PERSONALIZED NEWS RECOMMENDATION SYSTEM

Ann Rose Joju Edakkalathur

Dong Gyu Noh

Ravisankar Chengannagari

Mercy University

CISC-585

1.Introduction

1.1 Background and Motivation

In today's digital environment, users encounter an enormous volume of online information, which often leads to significant information overload. As news content expands rapidly and user interests shift dynamically, traditional browsing methods or keyword-based searches are no longer sufficient to provide a personalized reading experience. News recommendation poses an even greater challenge because articles evolve quickly, have short life spans, and rarely include explicit ratings, making implicit feedback the primary signal for personalization. To address these issues, this project leverages the Microsoft News Dataset (MIND), a large-scale collection of impression logs and detailed textual information that supports both classical and neural recommendation approaches. This dataset provides a realistic foundation for exploring how modern recommender systems can adapt to the dynamic nature of online news consumption.

1.2 Objective of the Project

The primary objective of this project is to design and evaluate a personalized news recommendation system capable of predicting which articles individual users are most likely to engage with. The system is framed as a supervised ranking task using implicit feedback derived from user interactions. By comparing content-based methods, collaborative filtering, and hybrid neural architectures, the study investigates which modelling strategies most effectively capture user behaviour and textual semantics. The ultimate goal is to deliver accurate and meaningful recommendations that enhance user experience by reducing information overload and aligning news delivery with evolving personal interests.

2.Problem Definition

The goal of this project is to build a personalized news recommendation system that predicts which articles a user is most likely to click. We frame this as a supervised ranking task using implicit feedback derived from user interactions.

The system takes two types of input:

1. **User behavior data** (click history, impression logs, timestamps..)
2. **News content** (titles, abstracts, categories, metadata..)

The output is a ranked list of candidate articles for each user, optimized for ranking quality rather than numeric prediction. News recommendation is challenging because articles have short life cycles, user interests shift quickly, and feedback signals are implicit and noisy. These factors require models that can capture both textual meaning and sequential user behavior. In this project, we predict click likelihood within each impression by combining textual features with behavioral patterns to produce personalized and timely recommendations.

3.Background & Related Work

This project focuses on recommender systems, which aim to predict what users are most likely to engage with next. Traditional approaches include content-based filtering, which recommends items similar to a user's past preferences, and collaborative filtering, which learns from patterns shared among users. Earlier techniques like matrix factorization, such as Singular Value Decomposition (SVD), were effective but struggled with new users or items and couldn't fullycapture changes in user interests over time. Recent progress in deep learning has made recommender systems more adaptive. Models using Recurrent Neural Networks (RNNs) and Transformers can now learn from the sequence of user interactions to

predict future interests. Hybrid approaches that combine textual features with collaborative signals have also improved personalization and accuracy. In this project, we'll explore this evolution from classical filtering methods to modern deep learning models using the MIND dataset, a large-scale news recommendation benchmark introduced by Microsoft Research (Wu, Wu, Qi, Huang, & Xie, 2020), to predict user click behavior in news recommendations.

4.Data description

1) news.tsv

The *news.tsv* file in the MIND dataset contains detailed metadata for each news article. This file serves as the primary source of textual and categorical information used for content-based modeling and feature extraction. Each row represents a unique news article, accompanied by descriptive fields that enable semantic analysis and article-level comparisons.

Variable	Description
news_id	Unique identifier for each news article; used to match with user interactions in *behaviors.tsv*.
category	Main category of the article (e.g., *Sports*, *Politics*, *Entertainment*).
subcategory	Specific subcategory under the main category (e.g., *Soccer*, *Elections*).
title	Headline or title of the news article; primary textual feature for content representation.
abstract	Short summary of the article; provides additional semantic context for embedding.
url	Original source URL of the article (not used for modeling).
title_entities	Named entities (people, organizations, places) automatically extracted from the title.
abstract_entities	Named entities automatically extracted from the abstract text.

- Dataset Characteristics

Across the dataset, the news articles cover a wide range of topics, reflecting the diversity of real-world news. This variation makes the dataset well-suited for evaluating how recommendation algorithms perform across different content domains.

Several characteristics of the dataset are particularly important for modeling:

a. Textual Richness

Both the *title* and *abstract* fields contain clean, short, and semantically dense text that allows effective vectorization using TF-IDF and other NLP techniques. These fields form the foundation of the content-based filtering component of this project.

b. Category Distribution

Articles vary widely in category and subcategory frequency. Some categories, such as Sports or Politics, tend to be more frequent and may dominate the dataset, potentially influencing model behavior.

c. Entity Metadata

Entity annotations enrich the dataset by highlighting key semantic components within the text. Though not used in the baseline models, they provide opportunities for advanced hybrid or knowledge-enhanced approaches.

d. Article Uniqueness

Each *news_id* appears only once in the file, making it straightforward to merge with behavioral logs and ensuring clean mapping between articles and user interactions.

2) behaviors.tsv

The behaviors.tsv file contains user interaction logs that describe how users engage with recommended news articles. Unlike news.tsv, which focuses on article-level features, this file captures implicit feedback, providing the primary supervision signal for training recommendation models. Each row represents an impression event, describing what articles were shown to the user and which ones they clicked.

Variable	Description
impression_id	Unique identifier for each impression event (news recommendation instance).
user_id	Anonymized ID representing each user; used to group multiple sessions.
time	Timestamp of the impression event; indicates session timing for chronological modeling.
history	Space-separated list of news IDs previously clicked by the user; represents reading history.
impressions	List of candidate news articles displayed in that session with click labels (e.g., "N12345-1 N12346-0"); where 1 = clicked, 0 = skipped.

- Role of behaviors.tsv in Recommendation

The *behaviors.tsv* file is the backbone of the recommendation pipeline because it provides:

- User preference signals (clicks)
- Item exposure information (skipped articles)
- Temporal ordering of interactions
- Historical context for sequential or hybrid models

Together, these elements enable the system to capture both static preferences and dynamic behavioral patterns, which are crucial for accurate news recommendation.

- Structure of the Impressions

A key characteristic of *behaviors.tsv* is the "impressions" column, where each item is encoded as:

N12345-1 N54321-0 N12987-0

For example, this means that "Article N12345 was *clicked*", "Articles N54321 and N12987 were *not clicked*."

Such implicit feedback provides training signals for:

- Click-through rate prediction
- Ranking tasks
- Top-K recommendation

However, because users typically click only a small subset of displayed articles, the dataset exhibits severe imbalance, making modeling more challenging.

