Meta Recommendation System

A. Chaitanya
IIT Gandhinagar
, Gandhinagar , Gujarat , India
23110052@iitgn.ac.in

B. Keerthan
IIT Gandhinagar
, Gandhinagar , Gujarat , India
23110068@iitgn.ac.in

G. Avinash
IIT Gandhinagar
, Gandhinagar , Gujarat , India
23110123@iitgn.ac.in

K. Dinesh Siddhartha
IIT Gandhinagar
, Gandhinagar , Gujarat , India
23110168@iitgn.ac.in

P. Praneeth
IIT Gandhinagar
, Gandhinagar , Gujarat , India
23110226@iitgn.ac.in

1 ABSTRACT

This paper introduces a novel Meta Recommendation System that automatically selects the most suitable recommendation algorithm based on dataset characteristics. By leveraging embeddings generated through CodeBERT, we encode both recommendation algorithms and dataset features. A deep neural network fine-tunes these embeddings to capture semantic relationships, with a contrastive loss function optimizing the matching process. Our approach minimizes manual intervention, enhances algorithm selection efficiency, and improves overall recommendation system performance, offering a more adaptive and scalable solution to recommendation tasks.

2 INTRODUCTION

In the vast and dynamic landscape of artificial intelligence, recommendation systems have become an essential component of personalized digital experiences — from e-commerce and entertainment to healthcare and education. Despite the abundance of specialized models, there exists no universal solution that determines which recommendation approach is best suited for a given problem or dataset. This absence of a guiding framework often forces developers to rely on trial-and-error experimentation, manually testing different algorithms to identify the optimal one. This is both inefficient and impractical, especially as datasets grow in complexity and diversity.

To address this critical gap, we propose a ***Meta Recommendation System***— a novel framework that aims to perform ***reverse recommendation***: instead of recommending products or content to users, it recommends the most suitable recommendation algorithm for a given use case. This meta-level approach has far-reaching applications. In scenarios where the nature of the data — whether it be textual, visual, or behavioral — significantly affects model performance, an automated way of choosing the right recommender can save time, improve accuracy, and enhance user satisfaction.

Traditionally, recommendation systems have relied on three primary methodologies. ***Collaborative filtering*** focuses on leveraging historical user-item interactions such as ratings, clicks, or purchases to identify patterns and make predictions. This method often employs neighborhood-based approaches or matrix factorization techniques like Singular Value Decomposition (SVD) and Alternating Least Squares (ALS). In contrast, ***content-based filtering*** uses the attributes of items — such as genres, tags, or textual descriptions — to build a profile of user preferences and suggest items with similar characteristics. This approach typically involves feature extraction and similarity computations, like cosine similarity. Recognizing the limitations of each method when used in isolation, ***hybrid recommendation systems*** combine elements of both collaborative and content-based filtering to achieve more balanced and robust performance, addressing issues such as the cold-start problem and improving recommendation diversity.

However, choosing the best approach among these — or their variations — depends heavily on the dataset's structure, modality, sparsity, and domain-specific requirements. Our Meta Recommendation System is designed to automate this decision-making process. By analyzing both the dataset features and the model characteristics through a blend of semantic embeddings and engineered attributes, the system is capable of suggesting the most appropriate recommendation strategy. This work represents a step toward making recommendation pipelines more intelligent, adaptive, and data-aware.

3 PROPOSED SYSTEM

3.1 Input-Output Based Categorization

We categorize the recommendation use-cases into four modalities:

- **Image-to-Image (I2I)**
- **Text-to-Image (T2I)**
- **Image-to-Text (I2T)**
- **Text-to-Text (T2T)**

3.2 Methodology

(Currently, we demonstrate only the **Text-to-Text (T2T)** modality—due to the availability of open-source code, availability of datasets, and ease of understanding. Our architecture, however, is designed to be extensible to all the above input-output formats. With appropriate embeddings and modality-specific processing, the same core recommendation logic can be applied across various image and text combinations.)

The proposed system leverages both recommendation system descriptions and dataset metadata to generate a unified embedding space suitable for recommendation tasks. The complete pipeline is structured as follows:

3.3 Input Embedding Generation

Recommendation Systems Embedding.

- The textual descriptions or codes of different recommendation systems are passed through a pre-trained **CodeBERT** model.
- CodeBERT converts the input into a dense **768-dimensional** embedding vector.
- This embedding captures the semantic meaning of the recommendation system.

