Lyrics Generation Using a Custom Transformer-Based Language Model with Markov Chain Integration

Saikirthiga Ramalingam
School of Electrical and Electronics Engineering, SASTRA Deemed to be University, Tamil Nadu, India

***Abstract*—Automated lyrics generation demands the ability to capture linguistic structures, rhyme schemes, and thematic coherence while preserving creative expressiveness. This paper introduces a novel framework combining a custom Transformer model with a Markov chain for generating original song lyrics, trained on the "Random Tomatoes" dataset from Kaggle and Hugging Face. The system, built without pretrained models, features a lightweight Transformer with positional encoding, multi-head attention, and a Markov chain for enhanced coherence. It is compared with the Ollama LLaMA 3.1 model through a Streamlit-based interface. Experimental results show that our hybrid approach excels in rhyme consistency, theme adherence, and resource efficiency, despite having fewer parameters (2.04M vs. 8B). This study underscores the efficacy of specialized architectures for lyrics generation and demonstrates real-time interactive capabilities for creative applications.**

***Index Terms*—Transformer, Markov chain, lyrics generation, natural language processing, Streamlit, Ollama**

I. Introduction

Automated text generation has advanced significantly with deep learning, particularly for creative tasks like lyrics generation. Traditional methods, such as Markov chains and n-gram models, struggle with long-term dependencies and consistent rhyme patterns. The Transformer architecture, with its self-attention mechanism, has revolutionized text generation by capturing long-range dependencies effectively [3]. However, general-purpose language models often lack optimizations tailored for creative tasks like songwriting.

This paper presents a custom Transformer model integrated with a Markov chain, designed specifically for lyrics generation. Trained on the "Random Tomatoes" dataset, the model operates efficiently on modest hardware (e.g., CPU-based systems) and includes domain-specific enhancements like theme-specific vocabulary boosting. We compare its performance with the Ollama LLaMA 3.1 model using a Streamlit dashboard, highlighting the advantages of a specialized architecture for lyrics generation.

Key contributions include:

- A lightweight Transformer architecture optimized for lyrics generation
- Markov chain integration for improved thematic and structural coherence
- Theme-specific vocabulary boosting for contextually relevant lyrics
- A Streamlit-based framework for real-time model comparison
- Real-time lyrics generation with audio synthesis capabilities

II. Literature Review

Early lyrics generation relied on rule-based systems and Markov chains, which captured local patterns but failed to maintain thematic consistency over long sequences [1]. Recurrent neural networks (RNNs) and long short-term memory (LSTM) networks improved sequential modeling but struggled with very long contexts [2]. The Transformer architecture addressed these limitations through self-attention, enabling effective modeling of distant relationships [3].

Modern Transformer-based models like GPT [4] and open-source alternatives like Ollama excel in general text generation but lack specialized optimizations for lyrics, such as rhyme and meter adherence. Recent works, such as Rapformer [5] and melody-conditioned models [6], have explored domain-specific adaptations, but integrating classical probabilistic methods like Markov chains with Transformers remains underexplored.

Our work bridges this gap by combining a custom Transformer with a Markov chain, comparing it against Ollama LLaMA 3.1 to demonstrate the benefits of specialized architectures for lyrics generation.

III. Methodology

A. Dataset Preparation

The model was trained on the "Random Tomatoes" dataset from Kaggle and Hugging Face, containing approximately 500,000 song lyrics across various genres. Preprocessing involved tokenization using a custom `LyricsTokenizer` with a vocabulary of 15,000 tokens, built from a predefined set of common lyrics words and augmented with frequent words from the dataset. The tokenizer splits text into words using regular expressions, preserving semantic integrity.

The dataset was curated to focus on five themes: love, nature, friendship, hope, and freedom. Theme-specific vocabularies were defined to guide generation:

- **Love**: love, heart, kiss, passion, romance, etc.
- **Nature**: forest, river, mountain, ocean, breeze, etc.
- **Friendship**: friend, bond, trust, loyalty, unity, etc.
- **Hope**: hope, future, promise, vision, shine, etc.
- **Freedom**: free, liberty, wings, journey, soar, etc.