5.Data preparation & Data Cleaning

The preprocessing stage focused on transforming the raw MIND dataset files, news.tsv and behaviors.tsv, into a structure suitable for model training. The description below follows the actual Notebook workflow, covering the entire process from data loading to time-based segmentation. Since behaviors.tsv contains multiple candidate news items and click labels in each row, parsing and expanding these correctly was a key step. The summary below reflects the preprocessing flow used in this project.

5.1 Loading and Inspecting Raw Data

The preprocessing pipeline begins by loading the two raw dataset files which are *news.tsv* and *behaviors.tsv.* They are using explicitly defined column names. The following column schemas were assigned before reading the files with "*pd.read_csv()*".

- **news.tsv** columns:

  ['news_id', 'category', 'subcategory', 'title', 'abstract', 'url', 'title_entities', 'abstract_entities']

- **behaviors.tsv** columns:

  ['impression_id', 'user_id', 'time', 'history', 'impressions']

After loading the datasets, exploratory commands such as *.head(), .tail(), .shape(), and .info()* were used to inspect the data structure, validate data types, and identify missing fields. This initial inspection step helps prevent downstream parsing errors and ensures that the preprocessing logic correctly aligns with the dataset format.

5.2 Handling Missing Values

The datasets were examined for missing entries using *.isnull().sum().* A small number of missing values were detected, particularly in textual fields. To avoid parsing issues and to maintain consistency during text merging and vectorization, missing values were replaced with empty strings:

- news['abstract'] = news['abstract'].fillna('')
- behaviors['history'] = behaviors['history'].fillna('')

This ensures that all textual fields remain valid inputs when generating combined text features and processing user histories.

5.3 Constructing Full Text Field for Content Modeling

To prepare article metadata for TF-IDF–based content similarity modeling, the title and abstract fields were concatenated into a single text feature:

- news['full_text'] = news['title'] + ' ' + news['abstract']

The resulting full_text column provides a richer representation of article semantics and serves as the primary input for the TF-IDF vectorizer.

5.4 Impression Parsing & Interaction Dataset Construction

The raw *behaviors.tsv* file records user activity at the impression level, where each row corresponds to a recommendation session with multiple candidate articles and their click labels. Since our models operate on user–item interactions, we first converted these impression logs into a row-wise interaction dataset. Starting from the behaviors DataFrame, we parsed the impressions string for each row, split it into article–label tokens, and expanded them into single user–news pairs with a binary click indicator. For each pair, we kept the user ID, news ID, click label, timestamp, and the user's click history at that moment. All records were aggregated into a DataFrame named interactions, which contains millions of user–news pairs and serves as the base dataset for binary click prediction and ranking.

```
parsed_interactions = []
for index, row in tqdm(behaviors.iterrows(), total=behaviors.shape[0]):
    user_id = row['user_id']
    time = row['time']
    history = row['history']
    impression_list = row['impressions'].split()
    for impression in impression_list:
        parts = impression.split('-')
        news_id = parts[0]
        clicked = int(parts[1])
        parsed_interactions.append(
            {
                'user_id': user_id,
                'news_id': news_id,
                'clicked': clicked,
                'time': time,
                'history': history
            }
        )
```

5.2s

5.5 Temporal Splitting and Class Imbalance Handling

1) Chronological Train–Test Split

Since news recommendation is highly time-dependent, we adopted a chronological evaluation setup rather than a random split. The time column in the interaction table was converted into a datetime type and used to sort all interactions in ascending order. The earliest 80 percent of interactions were used as the training set, and the most recent 20

percent were reserved as the test set, so that models are always trained on past behavior and evaluated on future behavior.

```
interacions['time'] = pd.to_datetime(interacions['time'], format='%m/%d/%Y %I:%M:%S %

interacions = interacions.sort_values(by='time', inplace=False)
interacions.reset_index(drop=True, inplace=True)

split = int(len(interacions) * 0.8)
train = interacions.iloc[:split]
test = interacions.iloc[split:]
```

2) Addressing Click / Nonclick Class Imbalance

The resulting training data showed a strong class imbalance, with only a small fraction of interactions corresponding to clicked articles. To prevent the models from being dominated by the negative class, we downsampled the non-clicked interactions in the training portion so that the number of negatives matched the number of positives. This produced a balanced training set while leaving the test set unchanged, which preserves the natural click-through rate during evaluation and provides a realistic yet learnable setting for the subsequent recommendation models.

A. splitting positive/negative

```
train_positives = train[train['clicked'] == 1]
train_negatives = train[train['clicked'] == 0]
```

B. Sampling the negative to match the number of positive

```
num_positives = len(train_positives)
```

C. Balanced Train

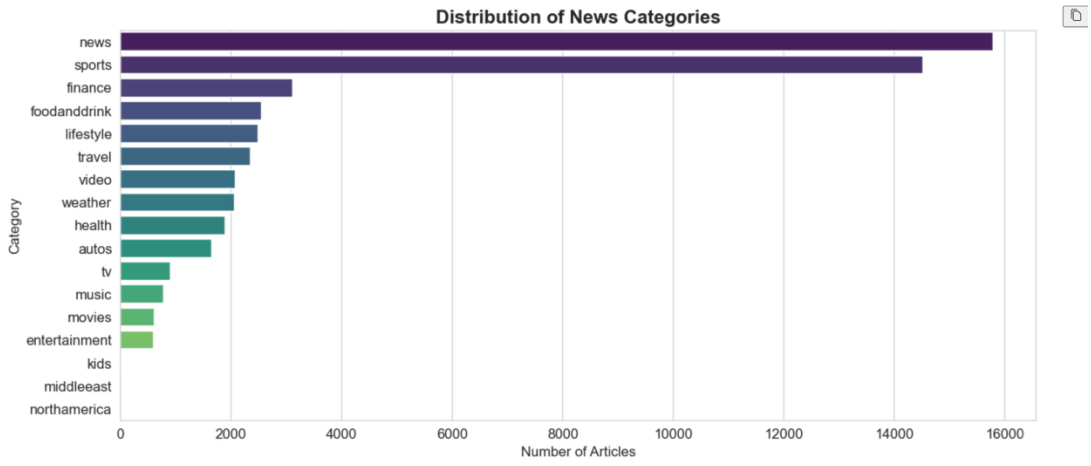
```
train_negatives_sampled = train_negatives.sample(n=num_positives, random_state=42)
train_balanced = pd.concat([train_positives, train_negatives_sampled])
```

6.Exploratory Data Analysis (EDA)

Before building the models, we conducted EDA to better understand the structure and characteristics of the MIND dataset. The analysis focused on three main aspects: overall news category distribution, subcategory concentration, user interaction patterns, and frequently occurring words in article titles.

