CSE 598 Project Report
A Comparative Analysis of Deep Learning Frameworks for Music Recommendation Systems
Group - 17

Aman Bhala [*] Ishan Bansal [†] Kartikeya Kansal [‡]

1 Problem Formulation

While music recommendation systems have been extensively explored, there exists a significant gap in the literature concerning the comprehensive performance analysis of such models across different deep learning frameworks, particularly in the context of PyTorch and TensorFlow2.x. Despite the critical role these frameworks play in the development and deployment of machine learning models, there is a noticeable lack of studies that systematically evaluate and compare their performance specifically for music recommendation tasks. This absence of comparative analysis not only hampers the advancement of more efficient and accurate music recommendation systems but also limits the ability of developers and researchers to make informed decisions regarding the selection of a framework that best suits their needs. Bridging this gap is essential for pushing the boundaries of what is possible in music recommendation, enhancing the user experience through more personalized and accurate suggestions, and optimizing the computational efficiency of these systems.

2 Proposed Solution

In this project, we propose to develop and conduct a comprehensive performance analysis of music recommendation engines using PyTorch and TensorFlow2.x. Our approach integrated both collaborative and content-filtering techniques to provide personalized music recommendations. For content-based filtering, we explored some clustering mechanisms whereas for collaborative filtering we analyzed user-item interactions and preferences, drawing insights from large-scale datasets to predict user preferences for music tracks or genres they have not yet encountered.

We evaluated these hybrid recommendation systems across various music datasets, including the Million Song Dataset, the FMA dataset and the Last.fm Dataset . These datasets offer a rich diversity of music genres, and user preferences, making them ideal for assessing the effectiveness of our recommendation models. By combining collaborative and content filtering, we anticipate our

[*]abhala1@asu.edu
[†]ibansal5@asu.edu
[‡]kkansal1@asu.edu

models will more accurately capture and predict user preferences, leading to more personalized and satisfying music-listening experiences.

To ensure a fair and meaningful comparison, we designed the model architecture and hyperparameters to be consistent across both frameworks. While there are pre-trained models available for tasks like image classification, the domain of music recommendation systems presents unique challenges that necessitate the development of specialized models. Given the lack of widely adopted pre-trained models for music recommendation, we created our models, paying close attention to feature extraction, sequence modeling, and personalization techniques, maintaining identical hyperparameters in both PyTorch and TensorFlow2.x to the extent possible.

Our analysis not only compares the performance of these models in terms of accuracy, precision, recall, and computational efficiency but also evaluates their scalability, ease of implementation, and adaptability to different datasets. This systematic evaluation aims to provide valuable insights into the strengths and limitations of PyTorch and TensorFlow2.x in the context of music recommendation, guiding future development and framework selection for researchers and practitioners in the field.

3 Literature Survey

The development and evaluation of music recommendation systems have been the subject of extensive research, exploring various methodologies ranging from collaborative and content-based filtering to more complex deep learning approaches. This literature survey highlights key studies and developments in the field, focusing on the utilization of collaborative filtering, content-based filtering, and deep learning techniques within music recommendation systems. Additionally, it sheds light on the comparative analysis of deep learning frameworks, particularly PyTorch and TensorFlow2.x, in various domains.

In [1], they provide a comprehensive overview of collaborative filtering techniques applied to music recommendation, emphasizing the challenges of recommending long-tail tracks. In [2], they explore the use of CNNs for extracting features from audio data, demonstrating the effectiveness of multi-modal approaches in music genre classification. [3], their research not only applies deep learning for sound classification but also discusses various evaluation metrics that can be adapted for assessing music recommendation systems, such as accuracy, precision, and recall.

4 Dataset

The dataset presented in Table 1 is a substantial compilation of lyrical content from the website LyricsFreak, meticulously gathered through web scraping techniques. It constitutes a rich repository of 57,650 songs in the English language, providing a robust foundation for the development of a content-based filtering music recommendation system. Each entry in the dataset is structured with several fields, offering a comprehensive snapshot of the song's metadata: the name of the artist, the title of the song, a unique link to the lyrics page on LyricsFreak, and a text snippet from the lyrics themselves.

