach was chosen over SentencePiece for performance reasons, providing significant gains during both training and inference, and allowing for explicit control over special token IDs.

3.2 Transformer Architecture

We implemented a custom transformer architecture with several optimizations:

3.2.1 Positional Encoding

Used sinusoidal positional encodings to address the fundamental limitation of transformers: their lack of inherent understanding of sequence order.

3.2.2 Multi-Head Attention

Our `ImprovedMultiHeadAttention` and `FlashMultiHeadAttention` modules include:

- Numerical stability improvements for mixed precision training
- Proper initialization for stable gradient flow
- Flexible masking support for different attention patterns

3.2.3 Feed-Forward Networks

Position-wise feed-forward networks with:

- GELU activation instead of ReLU for smoother gradients
- Careful parameter initialization to prevent vanishing/exploding gradients

3.2.4 Encoder-Decoder Architecture

The encoder and decoder follow the standard transformer design with optimizations:

- Pre-layer normalization (instead of post-layer norm) for improved training stability
- Parameter sharing between encoder/decoder embeddings and output projection
- Beam search for generation with top-k sampling

4 Training

4.1 Hyperparameters

We experimented with different hyperparameter settings across our experiments:

Table 1: Model Hyperparameters for Different Experiments

Parameter	**Initial Model**	**Base Model**	**Larger Model**
Vocabulary Size	32,000	32,000	32,000
Model Dimension (d_{model})	256	512	768
Number of Heads	4	8	12
Feed-Forward Dimension	1024	2048	3072
Number of Layers	3	6	6
Dropout	0.1	0.1	0.2

Table 2: Training Hyperparameters

Parameter	**Initial Model**	**Base Model**	**Larger Model**
Batch Size	8	16	4
Gradient Accumulation Steps	4	4	8
Effective Batch Size	32	64	32
Learning Rate	1e-4	3e-4	5e-5
Weight Decay	0.01	0.01	0.01
Number of Epochs	10	10	10
Label Smoothing	0.1	0.1	0.15

4.2 Loss Function

We implemented a custom `LabelSmoothingLoss` that:

- Redistributes some probability to other tokens (smoothing factor of 0.1-0.15)

- Prevents overconfidence in predictions
- Properly handles padding tokens with an `ignore_index` parameter
- Includes safeguards to prevent index errors

4.3 Learning Rate Schedule

We used a warmup scheduler that:

- Increases learning rate gradually during the initial training phase
- Follows a decay pattern after the warmup period
- Helps stabilize training, particularly in the early phases

5 Experiments and Results

5.1 Experiment 1: Initial Smaller Transformer

Our first experiment implemented a smaller, more lightweight transformer architecture:

- 256-dimensional model with 4 attention heads
- 3 encoder and decoder layers
- Reduced feed-forward dimension (1024)
- Focused on efficiency and faster training time

This model, described in the NLP_Project_Working_Draft1 notebook, served as our initial prototype, allowing us to quickly iterate and establish baseline performance. Despite its smaller size, it demonstrated the ability to learn the summarization task, achieving a validation loss of 4.36 after 10 epochs.

5.2 Experiment 2: Base Transformer Architecture

Based on insights from our initial model, we scaled up to the base transformer configuration:

- 512-dimensional model with 8 attention heads
- 6 encoder and decoder layers
- Standard multi-head attention mechanism

- Trained for 10 epochs on the CNN/DailyMail dataset

The model showed steady improvement in both training and validation loss throughout training. After 10 epochs, the model achieved a validation loss of 3.43, demonstrating better capabilities in capturing document structure and generating coherent summaries.

5.3 Experiment 3: Enhanced Transformer with Layer Scaling

In our third experiment, we implemented an enhanced transformer architecture with:

- Layer scaling parameters in both encoder and decoder
- GELU activation instead of ReLU
- Increased model dimension to 768 with 12 attention heads
- Higher dropout rate (0.2) for better regularization

This model showed faster convergence and better handling of complex patterns in the data, achieving a validation loss of 3.94 after just 5 epochs.