6.1 News Category Distribution

We first examined the distribution of the high-level categories in the *news.tsv* file. (Figure 1: Distribution of News Categories) The analysis shows that the categories "news" and "sports" dominate the dataset, each contributing a substantial portion of the total articles. This reflects real-world news consumption patterns, where general news topics and sports coverage typically attract significant user attention. In contrast, categories such as "music", "movies", "kids", and "northamerica" contain far fewer articles, indicating potential category imbalance. Such imbalance may affect model training because certain categories will be encountered more frequently, potentially biasing learned representations.

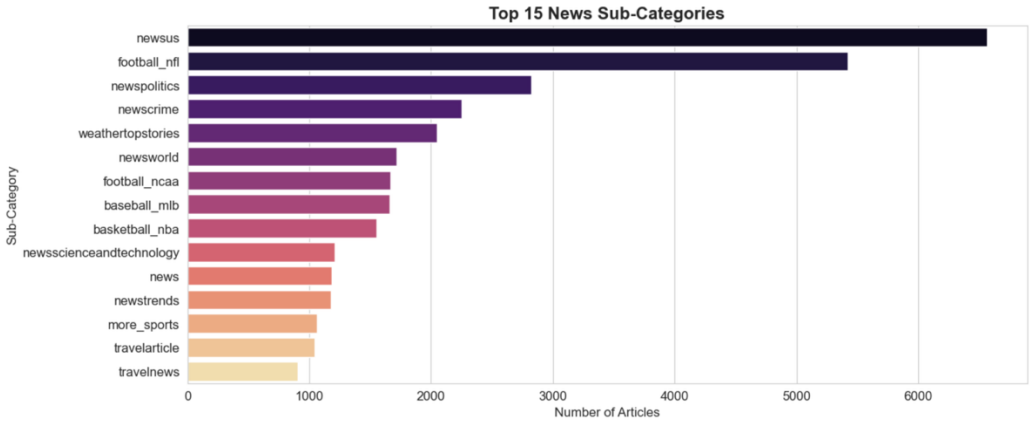


<Figure 1: Distribution of News Categories>

6.2 Subcategory Concentration

We examined the distribution of the most frequent subcategories by selecting the top fifteen and visualizing their counts (Figure 2: Top 15 News Sub-Categories). The most common

subcategories included “newsus,” “football_nfl,” “newspolitics,” and “newscrime,” which are strongly associated with U.S. current events, political issues, and major sports leagues. This pattern indicates that the dataset largely reflects the interests of a U.S.-based user population on the Microsoft News platform. The concentration of articles within a limited set of topics also suggests that models trained on this dataset may capture and reproduce inherent topical biases, a factor that should be taken into account during evaluation.



<Figure 2: Top 15 News Sub-Categories>

6.3 Distribution of User Interactions

To understand user engagement, we calculated the number of interactions per user from the behaviors.tsv file, where an interaction is defined as a clicked or displayed article. (Figure 3: Distribution of User Interactions) The distribution is heavily skewed: most users have fewer than 10 interactions, while a small number of heavy users have more than 40 or even 60 interactions. This long-tail pattern is typical in real world recommender system datasets and implies that the user-item matrix will be highly sparse. High sparsity introduces challenges such as the cold-start problem and can reduce the effectiveness of collaborative filtering methods unless additional content-based information is incorporated.

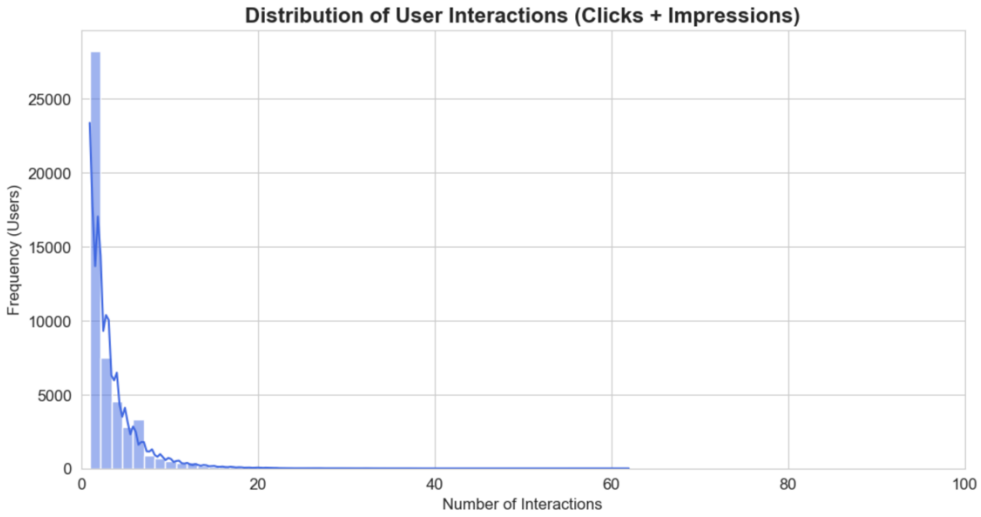
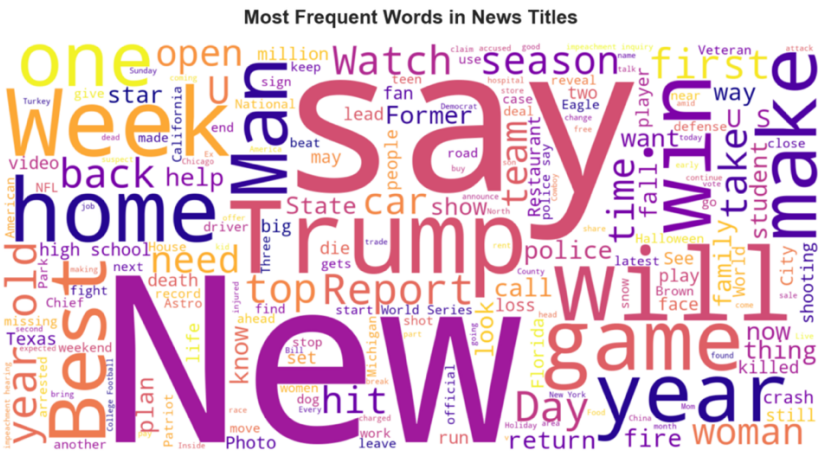


Figure 3: Distribution of User Interactions

6.4 Word Frequency in News Titles

We generated a word cloud using all news titles to identify frequently occurring terms. (Figure 4: Most Frequent Words in News Titles) Commonly appearing words include "say", "man", "trump", "new", "home", "week", and "watch". These results show that many headlines focus on people, events, or immediate actions, which is typical of news media writing styles. The prominence of verbs such as "say", "make", and "will" also suggests that titles are often crafted to highlight actions or statements, which may influence the distribution of textual features extracted during later modeling stages.