This meticulous assemblage of data is of paramount importance for machine learning models that aim to analyze and understand patterns within musical content. The 'artist' field is crucial as it enables the recommendation system to identify and suggest songs by the same artist, catering to listeners' preferences for a particular singer or band. The 'song' field, which holds the title, allows for the easy identification of individual tracks. Most importantly, the 'link' field provides a direct URL to the full lyrics, facilitating a deeper analysis of the song's textual content, such as themes, vocabulary, and sentiment. The 'text' field containing the lyrics excerpt is an essential feature for preliminary filtering and quick content analysis, which can be used for immediate categorization of songs based on the language used.

This dataset is designed not only to fuel the recommendation engine but also to enable detailed textual analysis that can lead to insights into the trends, patterns, and emotional landscapes that prevail in music lyrics. Such analysis can underpin the engine's capability to offer personalized song suggestions based on lyrical content, thereby enhancing the user experience. Overall, this dataset serves as a foundational component in bridging the gap between complex algorithmic processes and the nuanced, emotional connections that users form with music.

Artist	**Song**	**Link**	**Text Snippet**
ABBA	Ahe's My Kind Of Girl	/a/abba/ahes+my+kind+of+girl_20598417.html	Look at her face, it's a wonderful face...
ABBA	Andante, Andante	/a/abba/andante+andante_20002708.html	Take it easy with me, please...
ABBA	As Good As New	/a/abba/as+good+as+new_20003033.html	I'll never know why I had to go...
ABBA	Bang	/a/abba/bang_20598415.html	Making somebody happy is a question of give and...
ABBA	Bang-A-Boomerang	/a/abba/bang+a+boomerang_20002668.html	Making somebody happy is a question of give and...

Table 1: Sample entries from the music recommendation dataset.

5 Model Architecture

This section is designed to detail the process by which our machine learning model, for content-based filtering, processes and learns from textual data to provide recommendations. This model is commonly used in recommendation systems for items like songs, movies, or products, where the content of the item is used to recommend similar items to users.

The first stage of the model involves **Text Tokenization and Sequence Padding**. It begins with the initialization of a tokenizer. In this context, the tokenizer is configured to handle a vocabulary size of 10,000 words, which helps in converting the text into a manageable set of tokens that the model can understand. Any word not included in the tokenizer's vocabulary is assigned an out-of-vocabulary token, commonly denoted as "*oov*".

Once the tokenizer is set up, it proceeds to **Fit Tokenizer**, where the tokenizer is exposed to the dataset—typically a collection of text, (song lyrics, in our case) and learns to map each unique word to a specific integer index. This mapping is crucial as it transforms the textual data into a numerical format that the neural network can process.

The next step is **Text to Sequences**, where the actual conversion of text data into sequences of integers takes place. Each integer represents a specific word in the learned vocabulary. This transformation allows the model to treat the text structurally, assessing each word according to its position and significance in the sequence.

Finally, **Pad Sequences** is the process of standardizing the length of all sequences. Since input data can vary in length, and neural networks require a uniform input shape, sequences are truncated or padded with zeros to ensure they are all of the same specified length, in this case, 120 words.

The architecture then outlines the layers involved in the neural network:

The **Embedding Layer** is the first layer of the neural network where each integer in the sequence is mapped to a dense vector of fixed size. This layer is essential for the model to understand and process the text data effectively. The **Average Pooling Layer** then simplifies the output of the embedding layer by averaging over the sequence dimension, reducing the data's complexity and preparing it for the dense layers. A **Dense Layer** follows, which is a fully connected layer of the neural network. Here, the model learns to detect patterns in the averaged embeddings, using the rectified linear unit (ReLU) activation function to introduce non-linearity into the model, allowing it to learn more complex patterns. The final **Dense Layer** is where the model outputs its predictions. The size of this layer corresponds to the number of unique items (e.g., songs) in the dataset, and it uses the softmax activation function to provide a probability distribution over all possible items. The softmax function ensures that the output probabilities sum to one, making it easier to interpret the model's predictions as confidence scores for recommending each item.