5.4 Experiment 4: Advanced Generation Strategies for Tweet Summarization

For applying our models to tweet summarization, we developed and implemented sophisticated generation strategies to address the unique challenges of social media content:

5.4.1 Approach 1: Standard Temperature-based Sampling

This approach used straightforward auto-regressive generation:

- Fixed maximum output length with early stopping upon EOS token generation
- Temperature-controlled sampling (temperature = 0.7) to balance creativity and coherence
- Applied to grouped tweets from each hashtag

5.4.2 Approach 2: Advanced Decoding with the Interactive Summarizer

We implemented a comprehensive interactive summarization function with numerous optimizations and tunable parameters:

- **Temperature-controlled sampling:** Uses a higher temperature value (0.9) compared to standard generation (0.7) to encourage more diverse and creative outputs while maintaining coherence.
- **Top-k and Top-p (nucleus) sampling:** Combines both methods where:
  - Top-k (value = 60) limits selection to the 60 most probable next tokens
  - Top-p (value = 0.92) dynamically selects the smallest set of tokens whose cumulative probability exceeds 92%
  - This dual approach prevents both overly generic outputs and unlikely token selections
- **Repetition penalization:** Implements a sophisticated n-gram tracking system that:
  - Identifies 2-gram, 3-gram, and 4-gram patterns in the generated text
  - Applies a penalty factor (1.5) to logits for tokens that would create repetitive patterns
  - Dynamically updates the list of tracked n-grams as generation proceeds
  - Particularly crucial for tweet summarization where source content often contains repetitive patterns
- **Abstractive focus priming:** Optionally prefixes the input with a directive to encourage more abstractive rather than extractive summarization, guiding the model to use its own vocabulary rather than copying phrases.
- **Complete sentence generation:** Implements intelligent sentence completion that:
  - Identifies sentence-ending punctuation tokens (periods, question marks, etc.)
  - Continues generation beyond the maximum length if needed to complete a sentence
  - Boosts the probability of sentence-ending tokens when exceeding the target length
  - Ensures summaries end naturally rather than being cut off mid-sentence
- **Minimum length enforcement:** Ensures summaries meet a minimum length threshold (15 tokens) before considering generation complete, preventing overly terse outputs.

The interactive summarizer also included several implementation optimizations:

- Pre-computing encoder outputs to avoid redundant processing
- Custom memory management to handle longer input documents
- Comprehensive error handling and logging for production use
- Configurable generation parameters that can be adjusted at runtime

Algorithm 1 Advanced Summarization Generation Algorithm

Input: Document text, generation parameters
Output: Abstractive summary
Tokenize input text
Compute encoder representation once
Initialize decoder with start token
Initialize n-gram tracking data structure
while not reached stopping criterion **do**
    Generate decoder representation
    Compute logits for next token
    Apply temperature scaling
    if using repetition penalty **then**
        Identify potential n-gram repetitions
        Apply penalties to repeated token logits
    end if
    Apply top-k filtering
    Apply top-p (nucleus) filtering
    Sample next token from filtered distribution
    Add token to generated sequence
    Update n-gram tracking data
    if token is EOS or (length >= max_length and token ends sentence) **then**
        Break
    end if
end while
Decode token sequence to text
return summary

5.4.3 Balancing Parameters for Tweet Summarization

For tweet summarization, we found that different parameter settings were optimal compared to news article summarization:

Table 3: Optimized Generation Parameters for Different Content Types

Parameter	**News Articles**	**Twitter Content**
Temperature	0.7	0.9
Top-k	50	60
Top-p	0.9	0.92
Repetition Penalty	1.2	1.5
Min Length	30	15
Complete Sentences	True	True
Abstractive Focus	False	True

These parameter adjustments reflect the different nature of tweet content compared to news articles:

- Higher temperature and sampling thresholds for tweets to promote more creative integration of often disjointed social media content
- Stronger repetition penalties for tweets due to the inherently more repetitive nature of hashtag-grouped social media content
- Lower minimum length for tweet summaries, matching the brevity of the source content
- Explicit abstractive focus for tweets to discourage direct copying of popular phrases or slogans

5.5 Results on CNN/DailyMail Dataset

Our models achieved the following ROUGE scores on the CNN/DailyMail test set:

Table 4: Results on CNN/DailyMail Test Set

Model	**ROUGE-1**	**ROUGE-2**	**ROUGE-L**
Initial Model	0.1973	0.0418	0.1533
Base Model	0.2315	0.0716	0.2192
Larger Model	0.2483	0.0820	0.2310

5.6 Results on Tweet Summarization

When applied to our Twitter dataset, the models produced the following types of summaries:

- For #AYODHYAVERDICT: Summaries focused on peace, harmony, and respect for the Supreme Court's decision
- For #Kashmir: Summaries highlighted disputes, protests, and international perspectives
- For #KejriwalMustResign: Summaries captured the debate around whether resignation would solve problems

However, we observed several challenges specific to tweet summarization:

- Tokenization issues with hashtags being split into multiple tokens
- Difficulty in handling Twitter-specific language patterns
- Repetition in generated summaries due to repetitive content in source tweets

6 Discussion

6.1 Challenges in Tweet Summarization

Our project highlighted several key challenges in adapting general text summarization models to the Twitter domain:

- **Data scarcity:** Lack of large-scale, high-quality tweet-summary pairs
- **Domain mismatch:** Significant differences between news articles and tweet language
- **Twitter-specific elements:** Hashtags, mentions, and abbreviations require specialized handling
- **Coherence challenges:** Tweets grouped by hashtags often lack narrative coherence

6.2 Individual Model Training

All team members attempted to train their own small transformer (encoder-decoder) models on Kaggle. This parallel experimentation approach allowed us to:

- Compare different implementation strategies

- Identify common challenges and bottlenecks
- Share insights about efficient training approaches
- Pool our knowledge to develop the final consolidated model

This collaborative approach was particularly valuable given the computational demands of transformer training and the complexity of implementing these architectures from scratch.

7 Conclusion

Our project successfully implemented a custom transformer architecture for abstractive text summarization, with a specific focus on tweet summarization. Despite the challenges in adapting models trained on news articles to Twitter content, our approach demonstrates promising steps toward specialized social media summarization.

Key takeaways include:

- The importance of domain-specific data for specialized summarization tasks
- Benefits of a phased approach using larger datasets for initial training
- Challenges in tokenization and generation specific to social media content
- Necessary architectural adaptations for handling shorter, more informal text

Future work could explore domain-specific pre-training strategies, specialized tokenization for social media content, or hybrid approaches that combine extractive and abstractive methods for improved performance on tweet summarization.

8 User Interface for Tweet Summarization

We developed an intuitive web-based interface titled "Interactive Text Summarizer" to make our summarization system accessible to users without technical knowledge. As shown in Figure 1, the interface features:

- A clean, minimalist design with clear section headings
- A large text input area with instructive placeholder text for entering tweets or documents
- Comprehensive parameter controls with sensible defaults:
  - **Temperature** (0.9): Controls randomness with visual slider (range 0.1-1.0) and explanatory text

  - **Max Length** (50): Sets maximum summary length with range indicators (10-100 words)
  - **Min Length** (15): Ensures summaries have sufficient content (range 5-50 words)
  - **Repetition Penalty** (1.5): Controls repetition avoidance with explanatory text
- Toggle options with detailed explanations:
  - **Abstractive Focus**: Prioritizes original wording over extracted phrases
  - **Complete Sentences**: Ensures summaries end with complete sentences

9 Contribution Summary

- **Nishad Bagade (MT2024102):** Led the transformer architecture design and training process. Implemented the custom transformer model, optimized the encoder-decoder components, and managed the training pipeline.
- **Rishabh Kumar Singh (MT2024125):** Focused on preprocessing and frontend development. Implemented the custom tokenization strategy, created the data preprocessing pipeline for CNN/DailyMail dataset, and handled the model interface.
- **Kuldeep Chamoli (MT2024081):** Worked on Twitter data preprocessing. Imported existing Twitter datasets, implemented language detection and cleaning for Twitter-specific content, and organized tweets by hashtags for summarization.
- **Abhishek Kumar Singh (MT2024006):** Developed generation functions for producing summaries. Implemented different decoding strategies, created evaluation metrics, and optimized the generation process for both news articles and tweets.