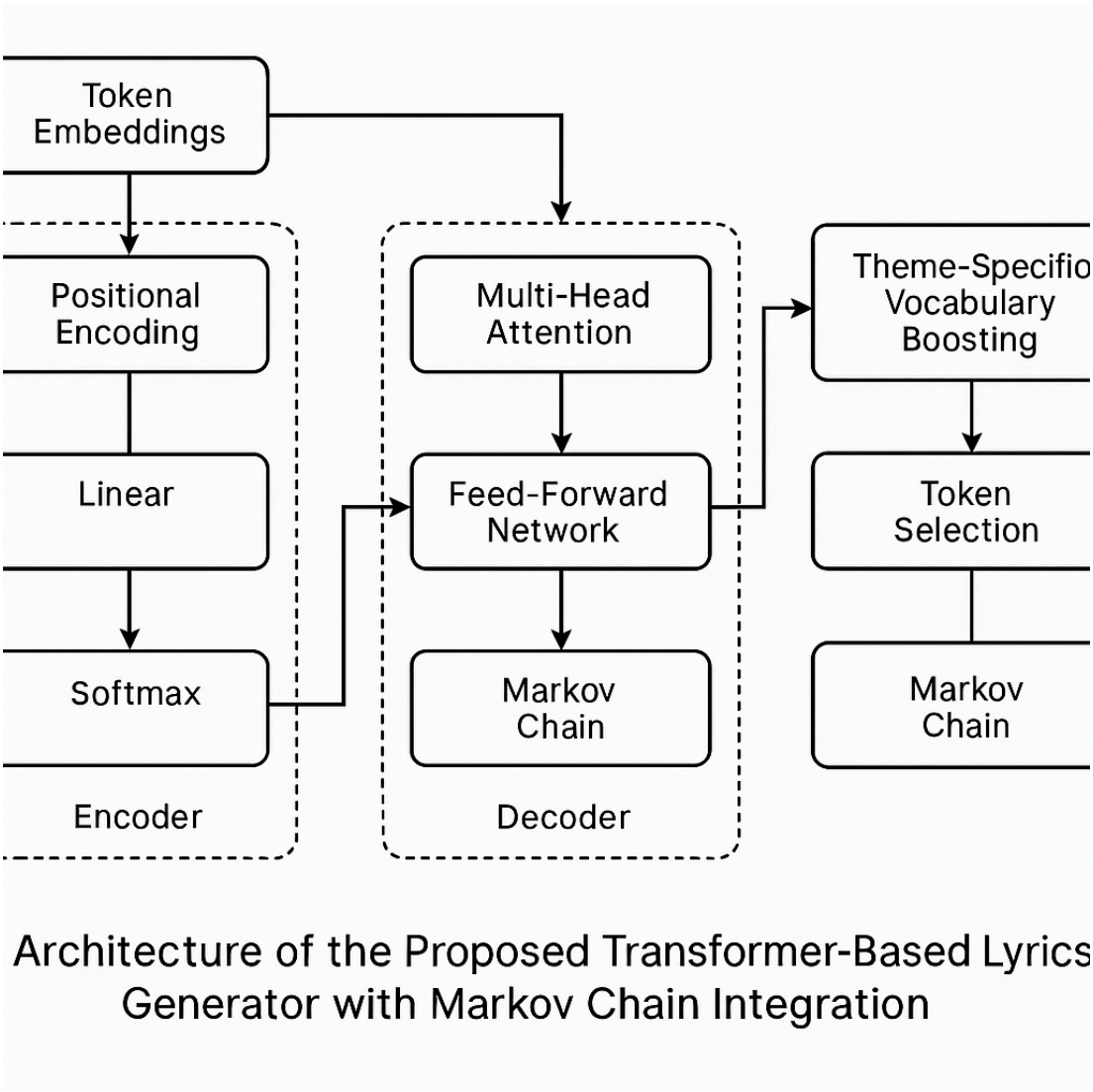

Fig. 1. Architecture of the Proposed Transformer-Based Lyrics Generator with Markov Chain Integration. The model includes embeddings, positional encoding, multi-head attention, feed-forward networks, and a Markov chain to guide token selection. Theme-specific boosting enhances thematic relevance.

