POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Enhancing Review-based Recommender Systems with Attention-driven Models Leveraging Large Language Model's Embeddings

Tesi di Laurea Magistrale in
Computer Science and Engineering - Ingegneria Informatica

Author: **Salvatore Marragony**

Student ID: 996233
Advisor: Prof. Maurizio Ferrari Dacrema
Academic Year: 2022-23

Abstract

This thesis aims to address a current gap in the field of recommender systems. While numerous techniques leverage user reviews as additional information to enhance recommendations, they often yield unsatisfactory results, leading to a decline in research interest in this direction. Conversely, there is a growing interest in utilizing Large Language Models (LLMs), a family of models born after 2017 and based on the Transformer architecture, demonstrating remarkable results across a variety of tasks. Although there are existing methods applying LLMs to recommender systems, literature currently lacks any technique focused on using an LLM to process user reviews for enriching recommendations. The underlying hypothesis of this work is that LLMs exhibit exceptional abilities in understanding human language and should, therefore, efficiently process user reviews, since written in natural language. To validate this hypothesis, seven new models are introduced in this thesis, ranging in complexity from simple content-based methods to neural models employing attention mechanisms with different settings. All models outperform the existing state-of-the-art models that leverage reviews for recommender systems. Notably, the final model, a straightforward hybrid ItemKNN incorporating both content-based and collaborative filtering, even outperforms effective collaborative filtering baselines like RP3Beta. These results confirm the validity of the hypothesis, paving the way for potential future developments in this direction.

Keywords: Recommender Systems, Large Language Models, Attention, Reviews

Sommario

Questa tesi ha lo scopo di riempire un'attuale lacuna nel mondo dei recommender systems. Esistono infatti numerose tecniche che sfruttano le recensioni lasciate da utenti a prodotti come informazione aggiuntiva per arricchire le raccomandazioni. Tuttavia questi metodi ottengono risultati spesso insoddisfacenti, portando a un declino dell'interesse per la ricerca in questa direzione. Al contrario, è sempre maggiore l'interesse verso l'utilizzo dei Large Language Models (LLMs), una famiglia di modelli nati dopo il 2017 e basati sull'architettura del Transformer, che stanno mostrando risultati incredibili in una grande varietà di task. Sebbene esistano metodi che applicano LLM ai recommender systems, al momento non esiste in letteratura nessuna tecnica basata sull'utilizzo di un LLM per processare le recensioni degli utenti e sfruttarle per arricchire le raccomandazioni. L'ipotesi alla base di questo lavoro è che i LLM dimostrano abilità eccezionali nel comprendere il linguaggio umano, e di conseguenza dovrebbero essere in grado di processare in modo molto efficace le recensioni degli utenti, in quanto composte da linguaggio umano. Per validare questa ipotesi, 7 nuovi modelli sono presentati in questa tesi, con un grado di complessità che va da semplici metodi content-based a modelli neurali che utilizzano meccanismi di attention con diverse configurazioni. Tutti i modelli ottengono risultati nettamente migliori rispetto ai modelli costituenti l'attuale stato dell'arte nell'utilizzo di recensioni in recommender systems. In particolare, l'ultimo modello (un semplice ItemKNN ibrido che utilizza sia content-based filtering che collaborative filtering) riesce persino a superare efficaci baseline di collaborative filtering come RP3Beta. Questi risultati confermano la veridicità dell'ipotesi, aprendo la strada a possibili futuri sviluppi in questa direzione.

Parole chiave: Recommender Systems, Large Language Models, Attention, Recensioni

Contents

1 Introduction

The world of recommender systems constitutes a continuously evolving field of research, captivating the interest of both academia and industry. The constant development of novel algorithms aims to enhance the existing state-of-the-art models to obtain improved results in real-world applications like social networks, streaming platforms and booking services. Recommender systems exploit any kind of collateral information to enhance the quality of recommendations provided, like user demographics, item attributes, temporal and location information, and contextual information.

Among these, also reviews given by users to items can be used as side information, as they are commonly found across various contexts. Whether as reviews of products purchased online, restaurant reviews, or movie feedback, users frequently leave reviews within platforms implementing recommender systems. These reviews can serve as supplementary information to enhance recommendations for the same users. Over the years, researchers have devised numerous techniques to harness user reviews in recommender systems [11]. Current state-of-the-art methodologies involve utilizing reviews to enrich both user and item profiles within the system. Examples of these techniques are HFT[34], NARRE[10], and HRDR[33]. However, as shown in Chapter 4, none of these methods has actually proven truly effective, as they struggle to achieve results comparable to simple collaborative filtering methods. Consequently, interest among researchers in utilizing reviews as additional information has waned.

Nevertheless, the world of artificial intelligence is undergoing a profound revolution, marked by the advent of a new class of models known as Large Language Models (LLMs). Emerging after 2017 and based on the Transformer architecture introduced in the paper "Attention is all you need" [56], LLMs represent a groundbreaking development in artificial intelligence, thanks to the introduction of the attention mechanism. The fusion of Transformer-based architectures and the continuous development of hardware technologies, particularly hardware accelerators enabling models with tens of billions of parameters (hence, "Large"), has led to the creation of increasingly powerful models. Models like ChatGPT[9] are now ubiquitous and employed across a myriad of artificial intelligence

tasks.

Given the remarkable capabilities demonstrated by these models, researchers are now exploring the application of LLMs to recommender systems. These techniques, however, are currently still in an experimental phase, as they are mainly based on prompt engineering techniques and they are hardly applicable in large-scale real-world scenarios; yet they exhibit promising results. Furthermore, at the time of writing of this thesis, there is no existing study in the literature about the use of LLMs on recommender systems based on user reviews. The aim of this thesis is to explore this new field and try to bridge the existing gap. The motivation behind this work lies in the fact that LLMs are designed with the primary aim of understanding human language, and they perform exceptionally well in this task. Consequently, they may effectively process user reviews, composed in human language, and use this processed information to enhance the recommendation process. Thus, the objective of this thesis is to evaluate this hypothesis for confirmation or refutation.

The structure of the thesis is as follows. In Chapter 2, current state of the art is presented, regarding both recommender systems, highlighting those that leverage user reviews, and LLMs, with a focus on models that use LLMs to produce recommendations. A brief discussion on evaluation techniques is also present. In Chapter 3, a detailed description of all models presented in the thesis is provided, covering both the theoretical description and the implementation details. In Chapter 4, all results are presented, comparing model performances against several baseline models on three different datasets, both from the perspective of pure metrics values and from the scalability side. Finally, in Chapter 5, the conclusions are drawn, highlighting novel results and possible directions for future works.

2 State of the art

This chapter will offer a comprehensive overview of the current state-of-the-art technologies relevant to the topics discussed in this thesis. It starts with an overview of Recommender Systems and of the most common and effective techniques used in the field, focusing then on the subsets of existing models that exploit reviews to enhance recommendations. The second part of the chapter will then focus on Large Language Models, which are the foundation of current top-performing AI models. After a description of the basic architecture on which these models are based, an overview of the applications of these models on recommender systems will be provided, highlighting the lack of methods based on LLMs that exploit reviews, which is the gap that this thesis aims to fill.

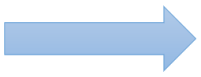
2.1. Recommender Systems

The term "Recommender Systems" refers to a vast and highly significant domain within the field of Artificial Intelligence, continually evolving and of profound interest to both researchers and companies. A recommender system is an algorithm with the main goal of suggesting to each user new items they have not yet interacted with. This broad definition basically finds practical application in every contemporary digital service. Streaming platforms, e-commerce websites, travel booking platforms, social networks, and so on are all examples of services that employ recommender systems. Depending on the context, the 'item' may be a movie, a product, or a hotel, and 'interaction' refers to actions such as viewing, purchasing, listening, or booking. Consequently, the efficacy of a recommender system holds paramount importance for businesses, as recommending the right products to users can lead to substantial growth in profits. It suffices to consider that 35% of Amazon's sales and 75% of what users watch on Netflix come from product recommendations based on their recommender systems[27].

A recommendation algorithm filters and analyses input data in order to provide recommendations, so the quality and variety of such data deeply affect the quality of the outcome. Input data can be divided into three categories: user data, item data, and interaction data. The information concerning users and items can be helpful in acquiring

valuable knowledge that can lead to higher-quality personalized recommendations, but interaction data are the foundation upon which a recommender system relies. Given a set of users and a set of items, it is possible to construct the so-called *User Rating Matrix* (URM), which is a matrix having users as rows, items as columns and every intersection $r_{u,i}$ between a row and a column represents the rating given by user u to item i. It is possible to distinguish between implicit ratings, representing just the presence or absence of interaction between a user and an item (so $r_{u,i}$ will be only either 0 or 1); and explicit ratings, where $r_{u,i}$ represents the actual rating given by u to i, for instance on a scale from 1 to 5 (1 means that the user didn't like at all the item, 5 means that he/she loved it). The URM is by nature very sparse, meaning that the great majority of the elements (more than 99%) will be 0. The reason is very easy to understand, considering for example that a Netflix user on average has watched between tens to a few hundred movies/tv series, which is just a very small fraction of the millions of products available on the catalogue. The main goal of the recommendation algorithm is to fill this sparse matrix, by assigning to each element the probability that a user will interact with each item, as shown in the example in Figure2.1. Therefore, if the goal of the system is to recommend n items to a user u, it will take the n items with the highest probability from the row of the URM representing u.

	Item1	Item2	Item3	Item4
User1	0	1	0	4
User2	5	0	0	0
User3	1	2	0	4
User4	5	4	1	0



	Item1	Item2	Item3	Item4
User1	0.3	0.8	0.2	0.7
User2	0.8	0.3	0.4	0.2
User3	0.6	0.7	0.4	0.9
User4	0.7	0.9	0.8	0.2

Figure 2.1: From sparse to dense URM.

2.2. Basic Recommenders

In this section, an overview of the main families of recommender systems is provided.

2.2.1. Non-personalized recommenders and global effects

The simplest approach to perform recommendations for items is to exploit interactions data considering only items, without any form of personalization for the different users, so that the same items will be recommended to all users. Possible options are to recommend

the top popular items (the ones with more interactions) or the best-rated ones (the items with higher average ratings). A step forward from non-personalized recommenders is to consider interactions data to identify global effects of both items and users. The predicted rating is computed through the formula $r_{u,i} = \mu + b_i + b_u$, where μ is the global bias (the average rating of the whole system), b_i is the item bias, so the average rating obtained by item i, and b_u is the user bias, so the average rating given by user u (since some users are more prone to give higher ratings than others).

2.2.2. Content-based recommenders

A step forward is to consider not only item interactions but also item attributes, in order to exploit the similarity between items and recommend items that are similar to the ones a user has interacted with[55]. These methods are based on the *Item Content Matrix* (ICM), which is a matrix having items as rows and attributes as columns. For each item, there is a 1 if the item has that attribute, 0 otherwise. Therefore, it is possible to compute the similarity between an item i and an item j ($s_{i,j}$) as the scalar product (possibly normalizing it and adding a shrink term) between the two vectors which are the rows of i and j in the ICM. Computing the similarity between all items, it is possible to obtain a similarity matrix S, that multiplied by the URM R gives the estimated URM $\tilde{R}$ ($\tilde{R} = R \cdot S$). This family of methods can be very effective with multiple and detailed attributes but often presents several issues: the similarity matrix is often too dense and therefore too much memory is needed; similarity values are often very small, so a popular solution is to take the k most similar items (*K-Nearest Neighbours - KNN*); the similarity matrix is only binary while it would be useful to weight attributes, which is a complex feature engineering task.

2.2.3. Collaborative Filtering

A more effective technique which is the main idea behind the most popular recommender systems is Collaborative Filtering (CF)[50]. This method doesn't exploit any side information about items or users, but it is entirely based on the information coming from interactions between users and items, based on the assumption that it is possible to understand what a user likes based on the similarity with the behavior of other users. Therefore, one great advantage of this method is that only the URM is needed, and the method is independent of item characteristics.

The simplest CF techniques, often referred to as "neighborhood methods", make recommendations by comparing a user's behavior and preferences with those of other users.

These methods compute a similarity matrix between elements, based on the URM, and use the similarity to predict the ratings for items the user has not interacted with. Depending on whether the similarity is computed between users or items, we can distinguish between user-based CF and item-based CF. They both exploit the same similarity functions, differing on whether it is computed between rows or columns of the URM. In case of implicit ratings, given two users u and v, it is possible to compute the cosine similarity between the two users as

$$s_{uv} = \frac{\vec{u} \cdot \vec{v}}{|\vec{u}|_2 \cdot |\vec{v}|_2 + C} \tag{2.1}$$

where $\vec{u}$ and $\vec{v}$ are the vectors containing all interactions for users u and v, while C is the shrink term to take into account the support. The same similarity can be computed between two items i and j exploiting the vectors $\vec{i}$ and $\vec{j}$. In the case of explicit ratings, other similarity functions can be used, like the Pearson correlation coefficient[41]. Once the similarity is computed, it is possible to use it to compute the predicted ratings just as mentioned in the previous section for content-based methods. The main difference between a content-based method and an item-based CF is that in the first case, the similarity is computed on the ICM, while in the second it is computed on the URM. To overcome the problem that most of the similarity values are very small, it is possible to take into account just the K nearest neighbors. To sum up:

- **Item-based KNN**

$$\hat{r}_{u,i} = \frac{\sum_{j \in KNN(i)} r_{u,j} \cdot s_{ij}}{\sum_{j \in KNN(i)} s_{ij}} \tag{2.2}$$

- **User-based KNN**

$$\hat{r}_{u,i} = \frac{\sum_{v \in KNN(u)} r_{v,i} \cdot s_{uv}}{\sum_{v \in KNN(u)} s_{uv}} \tag{2.3}$$

These methods excel in capturing immediate user feedback and often perform well in scenarios with a rich dataset and well-defined user patterns, but they struggle in cases of big sparsity or, in case of user-based models, with the cold-start problem (when a new user enters the system).

More complex techniques can be used to build predictive models using machine learning or statistical algorithms. These models are trained on user-item interaction data and learn patterns and relationships between users and items. Model-based approaches are typically more complex and can often offer better performance. Examples include matrix factorization models, like Singular Value Decomposition (SVD) or Alternating Least Squares (ALS). These methods decompose the user-item interaction matrix into latent factors. These factors capture hidden patterns in the data and are used to make recommenda-

tions. Netflix's movie recommendation system, for instance, uses matrix factorization[53].

Other examples are graph-based algorithms, which are based on the construction of a weighted bipartite graph based on user-item interaction data and item-item relationships. This graph connects users to items through edges, with edge weights determined by the interaction ratings. In P3 algorithms, like **RP3Beta**[14], the probability of the interaction between user u and item i is computed by performing 3 jumps on the bipartite graph, leveraging both on user-user similarity and on item-item similarity, as shown in Figure2.2.

1. The first jump is from user u to a seen item j, with the probability of this jump computed as $P_{u,j} = \frac{r_{u,j}}{N_u}$, with $r_{u,j}$ being the rating by u to j and N_u being the number of ratings given by u.
2. The second jump is from item j to a user v that has interacted both with j and i, with the probability of this jump being $P_{j,v} = \frac{r_{v,j}}{N_j}$, where $r_{v,j}$ is the rating given by v to j and N_j is the number of ratings received by j.
3. The last jump is from user v to item i, with probability $P_{v,i} = \frac{r_{v,i}}{N_v}$, where $r_{v,i}$ is the rating given by v to i and N_v is the number of ratings given by v.

The final predicted rating is then computed as

$$\hat{r}_{u,i} = \sum_{j} \sum_{v} P_{u,j} \times P_{j,v} \times P_{v,i} \tag{2.4}$$

RP3Beta introduces two hyperparameters, α and β: α is a hyperparameter that controls the weight of user-item interactions. A higher α value assigns more importance to user-item interactions. β is a hyperparameter that regulates the incorporation of item-item relationships in the recommendation process. It determines the influence of the item-item similarity matrix on the recommendations. A higher β value increases the significance of item associations, making recommendations more focused on suggesting items that are related to those a user has interacted with. The complete equation for RP3Beta is:

$$\hat{r}_{u,i} = \sum_{j} \sum_{v} P_{u,j} \times \frac{(P_{j,v})^{\alpha} \times (P_{v,i})^{\alpha}}{(N_j)^{\beta}} \tag{2.5}$$

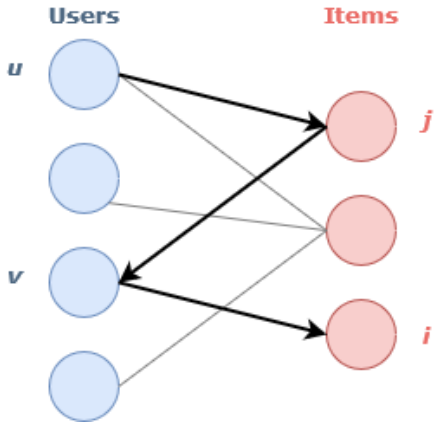


Figure 2.2: Example of 3 steps on the bipartite graph. To predict the interaction between user u and item i, the interactions between u and j, j and v and v and i are used

2.2.4. Hybrid recommenders

Hybrid recommender systems combine the strengths of different recommendation techniques to provide users with more accurate and diverse recommendations. These systems leverage both collaborative filtering and content-based filtering. By integrating these approaches, hybrid recommenders aim to mitigate the limitations of each method while capitalizing on their respective advantages. It is possible to combine multiple models in different ways:

- **Linear combination:** the resulting predicted ratings are a linear combination of the ratings predicted by different models
- **Pipelining:** a first model is used to enrich the URM that is used as input to a second model
- **Merging models:** different models are merged in some way before computing the predicted ratings. An example is to use as a similarity matrix the weighted average of the similarity matrices produced by different models.