Dataset Embedding.

- The datasets corresponding to the recommendation systems undergo manual feature engineering.
- Important metadata and structured features are manually extracted and encoded into a **50-dimensional** embedding vector.
- This 50D vector represents the characteristics of the dataset associated with each recommendation system.
- The full description and details of the features can be found here: **DatasetFeatures Text File**
- These features were thought to be used after going through the dataset2vec model.

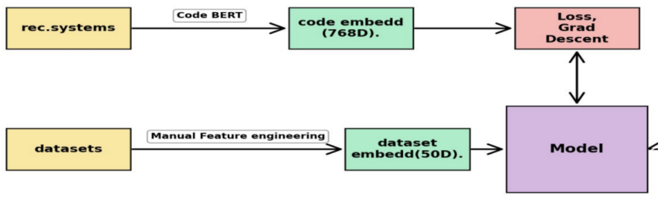


Figure 1: Methodology

3.4 Training Model

The goal of the model is to align the 50-dimensional dataset embeddings to the same semantic space as the 768-dimensional recommendation system embeddings.

- The dataset embeddings (50D) are passed through a deep neural network with **residual connections**, **layer normalization**, **ReLU activations**, and **dropout regularization** to generate a final output embedding.

The network architecture can be summarized as:

- **Fully Connected Layer (fc1)** → Output Size: 256 → LayerNorm (ln1) → ReLU + Dropout
- **Fully Connected Layer (fc2)** → Output Size: 256 → LayerNorm (ln2) → ReLU + Dropout
- **Residual Connection**: Skip connection adds the output of fc1 to the output of fc2.
- **Fully Connected Layer (fc3)** → Output Size: 256 → LayerNorm (ln3) → Residual Add
- **Final Fully Connected Layer (fc_out)** → Output Size: 768 (to match BERT output)

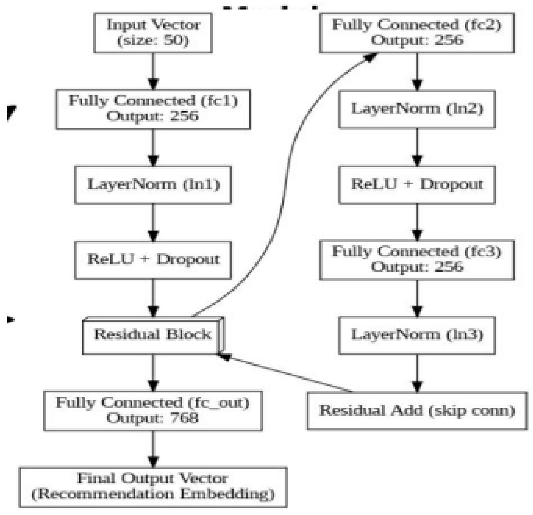

Input Vector (size: 50)
Fully Connected (fc1) Output: 256
LayerNorm (ln1)
ReLU + Dropout
Residual Block
Fully Connected (fc_out) Output: 768
Final Output Vector (Recommendation Embedding)
Fully Connected (fc2) Output: 256
LayerNorm (ln2)
ReLU + Dropout
Fully Connected (fc3) Output: 256
LayerNorm (ln3)
Residual Add (skip conn)

Figure 2: Neural Network Architecture within the Model.

3.5 Loss Function

To train the model, we use a contrastive loss function, which is designed to encourage similarity between correct pairs and dissimilarity between incorrect pairs. Specifically, the loss function operates as follows:

- **Cosine Similarity:** Cosine similarity is computed between the model's predictions and the target vectors.
- **Positive and Negative Pairs:** For each sample, the model is trained to:
  - Maximize similarity (i.e., reduce distance) for correct pairs (diagonal elements in the similarity matrix).
  - Minimize similarity for incorrect pairs (off-diagonal elements), where the model is encouraged to differentiate between the correct and most similar incorrect targets.
- **Margin:** A margin is introduced to prevent the similarity from being too close for incorrect pairs. This margin ensures that the model learns to distinguish between similar but incorrect vectors.

3.6 Training Procedure

The training process involves the following steps:

- **Input:** The dataset vectors and corresponding recommendation vector.
- **Forward Pass:** In each epoch, the input vectors are passed through the network to obtain predictions. The contrastive loss is computed based on the predicted and target vectors.
- **Backpropagation:** The loss is backpropagated through the network, and the optimizer updates the model's parameters.
- **Progress Tracking:** The training progress is tracked through the loss and average cosine similarity between predictions and targets over each epoch.