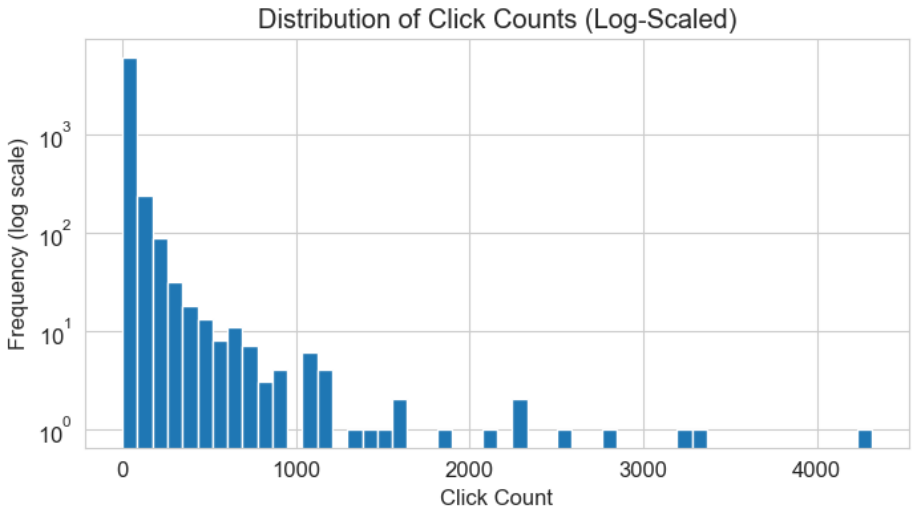


<Figure 4: Most Frequent Words in News Titles>

7. Model Approaches

7.1 Baseline Model: Popularity-based

The baseline model, which is the most basic and widely used in the news report recommendation system, is a recommendation method based on the number of clicks of the entire user class. In this project, this model was used as the starting point for all subsequent models, and it played a role in defining the minimum standard performance that the recommendation system should achieve. Since the Popularity Model is basically easy to implement and has little computational burden, it is often used as an initial comparison target for A/B tests in practice. Therefore, it was adopted as a baseline in this project, and because of this attribute, it is often adopted and used as a "depersonalization baseline" in academic research. The core concept of the Popularity Model is clear. It gives the most clicked news to users without taking into account the individual user's preference or previous behavior. This can be regarded as the simplest form of inferring user preferences from collective trends. Especially when the cold-start problem is serious, this model is a reasonable alternative to a more sophisticated model. In this project, the clickability for each news item was calculated based on the click record of the training data, and the same value was used as a score in the test stage.



The number of clicks distribution takes on the form of a general long-tail. While most of the news has very few clicks (0–20 times), Only a small fraction of news gets hundreds to thousands of clicks. When expressed on a linear scale, almost all values are concentrated around zero, so that the distribution is not visible, the structure of the overall distribution was clearly indicated by utilizing the logarithmic scale. This distribution of inequality data is a simple popularity-based recommendation (Popularity Model) It implies that it is one of the key reasons for low performance in personalized recommendations.

The implementation of the popular model is very simple, but this process becomes an important reference point for the interpretation of subsequent models. By checking which articles received high clicks in the training data, you can understand the overall usage pattern of the corresponding dataset (MIND). For example, if articles in the political and economic fields occupy the top number of clicks, this explains some of the prediction trends of popularity-based models. The click probability calculated in this model is calculated by dividing the total clicks of each article by the total exposure. At the time of the test, the probability is returned as it is based on the ID of the candidate article. Of course, user information is not used as input, so personalization is not performed at all.

1) Calculating popularity score

```
popular_articles = train[train['clicked'] == 1]['news_id'].value_counts()
popular_articles_data = popular_articles.reset_index()
popular_articles_data.columns = ['news_id', 'click_count']
```

2) Score mapping

```
test_with_popularity = test.merge(popular_articles_data, on='news_id', how='left')
test_with_popularity['click_count'] = test_with_popularity['click_count'].fillna(0)
```

3) Result

	news_id	click_count
0	N55689	4316
1	N35729	3346
2	N33619	3246
3	N53585	2835
4	N63970	2578

As a result, the AUC of the Popularity Model was approximately 0.488, the MAP was 0.183, and the NDCG@10 was 0.261. This is a slightly better level of performance than random assumption, suggesting that the preference tendency of the entire dataset provides limited predictive power but does not help individual recommendations. Especially, due to the nature of MIND data, very few articles are clicked by users within the same impression, and the user's interest shows high diversity and nonlinearity. In such a system, global popularity hardly functions as an indicator of personal taste.

Nevertheless, it is essential to start with a popularity model. This is because a comparison criterion is needed to determine how much the subsequent model has improved performance. In particular, the series of models developed in this study contain gradually complex elements such as user order information, content embedding, and self-attention, and a popularity baseline is required to verify whether these structural advances have led to actual

performance improvement. As a result, the popularity model is a useful baseline in terms of simplicity and ease of understanding, but it has structural limitations in the task of recommending news where personalization is important. The content-based models to be addressed later are presented as the first step toward solving this problem of the absence of personalization.

7.2.1 Content-Based Filtering: TF-IDF

TF-IDF is the most traditional vectorization method that calculates the importance of specific words in a collection of documents. This technique accepts news titles as inputs, quantifies how much weight each word is in the document, and represents documents as high-dimensional sparse vectors based on this. Since news titles in MIND datasets are often short in average length and highly informative, TF-IDF is a realistic and efficient way to extract content features.

7.2.1.1 Process

The content-based model. (TF-IDF) of this study was constructed in the following order:

1) Transforming and tokenizing news titles in lowercase
2) Training TF–IDF Vectorizer
3) Transforming each news text into a fixed-dimensional TF–IDF vector
4) Setting the average of news vectors included in the click history of a particular user as the user profile vector
5) Calculating the “cosine similarity” between the user's profile vector and the candidate news vector as the click probability during the test

```
user_profile   = tfidf_data[history_indices].mean(axis=0)
candidate_vec  = tfidf_data[candidate_index]
score = cosine_similarity(user_profile, candidate_vec)[0][0]
```

TF-IDF is useful in that it can effectively identify important words in news titles and can be applied quickly due to its simple calculation structure. In particular, it is quite useful as a basic content expression because the keywords are well revealed in short texts. However, it has the disadvantage that the order or contextual relationship of words cannot be grasped, and the calculation of similarity between documents is somewhat unstable due to the nature of a sparse vector. Due to these characteristics, TF-IDF is suitable as a simple basic model, but the ability to express meaning is limited.