2.3. Review-based recommenders

Reviews provided by users represent a rich source of additional information that can be harnessed. Consequently, over the years, numerous studies have been conducted on how to leverage this information to enhance recommender systems[11]. Various techniques have been employed to make use of reviews:

- **Term-based profile**: use TF-IDF[49] give weights to words and decide which are

the most relevant to use as attributes to a user/item and use them to recommend[17]. Based on the conclusions from Chen et al. in the survey [11], compared to traditional rating-based CF, these methods perform slightly worse but recommend new and unpopular items more often.

- **Ratings inference:** use reviews to infer item ratings given by users. Some techniques use aggregation[58] or machine learning[42] to perform sentiment analysis on reviews. In this way, it is possible to discern whether a review refers to a positive or negative rating, in order to build the URM used for CF techniques. Results are comparable with CF using real ratings [11].
- **Ratings enhancing**: reviews can still be exploited in the presence of ratings, as an auxiliary resource to weight them. These methods exploit reviews considering their helpfulness[44], their topics[36], their contexts[24] or the emotions expressed in them[35], using this information to improve the quality of the ratings. All of these methods show an overall improvement in performance, especially regarding the cold-start and the data sparsity problems.
- **Item profile enriching:** these methods aim to enrich the item profile rather than the user profile. Some extract opinions on an item from different users, taking into account also the expertise of the reviewer or the popularity of the item, or use them together with the item specifications[4]. Others use reviews to rank items so that opinions on two comparable items are used to decide which is better[20].

Despite the large number of research on how to exploit reviews, these methods' popularity has significantly decreased, because of the unsatisfactory results. The majority of these techniques date back more than ten years, and the application of reviews has received less and less attention over time. Currently, only a few methods actively harness user reviews in the recommendation process, and many struggle to outperform basic collaborative filtering baselines[47]. In the following sections, the three models that currently represent the state-of-the-art recommender systems that leverage reviews will be described. These models will serve as baselines for the proposed model.

2.3.1. HFT

Hidden Factors and Hidden Topics [34], is a model based on the combination of two main techniques.

The first is a 'standard' latent-factors model[31], which exploits latent representations of

users and items to predict the rating $r_{u,i}$ for a user u and item i, according to

$$\hat{r}_{u,i} = \alpha + \beta_u + \beta_i + q_u \cdot p_i \tag{2.6}$$

where α is an offset parameter, β_u and β_i are user and item biases, and q_u and p_i are K-dimensional user and item factors (respectively).

The second technique is Latent Dirichlet Allocation (LDA)[7], which aims to uncover hidden dimensions in review texts. Each document d is associated with a K-dimensional topic distribution θ_d, which is a stochastic vector that encodes the fraction of words in d that discuss each of the K topics. So, words in the document d discuss topic k with probability $\theta_{d,k}$. Each topic k also has an associated word distribution, ϕ_k, which encodes the probability that a particular word is used for that topic. Finally, the topic distributions themselves (θ_d) are assumed to be drawn from a Dirichlet distribution.

The goal of HFT is not to treat these two latent dimensions independently but to link them together, based on the idea that the latent representations derived from ratings are somehow based on properties from users and items, and these properties can be enriched by the latent dimensions obtained by the textual reviews. It is not trivial to link p_i and θ_i, as the first is a rating factor in the domain $\mathbb{R}^K$ while the second is a probability between 0 and 1. Therefore, the mapping is defined by the following transform:

$$\theta_{i,k} = \frac{\exp(\kappa p_{i,k})}{\sum_{k'} \exp(\kappa p_{i,k'})} \tag{2.7}$$

where κ is an hyperparameter with the intuition that a large κ means that users only discuss the most important topics, while small κ means that users discuss all topics evenly. For the final model, both latent factors are used, as the rating factor is used to predict the real rating while the latent review factors, which are used to define the corpus likelihood, are used as a regularization term in the loss function chosen to learn how to predict the real ratings:

$$\arg\min_{\theta,\phi,k,z} \sum_{r_{u,i} \in T} (\hat{r_{u,i}} - r_{u,i})^2 - \mu l(T|\theta, \phi, z) \tag{2.8}$$

where the first part is the rating error, l is the likelihood for corpus T, θ, ϕ, z, k are parameters and μ is an hyperparameter.

As described in the paper that presents it[34], HFT has been tested using three datasets, two of which are used in this thesis as well and will be described accurately in section

4.1.1. The model outperforms recommender systems based on either of the two techniques which are instead combined in HFT (latent factor model and LDA), by obtaining better results in terms of *Mean Squared Error* (MSE) (squared version of eq. 2.19).

2.3.2. NARRE

Neural Attentional Regression model with Review-level Explanations [10], is a neural network model with the aim to exploit reviews not only to improve ratings prediction but also to use them as explanations for users recommendations. By learning the usefulness of each review, it is possible to select highly useful ones to improve both recommendations and explainability.

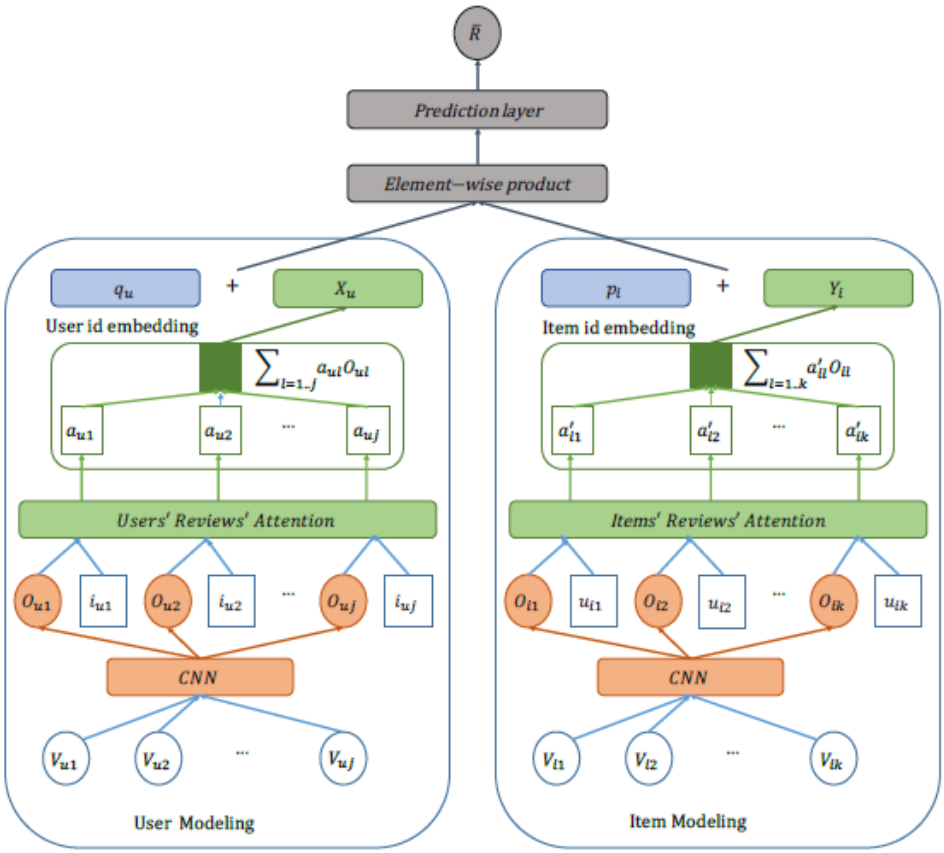


Figure 2.3: NARRE model architecture[10].

As shown in Figure2.3, the architecture of the model is based on two parallel networks, one for user modeling and one for item modeling. When predicting a rating $r_{u,i}$, all reviews from user u are processed by the user network and all reviews for item i are processed by the item network; then the outputs of the two networks are merged together in a last prediction layer. Each network starts with a CNN text processor to process reviews,

already used in previous state-of-the-art models like DeepCoNN [60]. In this processor, a word is mapped into an embedding vector of dimension d, so each review composed of n words is transformed in an embedding matrix of dimension $n \times d$. Then a convolutional neural network transforms each matrix into a feature vector O, so that an item i (or similarly an user u) will be represented by the vectors $O_{i,1}$, $O_{i,2}$, ... $O_{i,j}$, with $O_{i,j}$ the feature vector of the j-th review associated to item i. The biggest novelty introduced by NARRE is the way these vectors are aggregated together: it is in fact the first model to introduce an attention mechanism in this context. For each review, an attention score is computed through the following formula:

$$a_{i,j} = \text{SOFTMAX}(h^T ReLU(W_O O_{i,j} + W_u u_{i,j} + b_1) + b_2) \tag{2.9}$$

where $W_O \in \mathbb{R}^{t \times k_1}$, $W_u \in \mathbb{R}^{t \times k_2}$, $b_1 \in \mathbb{R}^t$, $h \in \mathbb{R}^t$, $b_2 \in \mathbb{R}^1$ are model parameters, t denotes the hidden layer size of the attention network, and ReLU[37] is a nonlinear activation function. Once obtained the attention weight of each review, the feature vector of item i is calculated as the following weighted sum:

$$O_i = \sum_{j=1}^{k} a_{ij} \cdot O_{ij} \tag{2.10}$$

To obtain the final representation Y_i of item i, O_i is sent to a fully connected layer. The same pipeline is applied in parallel to user and item, to obtain X_u and Y_i respectively. Finally, the outputs of the two networks are merged together using a neural form [26] of Latent-Factors Models (equation 2.6). First, the interaction between user u and item i is modeled as

$$h_0 = (q_u + X_u) \odot (p_i + Y_i) \tag{2.11}$$

where q_u and p_i are user preferences and item features based on ratings, as in equation 2.6, X_u and Y_i are user preferences and item features mentioned above, and $\odot$ denotes the element-wise product of vectors. Finally, the predicted rating $\hat{r}_{u,i}$ is computed as:

$$\hat{r}_{u,i} = W_1^T h_0 + b_u + b_i + \alpha \tag{2.12}$$

where $W_1 \in \mathbb{R}^n$ denotes the edge weights of the prediction layer, and b_u, b_i, and α represent the user bias, item bias, and global bias, respectively. The model is trained to optimize the square loss between the real and the predicted ratings. The datasets used to test NARRE, as reported in the paper[10], are the same described in Section 4.1.1. As baselines, both methods that do not include reviews (like Matrix Factorization

techniques[32]) and methods that leverage reviews (HFT[34] and DeepCoNN[60]) were used. As reported in its paper[10], NARRE outperforms all baselines in terms of *Root Mean Square Error* (RMSE)(2.19).

2.3.3. HRDR

Hybrid Neural Recommendation with joint Deep Representation learning of ratings and reviews [33], is a deep learning model with the unique intuition of combining information coming from both ratings and reviews, in order to exploit ratings to understand the different usefulness of reviews.

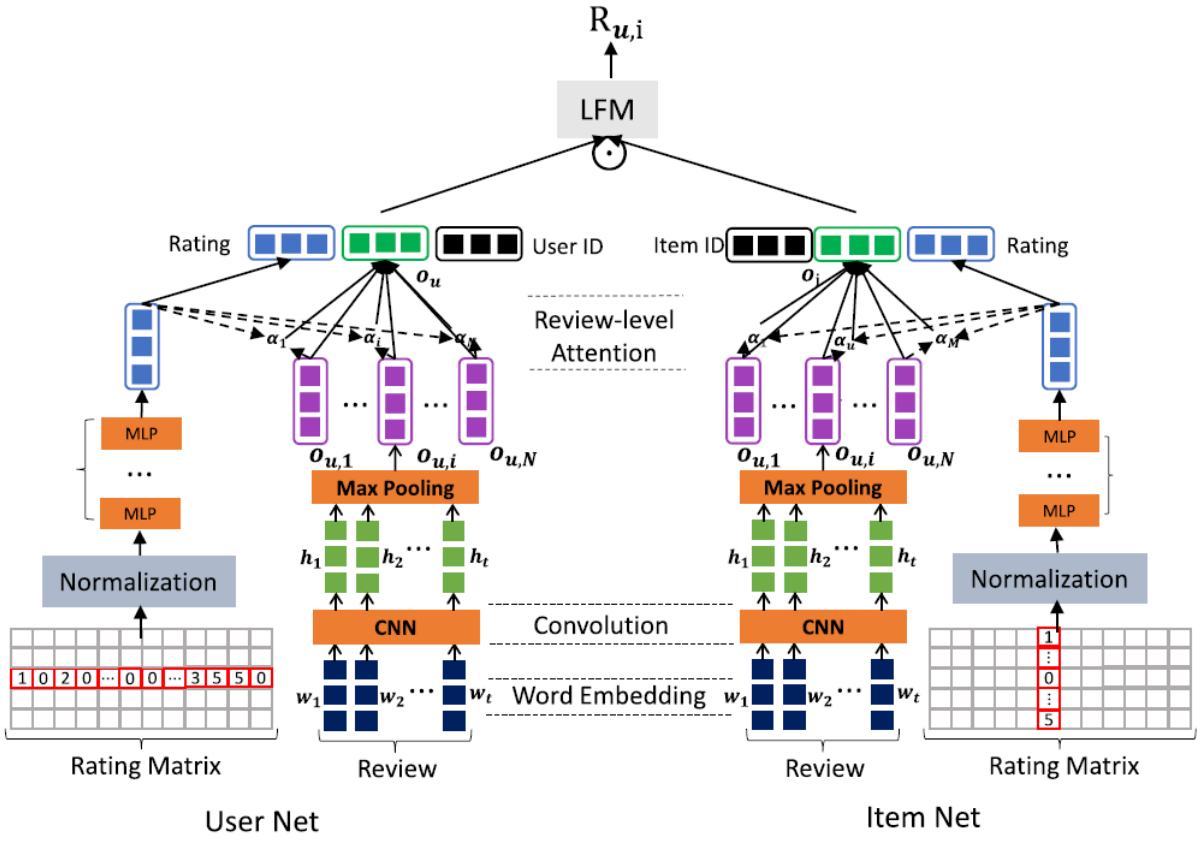


Figure 2.4: HRDR model architecture.

The model is an evolution of NARRE, as the architecture is very similar (Figure2.4). The main differences are in the attention mechanism and how rating embeddings are computed. In order to compute the latent features describing the rating patterns, a Multi-Layer Perceptron (MLP) network is used, where the output of the k-th layer is

$$x_u = \sigma(W^{(k)} x^{(k-1)} + b^{(k)}) \tag{2.13}$$

where the input x is the normalized rating pattern of user u (i.e., the row in rating matrix R of user u), δ is an activation function, W, b are the parameters in hidden layers. Reviews are processed in the same way described in NARRE, obtaining latent feature vectors describing each review belonging to a user/item. The biggest novelty introduced by HRDR lies in the way attention scores are computed, as the rating embeddings are used inside the attention computation, based on the idea that there's a correlation between ratings and reviews and ratings can help to understand reviews better. For a user u, given the latent feature vector $O_{u,m}$ obtained from the m-th review, the attention score $a_{u,m}$ can be computed as:

$$\begin{aligned} a_{u,m} &= \text{SOFTMAX}(O_{u,m} \odot q_r) \\ q_r &= W^q x_u + b^q \end{aligned} \tag{2.14}$$

where q_r is the attention query vector derived from the rating-based representation x_u via a linear transformer, $\odot$ denotes the dot product operator, W^q and b^q are the parameters. Then, the review-based representation of user u is obtained through a weighted sum and then passed through a linear layer, just as in NARRE. The final predicted rating $\hat{r}_{u,i}$ is computed as in equation 2.12, with h_0 being the product of two vectors containing rating-based representation, review-based representation and ID embedding of user and item respectively. The model is trained to optimize the square loss between the real and the predicted ratings. As reported in its paper[33], the datasets used to test HRDR are the same described in Section 4.1.1. Baselines for this method include both Matrix Factorization techniques[32] and methods previously described (HFT and NARRE). According to [33], HRDR outperforms all baselines in terms of *Root Mean Square Error* (RMSE)(2.19).

2.4. Large Language Models

Traditional language models have served as foundational tools in natural language processing for decades. These models, often based on techniques like N-grams[8] and statistical language models, aimed to capture the essence of language through patterns and probabilities. Their main goal was to predict the likelihood of a sequence of words given the preceding context. While effective for certain applications like spell-checking and speech recognition, traditional language models had notable limitations. They struggled with contextual understanding and could not capture the rich semantic nuances of human language. Additionally, these models often failed to generate coherent and contextually relevant text beyond simple sentence completions. Their limited training data and smaller model sizes constrained their capacity to handle the complexities of language and resulted in often rigid and occasionally incomprehensible outputs. These constraints

led to the evolution of *Large Language Lodels* (LLMs), which aimed to overcome these limitations. Large language models are a new family of language models with two main characteristics: they are composed of billions of learning parameters, trained using massive amounts of data; and are based on the Transformer[56], an architecture introduced in 2017 by Vaswani et al. that has led to a massive revolution in the field of artificial intelligence, especially in Natural Language Processing (NLP). A more detailed description of the Transformer follows.

2.4.1. Transformer

The Transformer is an encoder/decoder architecture based on self-attention. The encoder is a stack of six identical layers, where each layer has two sub-layers: a multi-head self-attention mechanism followed by a simple fully connected feed-forward network. The decoder is also a stack of six identical layers, each composed of three sub-layers: two are the same as the encoder plus another multi-head self-attention layer that takes as input the output of the encoder. Before the encoder/decoder part, inputs and outputs are embedded in vectors of fixed dimension, to which positional encoding vectors are summed in order to account for the relative or absolute position of the tokens in the sequences. Finally linear transformation and a softmax function are used to map the output of the decoder to probabilities of predicted next-token. The full architecture is shown in Figure2.5.