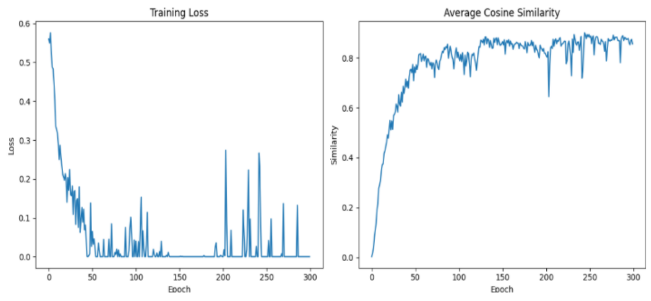


Figure 3: Overview of the training process including input, forward pass, backpropagation, and progress tracking.

3.7 Evaluation

After training, the model's performance is evaluated by comparing the predicted vectors against the recommendation vectors:

- **Cosine Similarity:** For each prediction, we compute the cosine similarity with all recommendation vectors and calculate the "similarity gap" between the correct recommendation and the most similar incorrect recommendation. This gap serves as a measure of how well the model can differentiate between correct and incorrect recommendations.

3.8 Hyperparameters

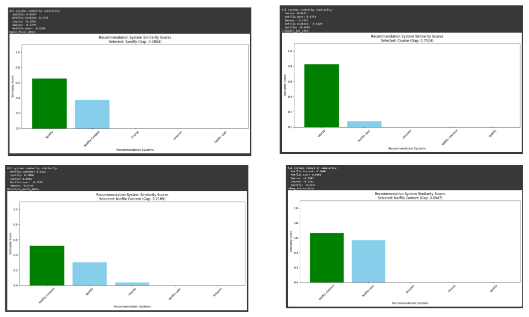
Key hyperparameters include:

- **Input/Output Dimensions:** The model is designed to handle input vectors of size 50 and output vectors of size 768.
- **Hidden Dimension:** The hidden layers have a dimensionality of 256.
- **Dropout Rate:** A dropout rate of 0.3 is used for regularization.
- **Epochs and Learning Rate:** The model is trained for 300 epochs with a learning rate of 0.001.
- **Optimizer:** The model is optimized using the **Adam optimizer**.
- **Loss Function:** The model uses the **Contrastive Loss Function**, designed to align the embeddings and differentiate between similar and dissimilar pairs.
- **Number of Learnable Parameters** = 342016
- **Inference time** = 342016
- **Training Time for 5 samples** = 5.39 seconds
- **Inference Time** = 1.1 milliseconds

3.9 Training Results

During training, the loss and similarity between the predictions and the targets are tracked. The model is expected to reduce the contrastive loss and improve similarity over time, as the optimization progresses.

This methodology effectively enables the model to learn meaningful embeddings that can be used for recommendation systems, with a focus on maximizing the alignment between dataset and recommendation vectors.



```
All systems ranked by similarity:
  Spotify: 0.6534
  Netflix Content: 0.3731
  Course: -0.2535
  Amazon: -0.3779
  Netflix user: -0.5288
Apple_Music_data:
```

```
All systems ranked by similarity:
  Course: 0.8267
  Netflix user: 0.0753
  Amazon: -0.2357
  Netflix Content: -0.4129
  Spotify: -0.5898
Linkedin_job_data:
```

```
All systems ranked by similarity:
  Netflix Content: 0.5213
  Spotify: 0.3024
  Course: 0.0361
  Netflix user: -0.3222
  Amazon: -0.6735
Movielens_movie_data:
```

```
All systems ranked by similarity:
  Netflix Content: 0.6640
  Netflix user: 0.5693
  Amazon: -0.1943
  Course: -0.2303
  Spotify: -0.3635
Udemy_Course_data:
```

Recommendation System Similarity Scores
Selected: Spotify (Gap: 0.2804)
Similarity Score
Spotify
Netflix Content
Course
Amazon
Netflix user
Recommendation Systems
Recommendation System Similarity Scores
Selected: Course (Gap: 0.7514)
Course
Netflix user
Amazon
Netflix Content
Spotify
Recommendation System Similarity Scores
Selected: Netflix Content (Gap: 0.2189)
Recommendation System Similarity Scores
Selected: Netflix Content (Gap: 0.0947)