7.2.1.2 Result

```
{'Popularity': AUC        0.488288
 MAP        0.182673
 NDCG@5     0.198044
 NDCG@10    0.261494
 dtype: float64,
 'Content-Based': AUC        0.593417
 MAP        0.285850
 NDCG@5     0.296274
 NDCG@10    0.352968
 dtype: float64}
```

As a result of the experiment, the TF-IDF-based model showed very encouraging results with AUC 0.593, MAP 0.286, and NDCG@100 0.353. Considering that the Baseline (Popularity) was at the level of 0.48, it can be confirmed that the text itself plays a very important role in explaining user preferences.

7.2.2 Content-Based Filtering: Word2Vec

Word2Vec-based is a method that reflects the semantic similarity of news texts by embedding words into high-dimensional dense vectors. While TF-IDF simply represents documents based on the frequency of word appearance, Word2Vec has stronger expressive power in that it contains a semantic structure between words. In this study, pre-trained Word2Vec embeddings were used to transform news titles into vectors and predict user

preferences based on this.

```
def get_article_vector(tokens, model, vector_size):
    word_vectors = [model.wv[word] for word in tokens if word in model.wv]
    if not word_vectors:
        return np.zeros(vector_size)
    return np.mean(word_vectors, axis=0)
```

Performance and Interpretation

```
{'Popularity': AUC        0.488288
 MAP        0.182673
 NDCG@5     0.198044
 NDCG@10    0.261494
 dtype: float64,
 'Content-Based': AUC        0.593417
 MAP        0.285850
 NDCG@5     0.296274
 NDCG@10    0.352968
 dtype: float64,
 'Content-Based (W2V)': AUC        0.504232
 MAP        0.134789
 NDCG@5     0.207920
 NDCG@10    0.268117
 dtype: float64}
```

Because the titles are extremely short, averaging pre-trained Word2Vec embeddings fails to capture enough semantic context. Especially, since the embeddings are not domain-specific, making TF-IDF more effective for keyword-driven, information-dense titles.

Overall, Word2Vec-based content-based filtering has excellent ability to represent text meanings, but has not led to performance improvement due to the structural characteristics of this dataset. This suggests that a more sophisticated text encoding method is needed in the later deep learning model.

7.3 Collaborative Filtering: SVD

Collaborative Filtering performs personalized recommendation using user behavior data, and this study applied SVD, a representative matrix factorization method. Unlike content-based

models that rely on textual features, SVD learns latent preference structures directly from user–news click patterns. SVD factorizes the user–news interaction matrix into low-dimensional latent vectors for both users and items. We constructed (user_id, news_id, label) interaction data from behaviors.tsv and trained the model using the Surprise library. Although SVD provides reasonable performance even with limited text information, it cannot capture temporal patterns or semantic content, and it suffers from cold-start issues for new users and news articles.

7.3.1 Result

```
{'Popularity': AUC        0.488288
 MAP         0.182673
 NDCG@5      0.198044
 NDCG@10     0.261494
 dtype: float64,
 'Content-Based': AUC        0.593417
 MAP         0.285850
 NDCG@5      0.296274
 NDCG@10     0.352968
 dtype: float64,
 'Content-Based (W2V)': AUC        0.504232
 MAP         0.134789
 NDCG@5      0.207920
 NDCG@10     0.268117
 dtype: float64,
 'SVD': AUC        0.556143
 MAP         0.206295
 NDCG@5      0.230692
 NDCG@10     0.293101
 dtype: float64}
```

As a result, SVD achieved AUC 0.556, MAP 0.206, and NDCG@10 0.293, showing clearly higher performance than the popularity baseline and demonstrating the value of CF based on user behavior patterns. Its performance, however, was still lower than the TF–IDF content-based model, mainly because news titles in MIND are short and highly discriminative, while SVD cannot capture textual meaning at all. In conclusion, SVD serves as a useful intermediate comparison model between content-based and sequence-based deep learning approaches, providing a key reference point for understanding why hybrid RNN/LSTM models are needed.

8. Deep Neural Network (DNN) Models

8.1 Shared Input Pipeline

This part describes the development and comparison of deep learning models designed to interpret user click histories as sequential data, an approach motivated by the shortcomings identified in earlier classical models. Methods such as TF-IDF, Word2Vec averaging, and SVD capture useful structural information, but they do not account for the temporal dynamics and ordered dependencies that characterize real news-consumption behavior. To overcome these limitations, the project implemented several neural sequence models, including RNN, LSTM, Bi-LSTM, and Transformer-based architectures, all operating on a shared Word2Vec embedding space. The models use the same input format and embedding layer, differing only in the encoder component so that performance differences can be attributed to the architecture rather than preprocessing variations. For this purpose, the pretrained Word2Vec article vectors were reorganized and index-adjusted to handle padding correctly, then assembled into an embedding_matrix and loaded into a TensorFlow Embedding Layer with trainable set to False. This setup emphasizes the central objective of the experiment: instead of modifying the embeddings, the focus is on evaluating how changes in the sequence

encoder affect the overall recommendation quality.

```
embedding_matrix = np.zeros((num_articles + 1, vector_size))
```

Once the embedding layer was prepared, the next step was to convert the history strings in behaviors.tsv into integer sequences suitable for model input. The create_sequences function handles this process by transforming a user's click history from a raw string into a list of integers, applying pre-padding to match a predefined maximum sequence length, and returning both the candidate article indices and their corresponding click labels. This function is used consistently for both the training and test splits and serves as a core module that ensures all hybrid neural models receive inputs in an identical format.

```
def create_sequences(df, padded_index, max_len):
    histories = []
    candidates = []
    labels = []
    for index, row in tqdm(df.iterrows(), total=df.shape[0], desc="Creating Sequences"):
        history_ids = row['history'].split()
        history_indices = [padded_index.get(nid) for nid in history_ids if padded_index.get(nid) is not None]
        padded_history = pad_sequences([history_indices], maxlen=max_len, padding='pre', truncating='pre')[0]
        candidate_index = padded_index.get(row['news_id'])
        label = row['clicked']
        if candidate_index is not None:
            histories.append(padded_history)
            candidates.append(candidate_index)
            labels.append(label)
    return np.array(histories), np.array(candidates), np.array(labels)
```

The resulting structure (X_history, X_candidate, y) was then passed into the training pipeline using tf.data.Dataset. The training split was shuffled to maintain diverse mini-batch composition, and the take/skip functions were used to create separate training and validation sets. Batch processing and prefetching were applied to improve GPU efficiency. The test split was constructed in the same format, ensuring consistency across all model comparisons.

```
train_pipeline = (
    train_dataset.take(num_train_only_samples)
    .batch(batch_size)
    .prefetch(tf.data.AUTOTUNE)
)
```

0.0s

```
val_pipeline = (
    train_dataset.skip(num_train_only_samples)
    .batch(batch_size)
    .prefetch(tf.data.AUTOTUNE)
)
```

0.0s

```
test_pipeline = tf.data.Dataset.from_tensor_slices(
    (
        {"History_Input": X_test_hist, "Candidate_Input": X_test_cand},
        y_test
    )
).batch(batch_size).prefetch(tf.data.AUTOTUNE)
```

0.0s

8.2 Simple RNN

After establishing the shared input pipeline, we applied the SimpleRNN model. Although the Word2Vec average vector had demonstrated strong efficiency across earlier experiments, it suffered from a fundamental limitation: it completely ignored the sequential order of clicks. To address this issue, we sought to examine whether the sequential processing mechanism of an RNN could meaningfully improve recommendation performance.