6 Methodology

6.1 Content Based Filtering

Content-based filtering is a recommendation technique that suggests items similar to those a user has previously liked or interacted with. This method primarily relies on the features of the items themselves, rather than user-user similarities. It creates a profile for each item, incorporating relevant characteristics such as genre, keywords, or any specific attributes pertinent to the item's category. For instance, in a movie recommendation system, features might include genre, director, and major actors.

The user profile is constructed based on the characteristics of items that the user has rated highly or interacted with frequently. Algorithms analyze these interactions to identify patterns or preferences in the user's behavior, which are then used to predict and recommend new items that share similar features.

Content-based filtering has several advantages. It allows for personalized recommendations as it builds a unique profile for each user. Additionally, it does not require data about other users, thus avoiding the cold-start problem associated with new users in collaborative filtering systems. However, it can lead to a lack of diversity in recommendations, as the system tends to recommend items that are too similar to those already consumed by the user. Moreover, its effectiveness heavily depends on the richness and accuracy of the metadata associated with each item.

6.2 Implementation

In the domain of music recommendation systems, content-based filtering has seen innovative applications, particularly through the analysis of song lyrics. A notable implementation of this technique is the development of a recommendation engine that leverages Natural Language Processing (NLP) to parse and understand the lyrical content of songs, aiming to recommend tracks with similar thematic or emotional content.

The first step in this process involves the extraction and preprocessing of lyrics. Lyrics are collected from a comprehensive database and are subjected to standard NLP preprocessing methods, such as tokenization, stop-word removal, and stemming. This preprocessing helps in reducing the complexity of the text and focuses on the most meaningful elements of the lyrics.

Following this, the next phase involves feature extraction where the preprocessed lyrics are converted into a numerical format that can be processed by machine learning algorithms. Techniques such as Term Frequency-Inverse Document Frequency (TF-IDF) or word embeddings like Word2Vec are used to vectorize the lyrics. These vectors effectively capture the semantic and syntactic essence of the lyrics, allowing for the quantification of similarities between songs based on their lyrical content.

Once the feature set is prepared, the recommendation engine employs clustering algorithms, such as K-means or hierarchical clustering, to group similar songs based on their lyrical similarities. Each cluster represents a niche of songs that share common themes or emotional tones. For personalized recommendations, the system analyzes the user's listening history to identify their preference patterns and suggests new songs from the clusters that align with their past interests.

Additionally, advanced implementations may incorporate sentiment analysis to further enhance recommendation accuracy. By assessing the emotional tone of the lyrics—whether they are happy, sad, angry, or relaxing—the system can tailor recommendations to fit not only the thematic preferences of the user but also their current mood or emotional state.

This implementation showcases a significant advancement in music recommendation systems by moving beyond traditional metadata like genre or artist and tapping into the rich, expressive layers of music offered by lyrics. This approach not only enhances the user experience by providing more nuanced recommendations but also demonstrates the potential of combining NLP with machine learning to unlock deeper insights in multimedia content.

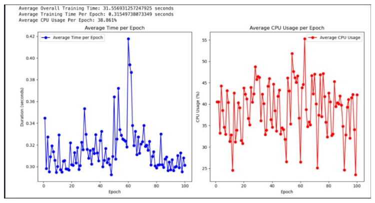
7 Experimental Setup

In our project on music recommendation systems utilizing PyTorch and TensorFlow2.x, the experimental setup was crafted to ensure precise and reproducible comparisons across both frameworks. We used the Million Song Dataset, the FMA dataset, and the Last.fm Dataset to assess our hybrid recommendation models, which integrate collaborative and content-filtering techniques. To provide a consistent basis for comparison, each framework was set up with identical model architectures and hyperparameters, including batch sizes, number of epochs, and learning rates.

For the hardware setup, we utilized a Mac M1 8-core CPU and leveraged Google Colab's Tesla T4 GPU to conduct our experiments. This allowed us to evaluate the performance differences between PyTorch and TensorFlow2.x under varying computational loads. We specifically tested the models with different numbers of data points—5,000, 20,000, and 50,000—to understand how each framework scales with increasing data volumes and complexity.