Additionally, all team members individually attempted to train their own small transformer (encoder-decoder) models on Kaggle, allowing for comparison of different implementation approaches and collaborative improvement of the final system.

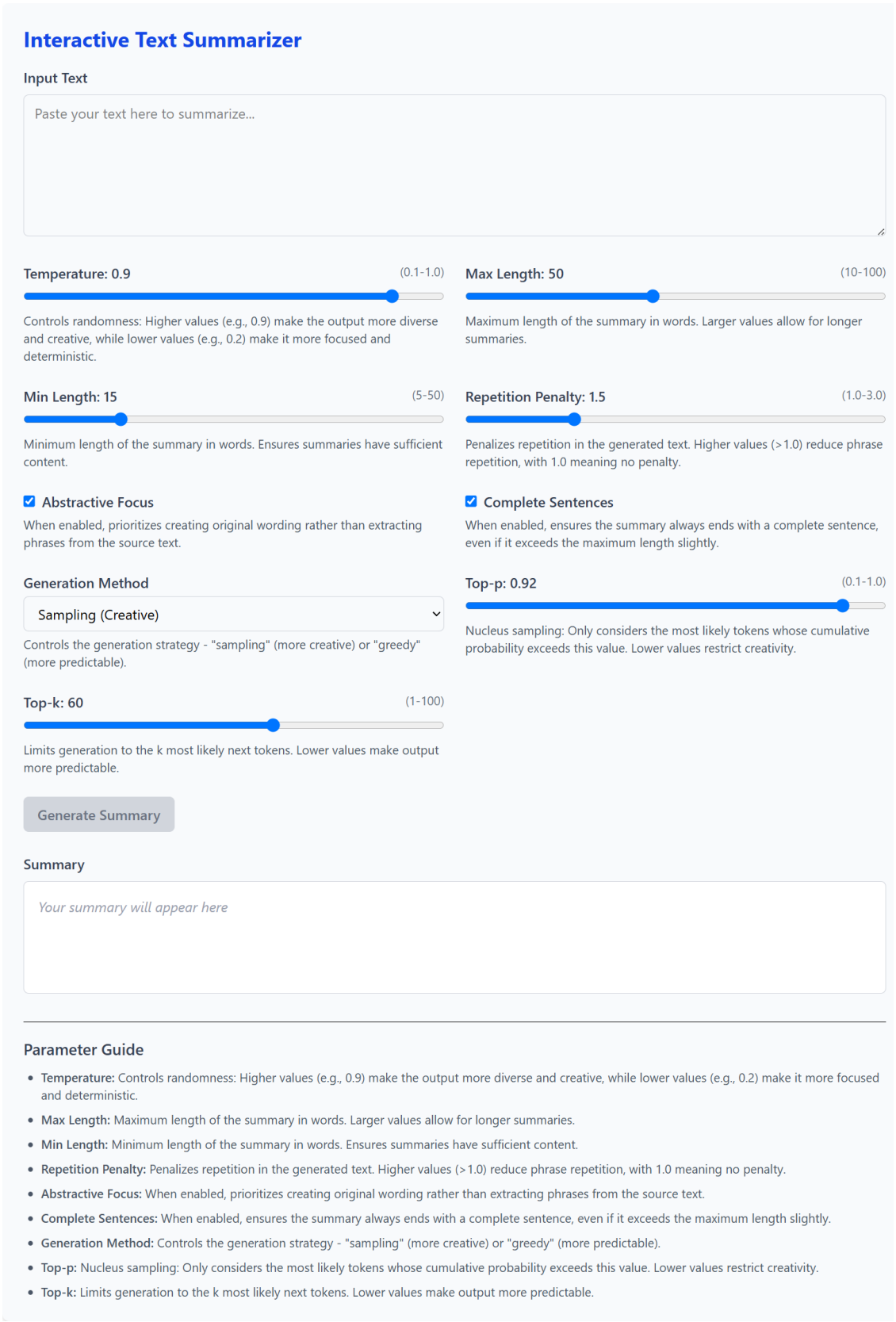

Figure 1: The Interactive Text Summarizer interface.