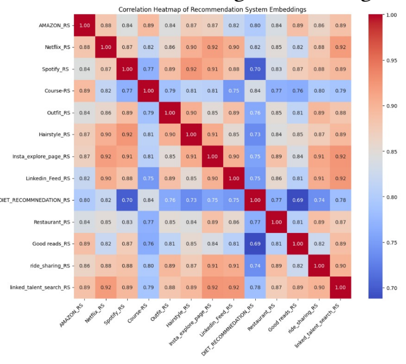
Figure 4: Visualization of similarity score distributions across different datasets. Each plot shows the top-ranked recommendation systems for a specific domain, with the selected system marked in green.

Image	Dataset	Recommended Content
Image 1	Apple Music Data	Spotify Recommendations
Image 2	LinkedIn Job Data	Course Recommendations
Image 3	MovieLens Movie Data	Netflix Content
Image 4	Udemy Course Data	Netflix Content

Table 1: Datasets and their Corresponding Recommended Content

4 WHY THE MODEL WORKS

The effectiveness of the proposed meta-recommendation system lies in its ability to intelligently map datasets to suitable recommendation algorithms through a unified and expressive architecture. By embedding both dataset features and algorithm representations into a joint vector space, the model enables meaningful interactions and similarity-based reasoning across previously unseen datasets and algorithms. CodeBERT encodes the source code of recommendation algorithms, capturing deep architectural patterns and operational logic beyond superficial naming conventions. On the dataset side, informative features—such as sparsity, rating scale, feedback type, and interaction density—are manually engineered to provide a rich and discriminative characterization of each dataset's structure and challenges. These embeddings are processed through a deep residual neural network, which enhances model capacity while preserving training stability via skip connections, thus enabling the capture of non-linear, high-dimensional relationships. Crucially, the model is trained in an end-to-end supervised manner, optimizing for the alignment between dataset embeddings and the embeddings of algorithms that perform best on them. This approach ensures that the recommendations are not only interpretable and semantically grounded but also directly optimized for performance, leading to a robust, generalizable, and task-aware recommendation engine. The contrastive loss improves the model's performance by maximizing similarity between dataset embeddings and the best-performing algorithm embeddings, while minimizing similarity with non-optimal ones. It operates by maximizing similarity for correct pairs (diagonal elements) and minimizing it for incorrect pairs (off-diagonal elements). A margin is introduced to ensure better separation between correct and incorrect matches, reinforcing accurate algorithm-dataset alignments.



The correlation heatmap (shown in above Figure) further supports the effectiveness of CodeBERT in encoding recommendation system embeddings. High intra-domain correlations and meaningful inter-domain relationships across various recommendation system datasets indicate that CodeBERT captures deep and semantically relevant patterns rather than surface-level noise, thereby validating the quality of the learned embeddings.

5 DATASETS AND LIBRARIES

We utilized publicly available datasets for training. The Datasets used are in the folder attached below.

5.1 Datasets

(1) **Spotify Music Dataset**: Link
(2) **Netflix Movie Dataset**: Link
(3) **Amazon Product Dataset**: Link

5.2 Libraries

(1) **Torch** – PyTorch for deep learning and neural network model implementation
(2) **Transformers** – Hugging Face library for CodeBERT embeddings generation
(3) **Pandas and Numpy** – Data manipulation, dataset processing and Numerical computing, array operations
(4) **Scikit-learn** – Feature engineering, normalization, and evaluation metrics
(5) **Matplotlib** – Visualization of results

6 COMPUTING RESOURCES

(1) **GPU:** All training and inference were performed using Google Colab's T4 GPU.
(2) **RAM:** Utilized Colab's provided RAM (typically a maximum of 15GB).
(3) **Storage:** Datasets, model checkpoints, and outputs were stored in Google Drive linked to Colab, with a minimum of 15GB of available space.
(4) **CPU:** Google Colab's hosted runtime was used for preprocessing and parallel data loading.
(5) **Cloud Resources:** No local hardware was used. All experiments were conducted fully on Google Colab.