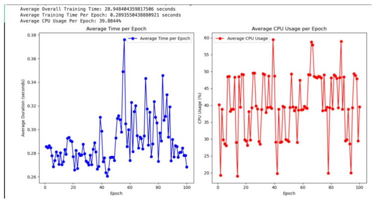
The rigorous data preprocessing steps ensured that the input data were standardized, and both frameworks operated under equivalent experimental conditions. This careful setup enabled us to isolate the frameworks' performance capabilities from other variables, providing a clear and detailed analysis of their strengths and limitations within the context of music recommendation.

8 Results



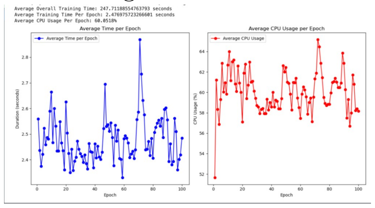
```
Average Overall Training Time: 31.556931257247925 seconds
Average Training Time Per Epoch: 0.31549738073349 seconds
Average CPU Usage Per Epoch: 38.861%
```

Figure 1: Model on TensorFlow - 5000 songs



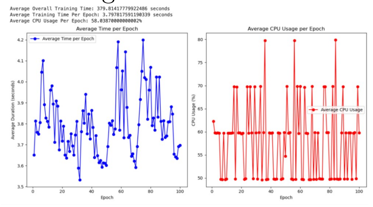
```
Average Overall Training Time: 28.948404359817506 seconds
Average Training Time Per Epoch: 0.2893550438880921 seconds
Average CPU Usage Per Epoch: 39.8844%
```

Figure 2: Model on Pytorch - 5000 songs



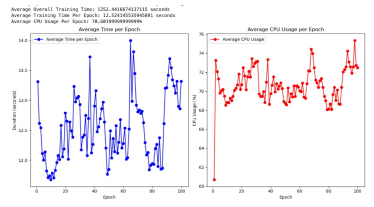
```
Average Overall Training Time: 247.71188554763793 seconds
Average Training Time Per Epoch: 2.476975723266601 seconds
Average CPU Usage Per Epoch: 60.0518%
```

Figure 3: Model on TensorFlow - 20,000 songs



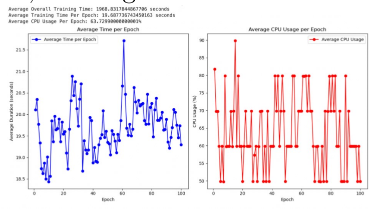
```
Average Overall Training Time: 379.81417779922486 seconds
Average Training Time Per Epoch: 3.797817591190339 seconds
Average CPU Usage Per Epoch: 58.03870000000002%
```

Figure 4: Model on Pytorch - 20,000 songs



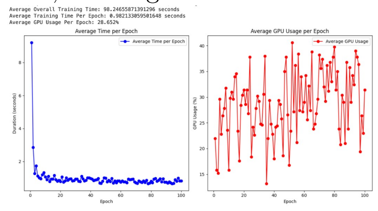
```
Average Overall Training Time: 1252.4416674137115 seconds
Average Training Time Per Epoch: 12.524145535945891 seconds
Average CPU Usage Per Epoch: 70.60199999999999%
```

Figure 5: Model on TensorFlow - 50,000 songs



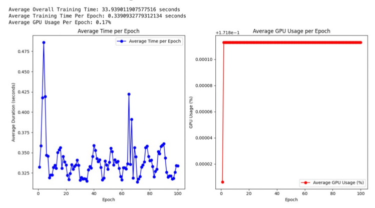
```
Average Overall Training Time: 1968.8317844867706 seconds
Average Training Time Per Epoch: 19.68773674345016З seconds
Average CPU Usage Per Epoch: 63.72990000000001%
```

Figure 6: Model on Pytorch - 50,000 songs



```
Average Overall Training Time: 98.24655871391296 seconds
Average Training Time Per Epoch: 0.982133059501648 seconds
Average GPU Usage Per Epoch: 28.652%
```

Figure 7: Model on TensorFlow - 50,000 songs



```
Average Overall Training Time: 33.939011907577516 seconds
Average Training Time Per Epoch: 0.3390932779312134 seconds
Average GPU Usage Per Epoch: 0.17%
```

Figure 8: Model on Pytorch - 50,000 songs

Above results indicate that for smaller datasets like 5000 songs, we see that PyTorch performs better than Tensorflow while resource consumption remains almost same. For 5000 songs, PyTorch takes 28.94 seconds as the overall training time and 0.28 seconds for training per epoch, while Tensorflow takes 31.55 seconds as the overall training time and 0.31 seconds as training time per epoch.