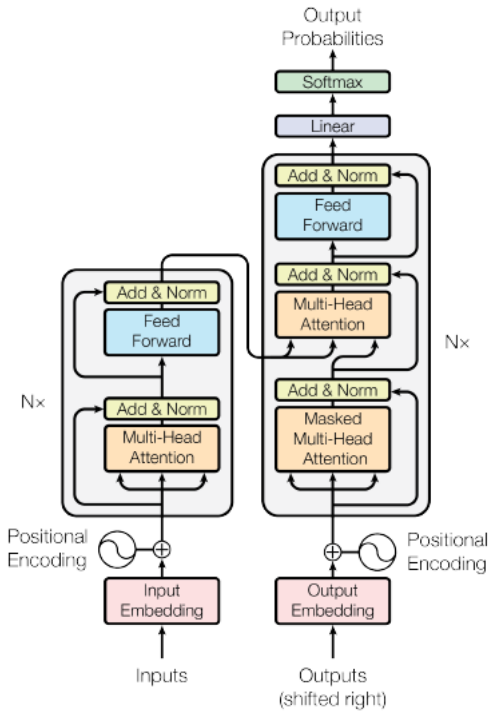


Figure 2.5: Transformer architecture[56].

Attention mechanism

Attention is a core concept in neural network models, enabling selective focus on specific input elements while dynamically weighting their importance. This mechanism allows models to capture contextual information, adapt to variable-length sequences, and significantly enhance performance in various tasks, particularly in natural language processing. While the concept of Attention was already introduced in previous works[22], the attention mechanism inside the Transformer was the biggest breakthrough in the field.

The foundation of this mechanism is the *Scaled Dot-Product Attention*. This type of Attention takes three inputs:

- Queries (Q): represent the elements in the input sequence that the model wants to focus on or retrieve information about.
- Keys (K): help establish relationships between the queries and the elements in the input sequence.
- Values (V): represent the actual information associated with the elements in the input sequence. They are used to construct the output based on the attention scores computed from the queries and keys.

Queries and keys have dimension d_k, values have dimension d_v. Given as input the matrices Q, K and V, the Scaled Dot-Product Attention is computed as:

$$\text{Attention(Q, K, V)} = \text{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.15}$$

Instead of performing a single attention function, queries, keys and values are linearly projected h times to dimensions d_k, d_k and d_v, so that attention is performed in parallel on different learned linear projections. The outputs are then concatenated and linearly projected again, to obtain the final output (Figure 2.6). This is called *Multi-Head Attention*, and is how attention layers inside the Transformer work. Here is the complete formula:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h) \cdot W^O \tag{2.16}$$

where

$$\text{head}_i = \text{Attention}(Q \cdot W^{Q_i}, K \cdot W^{K_i}, V \cdot W^{V_i}) \tag{2.17}$$

Where the projections are parameter matrices: $W^{Q_i} \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W^{K_i} \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W^{V_i} \in \mathbb{R}^{d_{\text{model}} \times d_v}$ and $W^O \in \mathbb{R}^{h \cdot d_v \times d_{\text{model}}}$.

In the decoder part of the Transformer, the queries come from the previous decoder layer, and the memory keys and values come from the output of the encoder.

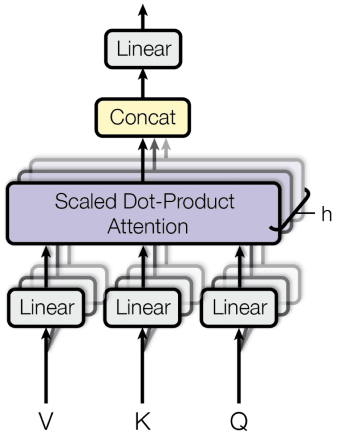


Figure 2.6: Multi-Head Attention.

The encoder contains *self-attention* layers, which are layers where queries, keys and values all come from the same output of the previous encoder layer. Self-attention has many benefits compared to previous attention techniques, as it allows a lower total computational

complexity, a greater amount of computation that can be parallelized, and shorter paths between long-range dependencies. This last aspect represents the biggest improvement from self-attention, as it allows to understand important relationships between elements that are very far in a sequence.

2.4.2. Main LLMs

In this section some of the current most powerful LLMs are described. It is interesting to notice how recent all these models are, and how a model can greatly evolve in a matter of a couple of years within the same company.

1. **GPT-3:** released in 2020 by OpenAI, GPT-3 [9] (GPT stands for Generative Pre-trained Transformer) was the first widely popular LLM, showcasing the enormous capabilities of LLMs. It can generate human-like text, answer questions, translate languages, and perform a wide range of language-related tasks. The greatest innovation of this model is its ability to achieve high performances on all tasks without requiring specialized fine-tuning for each different activity. When given challenging tasks like one-shot or zero-shot learning, which require performing a task with either one or zero examples, it demonstrates exceptional ability. It was trained using the Common Crawl dataset[1], composed of petabytes of data from page crawling collected over 12 years.

2. **GPT-4:** released in 2023 by OpenAI[39], it is the evolution of GPT-3. It shows great improvements in performance compared to its predecessor, thanks to its fine-tuning using *Reinforcement Learning from Human Feedback* (RLHF)[13]. The primary issue with reinforcement learning is defining a practical reward function. Human feedback works incredibly well as reward functions, but the number of feedbacks needed is massive. In traditional reinforcement learning, an agent is given an observation, acts upon it, and is rewarded accordingly. With this technique, a human agent evaluates two sets of observations-actions through brief videos to determine which is better, rather than awarding a reward for each action. The optimization function receives the outcome. The main drawback of GPT-4 is that it lacks knowledge of events happened after September 2021 (when its pre-training ended), and it does not learn from experience.

3. **PaLM 2:** released in 2023 by Google, it's the evolution of PaLM[12], outperforming it in many aspects. Its performance is possible thanks to an enormous amount of training data as well as trainable parameters (340 billion), trained thanks to highly efficient parallelization. It is based on Google's Pathways architecture[6], which aims

to develop a versatile model capable of excelling in a wide range of tasks, resembling the way humans learn new skills by building upon prior knowledge instead of starting from scratch. This architecture creates a sparse model, activating only the necessary neurons for specific tasks, similar to the way the human brain works, allowing it to have a much lower energy cost.

4. **LLaMA:** released in 2023 by Meta[52], is a group of LLMs with different sizes of trainable parameters. According to Meta, these models take the best intuitions from the best other existing models in order to outperform SOTA baselines, enabling even smaller versions to achieve impressive results. The main architecture is a "simple" Transformer with many little improvements.

5. **MT-NLG:** developed in collaboration by Nvidia and Microsoft, the Megatron-Turing Natural Language Generation model[51] is a huge LLM with 540 billion parameters. It exploits latest-generation GPUs produced by Nvidia to enable training with massive amounts of data and numerous parameters. Rather than impressing with its effectiveness, it demonstrates the utility of highly powerful hardware accelerators in the realm of LLMs.

Model	Company	Year	#Parameters
GPT-4	OpenAI	2023	Unknown
PaLM 2	Google	2023	340B
LLaMA	Meta	2023	7-65B
MT-NLG	Nvidia/Microsoft	2021	530B
GPT-3	OpenAI	2020	175B

Table 2.1: Overview of the main LLMs

2.4.3. LLMs to generate embeddings

Embeddings obtained from text are numerical representations of words, phrases, or sentences that capture their semantic and contextual information in a dense vector space of fixed dimension.

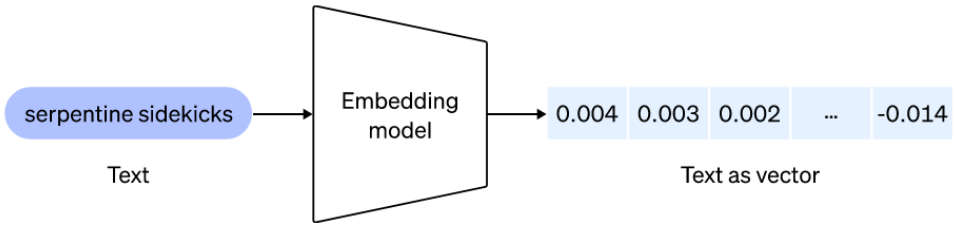


Figure 2.7: Embedding generation process (from [23])

The underlying intuition is that by mapping two embeddings as points within the vector space of their embedding dimension, they will be close if they correspond to two words/phrases with similar meanings; otherwise, they will be distant. Thanks to the strong ability to understand and interpret text, LLMs have also shown great versatility in generating embeddings. These embeddings capture intricate semantic relationships, enabling them to represent textual information in a dense, high-dimensional space. LLMs leverage their pre-trained knowledge to transform words, phrases, or sentences into continuous vectors, offering a rich source of contextual information. Current state-of-the-art embedding models[38] are pre-trained LLMs with slight modifications and fine-tuned to produce better results. In particular, these models are based on a Transformer encoder, which maps the input to a dense vector representation. Two special tokens, [SOS] and [EOS], are inserted to the start and end of the input sequence respectively. When given the sequence as input to the encoder, the last layer hidden state corresponding to the token [EOS] is extracted as the output embedding of the input sequence. The best embedding model currently available through OpenAI's APIs is *text-embedding-ada-002*[23]. It replaces five separate models for text search, text similarity, and code search, and outperforms OpenAI's previous most capable model, with a much lower price. According to the company, the model outperforms all other models in almost all baselines, has a longer context length and a smaller embedding size (1536).

2.4.4. LLMs for Recommender Systems

As demonstrated in the previous section, the development of Large Language Models is highly dynamic and continuously evolving. Each year, new models are released, improving upon the previous state-of-the-art. Consequently, the use of LLMs in recommender systems is still a relatively new yet highly fascinating area of study. Researchers are continually investigating the optimal approaches to leverage the vast potential of LLMs to enhance recommendation systems. Some of the popular techniques used in this field at the moment are described here:

1. **P5 model:** Pretrain, Personalized Prompt and Predict Paradigm[21] is a text-to-text paradigm to develop recommender systems. The model is able to unify five types of recommendation tasks: Sequential Recommendation, Rating Prediction, Explanation Generation, Review Summarization, and Direct Recommendation. All accessible data sources, including user profiles, item details, user feedback, and interactions, are transformed into natural language sequences. This extensive dataset in a multitasking setup enables P5 to acquire the semantic knowledge necessary for generating personalized recommendations. The features input is fed to the model using adaptive personalized prompt templates. P5 generates recommendations treating all personalized tasks as a conditional text generation problem and solving them using an encoder-decoder Transformer model pre-trained with instruction-based prompts. The model exhibits great zero-shot and few-shots capabilities.

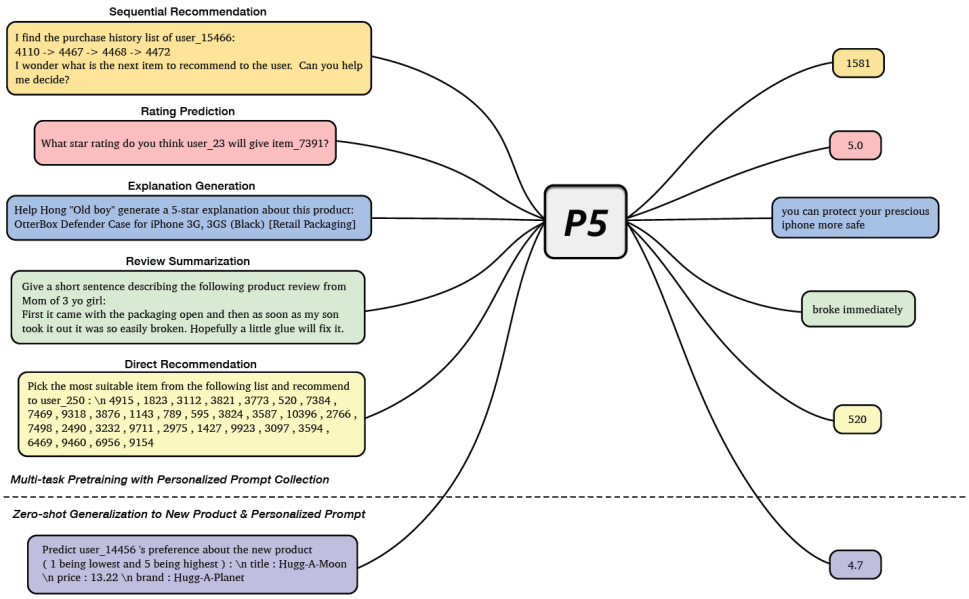


Figure 2.8: P5 model[21].

2. **M6-Rec:** developed using Alibaba's M6 LLM, M6-rec[15] is a recommendation model that uses textual prompt as inputs to perform several recommendation tasks, which are converted to either language understanding or generation. To train the model, only a negligible amount (1%) of task-specific parameters were added, without making changes to the original M6 Transformer model.

3. **LMRecSys:** Language Model Recommender Systems[59] uses the user's past interaction history (watched movies), taken from the MovieLens-1M dataset, to predict the next movie that the user would watch, with a textual prompt-based input. The primary objective is to employ Pretrained Language Models (PLMs) for zero-

shot recommendations, as these models come with pre-existing knowledge of various items and do not need to rebuild it based on past interactions. This proves to be highly advantageous, but overall this model shows quite poor performance[59].

4. **NIR:** Zero-Shot Next-Item Recommendation[57] is a model based on a three-steps prompting strategy to perform next-item recommendations in a zero-shot setting. First, they prompt GPT-3 to summarize the user's preferences using the items from the watch history. Next, they use GPT-3's response and the candidate movies in a prompt to request GPT-3 to select representative movies in descending order of preference. Finally, they create a recommendation prompt to guide GPT-3 to recommend 10 movies from the candidate set that are the most similar to the representative movies. It is the only model explicitly built to perform top-n recommendation, which is the most common task with Recommender Systems.

It is interesting to notice how none of these methods focuses on the use of reviews to exploit LLMs' great capability to understand human language. The models presented in the next chapter aim to fill this gap.

2.5. Evaluation

To evaluate recommendation models, it is possible to use two different groups of metrics

2.5.1. Regression metrics

This family of metrics aims to evaluate the ability of a model to reconstruct the real ratings given to each user-item interaction. Therefore these metrics measure the distance between the ground truth value and the predicted value. Some of the most popular are:

- **MAE:** Mean Absolute Error, measure the mean absolute distance between the real ratings and the predicted ones, through the following formula:

$$\text{MAE} = \frac{1}{n}\sum_{u,i}|r_{u,i} - \hat{r}_{u,i}| \tag{2.18}$$

- **RMSE:** Root Mean Squared Error, measure the square root of the squared of the distance between the two values. It gives higher weight to larger error and is more sensitive to outliers.

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{u,i}(r_{u,i} - \hat{r}_{u,i})^2} \tag{2.19}$$

In both equations $r_{u,i}$ is the real rating while $\hat{r}_{u,i}$ is the predicted rating.

Regression methods like HFT, NARRE and HRDR aim to minimize the values of these metrics

2.5.2. Ranking metrics:

Ranking metrics are more often used in the field of recommender systems as they are able to assess whether a model is recommending the right items to a user, regardless of the predicted rating. Once computed the probabilities for each item to be liked by each user, the top k items with higher probabilities are taken for each user to compute the following metrics:

- **Precision@k:** measures the proportion of relevant items (recommended items that correspond to an interaction in the test set) among the top-k recommendations.

$$\text{Precision@k} = \frac{\text{Number of relevant items in the top-k recommendations}}{k} \tag{2.20}$$

- **Recall@k:** assesses the ability of the recommendation system to retrieve relevant items within the top-k positions, considering all relevant items in the dataset.

$$\text{Recall@k} = \frac{\text{Number of relevant items in the top-k recommendations}}{\text{Total number of relevant items}} \tag{2.21}$$

- **MAP@k:** Mean Average Precision, takes into account also the order of the recommended items. It is the mean of the Average Precision at k (AP@k), which is the average of the precision computed for every element in the top-k list.

$$\text{MAP@k} = \frac{1}{N_u} \sum_{u \in U} AP@k_u \tag{2.22}$$

where

$$\text{AP@k} = \frac{1}{m} \sum_{k} P(k) \cdot rel(k) \tag{2.23}$$

whit N_u being the number of items, m being the number or relevant items for each user and $rel(k)$ tells whether the recommendation at position k is relevant (1) or not (0).

- **NDCG@k:** Normalized Discounted Cumulative Gain, measures the usefulness, or gain, of a recommendation based on its position in the recommendation list. The

gain is accumulated from the top of the list to the bottom, with the gain of each recommendation discounted at lower ranks.[29] The normalized version corrects for the fact that different users may have different numbers of relevant items.

$$\text{NDCG@k} = \frac{DCG@k}{IDCG@k} \tag{2.24}$$

where

$$DCG@k = \sum_{i=1}^{k} \frac{rel_i - 1}{\log_2(i+1)} \tag{2.25}$$

rel_i represents the relevance score of the item at position i in the list. DCG@k calculates the cumulative gain of the top-k items in a ranked list, giving higher importance to relevant items at higher positions. IDCG@k represents the theoretical maximum DCG@k achievable for a set of items and their relevance scores, reflecting the ideal ranking order.