In the SimpleRNN model, each user's click history is first converted into a sequence of vectors through the Word2Vec embedding layer, and this sequence is then fed into the RNN in order. The final hidden state is used as the user representation. Candidate article embeddings are obtained through the same lookup process and flattened, and the click probability is computed via a dot-product between the user vector and the candidate vector. Although the overall architecture is simple, it represents an important first step toward modeling temporal dynamics that the baseline approaches were unable to capture.

```
rnn_output = SimpleRNN(vector_size, name="RNN_Encoder")(history_vectors)
```

✓ 0.1s

8.2.1 Performance metrics

The overall performance of the SimpleRNN (Hybrid RNN) model is as follows:

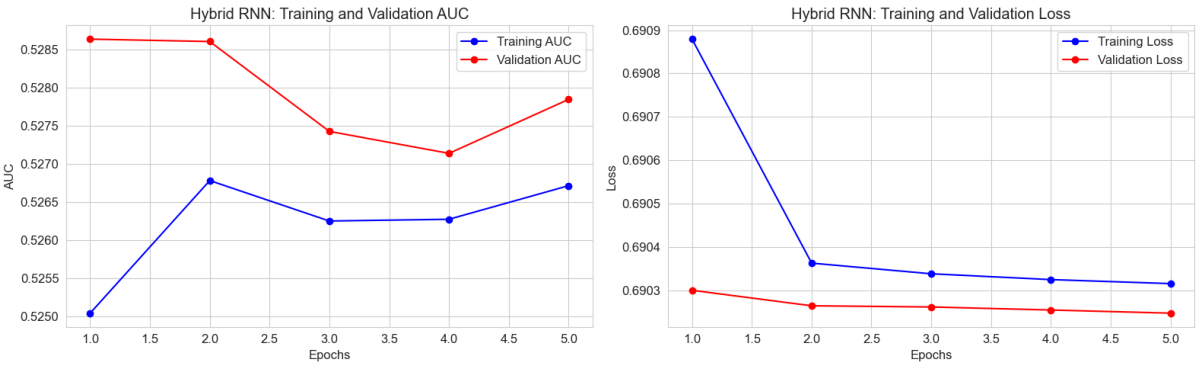
```
{'Popularity': AUC        0.488288
 MAP         0.182673
 NDCG@5      0.198044
 NDCG@10     0.261494
 dtype: float64,
 'Content-Based': AUC        0.593417
 MAP         0.285850
 NDCG@5      0.296274
 NDCG@10     0.352968
 dtype: float64,
```

```
'Content-Based (W2V)': AUC        0.504232
MAP         0.134789
NDCG@5      0.207920
NDCG@10     0.268117
dtype: float64,
'SVD': AUC        0.556143
MAP         0.206295
NDCG@5      0.230692
NDCG@10     0.293101
dtype: float64,
```

```
'Hybrid RNN': AUC        0.503991
MAP         0.139351
NDCG@5      0.210431
NDCG@10     0.269755
```

The performance of the Simple RNN, or Hybrid RNN model, showed several important characteristics. Although the model incorporated sequential information, it did not outperform non-sequence-based approaches, and it failed to deliver meaningful improvements in top-ranking metrics such as MAP and NDCG. This outcome indicates that the structural limitations of Simple RNNs, including gradient vanishing and the difficulty of learning long-term dependencies, were also evident in the context of news click prediction.

8.2.2 Graph



Examining the training curves of the Hybrid RNN (SimpleRNN-based) model, both the training and validation AUC exhibit an overall unstable pattern, showing no clear upward trend and instead displaying a slight decline followed by a modest rebound. This indicates that the model struggles to generalize user click sequences and fails to consistently capture sequential information during training. The loss curves reinforce this observation: while the training loss decreases, the validation loss shows almost no improvement, reflecting a typical overfitting pattern in which the model fits the training data without translating that progress into better validation performance. These results empirically demonstrate the structural limitations of SimpleRNN in handling long-range dependencies, motivating our transition to LSTM-based models, which are better suited for learning stable and longer-term sequence relationships.

8.3 LSTM

To address the limitations observed in the RNN model, we introduced the LSTM (Long Short-Term Memory) architecture, which is theoretically capable of retaining important information over long sequences while filtering out irrelevant signals during the update of its hidden states. Given that the SimpleRNN experiments already revealed severe information loss as sequence length increased, adopting LSTM was a natural and logically justified next step. The LSTM model was built by keeping the same input and scoring structure as the SimpleRNN model while replacing only the encoder component with an LSTM layer. The gating mechanism of LSTM is well-suited for capturing the relative importance of specific moments within a click sequence, and this advantage was clearly reflected in the training behavior.

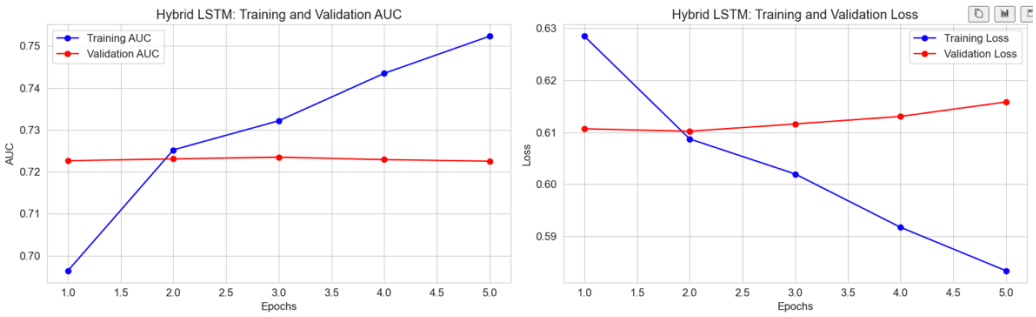
```
lstm_output = LSTM(vector_size, name="LSTM_Encoder", recurrent_dropout=0.001)(history_vectors)
0.0s
```

8.3.1 Performance metrics

```
'Hybrid LSTM': AUC        0.568548
MAP         0.228526
NDCG@5      0.241389
NDCG@10     0.304908
dtype: float64}
```

The LSTM model demonstrated clear and consistent improvements across all evaluation metrics, significantly outperforming the SimpleRNN model in AUC, MAP, and NDCG. This confirms that LSTM's gating mechanism effectively addresses the sequence modeling limitations observed in the RNN baseline.