For larger datasets like 20000 songs, tensorflow takes 247 seconds for overall training time and PyTorch takes 379.81 seconds and for 50000 songs, Tensorflow takes 1252 seconds and PyTorch takes 1968 seconds. This shows that for smaller datasets, Pytorch performs slightly better than Tensorflow while for larger datasets, Tensorflow works better than PyTorch in terms of overall training time and training time per epoch.

9 Conclusion

Our experimentation has yielded insightful conclusions regarding the comparative strengths of PyTorch and TensorFlow in various computational contexts. PyTorch, with its pythonic and intuitive coding style, excels primarily in research and development environments. This framework is particularly favored for its flexibility, allowing researchers and developers to easily experiment with novel ideas and algorithms. Its straightforward structure not only simplifies debugging but also makes PyTorch highly accessible for small-scale projects and educational purposes, where understanding and tweaking the inner workings of models is crucial.

On the other hand, TensorFlow proves to be the superior choice for large-scale deployments. Its robust suite of tools and an extensive ecosystem are well-suited for developing and managing complex neural network architectures. TensorFlow's strength lies in its scalable nature and advanced features such as distributed training capabilities, which are essential for processing and analyzing vast datasets efficiently. This makes TensorFlow particularly valuable in production environments where performance, reliability, and scalability are critical.

Overall, both frameworks offer unique advantages tailored to specific needs and environments—PyTorch for flexibility and ease of use in research and small projects, and TensorFlow for robustness and scalability in large-scale industrial applications.

10 Future Works

To enhance the current project, several key improvements are planned for the near future:

Firstly, there will be a significant expansion of the dataset to include 50,000 songs, a move aimed at providing a broader spectrum of user preferences and improving the granularity of the music recommendation system. By tapping into a larger array of songs, the system is expected to capture more nuanced tastes and listening patterns, leading to a more personalized user experience.

The project also intends to refine the underlying recommendation engine by restructuring its architecture. This will involve incorporating collaborative filtering techniques that not only rely on individual listening histories but also draw from collective user data. Such methods are particularly adept at identifying and predicting user preferences by examining similarities among various user profiles. The metadata associated with the songs, such as genre, release date, and artist popularity, will be utilized more effectively to enhance recommendation accuracy.

In addition to improving the current system, alternative techniques for music recommendation will be explored. One such avenue is the use of sophisticated deep learning models that can potentially uncover complex patterns in music preference that simpler models may overlook. The adaptability and predictive power of deep learning models offer promising enhancements to recommendation systems.

Finally, to ensure the reliability and adaptability of the system across different platforms, comprehensive testing will be conducted on multiple devices. This rigorous testing protocol will help in identifying any device-specific issues and ensure a seamless and robust user experience regardless of the hardware used. The aim is to deliver a universally accessible system that provides high-quality music recommendations on any device.

These planned enhancements are designed to make the music recommendation service more intuitive, accurate, and enjoyable for users across various interfaces.

References

[1] K. Choi, G. Fazekas, M. Sandler, and K. Cho, "Convolutional recurrent neural networks for music classification," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2017, pp. 2392–2396.

[2] S. Oramas, F. Barbieri, O. Nieto Caballero, and X. Serra, "Multimodal deep learning for music genre classification," *Transactions of the International Society for Music Information Retrieval. 2018; 1 (1): 4-21.*, 2018.

[3] O. Celma, *Music Recommendation and Discovery: The Long Tail, Long Fail, and Long Play in the Digital Music Space*, 01 2010.