3 | Models and methods

In this chapter, an overview of the models presented in this thesis is provided. The structure and the theoretical foundations underlying the decision-making choices are described, as well as implementation aspects. The work in this thesis is based on seven different models, each adding a degree of complexity to the previous one, to clearly understand the effect of single modifications on the model performance. All models are based on the use of embeddings obtained from user reviews through an LLM. The focus of the models was chosen to address the current gap in the literature. As outlined in Chapter 2, there are existing models for making recommendations that either utilize reviews or are based on LLMs. However, no existing model combines these two characteristics. Therefore, the hypothesis is that the combination of these two aspects may yield promising results, given that LLMs are designed to effectively understand human language, and reviews consist of human language. Therefore, the aim of this thesis is to empirically validate or refute this hypothesis.

To generate embeddings, OpenAI's text-embedding-ada-002 model (see Section 2.4.3) was used. Therefore, all reviews were embedded in a vector of fixed dimension $d = 1536$. All models have the same basic structure. They are based on the same intuition used in NARRE (Section 2.3.2) and HRDR (Section 2.3.3) to use reviews to model users and items separately and then merge the information coming from both in the final layer. Hence, all models are two-tower models, with a tower responsible for computing the user profile given all the information associated with the user, and the other to compute the item profile in the same way. Both towers are identical, thus, for each model, the description of just one tower will be provided, as the other works the same way. Compared to the previously mentioned baselines, the last layer is a simple linear layer, without considering any external bias, to keep the structure as simple as possible. Therefore, calling the output of the user tower p_u and the output of the item tower q_i, the resulting predicted rating is

$$\hat{r}_{u,i} = W(p_u \times q_i) + b \tag{3.1}$$

where W and b are parameters of the linear layer.

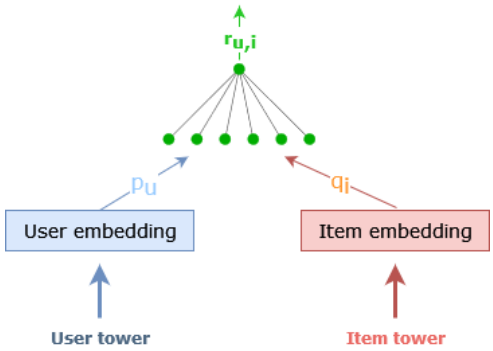


Figure 3.1: Basic structure for every model.

In the following sections, each model is described.

3.1. Model 1 - Review Embeddings only (RE)

3.1.1. Aim of the model

The first model focuses entirely on the effect generated by the review embeddings, without any kind of collaborative filtering information. The aim of this model is to understand whether the sole information coming from embedded reviews can be meaningful enough to build a valid user/item profile, aggregating all reviews associated with the user/item. Therefore, for each user-item interaction, the model only takes as input all the embedded reviews given by that user and all the embedded reviews for that item.

3.1.2. Architecture

Each tower takes a matrix as input. For the user tower, the matrix has dimension $n_{rev\_u} \times d$, with n_{rev_u} being the number of reviews given by the user; for the item tower, the matrix has dimension $n_{rev\_i} \times d$, with n_{rev_i} being the number of reviews received by the item. In both matrices, d is the embedding dimension (1536 in this case). Each group of embeddings is processed through a *Multi-head Attention* layer (see Equation 2.16) in a self-attention manner (so embedded reviews are used as keys, queries and values). The attention layer maps a set of embeddings into another set with the same cardinality, which models the interrelationship among reviews referring to the same user/item. This new set is then aggregated through a pooling layer to obtain the user/item embedding. The pooling strategy can be either the sum or the mean of all reviews after being processed by the multi-head attention. The decision on the pooling strategy was taken in the hyperparameter tuning phase (see Section 4.3), where the sum turned out to be the most effective

choice, while the mean led to very poor results. This result already highlights a potential bias towards more active users and popular items, whose embeddings, after aggregation, will have a very high norm. This provides an initial indication that embeddings alone may not be sufficient to generate effective recommendations for every user.

The result of the aggregation is then given to a linear layer, that produces p_u for the user tower and q_i for the item tower. These embeddings are then grouped together as shown in Figure 3.1.

3.1.3. Hyperparameters

The only hyperparameters for this model are the dropout rate in the attention layer and in the linear layer, the number of heads for the multi-head attention, and the pooling strategy.

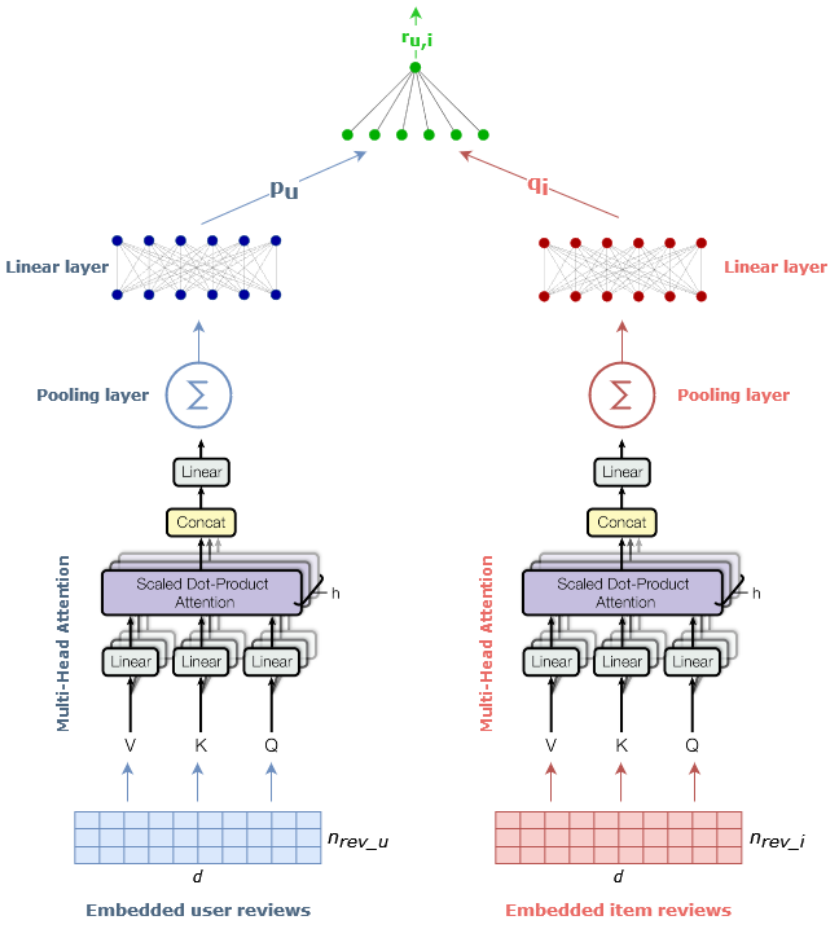


Figure 3.2: Model RE architecture.

3.2. Model 2 - Aggregation of Review and Collaborative Embeddings (RE+CE)

3.2.1. Aim of the model

The second model introduces collaborative filtering information, taking into account not only embedded reviews but also ratings assigned to user-item interactions. The purpose of this model is to understand how much an embedding generated from information extracted from interactions can enrich the user/item profile resulting from the embedded reviews. Therefore the input is composed of both embedded reviews and ratings information.

3.2.2. Architecture

For each user, all ratings given by them are taken (the corresponding row in the URM) and stored in a vector of dimension n_{items}. Similarly, for each item, the corresponding column in the URM is taken to store the ratings in a vector of dimension n_{users}. Each vector is then processed by a *Multi-Layer Perceptron*(MLP) that maps it from the original dimension to a smaller dimension $n_{factors}$. The reason behind this choice is to learn embeddings that can carry useful information from the rating patterns. Each MLP is composed of three linear layers, each followed by a ReLU [37] activation function. The final layer conducts batch normalization[28], which standardizes the activations from the preceding layer to have a mean of around 0 and a standard deviation of approximately 1 within the mini-batch. This layer introduces two learnable parameters, γ and β, which allow the network to scale and shift the normalized activations adaptively. These parameters are learned during training. The final user profile p_u and item profile q_i are the concatenation of the resulting collaborative filtering embeddings from this part and the vectors obtained from embedded reviews, processed as Model 1. Both profile vectors have dimension $d_{collab} + d_{reviews}$, where $d_{collab} = n_{factors}$ and $d_{reviews} = d(1536)$. The two vectors are then merged together, as described at the beginning of the chapter.

3.2.3. Hyperparameters

The hyperparameters for this model are the number of nodes in the hidden layers for both the user and the item MLP, and the dimension of the collaborative filtering embedding $n_{factors}$, as well as the dropout rate for all linear layers and for the attention layer, the number of heads for the multi-head attention, and the pooling strategy to aggregate the embedded reviews.

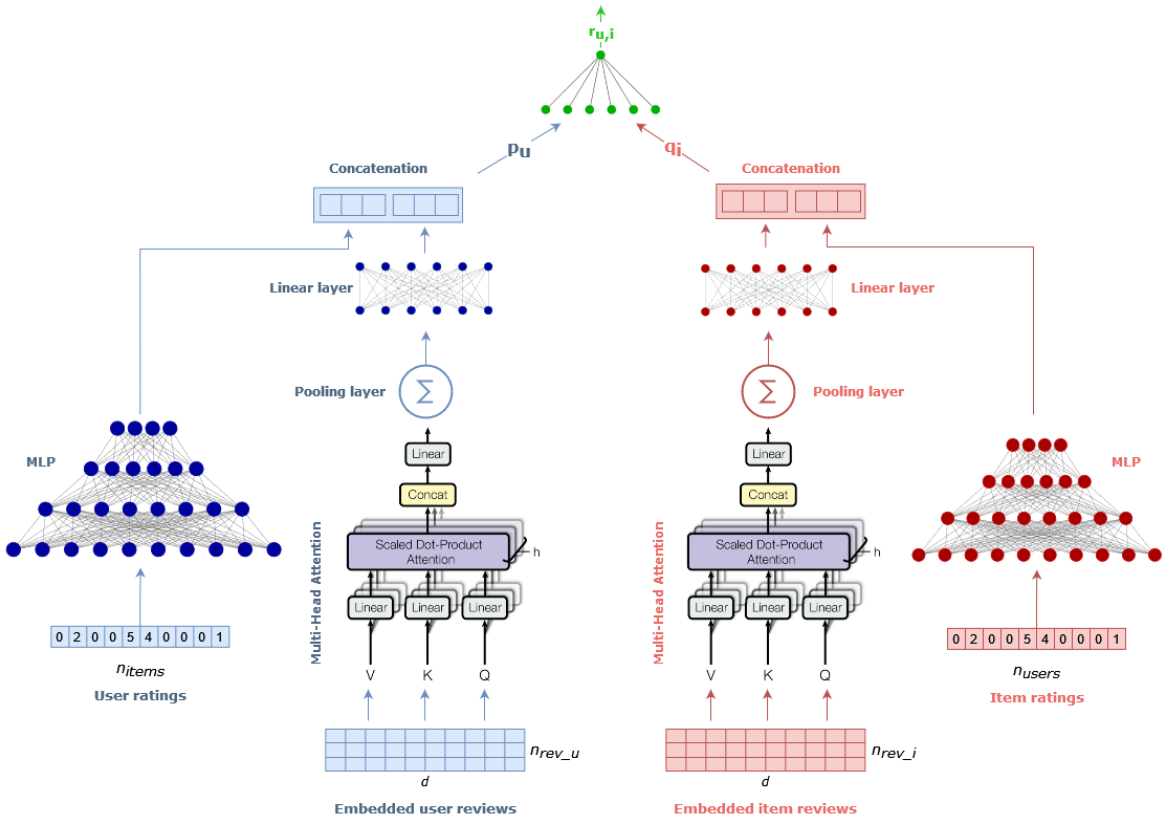


Figure 3.3: Model RE+CE architecture.

3.3. Model 3 - RE+CE with Transformer Encoder (RE+CE TE)

3.3.1. Aim of the model

The purpose of this model is to investigate the effects of a more intricate attention mechanism, increasingly similar to the Transformer's implementation which is showing to be so successful in currently existing AI models. The idea is to understand to what extent a more complex architecture can improve the quality of the produced information before potentially confusing the model due to an excessive number of parameters. Since the attention mechanism aims to map one set of embeddings to another set with the same cardinality, the Transformer's encoder block, designed for this purpose, is used in place of the multi-head attention layer.

3.3.2. Architecture

The architecture of this model is the same as the model employing collaborative filtering information merged with information obtained through self-attention (Model RE+CE), employing a more complex processing of the embedded reviews instead of a single attention layer. This processing is performed using the encoder part of the Transformer, described in Subsection 2.4.1. The encoder is a stack of n_{layers} identical sub-layers. Each sub-layer performs the operations described in the following function:

Algorithm 3.1 Encoder sub-layer

0: **function** SUB_LAYER(Q, K, V)
0: $att \leftarrow$ multiheadattention(Q, K, V)
0: $x \leftarrow$ norm$(V + att)$
0: $ff \leftarrow$ feed_forward(x)
0: $x2 \leftarrow$ norm$(ff + x)$
0: **return** $x2$

where $norm$ is layer normalization and $feed\_forward$ is a feed-forward neural network, composed of two linear layers with a ReLU[37] activation function.

The result of each sub-layer is used as Q, K, and V as input for the subsequent sub-layer. The result of the last sub-layer is then aggregated with a pooling strategy, which is the mean of all reviews. This choice was made since using the sum caused infinities during training. The rest of the model is identical to Model RE+CE. Maintaining the same structure across the rest of the model allows to identify the precise improvement — if any — provided by this one alteration. It is significant to note that this model has many more parameters and is far heavier than the preceding models. There will be a discussion on whether using such a sophisticated model is worthwhile in Chapter 4.

3.3.3. Hyperparameters

The hyperparameters are the same as Model RE+CE, with the addition of the number n_{layers} of sub-layers in the decoder and d_{ff}, which is the number of nodes in the intermediate layer of the feed-forward network.

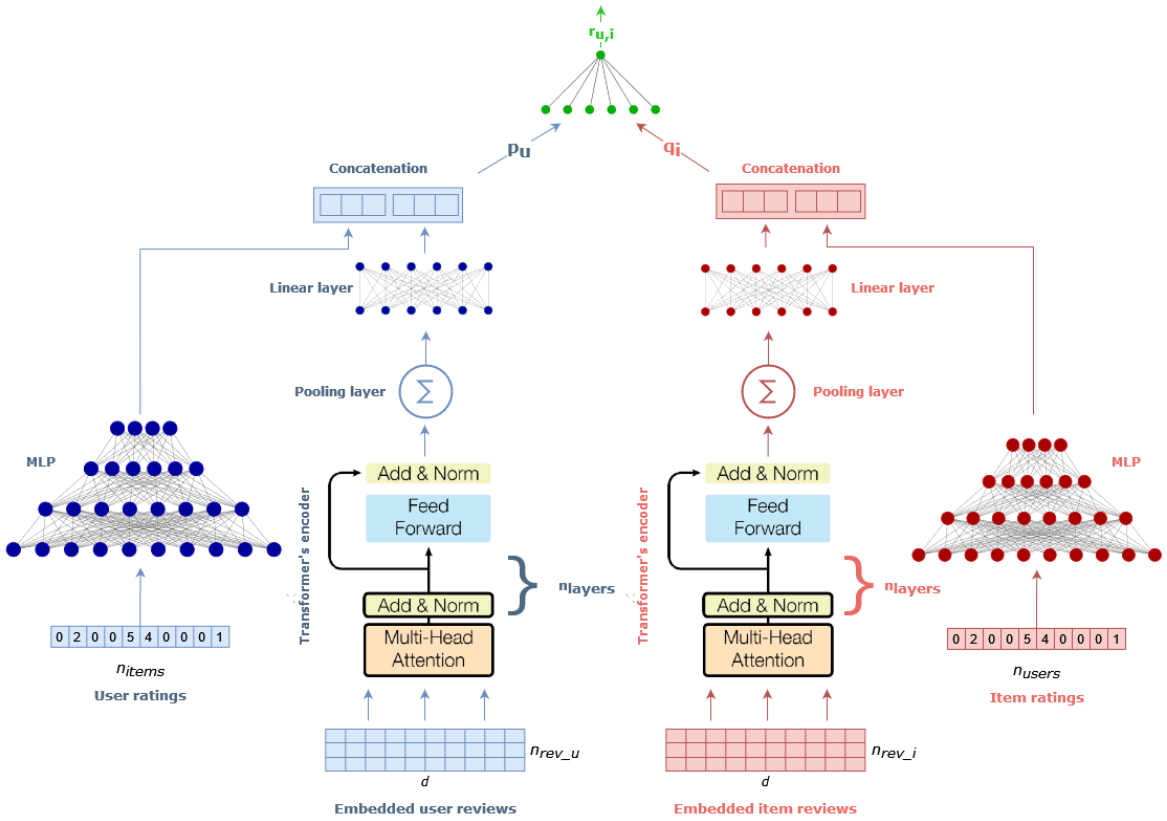


Figure 3.4: Model RE+CE TE architecture.

3.4. Model 4 - Collaborative Embeddings as Attention Queries (CE+AQ)

3.4.1. Aim of the model

This model focuses on the use of collaborative filtering information in the attention mechanism. The idea behind the model is to more closely connect the collaborative embedding and the information produced from embedded reviews. Specifically, the goal is to introduce the embedding produced by processing information obtained from interactions into the attention mechanism, to focus on reviews more consistent with the collaborative filtering profile and give them more importance. The hope is that this type of processing can yield a more significant result compared to self-attention.

3.4.2. Architecture

The structure is just like Model RE+CE, still with two separate towers for user and item that are then merged at the end, and with information from both ratings and reviews as

input for each tower.

Ratings are processed with the same collaborative MLP as Model RE+CE, while embedded reviews are processed with a different attention mechanism. In this model, embedded reviews are just used as keys and values in Equation 2.15, while collaborative filtering embeddings (the output of the collaborative MLP) are used as queries.