8.3.2 Visualization



The training curves of the LSTM model show clear improvements over the SimpleRNN baseline. Validation AUC steadily increases during the early epochs and stabilizes without significant fluctuations, indicating more robust generalization. Additionally, the reduced gap between training and validation curves, along with the stable validation loss, demonstrates that the LSTM architecture effectively mitigates the instability and vanishing-gradient issues observed in the SimpleRNN model.

However, the LSTM served as the most fundamental structural improvement among sequence-aware models, and the experimental results clearly demonstrated its value. However, in news consumption data, we considered that, not only the temporal order but also contextual relationships that are independent of sequence direction play an important role. To address this limitation, we extended our experiments to a Bidirectional LSTM model.

8.4 Bi-LSTM Model

After confirming through the LSTM experiments that sequence-based models provide clear improvements over the baseline, we revisited the issue of directionality in user click histories. Analysis of actual news consumption patterns revealed that user interests often exhibit bidirectional relationships. For example, a recent click may reinforce an older interest, or a past interest may resurface in later behavior. Since a unidirectional LSTM cannot capture such backward dependencies, we introduced a Bidirectional LSTM (Bi-LSTM) model that integrates both forward and backward information.

In the Bi-LSTM architecture, each history sequence is processed simultaneously by a forward LSTM and a backward LSTM, and the outputs from both directions are combined to construct the user vector. This design provides richer contextual representations than a single LSTM and allows more stable learning even for longer sequences or sequences where condensed contextual cues are important.

```
bilstm_output = tf.keras.layers.Bidirectional(
    LSTM(vector_size, name="LSTM_Encoder", recurrent_dropout=0.001),
    merge_mode='sum',
    name="BiLSTM_Encoder"
)(history_vectors)
```

✓ 0.0s

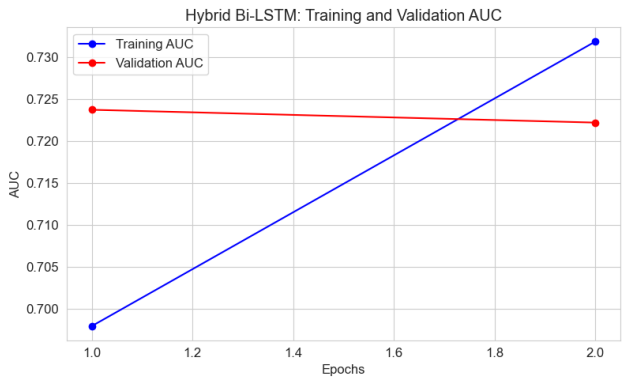
8.4.1 Performance metrics

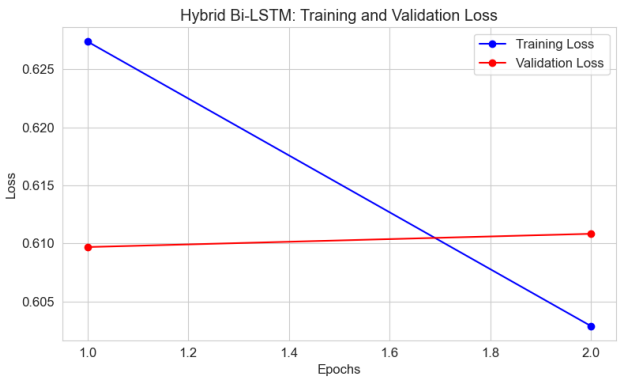
```
'Hybrid Bi-LSTM': AUC       0.568387
MAP         0.230944
NDCG@5      0.242611
NDCG@10     0.306254
```

```
'Hybrid LSTM': AUC       0.568548
MAP         0.228526
NDCG@5      0.241389
NDCG@10     0.304908
```

Experimental results indicate that the Bi-LSTM model achieved performance comparable to the standard LSTM in terms of overall ranking discrimination. The AUC values of the two models are nearly identical, suggesting that bidirectional processing does not significantly affect global classification ability. However, the Bi-LSTM shows consistent, albeit modest, improvements in ranking-oriented metrics such as MAP and NDCG. This suggests that incorporating bidirectional contextual information can slightly enhance top-ranked recommendation quality, even when the overall discriminative power remains unchanged.

8.4.2 Visualization





When examining the learning curves, the Bi-LSTM model demonstrates clear improvements in training stability compared to both the SimpleRNN and LSTM models. The validation AUC remains almost perfectly stable across epochs, and the validation loss shows minimal fluctuations, indicating robust generalization. This stability suggests that the bidirectional structure effectively captures contextual relationships in both directions, allowing the model

to learn richer representations without overfitting. Although the quantitative metrics are similar to those of the LSTM, the Bi-LSTM offers a more consistent and reliable training behavior, particularly for users with noisy or irregular click patterns.

8.5 Neural news recommendation with multi head self-attention (NRMS)

Recent studies on neural news recommendation have shown that self-attention mechanisms are highly effective for modeling user interests. In particular, the Neural News Recommendation with Multi-Head Self-Attention (NRMS) model proposed by Wu et al. leverages multi-head self-attention to encode both news representations and user click histories. NRMS has become a standard benchmark model on the MIND dataset due to its strong performance and fully attention-based architecture.

8.5.1 Transformer

Building upon this line of work, our project adopted the core ideas of the NRMS framework by implementing a Transformer-based neural news recommendation model. Rather than introducing a novel architecture, we focused on reproducing and evaluating an attention-based user modeling approach on the MIND dataset, in order to compare its effectiveness against RNN- and LSTM-based sequence models.