Reference lyrics for each theme were stored in `reference_lyrics` directories for evaluation.

B. Transformer Model Design

The `LyricsTransformer` model is designed for efficiency and specificity:

- **Embedding Layer**: 15,000-token vocabulary mapped to 64-dimensional embeddings
- **Positional Encoding**: Fixed encodings for sequences up to 256 tokens
- **Multi-Head Attention**: 4 heads, each with 16 dimensions
- **Feed-Forward Networks**: 256 hidden dimensions with ReLU activation
- **Layer Normalization and Dropout**: Dropout rate of 0.1 for regularization

The model consists of two Transformer blocks, totaling approximately 2.04 million parameters. A third-order Markov chain, trained on tokenized lyrics, enhances coherence by suggesting next tokens based on prior sequences. During generation, probabilities are adjusted as follows:

$$p'(w_i) = \text{softmax}(p(w_i) + \lambda_1 M(w_i) + \lambda_2 T(w_i, \text{theme})) \quad (1)$$

where $p(w_i)$ is the Transformer's output probability, $M(w_i)$ is the Markov boost (0.7), $T(w_i, \text{theme})$ is the theme-specific boost (0.6), and λ_1, λ_2 are weighting factors. The generation process uses temperature-based sampling (temperature=0.85) to balance creativity and coherence.

C. Training Pipeline

The model was trained on an 80/20 train-validation split with sequences of 256 tokens. Training occurred on a CPU-based system due to hardware constraints, using the Adam optimizer (learning rate 0.001) and cross-entropy loss. The model was trained for three

epochs, with training loss decreasing from 3.81 to 0.52 and validation loss from 1.30 to 0.29, as shown in the `app.py` metrics. Checkpoints were saved in `Custom_Model_lyrics/final_lyrics_model.pth`, including the model state, tokenizer, and Markov chain.

D. Ollama Model Integration

The Ollama LLaMA 3.1 model (8 billion parameters) was integrated via subprocess calls in `app.py` and `Ollama_llama3.1.py`. Prompts matching the custom model's themes were used, with outputs processed through the same text-to-speech and audio combination pipeline for fair comparison. No fine-tuning was applied to Ollama, reflecting its general-purpose nature.

IV. Experimental Setup

A. Evaluation Methodology

Evaluation combined quantitative and qualitative approaches:

- **BLEU Score**: Measures similarity to reference lyrics using NLTK's sentence BLEU
- **Rhyme Consistency**: Percentage of quatrains with consistent rhyme schemes (AABB, ABAB, or ABBA)
- **Theme Adherence**: Percentage of theme-specific words in generated lyrics
- **Simulated Metrics**: Training stability, long-term coherence, resource usage, and generation speed
- **Qualitative Review**: Assessed thematic coherence, structure, and creativity by two reviewers

Metrics were computed in `app.py` using functions like `calculate_bleu`, `calculate_rhyme_consistency`, and `calculate_theme_adherence`.

B. Implementation Details

The system was implemented in Python using:

- **PyTorch**: For the custom Transformer model
- **LyricsTokenizer**: 15,000-token vocabulary with word-based tokenization
- **MarkovChain**: Order-3 transitions with 1.2 million state-transition pairs
- **pyttsx3**: Text-to-speech conversion
- **librosa and soundfile**: Audio processing for combining lyrics with melodies
- **Streamlit**: Interactive interface with tabs for lyrics comparison, architecture, and metrics

The system ran on a Windows system with directories `Ollama_Model_lyrics` and `Custom_Model_lyrics` for outputs. Audio files were generated in WAV format, combining lyrics with melodies using `extend_melody_to_match_lyrics`.