In order to compute the attention, queries, keys, and values must have the same dimension, so it is necessary to map embedded reviews from $d = 1536$ to $n_{factors}$, which is the dimension of the collaborative filtering embedding. Therefore, the *reviews MLP* is employed. It is an MLP with three linear layers followed by ReLU activation, mimicking the structure of the collaborative MLP.

The processing after the attention layer is the same as in Model RE+CE. The only difference lies in the dimension of the profiles p_u and q_i, since both d_{collab} and $d_{reviews}$ are equal to $n_{factors}$.

3.4.3. Hyperparameters

The hyperparameters for this model are the same as Model RE+CE, with the introduction of the number of nodes in the hidden layers for the reviews MLP.

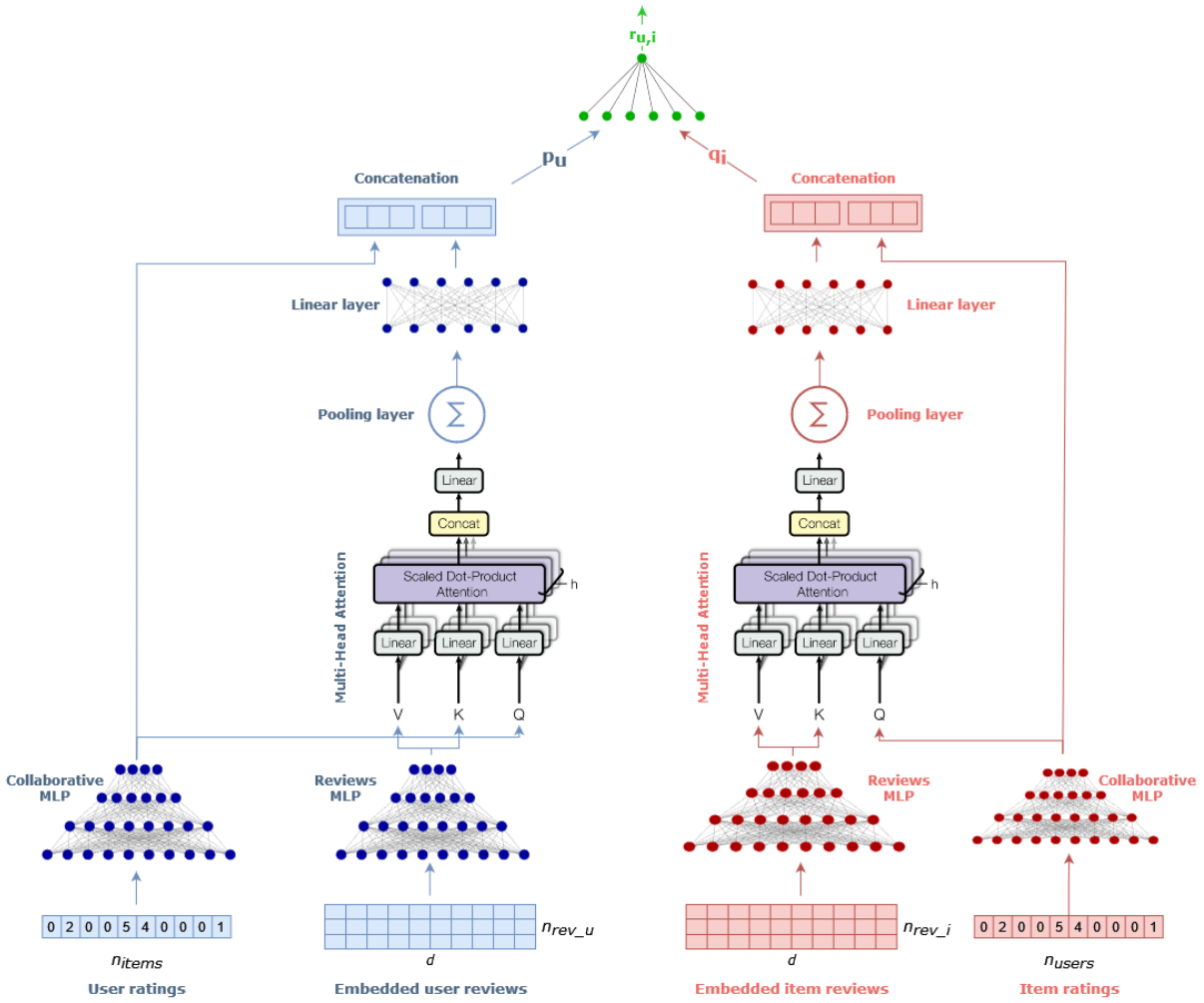


Figure 3.5: Model CE+AQ architecture.

3.5. Model 5 - CE+AQ with Transformer Encoder (CE+AQ TE)

3.5.1. Aim of the model

Model 5 employs the most complex processing of embedded reviews, combining the intuitions of the previous models. In theory, this model should be the one achieving the best results, as it combines the power of the Transformer's encoder with the rich information coming both from rating embeddings and review embeddings for the attention mechanism, where collaborative filtering embeddings are used as queries.

3.5.2. Architecture

This model has the same architecture as the model exploiting collaborative filtering embeddings inside the multi-head attention layer (Model CE+AQ), employing a more complex attention mechanism. The architecture of this model indeed uses the same encoder block described in Model RE+CE TE, with the substantial difference of using embedded reviews only as keys (K) and values (V), while using rating embeddings as queries (Q). For each sub-layer in the encoder, the result of the previous layer is used as keys and values while the same rating embedding is used as query, to ensure that its effect doesn't vanish after the first sub-layer. This is the pseudo-code for the encoder block:

Algorithm 3.2 Encoder block structure

```
layer_result = embedded_reviews
for sub_layer in encoder_layers do
    layer_result=sub_layer(rating_embedding, layer_result, layer_result)
end for=0
```

where sub_layer performs the operations described in Algorithm 3.1.

3.5.3. Hyperparameters

The hyperparameters are the same as Model RE+CE TE, with the addition of the number of nodes in the hidden layers of the MLP used to map the dimension of the embedded reviews to d_{collab}

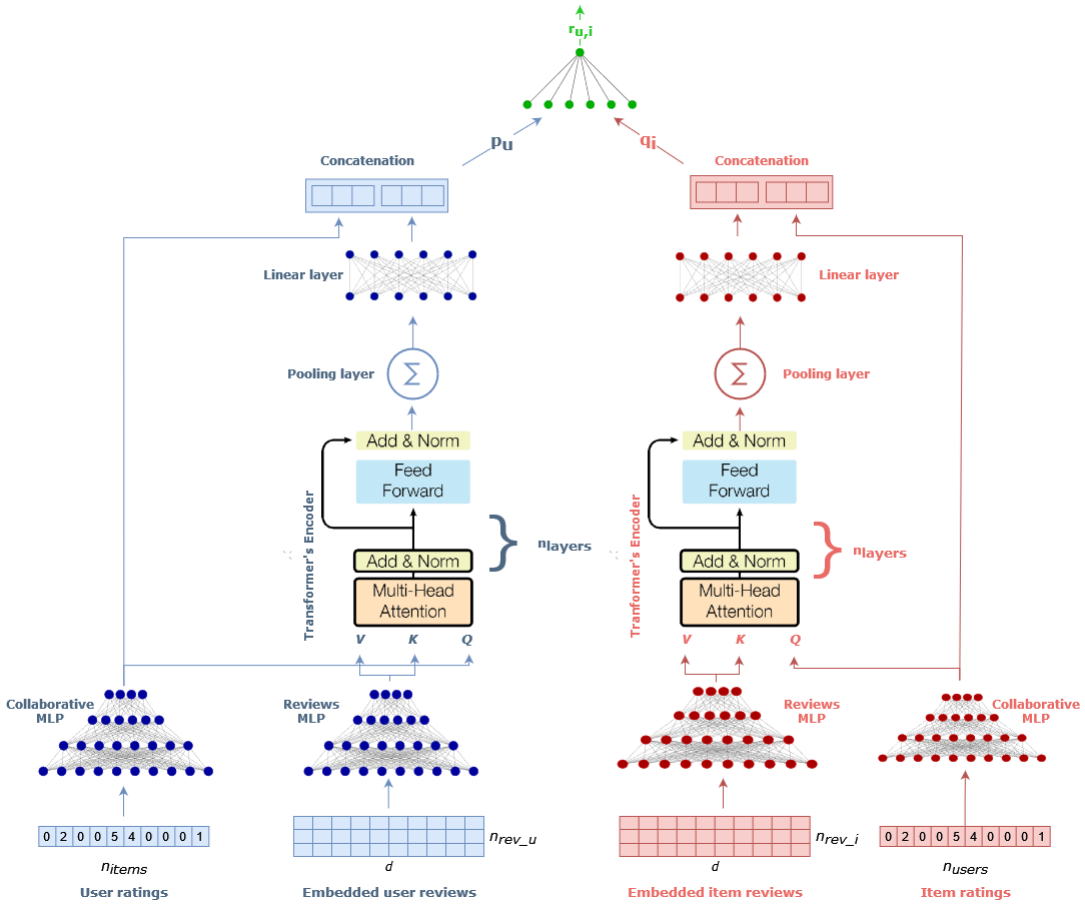


Figure 3.6: Model CE+AQ TE architecture.

3.6. Implementation

In this section, the implementation details for the presented models are described, including coding aspects and considerations on hardware resources.

3.6.1. Embeddings retrieval

One of the core aspects in this work is the use of embeddings generated by an LLM, more specifically OpenAI's text-embedding-ada-002 model[23]. This model can be accessed through OpenAI's API, which allows to send requests containing review texts and get responses with the corresponding embeddings. The API is accessed thanks to a private key, which can be generated from the personal OpenAI account. The model has a cost per token of \$0.0001 / 1K tokens, which guarantees the possibility of embedding a whole dataset with just a few dollars (more details on the specific cost for each dataset will be provided in Section 4.1.1). Each request to the model cannot exceed the maximum

token size of 8191, and there are rate limits of a maximum of 3 requests per minute and 200 requests per day. Therefore a script was created to automatically send requests until the daily limit was reached. First, the number of tokens for each review in the dataset is computed, using the *tiktoken* library, then a batch of reviews is created, sequentially appending reviews in a list and ensuring that the total number of tokens doesn't exceed the maximum size. Next, a request containing the batch of reviews is sent to the API. The embeddings contained in the response are sequentially stored in a dictionary containing the user ID, the item ID, the rating associated with the interaction and the embedding corresponding to the associated review. Finally, the dictionary is appended in a pickle file, and the script waits for 60 seconds before repeating the whole process, which is repeated 200 times each day.

3.6.2. Code details

All models are implemented using the *Neural Network* library from Pytorch[43], which is one the most popular and commonly used frameworks to build neural network models in Python. It provides pre-implemented modules that cover the majority of the most common layers used deep learning models, including most layers necessary for the implementation of the models in this thesis. To speed up the computation, batched data was given as input to the model. Batches couldn't be too large (>32) because of GPU memory constraints. Before applying multi-head attention to embedded reviews, a padding mask was necessary to account for different review numbers in batches. In fact, a batch is made of different users/items, and each user/item is associated with a different number of total reviews given/received. Therefore, the maximum number of reviews in the batch was taken as a dimension for the shape of the tensor containing the batch ([batch_size, max_num_reviews, d]). For each user/item in the batch, if the total number of associated reviews was lower than the maximum for the batch, tensors containing all zeros were added to pad each element in the batch to the same dimension. These tensors were then ignored by the attention mechanism thanks to a padding mask. Pytorch neural models are based on two methods for the training phase: *forward* and *backward*. The *forward* method defines how input data flows through the neural network, while the *backward* method, often referred to as the "backward pass," is responsible for computing gradients and facilitating backpropagation. During the forward pass, the *forward* method specifies the operations that transform input data into output. The *backward* method, on the other hand, computes gradients of the model's parameters with respect to a given loss function. This is crucial for optimizing the model using gradient-based techniques like stochastic gradient descent. Together, these methods simplify both model architecture design and

the training process.

3.6.3. Hardware resources

Since all models contain neural network components, a hardware accelerator is necessary to greatly speed up the computation. A single GPU is enough to train and evaluate all models with low computational time, needing around one minute per epoch, with small variations depending on the complexity of the model and the size of the dataset. The computation of matrices inside the attention mechanism is the part of the model that needs more memory allocation. The multiplications of parameter matrices of dimension $[d \times d]$ for each element in the batch require the batch size not to exceed 32, to guarantee that the model could fit in the GPU memory. The GPU used to train and evaluate the models and to perform hyperparameter tuning is the Nvidia GeForce RTX 3090.

One of the key factors that significantly increases the computational time when using hardware accelerators is the data transfer between the CPU and GPU. Therefore, the code was optimized to minimize the frequency of these transfers, enhancing computational efficiency. More details are provided in Subsection 3.7.1.

3.6.4. BPR loss

All models were trained to optimize the *Bayesian Personalized Ranking* (BPR) loss [46], which is often used in recommender systems to optimize ranking problems. The primary objective of BPR is to optimize the ranking of items for each user based on their historical interaction data. It is designed to model the preference of a user for one item over another by maximizing the likelihood that the user would prefer the observed positive item over a randomly sampled negative item. In this work, the focus is on the ability of models to recommend to a user new items, regardless of the ratings from past interactions, so the positive item is any item the user has interacted with, while a negative item is any item without interaction. Therefore, all the feedback in the datasets was made implicit, as the focus is solely on the presence or absence of interaction. This assumption makes perfect sense when considering the context in which these models are applied. If users purchase a product on Amazon, it's likely because they are interested in that type of product. If they are unsatisfied with the product and write a negative review, they probably still have an interest in buying a similar product, as they were not content with the first one. Similarly, if a user negatively reviews a restaurant, it may still imply that the user is likely interested in that type of restaurant and may wish to try similar ones (hopefully better). On the opposite, a vegetarian would never leave a review, even a negative one, at a steak

house.

The loss is computed through the following formula:

$$\text{BPR Loss} = -\sum_{(u,j,k)\in B} \ln(\sigma(\hat{r}_{u,j} - \hat{r}_{u,k})) + \lambda||\Theta||^2 \tag{3.2}$$

where u is the user, j is the sampled positive item and k in the sampled negative item; B is the batch; $\hat{r}_{u,j}$ and $\hat{r}_{u,k}$ are the predicted ratings for the interactions between the user and the two items; $\sigma(z)$ is the sigmoid function, defined as $\sigma(z) = \frac{1}{1+e^{-z}}$, which maps a real number 'z' to the range [0, 1]; λ is the regularization hyperparameter; $||\theta||^2$ is the L2 norm (Euclidean norm) of the model's parameter vector θ, squared.

3.7. Experimental pipeline

In this section, the whole pipeline for the models proposed is described, ranging from data preparation to how evaluation is performed to obtain the results that will be discussed in the next chapter.

3.7.1. Data preparation

All models require, as input, a tensor containing the embeddings of the reviews for both the user and the item, which are used in their respective towers. With the exception of the simplest model, all models also require information about the ratings. Therefore, starting from the user/item IDs, it is necessary to create a two-dimensional vector of size $batch\_size \times n_{items}$ (in the case of users) or $batch\_size \times n_{users}$ (in the case of items). In this vector, for each element in the batch, a value of 1 indicates interaction, while 0 indicates no interaction. Additionally, a three-dimensional vector of size $[batch\_size, max\_num\_reviews, embedding\_dim]$ needs to be created, as described in the section 3.6.2. To obtain these tensors, the following pipeline was excecuted:

1. **Embedding list:** as first step, a list containing all review embeddings was created and saved directly on the GPU memory, to guarantee that all embeddings were always already on the GPU and didn't have to be transferred from CPU to GPU for each batch. To link each interaction to the right embedding, a column was added to the dataset containing for each interaction the index of the corresponding embedding in the list.

2. **URM creation:** starting from the whole dataset, URM matrices containing rating information for training, validation, and testing were created, as further described

in sec. 4.1.2. Each URM was saved as a Compressed Sparse Row (CSR) matrix, which is a data format that guarantees memory efficiency for sparse matrices.

3. **Training dictionary:** to quickly retrieve all reviews for each user/item without scanning the whole dataset every time, two dictionaries were created (one for the users and one for the items), containing all the user/item IDs in the training set as keys, and all the indices of the corresponding review embeddings in the embedding list as values.

4. **Review and rating tensors:** finally, to train/evaluate the model, it was possible to create the previously described tensors needed as input in a very efficient way. The rating vector is just a row/column of the URM, so it is extremely fast to retrieve, while the review vector can be created directly on the GPU memory by accessing the training dictionary and retrieve the corresponding embeddings, which are already on the GPU memory.

As highlighted in the Section 3.6.3, with this pipeline the data transfer between CPU and GPU is minimized.

3.7.2. Training

The models are trained to optimize the BPR Loss, using Adam[30] as optimization algorithm. The choice of the right learning rate was crucial in the hyperparameter optimization, as "large" values ($> 10^{-3}$) led to gradient explosion with consequent poor results. Early stopping was implemented to prevent overfitting, and monitoring values of ranking metrics on the validation set. Since the computation of these metrics requires a non-negligible amount of time, it was chosen to evaluate the model on the validation set every 5 epochs, stopping the training of the model whenever the value of NDCG@10 had stopped increasing for more than 5 monitoring (so 25 epochs).

3.7.3. Evaluation procedure

All models are trained through the basic *forward* function for Pytorch models, which updates weights based on the value of the BPR loss. In addition, they implement two functions that are used for evaluation: *compute_item_info* and *predict*.