7 TIMELINE FOLLOWED

- **Phase 1: Background Research** Reviewed existing recommendation systems—analyzed their inputs, outputs, recommendation types, methods, features used, and evaluation metrics.
- **Phase 2: Problem Framing** Structured findings and identified the need for a universal evaluation metric to compare different systems effectively.
- **Phase 3: Feature Extraction** Focused on text-to-text recommendation systems. Extracted dataset features and used CodeBERT to generate vector embeddings of code.
- **Phase 4: Model Design** Designed the model architecture using combined dataset and code vectors. Discussed suitable loss functions for effective recommendation learning.
- **Phase 5: Model Development** Built and refined the prototype model. Began initial evaluations and improved recommendation quality through feature alignment.
- **Phase 6: Finalization** Prepared the final report and presentation summarizing key insights, model design, and future directions.

8 EXTRA FINDINGS

8.1 Motivation

Our goal was to identify an alternative similarity function that closely approximates **cosine similarity** across various recommendation system datasets, while potentially offering improved properties for certain applications.

8.2 Experiment

We compared several alternative similarity metrics — **Sigmoid Dot Product**, **ReLU-Cosine**, **Tanh-Dot**, **Inverse Hyperbolic Tanh**, and **Norm-Contrast** — across three datasets: **Netflix**, **Spotify**, and **Amazon**. Evaluation criteria included **Top-K Overlap** with cosine similarity and **Spearman Rank Correlation** of the similarity scores.

max width=0.1

Metric	Netflix Overlap	Netflix Spearman	Spotify Overlap	Spotify Spearman	Amazon Overlap	Amazon Spearman
Sigmoid Dot	52/100	0.05	95/100	0.951	40/250	-0.152
ReLU-Cosine	100/100	1.00	100/100	1.00	100/250	1.00
Tanh-Dot	39/100	-0.12	97/100	0.803	40/250	-0.152
Inv Hypb Tanh	**100/100**	**0.86**	**100/100**	**0.97**	**240/250**	**0.94**
Norm-Contrast	100/100	0.76	99/100	0.975	46/250	0.18

Table 2: Comparison of alternative similarity metrics across datasets wrt recommendations obtained from cosine similarity.

8.3 Key Finding

The **Inverse Hyperbolic Tanh (IHT)** similarity consistently demonstrated the highest overlap and strongest correlation with cosine similarity across all datasets.

It is defined as:

$$\text{IHT}(x, y) = 1 - \tanh\left(\arccos\left(\text{cosine\_sim}(x, y)\right)\right)$$

8.4 Closure

The **Inverse Hyperbolic Tanh Similarity (IHT)** is a robust and reliable proxy for cosine similarity, making it a promising candidate for use in recommendation systems across diverse datasets.

9 CONCLUSION

This meta-recommendation system provides a scalable, principled, and data-driven framework for algorithm selection in recommendation tasks. By leveraging advanced code embeddings, informative dataset features, and a robust neural architecture, the system learns to accurately predict the most suitable algorithm for any given dataset, enabling automated and intelligent model selection in real-world applications.

10 USEFUL LINKS

Link to PPT: Link
Link to documentations done for 13 recommendation systems: Link
Schema Design: Link
Link to Datasets and all Codes used as data: Link
Link to Github containing all Codes: Link

REFERENCES

(1) https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=8862048
(2) https://www.mdpi.com/2624-6511/8/2/40
(3) https://www.researchgate.net/publication/379074761_Enhancing_Book_Recommendations_on_GoodReads_A_Data_Mining_Approach_Based_Random_Forest_Classification
(4) https://www.researchgate.net/publication/376517685_Diet_Recommendation_System_Using_Machine_Learning
(5) https://rhydhamgupta.medium.com/building-a-recommender-system-from-scratch-1-dd140492afa6
(6) https://pyimagesearch.com/2023/10/30/spotify-music-recommendation-systems/
(7) https://stratoflow.com/spotify-recommendation-algorithm/
(8) https://www.youtube.com/watch?v=gaZKjAKfe0s
(9) https://pyimagesearch.com/2023/07/03/netflix-movies-and-series-recommendation-systems/
(10) https://dl.acm.org/doi/pdf/10.1145/3543873.3587675
(11) https://www.ijnrd.org/papers/IJNRD2304674.pdf
(12) https://ai.meta.com/blog/powered-by-ai-instagrams-explore-recommender-system/
(13) 1.https://pyimagesearch.com/2023/08/07/linkedin-jobs-recommendation-systems/
(14) https://pyimagesearch.com/2023/08/07/linkedin-jobs-recommendation-systems/