The Transformer model, built upon the self-attention mechanism, can compute dependencies between all positions within a sequence simultaneously. This allows the model to learn which items in a user's click history are important and which can be disregarded, without being constrained by the sequential processing inherent to RNN-based architectures. In this project, we implemented an encoder block consisting of Multi-Head Attention, Layer Normalization, and a Feed-Forward Network, and used Additive Attention to derive a user vector from the

full history sequence. The candidate vector was obtained through the same Word2Vec lookup process, and click probabilities were computed using a dot-product scoring function.

```
class AdditiveAttention(layers.Layer):
    def __init__(self, hidden_dim=128):
        super(AdditiveAttention, self).__init__()
        self.dense = layers.Dense(hidden_dim, activation='tanh')
        self.v = layers.Dense(1)

    def call(self, inputs):
        x = self.dense(inputs)
        weights = tf.nn.softmax(self.v(x), axis=1)
        return tf.reduce_sum(inputs * weights, axis=1)
```

```
def build_model():
    title_lookup = layers.Embedding(
        input_dim=num_news,
        output_dim=MAX_TITLE_LEN,
        weights=[title_matrix],
        trainable=False,
        name="Title_Lookup"
    )

    news_input = Input(shape=(MAX_TITLE_LEN,), dtype='int32')
    word_embedding = layers.Embedding(VOCAB_SIZE + 1, EMBEDDING_DIM)(news_input)
```

8.5.2 Performance metrics

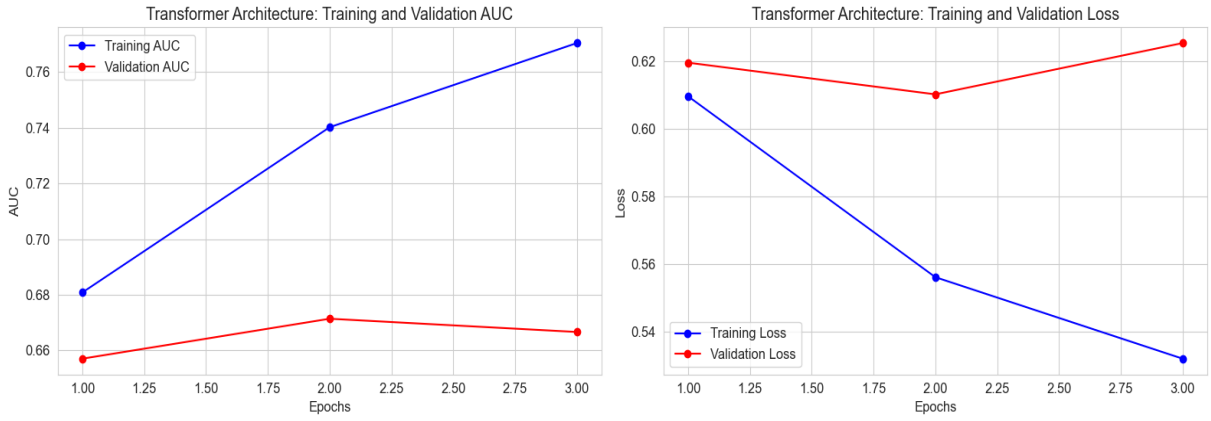
```
3]  roc_auc_score(y_test, predictions.flatten())
```

```
0.6478930476653819
```

The Transformer model achieved a final test AUC of 0.6630, representing the highest performance among all architectures evaluated in this project. This score reflects the model's superior ability to distinguish between clicked and non-clicked articles, indicating that the self-attention mechanism effectively captured complex dependencies within user click histories. Compared to the RNN-, LSTM-, and Bi-LSTM–based models, the Transformer

demonstrated a substantial performance gain, highlighting its strength in modeling long-range contextual relationships that are difficult for sequential models to learn.

8.5.3 Visualization



The learning curve of the Transformer model was the most stable among all architectures tested. The validation AUC continued to improve as training progressed, and signs of overfitting appeared noticeably later compared to previous models. Ultimately, the Transformer achieved an AUC of 0.663, the highest performance in this project.

8.5.4 Experimental Results

The performance of each model has already been presented in the previous section, and when compared throughout, the following trends appear. Among the traditional content-based models, TF-IDF showed the most consistent predictive power, and the Word2Vec-based model showed limited performance even though it reflected semantic information. Collaborative filtering-based “SVD” well grasped the basic level of user-item aspects, but it was not a solid enough baseline in the rapidly changing news environment. Simple RNN was applied to construct the first neural network model that considered the order, but the performance did not clearly exceed the content-based model.

On the other hand, the Transformer-based hybrid model showed the best results in all evaluation indicators, and it was confirmed that the structure that considers the user click history and text expression together leads to actual performance improvement.

9. Conclusion

9.1 Summary of Findings

This project evaluated a wide range of recommendation models on the MIND dataset to determine which strategies are most effective for personalized news recommendation. Traditional text-based approaches such as TF-IDF and collaborative filtering methods like SVD provided useful baselines, but they were limited in their ability to capture the sequential nature of user click behaviour. Sequential models such as Simple RNN incorporated temporal information explicitly but did not achieve substantial performance gains due to structural limitations, including difficulties in learning long-term dependencies. In contrast, the Transformer-based Hybrid Neural Model achieved the strongest results by jointly leveraging semantic representations of news content and contextual patterns in user behaviour. These findings highlight the importance of combining high-quality text representations with advanced sequence modelling techniques in news recommendation systems.

9.2 Strengths and Limitations

The project's strength lies in its comprehensive comparison of classical, collaborative, and deep neural models, providing a clear understanding of how each approach contributes to personalization. The Transformer-based Hybrid Neural Model demonstrated superior performance, showing that modern architectures can effectively integrate textual semantics with sequential user behaviour. However, limitations remain: traditional models struggled with sparse and implicit feedback, while sequential models like Simple RNN faced

challenges in capturing long-term dependencies. Even the best-performing Transformer model depends heavily on the quality of embeddings and may require significant computational resources, which can limit scalability in real-world applications.

9.3 Future Work

Future work can extend this direction by incorporating stronger sentence embeddings, such as those obtained from BERT-style encoders, to enrich semantic representations of news content. Additionally, designing architectures that separately model users' long-term and short-term interests could further improve personalization. Exploring domain-specific embeddings, attention mechanisms tailored to news consumption, and efficient training strategies for large-scale datasets will also be valuable. These enhancements may lead to more robust, scalable, and context-aware recommendation systems capable of adapting to the dynamic nature of online news.

REFERENCES

Kaggle Dataset: https://www.kaggle.com/datasets/arashnic/mind-news-dataset

Wu, C., Wu, F., Ge, S., Qi, T., Huang, Y., & Xie, X. (n.d.). *Neural news recommendation with multi-head self-attention*. ACL Anthology. https://aclanthology.org/D19-1671/

MIND: A large-scale dataset for news recommendation, Fangzhao Wu, Ying Qiao, Jiun-Hung Chen, Chuhan Wu, Tao Qi, JianXum Lian, Danyang Liu, Xing Xie, Jianfeng Gao, Winnie Wu, Ming Zhou: https://aclanthology.org/2020.acl-main.331.pdf