To evaluate the models, it is necessary to compute scores for all items for each user. Unlike the training phase, where the loss is calculated on a batch of users and items, during evaluation scores must be computed for a single user across all n_{items} items. In this process, the part related to item embeddings (q_i in Figure 3.1) can be computed only

once since it remains constant, while the user's embedding (p_u) needs to be computed every time. To compute q_i, the first function, *compute_item_score*, is called, which activates only the item tower of the model and returns the embedding vector q_i before it is processed alongside p_u. Since calculating q_i for all items at once would be too memory-intensive, the total number of items is divided into batches of the same size as those in training, and the function is called multiple times. The results are stacked to obtain the total q_i for all items. Once q_i is obtained, it is possible to calculate the score for each user using the *predict* function. This function takes the information of a single user as input and processes it through the user tower of the model. The result is an embedding vector for the individual user. To obtain scores for all items, it is necessary to replicate the user's p_u vector n_{items} times to match dimensions. The function then returns a vector of n_{items} scores. Once the scores are obtained, they are sorted in descending order, and the top-k scores are selected and compared with the interactions of each user present in the test set, to calculate ranking metrics. This evaluation process allows for the assessment of how well the model's recommendations align with the actual interactions of users, providing insights into the model's performance and its ability to make relevant recommendations.

3.8. Content-based models

After the complete description of the 5 complex models that employ neural techniques with the same intuitions of state-of-the-art methods like NARRE[10] and HRDR[33], in this section two more models are introduced, which substantially differ from the previous ones. These models are based on simple content-based techniques, with the specific purpose of simply understanding the quality of the embeddings generated by the LLM.

3.8.1. Model 6 - Content-based with review embeddings (CB-KNN)

This model is a simple KNN content-based recommender (see Section 2.2.2), where for each item, the mean of the corresponding embedded reviews is taken as features. Therefore, the ICM is a matrix of shape $[n_{items} \times d]$, where each feature is the mean value in that dimension of the embedding space. The similarity between items is then computed using adjusted cosine similarity, which is a variation of cosine similarity described in Equation 2.1, where the deviation from the average embedding is considered when computing the

similarity between two items, through the formula:

$$\text{Adjusted Cosine Similarity}(i,j) = \frac{\sum_k (E_{ik} - \bar{E}_i)(E_{jk} - \bar{E}_j)}{\sqrt{\sum_k (E_{ik} - \bar{E}_i)^2 \cdot \sum_k (E_{jk} - \bar{E}_j)^2}} \tag{3.3}$$

Where E_{ik} and E_{jk} are the values of the embeddings for items i and j at dimension k, and $\bar{E}_i$ and $\bar{E}_j$ are the average embeddings for items i and j respectively. Hyperparameters for this model are the k value for the k-nearest neighbors, the shrink term described in Equation 2.1, and whether or not to normalize the similarity (as in Equation 2.1).

3.8.2. Model 7 - Hybrid ItemKNN with CF and content-based (CFCB-KNN)

This model aims at understanding whether content-based information can enrich a simple collaborative filtering technique like ItemKNN. In item-item collaborative filtering, the similarity is computed among columns of the URM, to understand how similar are items based on the interaction information. In content-based methods, the similarity is computed among rows of the ICM, to understand how similar are items based on their features. This model aims at combining these two sources of information. To do so, the URM is transposed and stacked horizontally together with the ICM, to obtain a new ICM having as columns not only features information (the d average values of the review embeddings), but also interaction information (users of the URM). Therefore, if the ICM has dimension $[n_{items} \times d]$ and the URM has dimension $[n_{users} \times n_{items}]$, the new matrix used for this model will have dimension $[n_{items} \times (d + n_{users})]$. The similarity is computed on this new matrix, just as described in Model CB-KNN. The hyperparameters are the same as Model CB-KNN, plus a weight to give to elements of the ICM to give less or more importance with respect to interaction data.

3.9. Implementation details for KNN models

Models CB-KNN and CFCB-KNN are just content-based methods with a simple similarity to be computed, without any neural technique that requires a training phase to upgrade weights. The input for Model CB-KNN is a simple ICM where each row is obtained by averaging the embedded reviews associated with each item, while for Model CFCB-KNN the ICM is stacked horizontally with the URM obtained from the dataset. To evaluate the models, each row in training URM is multiplied by the similarity matrix computed by the models through a dot product, to obtain a vector of scores associated with items. This vector is sorted in descending order, and the top-k items are taken to compute ranking

metrics, by comparing them with seen items in the validation/test URM.

4 | Results

In this chapter, a discussion on the results obtained by different models is provided, highlighting performances against the chosen baselines.

4.1. Data

To train and evaluate the models, data containing both reviews and ratings are required. Not many publicly available datasets contain both types of information, as the majority of users are already reluctant to rate products, so it is even more uncommon to find contexts where users are eager to write reviews. Therefore, most of the data containing reviews is too sparse to be effectively utilized by recommender systems, and only very few datasets are genuinely useful. A piece of evidence is the fact that basically all the methods developed in recent years that leverage reviews for recommendations use the same datasets. Consequently, the methods presented in this thesis also rely on these common datasets.

4.1.1. Datasets

In this section, the three datasets chosen to train and evaluate the models are accurately described. They are all available for free and easy to download, as data are stored in json format. All datasets are 5-core versions, which guarantees that there are at least 5 interactions for each item and for each user. This characteristic enables models to have enough information to build a meaningful user/item profile.

Amazon Music

The first dataset is the Digital Music category from Amazon Reviews Dataset[25]. This dataset is an extensive collection of customer reviews and ratings for products available on the Amazon platform. It includes unstructured textual reviews, numerical ratings, and associated metadata, like the helpfulness of each review or the summary, as well as additional product data (descriptions, category information, price, brand, and image fea-

tures). It was created by Julian McAuley at the University of California, San Diego. The presented models only need four features from the data: the user ID (called *reviewerID* in the dataset), the item ID (called *asin*), the textual review (called *reviewText*) and the rating (called *overall*). The data is saved in a json file; this is an example of an entry:


```
{
  "reviewerID": "A2SUAM1J3GNN3B",
  "asin": "0000013714",
  "reviewerName": "J. McDonald",
  "helpful": [2, 3],
  "reviewText": "I bought this for my husband who plays the piano.  He is having a..."
  "overall": 5.0,
  "summary": "Heavenly Highway Hymns",
  "unixReviewTime": 1252800000,
  "reviewTime": "09 13, 2009"
}
```


A built-in version of this dataset is available in the Cornac library [48] and was hence used. It refers to the 5-core version from 2014, including data collected from May 1996 to July 2014. The dataset is composed of a total of 64706 interactions, with 5541 unique users, 3568 unique items and a sparsity of around 99.67%. The total number of tokens is 17208197, therefore the cost to embed the whole dataset, as described in Section 3.6.1, is roughly 1.72$.

Amazon Toys and Games

The second dataset is the Toys and Games from the Amazon Reviews Dataset. The description is the same as the previous dataset, with the only difference in the category of reviewed products. Many previous works ([10], [33]) use different categories of products from the Amazon Reviews Dataset, as it represents the largest and richest source of information with reviews currently available. Using different categories of the same dataset is totally reasonable, as reviews have completely different topics and it is likely that the user demographics for these two product categories are substantially different. The dataset contains a total of 167597 interactions, with 19412 unique users and 11924 unique items, for a data sparsity of 99.93%. The total number of tokens is 20479777, therefore the whole dataset can be embedded with a cost of around 2.05$.

Yelp

The third dataset used is the Yelp Reviews Dataset[2]. This dataset is a comprehensive collection of user-generated reviews and ratings from the Yelp platform, covering a wide range of businesses, such as restaurants, bars, and retail establishments. This dataset includes textual reviews, star ratings, and lots of metadata about the businesses and users. It comprises different json file, but the information used by the models is all inside the *review.json* file, where each query is as follows:

```
{
// string, 22 character unique review id
"review_id": "zdSx_SD6obEhz9VrW9uAWA",

// string, 22 character unique user id, maps to the user in user.json
"user_id": "Ha3iJu77CxlrFm-vQRs_8g",

// string, 22 character business id, maps to business in business.json
"business_id": "tnhfDv5Il8EaGSXZGiuQGg",

// integer, star rating
"stars": 4,

// string, date formatted YYYY-MM-DD
"date": "2016-03-09",

// string, the review itself
"text": "Great place to hang out after work: the prices are decent, and the amb

// integer, number of useful votes received
"useful": 0,

// integer, number of funny votes received
"funny": 0,

// integer, number of cool votes received
"cool": 0
}
```

The only features used are: *user_id, business_id, text* and *stars*. This dataset is much

larger than the previous ones, including almost 7 million interactions and more than 150000 businesses. This amount of data is too large to be processed by the models, mainly due to hardware constraints (size limit of the GPU memory) and massive expected computational time, therefore a sub-sampling was performed to obtain a dataset size comparable to the other two datasets used. Since the dimension of the sub-sampled dataset is expected to be approximately 1% of the original dimension, it is impossible to ensure that the data distribution is preserved. Therefore, it was decided to perform a subsampling that not only reduces the dataset's size but also transforms it into a 5-core dataset, aligning it with the characteristics of the other two datasets employed. The subsampling was composed of different steps:

- First, the json file was loaded into a Pandas dataframe and a random subsampling of rows was performed. Several attempts were made to find the best percentage of interactions to take in the subsampling, with 18% being the value that led to the best final size and that was hence chosen.
- Then, a graph was built from the subsampled data, using the *networkx* library. In the graph, user ids and item ids are used as nodes, and each interaction between a user and an item is an edge of the graph, carrying as information the rating value and the review associated to the interaction.
- From this graph, the 5-core subgraph[16] was extracted. This subgraph includes only nodes (both users and items) that have at least 5 interactions.
- Extracting the 5-core subgraph doesn't guarantee that all nodes in the subgraph will still have at least 5 connections, so the subgraph was transformed again in a Pandas dataframe and from it users and items with less than 5 interactions were iteratively removed, according to the following algorithm:

Algorithm 4.1 Iterative removal of users and items

```
u_less_5 = get user ids with less than 5 interactions
i_less_5 = get item ids with less than 5 interactions
while u_less_5 ≠ 0 and i_less_5 ≠ 0 do
    df = df[user ids with at least 5 interactions]
    df = df[item ids with at least 5 interactions]
    u_less_5 = get user ids with less than 5 interactions
    i_less_5 = get item ids with less than 5 interactions
end while=0
```

After the preprocessing, the new dataset obtained has a total dimension of 83505 in-

teractions, with 9224 unique users and 9093 unique businesses. The average number of interactions per user is 9.05, the average number of interactions per item is 9.18, for a data sparsity of 99.90%. The total number of tokens in the whole dataset is 15321800, thus requiring around 1.53$ to be entirely embedded.

4.1.2. Data splitting

In the process of training, evaluating, and testing the models, the dataset was divided into three distinct sets with a predetermined ratio of interactions of 80% for training, 10% for evaluation, and 10% for testing. The randomization of this split was carried out to prevent any inherent biases in the data distribution. However, a critical consideration in this randomization process was the necessity to ensure that each user and each item had at least one interaction included in the training set. This choice was made to guarantee that the trained models could effectively process information related to every user and item present in the evaluation and test sets. By having this foundational interaction data, the models are equipped to make informed predictions and recommendations for all users and items in the dataset, minimizing the potential challenges of dealing with entirely unseen users or items during the evaluation and testing phases. To ensure consistency among results, the split was performed just once for each dataset, storing data in files that were used by all models.

4.2. Baselines

To compare the performance of the proposed models, 5 models have been chosen as baselines:

- **HFT, NARRE, HRDR:** these methods, presented in Section 2.3, are the current state-of-the-art models that leverage user reviews to make recommendations.
- **ItemKNN, RP3Beta:** as described in Section 2.2.3, these are popular collaborating filtering techniques that lead to excellent results in many contexts.

The first group of methods was implemented using the Cornac library[48], which is one of the frameworks recommended by ACM RecSys 2023[3] for the evaluation and reproducibility of recommendation algorithms. This library provides the implementation of many popular recommender systems, including the three mentioned models, as well as a complete pipeline for data loading and splitting and model evaluation. The pipeline is based on three objects:

- *ReviewModality*: is the object designed to incorporate textual reviews as part of the

recommendation process. It takes the reviews as input and applies different preprocessing steps to create a vocabulary of words from the reviews. First, reviews are tokenized, with different possible tokenization choices. The base English tokenizer was employed for the baseline models, which excludes common English stop words, such as "the," "and," and "is," which are often considered noise in natural language processing tasks. Then, words are filtered based on two parameters: the maximum vocabulary size m, which allows only the m most frequent words in the reviews to be considered during analysis; and the maximum document frequency f, which guarantees that words appearing in more than $f\%$ of the reviews are excluded, to remove very common words that may not carry much discriminatory information. f and n were both hyperparameters to be tuned.

- *RatioSplit*: is the object that takes as input both interaction and review data (the *ReviewModality* object) and splits it to obtain training, validation, and testing sets to be used in the experiments.
- *Experiment*: is the object performing training and evaluation of the model(s). It takes as input the *RatioSplit* object, one or more models to be trained and evaluated on the input data, and the metrics to be used to evaluate the model(s). The object outputs the results of each model both on the validation and on the test set in all metrics.

Hyperparameters tuning was performed on all models in order to find the best sets of hyperparameters to maximize NDCG@10, as described in Section 4.3. The collaborative filtering methods were implemented using the official repository of the Recommender Systems course at Politecnico di Milano[19], which provides the implementation of many recommendation algorithms, including these two. As pure collaborative filtering techniques, these two methods only require the URM as input data. The URM is stored as a Compressed Sparse Row (CSR) matrix, which is split into training, validation and testing URMs. During training, both models compute the similarity matrix of dimension $[n_{items} \times n_{items}]$, which is used during evaluation. Hyperparameters tuning was performed on these models as well, maximizing NDCG@10 as described in Section 4.3. Since the models presented focus on the impact of LLMs in recommender systems based on reviews, one would expect to include recommender systems based on LLMs among the baselines. However, as outlined in Section 2.4.4, currently, all LLM-based methods are primarily prompt engineering techniques that leverage interactions with models such as OpenAI's GPTs to provide specific recommendations for individual users, given the provided context. However, so far none of these methods is yet really applicable to large-scale datasets like those used to evaluate the methods in this study, as the number of queries to the

LLM would be excessive, and automating the process for such intricate data would be too challenging. Moreover, these approaches frequently emphasize their efficacy in situations like the cold-start problem, zero-shot recommendation, or recommendation explanation rather than predicting top-k suggestions. Consequently, it was decided not to implement any LLM-based method as a baseline. Still, these approaches have a potentially promising future, and it is likely that even these kinds of procedures will soon be able to be used as baselines.

4.3. Hyperparameter tuning

Hyperparameter tuning is a crucial aspect when using deep learning models, as it strongly influences the performance and effectiveness of recommendation algorithms. All presented models involve a multitude of hyperparameters, such as learning rates, batch sizes, and network architectures, which need to be optimized to achieve the best results. Optuna[5], a popular Python library for hyperparameter optimization, was used to perform hyperparameter tuning. Optuna is an open-source hyperparameter optimization framework that uses a Bayesian optimization approach to efficiently search the hyperparameter space and identify the optimal configuration for a given deep learning model. It adopts state-of-the-art algorithms for sampling hyperparameters and pruning unpromising trials. This helps to speed up optimization time and performance greatly compared to traditional methods such as grid-search or random-search. Optuna relies on the TPESampler, or Tree-Structured Parzen Estimator, which is a Bayesian optimization algorithm that iteratively selects, evaluates, and refines hyperparameters to find the best configuration for the given model. It does so by randomly selecting a subset of hyperparameters and sorting them based on their performance scores. These hyperparameters are divided into two groups and modeled using Parzen Estimators[40] to estimate their densities. The algorithm then identifies the hyperparameters with the highest expected improvement, evaluates them, and repeats the process until a predefined budget is exhausted. Ultimately, the TPESampler returns the best hyperparameters for the task. In Python, Optuna is based on the *study* object, which allows to perform hyperparameter optimization on a predefined set of hyperparameters for a given model. The key element of a study is the objective function, which is used to evaluate and score the performance of a particular set of hyperparameters for a given model. The goal is to find the set of hyperparameters that maximizes or minimizes the objective function, depending on whether it represents a maximization or minimization problem (minimization if the value to optimize is a regression metric, maximization if it is a ranking metric). When defining a study, the number of trials to perform must be specified. A *trial* represents a single execution of

the objective function with a specific set of hyperparameters. Optuna performs a series of trials, each with a different combination of hyperparameter values, to systematically search for the best configuration. The objective function is called for each trial, and it returns a score that quantifies the performance of the model with those hyperparameters. Optuna then uses the results of these trials to decide how to explore the hyperparameter space more effectively in subsequent trials. For each of the presented models, 50 trials of optimization were performed in order to find the best hyperparameter configuration. In addition to the model-specific hyperparameters, all neural models shared three additional hyperparameters: the *batch size* for the batches of samples in the training and evaluation phase, the *learning rate* for the Adam optimizer and the λ parameter in the loss function. Depending on the dataset, the *batch size* may have a maximum value that should not be exceeded. If it surpasses this limit, it can lead to the model not fitting within the GPU's memory capacity, typically not exceeding 24GB. The last two hyperparameters are especially crucial because incorrect values, particularly when set too high, can result in gradient explosions and substantially degrade the model's performance.

4.4. Baselines results

Before analyzing the results of different models against the baselines, the results of all baselines are described in this section, divided for each dataset.