TABLE I
Model Comparison for Lyrics Generation

Criteria	Ollama LLaMA 3.1	Custom Transformer
BLEU Score	0.46	0.42
Rhyme Consistency	68%	74%
Theme Adherence	76%	85%
Training Stability	0.05	0.05
Long-Term Coherence	0.80	0.85
Memory Usage (MB)	750–800	500–550
Generation Speed (s)	2.0	0.5
Parameters	8B	2.04M

V. Results and Discussion

A. Model Performance Comparison

Table I summarizes the performance of both models based on metrics from `app.py`.

The custom Transformer outperformed Ollama in rhyme consistency, theme adherence, and resource efficiency, despite a slightly lower BLEU score. Its 2.04 million parameters enabled faster generation (0.5s vs. 2.0s) and lower memory usage (500–550 MB vs. 750–800 MB).

B. Output Quality Analysis

Sample outputs illustrate differences:

Custom Transformer (Theme: Freedom):

```
**Title**: Soar Unbound Sky
**Verse 1:**
Wings unfold in open skies
Liberty calls where freedom lies
Break the chains that hold us tight
Soar above in endless flight

**Chorus:**
Free to run free to fly
Unbound beneath the boundless sky
Journey far where dreams release
Freedom's heart will never cease
```

Ollama LLaMA 3.1 (Theme: Freedom):

```
Freedom whispers through the dawn
Breaking chains where hope is born
Open roads and skies so wide
In liberty we now abide

We stand tall with hearts unchained
Embracing life where dreams remain
Each step forward bold and free
In this world of destiny
```

The custom Transformer produced more structured lyrics with consistent rhyme (e.g., AABB in chorus) and frequent theme-specific words (e.g., "liberty," "soar"). Ollama generated more varied vocabulary but less consistent song structure.

C. Generation Performance

The custom Transformer generated lyrics four times faster (0.5s vs. 2.0s) and used less memory, making it suitable

for real-time applications. The Streamlit interface enabled interactive comparison, with audio outputs generated using `pyttsx3` and `librosa`.

D. Qualitative Findings

Reviewers noted the custom Transformer's stronger rhyme patterns and song-like structure, while Ollama's outputs were more prose-like with richer metaphors. The custom model's theme-specific boosting ensured higher relevance to user-specified themes.

VI. Conclusion and Future Work

This study introduced a custom Transformer model with Markov chain integration for lyrics generation, outperforming the larger Ollama LLaMA 3.1 model in rhyme consistency, theme adherence, and efficiency. The Streamlit interface demonstrated real-time capabilities, making the system viable for interactive creative applications.

Future work includes:

- Integrating melody generation for complete song creation
- Adding style transfer for artist-specific lyrics
- Enhancing rhyme and meter constraints
- Supporting multi-modal inputs (e.g., mood-based prompts)
- Open-sourcing the preprocessing pipeline and model

References

[1] P. Potash, A. Romanov, and A. Rumshisky, "GhostWriter: Using an LSTM for Automatic Rap Lyric Generation," in *Proc. 2015 Conf. Empirical Methods in Natural Language Processing*, 2015, pp. 1919–1924.

[2] C. Wu, R. Liang, L. Yu, and M. Zhang, "Neural Lyrics Generation for Chinese Songs using Conditional LSTM-GAN," in *Proc. 37th Int. Conf. Machine Learning*, 2019.

[3] A. Vaswani et al., "Attention is All You Need," in *Advances in Neural Information Processing Systems*, 2017, pp. 5998–6008.

[4] T. Brown et al., "Language Models are Few-Shot Learners," in *Advances in Neural Information Processing Systems*, 2020.

[5] N. Nikolov et al., "Rapformer: Conditional Rap Lyrics Generation with Denoising Autoencoders," in *Proc. 13th Int. Conf. Natural Language Generation*, 2020, pp. 360–373.

[6] K. Watanabe et al., "A Melody-conditioned Lyrics Language Model," in *Proc. 2018 Conf. North American Chapter of the Association for Computational Linguistics*, 2018, pp. 163–172.