Amazon Music				
	Precision@10	**Recall@10**	**MAP@10**	**NDCG@10**
RP3Beta	**0.0337**	**0.2652**	**0.1241**	**0.1629**
ItemCFKNN	0.0304	0.2394	0.1156	0.1500
HFT	0.0024	0.0071	0.0048	0.0047
NARRE	0.0004	0.0011	0.0021	0.0008
HRDR	0.0055	0.0194	0.0092	0.0113

Table 4.1: Results on Amazon Music dataset

Amazon Toys				
	Precision@10	**Recall@10**	**MAP@10**	**NDCG@10**
RP3Beta	**0.0137**	**0.1251**	**0.0636**	**0.0791**
ItemCFKNN	0.0129	0.1180	0.0602	0.0748
HFT	0.0003	0.0013	0.0011	0.0008
NARRE	0.0002	0.0009	0.0009	0.0005
HRDR	0.0001	0.0005	0.0008	0.0004

Table 4.2: Results on Amazon Toys dataset

Yelp				
	Precision@10	**Recall@10**	**MAP@10**	**NDCG@10**
RP3Beta	**0.0062**	**0.0569**	**0.0214**	**0.0301**
ItemCFKNN	0.0060	0.0548	0.0209	0.0286
HFT	0.0003	0.0015	0.0013	0.0008
NARRE	0.0003	0.0014	0.0012	0.0008
HRDR	0.0002	0.0011	0.0014	0.0008

Table 4.3: Results on Yelp dataset

Tables 4.1, 4.2 and 4.3 clearly highlight an interesting result: state-of-the-art methods leveraging user reviews obtain performances that are by far worse than collaborative filtering techniques, even simple ones like ItemKNN. It is important to remember that HFT, HRDR and NARRE are explicitly trained to minimize the error between the real and the predicted rating, so they aim at optimizing regression metrics rather than ranking metrics, which are instead used to compare models in this thesis. This significant difference in results clearly demonstrates that these methods are not suitable for ranking recommendations, highlighting the current lack of appropriate methods based on reviews that can actually bring effective results in real-world applications. This evidence therefore justifies the increasing lack of interest by researchers in this kind of methods. Moreover, it is possible to observe the significant difference in performance among the datasets: the top-performing baseline models obtain neatly better results on the Amazon Music dataset, with values for all metrics being five times higher than the ones on the Yelp dataset; while results on the Amazon Toys dataset lie in between the other two. These results can be explained by various differences in the data distribution among datasets,

as well as characteristics on the items inside the datasets. It is likely more difficult to understand the similarity among businesses in the Yelp dataset (both using review-based and collaborative filtering information) rather than among products on Amazon.

4.5. Results Model RE

In this section, results for the first model are described. Model RE leverages only embedded reviews, without considering any kind of collaborative filtering information, being the simplest neural model among the ones proposed. The results on all datasets against the baselines are shown in Table 4.4.

	Music		Toys		Yelp	
	Prec	**NDCG**	**Prec**	**NDCG**	**Prec**	**NDCG**
RP3Beta	**0.0337**	**0.1629**	**0.0137**	**0.0791**	**0.0062**	**0.0301**
ItemCFKNN	0.0304	0.1500	0.0129	0.0748	0.0060	0.0286
Model RE	0.0112	0.0459	0.0030	0.0143	0.0027	0.0130
HFT	0.0024	0.0047	0.0003	0.0008	0.0003	0.0008
NARRE	0.0004	0.0008	0.0002	0.0005	0.0003	0.0008
HRDR	0.0055	0.0113	0.0001	0.0004	0.0002	0.0008

Table 4.4: Precision@10 and NDCG@10 of Model RE against the baselines

The results of Model RE already yield interesting conclusions. Firstly, despite its simplicity, the model outperforms review-based baselines significantly, with an NDCG value four times higher than the best baseline model (HRDR). This surprising outcome leads to two conclusions: embeddings generated by the LLM are meaningful and can help to achieve satisfactory results, and creating an embedding for each review, rather than each word in the dictionary, yields higher-quality outcomes. As expected, on the other hand, compared to collaborative filtering baselines this model struggles to achieve comparable results. The sole information from reviews helps to construct a user and item profile, but it fails to capture enough information on user-item interactions to provide quality recommendations. The detailed results for all metrics are shown in Section 4.12.

This model is also the simplest in terms of hyperparameters, which are quite similar for all datasets. After hyperparameter tuning (described in Section 4.3), for all models, the best pooling strategy turns out to be the sum and the best number of heads is 2. The other hyperparameters are reported in Table 4.5.

	Amazon Music	Amazon Toys	Yelp
Dropout rate	0.4	0.21	0.4
Learning rate	1.9e-5	1.9e-5	1.3e-4
Batch size	6	19	20
λ_{reg}	4e-6	1.2e-4	7.1e-6

Table 4.5: Model RE hyperparameters

4.6. Results Model RE+CE

Model RE+CE extends Model RE introducing collaborative filtering information from interactions between users and items. Therefore, the results are expected to be better than those from Model RE. This hypothesis is indeed confirmed, as shown in Table 4.6.

	Music		Toys		Yelp	
	Prec	**NDCG**	**Prec**	**NDCG**	**Prec**	**NDCG**
RP3Beta	**0.0337**	**0.1629**	**0.0137**	**0.0791**	**0.0062**	**0.0301**
ItemCFKNN	0.0304	0.1500	0.0129	0.0748	0.0060	0.0286
Model RE+CE	0.0248	0.1191	0.0072	0.0397	0.0060	0.0280
HFT	0.0024	0.0047	0.0003	0.0008	0.0003	0.0008
NARRE	0.0004	0.0008	0.0002	0.0005	0.0003	0.0008
HRDR	0.0055	0.0113	0.0001	0.0004	0.0002	0.0008

Table 4.6: Precision@10 and NDCG@10 of Model RE+CE against the baselines

Collaborative filtering shows an impressive contribution to the model, as results are greatly enhanced in all datasets. In fact, all metric values are more than doubled. The performance, however, still falls short of collaborative filtering baselines, even though the NDCG value obtained in the Yelp dataset is very close to that of ItemKNN. This finding demonstrates that processing information obtained from interactions using neural techniques is not as effective as the use of simpler algorithms that do not employ any learning mechanisms, as already evidenced in [18]. The knowledge gained through the processing of embedded reviews fails to enrich collaborative filtering information to the extent of surpassing the baselines.

In contrast, compared to review-based baselines, this model also highlights significantly

improved results, confirming the conclusions drawn from the previous model. The detailed results for all metrics are shown in Section 4.12.

In addition to the hyperparameters already present in Model RE, this model introduces new hyperparameters mainly related to the architecture of the MLP processing the ratings. These include the number of nodes in the first layer of the MLP in the user tower ($n\_user\_mlp\_factors$), those in the first layer of the MLP in the item tower ($n\_item\_mlp\_factors$), and the final dimension $n_{factors}$. Hyperparameters are significantly different between datasets, confirming the differences evidenced by the results. It is interesting to notice the large values in the MLP architecture, meaning that it is possible to obtain a high-dimensional latent representation of the interaction pattern. The full list of hyperparameter values is provided in Table 4.7.

	Amazon Music	**Amazon Toys**	**Yelp**
Dropout rate	0.61	0.35	0.37
Learning rate	6.3e-7	2e-6	5.8e-5
Batch size	5	5	15
λ_{reg}	4.5e-4	2.4e-6	0.01
$n_{factors}$	1475	1483	348
n_user_mlp_factors	2951	2967	949
n_item_mlp_factors	2951	2967	884
n_{heads}	2	2	4
Pooling	sum	mean	sum

Table 4.7: Model RE+CE hyperparameters

4.7. Results Model RE+CE TE

Model RE+CE TE has the same structure as Model RE+CE, with the only difference in the way embedded reviews are processed: instead of a single multi-head attention layer, the encoder block of the Transformer is employed. The aim of this model is to understand whether adding more complexity in the way embeddings are interpreted by the model can lead to an improvement in performance or not. Therefore, the model has been evaluated on all datasets. The results are shown in Table 4.8.

	Music		Toys		Yelp	
	Prec	**NDCG**	**Prec**	**NDCG**	**Prec**	**NDCG**
RP3Beta	**0.0337**	**0.1629**	**0.0137**	**0.0791**	0.0062	0.0301
ItemCFKNN	0.0304	0.1500	0.0129	0.0748	0.0060	0.0286
Model RE+CE TE	0.0218	0.1035	0.0069	0.0375	**0.0071**	**0.0346**
HFT	0.0024	0.0047	0.0003	0.0008	0.0003	0.0008
NARRE	0.0004	0.0008	0.0002	0.0005	0.0003	0.0008
HRDR	0.0055	0.0113	0.0001	0.0004	0.0002	0.0008

Table 4.8: Precision@10 and NDCG@10 of Model RE+CE TE against the baselines

The results lead to interesting considerations. While for the first two models results were coherent among all datasets, this is the first model where a discrepancy in performance between datasets is evidenced. Specifically, the outcomes diverge between the Amazon datasets and the Yelp dataset. In the case of Amazon, the increased complexity of Model RE+CE TE does not yield better results than Model RE+CE, for both Amazon Music and Amazon Toys. Conversely, for the Yelp dataset, the results are significantly superior to those of Model RE+CE. Moreover, while all other models failed to surpass collaborative filtering baselines on all datasets, this model on the Yelp dataset is the first to obtain the best metric values among all models considered, including popular collaborative filtering methods like RP3Beta. These results allow for reasoning on two aspects. First, differences in datasets translate into strong differences in results when comparing different models; in particular, more complex processing of the LLM-produced embeddings results in degraded performances on the Amazon datasets while it benefits the results on the Yelp dataset. Second, this model is the first to fully confirm the hypothesis underlying this thesis. In fact, this model is able (under certain data characteristics) to overcome not only review-based baselines (which are consistently outperformed across all datasets) but also to outperform collaborative filtering methods like RP3Beta, indicating a potential advancement in the state-of-the-art. The detailed results for all metrics are shown in Section 4.12.

In addition to the hyperparameters in Model RE+CE, two new hyperparameters are introduced for this model: the number of sub-layers inside the encoder block, and the number of nodes d_{ff} in the hidden layer of the feed-forward neural network in each sub-layer. All hyperparameters are provided in Table 4.9.

	Amazon Music	Amazon Toys	Yelp
Dropout rate	0.5	0.69	0.2
Learning rate	7.4e-6	2.7e-6	3.6e-6
Batch size	17	14	5
λ_{reg}	1e-4	8.1e-4	4.8e-4
$n_{factors}$	727	1361	1197
n_user_mlp_factors	1478	2723	2395
n_item_mlp_factors	1486	2723	2395
n_{heads}	2	1	6
d_{ff}	984	4254	2822
num_layers	6	5	2

Table 4.9: Model RE+CE TE hyperparameters

4.8. Results Model CE+AQ

Model CE+AQ focuses on the introduction of the rating pattern inside the attention mechanism, to evaluate whether it can carry more information rather than using self-attention. Therefore, the output of the collaborative MLP used to process the ratings is used as query in the multi-head attention layer. The trade-off for this new attention mechanism is a compression of the dimension of the embeddings produced by the LLM. The results on all datasets are shown in Table 4.10.

	Music		Toys		Yelp	
	Prec	**NDCG**	**Prec**	**NDCG**	**Prec**	**NDCG**
RP3Beta	**0.0337**	**0.1629**	**0.0137**	**0.0791**	0.0062	0.0301
ItemCFKNN	0.0304	0.1500	0.0129	0.0748	0.0060	0.0286
Model CE+AQ	0.0197	0.0921	0.0059	0.0326	**0.0066**	**0.0316**
HFT	0.0024	0.0047	0.0003	0.0008	0.0003	0.0008
NARRE	0.0004	0.0008	0.0002	0.0005	0.0003	0.0008
HRDR	0.0055	0.0113	0.0001	0.0004	0.0002	0.0008

Table 4.10: Precision@10 and NDCG@10 of Model CE+AQ against the baselines

The results lead to interesting considerations. The most noticeable observation is the dis-

crepancy in results across datasets. Specifically, once again the outcomes diverge between the Amazon datasets and the Yelp dataset. Indeed, for the Amazon datasets (Music and Toys) the results from this model are worse than those from Model RE+CE, highlighting that self-attention provides a more accurate processing on embedded reviews rather than introducing external information, and that the compression of the dimension of embedded reviews can degrade the performance. Therefore, this model is not to be preferred when compared with Model RE+CE. On the other hand, on the Yelp dataset, results from this model are better than those from Model RE+CE, even outperforming collaborative filtering methods like RP3Beta, highlighting a great benefit from embedding compression and collaborative information in the attention. It can be concluded once again that distinct dataset characteristics, despite similar sparsity and number of interactions, translate into different outcomes among models. Nevertheless, all results are satisfying and markedly outperform review-based baselines, just as all previous models. In one case, the performance even surpasses effective collaborative filtering techniques like RP3Beta, representing a tangible enhancement of the current state-of-the-art. The detailed results for all metrics are shown in Section 4.12.

The hyperparameters for this model are the same as for Model RE+CE, with the addition of the number of nodes of the MLP used to reduce the dimension of the embedded reviews (*embedding_mlp_factors*). All hyperparameters are listed in Table 4.11.

	Amazon Music	**Amazon Toys**	**Yelp**
Dropout rate	0.56	0.4	0.2
Learning rate	2e-6	2.5e-6	3e-6
Batch size	6	13	4
λ_{reg}	4.6e-6	6.8e-5	0.0095
$n_{factors}$	609	552	664
n_user_mlp_factors	1335	1265	1384
n_item_mlp_factors	1391	1374	1398
embedding_mlp_factors	1485	1395	1390
n_{heads}	1	2	1
Pooling	mean	sum	mean

Table 4.11: Model CE+AQ hyperparameters

4.9. Results Model CE+AQ TE

Model CE+AQ TE has the same structure as Model CE+AQ, employing the same attention mechanism as Model RE+CE TE. Hence, embedded reviews are not processed through a single multi-head attention layer but are passed through the encoder block of the Transformer. For each sub-layer in the block, the output of the previous layer is used as key and value to compute attention, while the output of the MLP processing ratings is used as query. Results on all datasets are shown in Table 4.12.

	Music		Toys		Yelp	
	Prec	**NDCG**	**Prec**	**NDCG**	**Prec**	**NDCG**
RP3Beta	**0.0337**	**0.1629**	**0.0137**	**0.0791**	0.0062	0.0301
ItemCFKNN	0.0304	0.1500	0.0129	0.0748	0.0060	0.0286
Model CE+AQ TE	0.0211	0.0991	0.0072	0.0399	**0.0065**	**0.0323**
HFT	0.0024	0.0047	0.0003	0.0008	0.0003	0.0008
NARRE	0.0004	0.0008	0.0002	0.0005	0.0003	0.0008
HRDR	0.0055	0.0113	0.0001	0.0004	0.0002	0.0008

Table 4.12: Precision@10 and NDCG@10 of Model CE+AQ TE against the baselines

Results on all datasets highlight an interesting evidence: this model is able to perform better than Model CE+AQ, meaning that a more complex processing allows to better incorporate the collaborative filtering information inside the attention mechanism rather than employing just a single multi-head attention layer. On all datasets, results are comparable with those of Model RE+CE TE, demonstrating coherence when using more complex architectures. As in all previous models, this model consistently outperforms review-based baselines. Coherently with previous models, there is a discrepancy among datasets for this model as well, which is able to outperform also collaborative filtering baselines on the Yelp dataset, while it fails to reach comparable performances on the Amazon datasets. The detailed results for all metrics are shown in Section 4.12.

The hyperparameters are the same as Model RE+CE TE, with the addition of *embedding_mlp_factors* from Model CE+AQ. Given the strong similarity between models, it was chosen to use the same hyperparameters as Model RE+CE TE, with the value of *embedding_mlp_factors* from Model CE+AQ.

4.10. Results Model CB-KNN

Model CB-KNN is the simplest model to be tested on all datasets, to assess the pure quality of the embeddings provided by the LLM. The results are shown in Table 4.13.

	Music		Toys		Yelp	
	Prec	**NDCG**	**Prec**	**NDCG**	**Prec**	**NDCG**
RP3Beta	**0.0337**	**0.1629**	**0.0137**	**0.0791**	**0.0062**	**0.0301**
ItemCFKNN	0.0304	0.1500	0.0129	0.0748	0.0060	0.0286
Model CB-KNN	0.0255	0.1216	0.0079	0.0401	0.0001	0.0006
HFT	0.0024	0.0047	0.0003	0.0008	0.0003	0.0008
NARRE	0.0004	0.0008	0.0002	0.0005	0.0003	0.0008
HRDR	0.0055	0.0113	0.0001	0.0004	0.0002	0.0008

Table 4.13: Precision@10 and NDCG@10 of Model CB-KNN against the baselines

Despite the simplicity of this model, the performance achieved on two of the three datasets is unexpectedly impressive, as the model is able to obtain results that are not that distant from collaborative filtering baseline results, despite being a content-based model. These methods notoriously perform much worse than collaborative filtering techniques, except for rare situations in which the content information associated with each item is so rich to allow a strong understanding of the hidden patterns in the data. Therefore, the fact that content information is entirely extracted from embeddings obtained through an LLM clearly demonstrates that the LLM is indeed able to understand human text well and generate meaningful embeddings. Results are instead very poor on the Yelp dataset, especially compared to other baselines. These results can be explained by the inherent nature of reviews. Since the model's similarity is entirely based on the content of the reviews, asserting that two items are similar requires the extraction of sufficient information about the characteristics of the items from the reviews. In the case of the Yelp dataset, where items are primarily restaurants or bars, most reviews are likely focused on aspects such as service quality or the taste of a specific dish, rather than providing a more generic description that would enable the extraction of information about the place's overall features. In Amazon datasets, on the other hand, a review is more likely to contain a comprehensive and all-encompassing description of the item, thus allowing a more direct assessment of how similar two objects are. The quality of the embeddings reflects this difference in datasets. The detailed results for all metrics are shown in Section

4.12.

4.11. Results Model CFCB-KNN

Model CFCB-KNN combines content-based information with collaborative filtering to create a hybrid model without employing any neural techniques. Among all the proposed models, it achieves the best results, as demonstrated in the Table 4.14.

	Music		Toys		Yelp	
	Prec	**NDCG**	**Prec**	**NDCG**	**Prec**	**NDCG**
RP3Beta	**0.0337**	0.1629	0.0137	**0.0791**	0.0062	0.0301
ItemCFKNN	0.0304	0.1500	0.0129	0.0748	0.0060	0.0286
Model CFCB-KNN	0.0336	**0.1638**	**0.0138**	0.0782	**0.0064**	**0.0313**
HFT	0.0024	0.0047	0.0003	0.0008	0.0003	0.0008
NARRE	0.0004	0.0008	0.0002	0.0005	0.0003	0.0008
HRDR	0.0055	0.0113	0.0001	0.0004	0.0002	0.0008

Table 4.14: Precision@10 and NDCG@10 of Model CFCB-KNN against the baselines

These results are truly remarkable and provide a definitive answer to the hypothesis underlying this thesis: applying LLMs to process reviews significantly enhances the performance of recommender systems. In two of the three datasets, this model not only substantially improves performance compared to neural baselines but even surpasses RP3Beta, the baseline that yields the best results and represents an effective and widely used collaborative filtering technique. Even on the Amazon Toys dataset, it achieves a result very close to RP3Beta and still outperforms ItemKNN, showcasing the efficacy of the approach. It is interesting to note that, although Model CB-KNN performs poorly on the Yelp dataset, the content-based information is still effectively utilized by this model. The addition of this information enables it to surpass the best baseline. Furthermore, the weight assigned to the ICM after hyperparameter tuning is not insignificantly low (approximately 0.5), demonstrating that superior results are obtained by assigning non-negligible weight to content-based information.

This model demonstrates that while the information contained in the embeddings of reviews alone can lead to satisfactory results (as shown in Model CB-KNN), associating this information with that obtained from interactions in a collaborative filtering approach can yield truly astonishing results, concretely improving the current state of the art. The

detailed results for all metrics are shown in Section 4.12.

4.12. Comparison among all models

To summarize, Tables 4.15, 4.16 and 4.17 show the comparison of results among all presented models for the three datasets.

Amazon Music				
Models	**Precision@10**	**Recall@10**	**MAP@10**	**NDCG@10**
(1) RE	0.0112	0.0864	0.0310	0.0459
(2) RE+CE	0.0248	0.1982	0.0895	0.1191
(3) RE+CE TE	0.0218	0.1774	0.0765	0.1035
(4) CE+AQ	0.0197	0.1532	0.0691	0.0921
(5) CE+AQ TE	0.0211	0.1693	0.0733	0.0991
(6) CB-KNN	0.0255	0.2056	0.0907	0.1216
(7) CFCB-KNN	**0.0336**	**0.2657**	**0.1255**	**0.1638**

Table 4.15: Results on Amazon Music dataset

Amazon Toys				
	Precision@10	**Recall@10**	**MAP@10**	**NDCG@10**
(1) RE	0.0030	0.0278	0.0010	0.0143
(2) RE+CE	0.0072	0.0672	0.0307	0.0397
(3) RE+CE TE	0.0069	0.0641	0.0288	0.0375
(4) CE+AQ	0.0059	0.0545	0.0255	0.0326
(5) CE+AQ TE	0.0072	0.0662	0.0312	0.0399
(6) CB-KNN	0.0079	0.0728	0.0295	0.0401
(7) CFCB-KNN	**0.0131**	**0.1194**	**0.0622**	**0.0757**

Table 4.16: Results on Amazon Toys dataset

Yelp				
	Precision@10	**Recall@10**	**MAP@10**	**NDCG@10**
(1) RE	0.0027	0.0252	0.0009	0.0130
(2) RE+CE	0.0060	0.0549	0.0193	0.0280
(3) RE+CE TE	**0.0071**	**0.0664**	**0.0245**	**0.0346**
(4) CE+AQ	0.0066	0.0611	0.0223	0.0316
(5) CE+AQ TE	0.0065	0.0597	0.0234	0.0323
(6) CB-KNN	0.0001	0.0012	0.0004	0.0006
(7) CFCB-KNN	0.0064	0.0587	0.0224	0.0313

Table 4.17: Results on Yelp dataset

As described in previous sections, the most impressive result is that a relatively simple model like ItemKNN, constructing the similarity matrix on a hybrid matrix containing both collaborative filtering and content-based information (Model CFCB-KNN), achieves excellent results across all datasets. It even emerges as the best-performing model for the two Amazon datasets. This clearly demonstrates the significance of embeddings produced by the LLM, providing an effective contribution when used straightforwardly. However, across all datasets, using only these embeddings in neural models, without any collaborative filtering information, proves less effective, as Model RE consistently yields inferior results.

Once again, there is a notable discrepancy among datasets. In Amazon datasets neural methods struggle to outperform simpler methods, whereas on the Yelp dataset, employing a more complex architecture ensures superior performance, with Model RE+CE TE achieving the overall best results.

4.13. Datasets comparison

Given the observed discrepancies in the results, this section provides a more in-depth comparative analysis of the three datasets to understand if there are any characteristics that can unequivocally justify the different behavior of the models. Table 4.18 shows some basic statistics for the three datasets.

	Amazon Music	Amazon Toys	Yelp
Num users	5541	19412	9224
Num items	3568	11924	9093
Num interactions	1485	1395	1390
Sparsity	0.9967	0.9993	0.9990
Average user interactions	11.68	8.63	9.05
Average item interactions	18.13	14.05	9.18
Average rating	4.22	4.35	3.96
Average token length	265.95	122.26	183.48

Table 4.18: Datasets basic statistics

Analyzing these basic statistics, it can be observed that all datasets have relatively similar characteristics, with no clear differences that could immediately explain variations in results. One possible explanation for the greater effectiveness of the two-tower models proposed in the thesis on the Yelp dataset may be due to the greater symmetry in the number of users and items. In the Yelp dataset, the number of users and items is very similar, as is the average number of interactions per user/item. Therefore, it is plausible that the two-tower models, being symmetric for users and items, can generate more effective information. Conversely, in both Amazon datasets, the number of users is significantly higher than the number of items, resulting in a much higher average number of interactions for items. This asymmetry may likely lead to less effective performance in symmetric models like the two-tower models proposed in this thesis.

To analyze the embeddings, t-distributed stochastic neighbor embedding (t-SNE) [54] was employed. It is a machine learning technique used for visualizing high-dimensional data in a lower-dimensional space, two dimensions in this case. It is particularly useful for exploring the inherent structure or patterns within the data. Results are shown in Figure 4.1, where ratings are used as labels to distinguish different points.

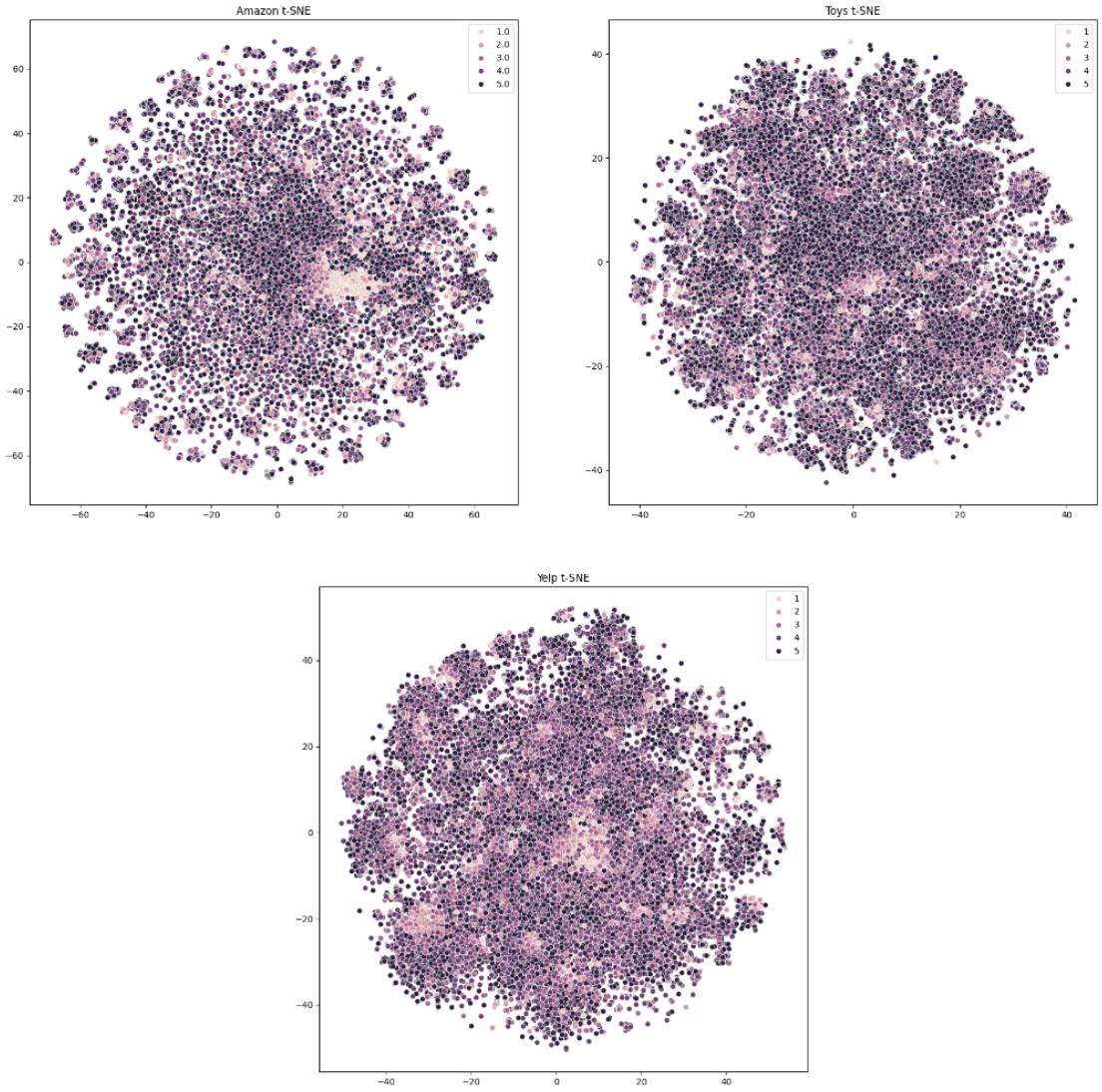


Figure 4.1: t-SNE of the three datasets.

The three plots do not provide any clear conclusion about the behavior in the various datasets, as they do not show any evident difference. It can be observed that in the Amazon datasets, the embeddings appear more clustered towards the center, especially those associated with rating 5 (the darker points in the plots), which seem to concentrate around 0. In contrast, in the Yelp dataset, the points appear more distributed in space, and the embeddings associated with the maximum rating do not seem to concentrate in any specific area. However, these observations do not allow for any definitive conclusions about the differences between datasets.

4.14. Scalability of experiments

When comparing different models, it is meaningful not only to evaluate them using metrics that assess their ability to perform the required task, but also to consider the time complexity and the memory allocation required. If two models perform similarly but the first requires 10 times the training and evaluation time needed by the second, it probably makes sense to prefer the second one, even if with slightly worse results. In this section, time complexity and memory constraints are discussed for each model. Collaborative filtering baselines and content-based methods (models CB-KNN and CFCB-KNN) are not deep learning models and they don't need any parameter to be trained; therefore they are much faster than all other methods, requiring less than a minute to compute the similarity matrix necessary to produce results. Neural baseline models like HFT, NARRE and HRDR, on the other hand, are extremely time-consuming and require hours to be trained. This time complexity is mainly due to the large parameter matrices that are necessary to process the word embeddings, as these methods don't create embeddings on entire reviews but on single words, therefore needing a much larger space. This aspect, together with the unsatisfactory results on ranking metrics, makes these models truly unpractical and ineffective in real-world scenarios, and by far the worst to be considered among all models compared in this thesis. The 5 neural models proposed in this thesis require a non-negligible amount of time to be trained, as they have many parameters to be learned. Particularly noteworthy is a clear difference in computation times between the three models which use a single multi-head attention layer and models RE+CE TE and CE+AQ TE, which instead employ the Transformer's encoder block composed of multiple sub-layers with numerous additional parameters. Still, the time complexity is lower than compared with the neural baselines aforementioned. This is due to the main innovation within the proposed models: embeddings are generated by an external model and just processed inside the model, and not single words but entire reviews are embedded in the latent space. The most time-consuming operation is the computation of the attention, so the average time needed for each epoch increases as the complexity of the attention mechanism grows. The complete overview of the average training time for all models, depending on the dataset, is provided in Table 4.19.

	Amazon Music	Amazon Toys	Yelp
RP3Beta	0.73 s	**2.90 s**	**3.10** s
ItemCFKNN	**0.68 s**	4.89 s	3.96 s
HFT	15.7 s/e	206 s/e	157 s/e
NARRE	172 s/e	385 s/e	194 s/e
HRDR	112 s/e	249 s/e	120 s/e
(1) RE	13.41 s/e	39.8 s/e	14 s/e
(2) RE+CE	24 s/e	125 s/e	18.4 s/e
(3) RE+CE TE	82.5 s/e	316 s/e	75 s/e
(4) CE+AQ	17.2 s/e	48 s/e	43.4 s/e
(5) CE+AQ TE	41 s/e	304.2 s/e	72 s/e
(6) CB-KNN	5.39 s	53.26 s	36.02 s
(7) CFCB-KNN	5.19 s	53.90 s	47.95 s

Table 4.19: Average training time for each model for all datasets. Time is expressed in seconds (s) or seconds per epoch (s/e).

The difference in times within the same model between datasets reflects the different sizes of datasets. In collaborative filtering and content-based methods, for instance, the time to compute the similarity matrix with Amazon Toys data is much longer than using Amazon Music data, because of the higher number of unique users/items within the dataset, resulting in a bigger URM/ICM.

From a scalability perspective, it can be concluded that Model CFCB-KNN is the best model to consider. Indeed, on the Amazon datasets, it outperforms all other models with a significantly lower time requirement compared to neural models. On the Yelp dataset, although it achieves slightly worse results than some neural models, the drastic reduction in the time needed justifies its use over other proposed models.

5 | Conclusion

The work presented in this thesis aims at filling an existing gap in the state of the art of recommender systems. Indeed, there are many existing methods leveraging reviews provided by users to items to enhance recommendations, but none of these methods seems to have resulted in a significant breakthrough in the field of recommender systems. The performances of the considered state-of-the-art models (HFT, NARRE, and HRDR) are rather disappointing when compared to popular collaborative filtering techniques such as ItemKNN and RP3Beta, as demonstrated in Section 4.4. As a result, interest in these methods has waned over the years. On the contrary, there is a growing interest in the application of Large Language Models (LLMs) to recommender systems, given the outstanding capabilities these methods are showcasing across various artificial intelligence tasks.

However, despite the exceptional ability of LLMs to comprehend and process natural language, there is currently no method that employs LLMs to enhance a recommender system by leveraging user reviews. Since these reviews are composed in natural language, it is expected that LLMs could process them very effectively. Therefore, the aim of this thesis is to verify this hypothesis, by presenting seven different models that, at different levels of complexity, exploit the power of an LLM to process user reviews.

All models are based on the use of embeddings obtained from reviews using an LLM, in particular OpenAI's text-embedding-ada-002 model[23]. These embeddings are then used as features describing items in a content-based method (Model CB-KNN), are combined with collaborative filtering information in a hybrid ItemKNN model (Model CFCB-KNN), are processed using multi-head attention in a neural model (Models RE, RE+CE and CE+AQ), and are processed using the Transformer's encoder in a more complex neural model (Models RE+CE TE and CE+AQ TE). All models produce different results but they all lead to the same conclusion: LLMs can indeed improve the performance of a recommender system that uses reviews. As shown in Sections from 4.5 to 4.9, neural models with the same structure of NARRE and HRDR obtain results that are by far more satisfactory on ranking metrics and are much closer to the results obtained by

collaborative filtering baselines. The impressive improvement in results demonstrates the quality of the embeddings produced by the LLM and the effectiveness of embedding the entire review rather than single words.

The differences in data characteristics between the datasets translate into a discrepancy in results among different models. The fact that neural models fail to perform as well as collaborative filtering baselines on two of the three datasets can be attributed not to the quality of the embeddings produced by the LLM but rather to the inherent architecture of the models themselves. Models CB-KNN and CFCB-KNN, in fact, demonstrate that the embeddings contain a substantial amount of meaningful information. For instance, a simple content-based method like Model CB-KNN, which would typically yield poor results when compared to collaborative filtering methods, manages to achieve outcomes that are not so distant from those of ItemKNN and RP3Beta in those datasets. Moreover, utilizing embeddings as features combined with collaborative filtering information, as demonstrated in Model CFCB-KNN, even leads to results surpassing those of effective models such as RP3Beta in some cases. This shows that, in the Amazon datasets, the primary issue causing the performance decline lies in the use of complex neural models, which often fail to deliver satisfactory results in the world of recommender systems, as evidenced in [18]. On the other side, this also demonstrates that, if employed in a straightforward manner without excessive processing, the information provided by the embeddings generated by the LLM can effectively enrich the recommendation model and yield results sometimes superior to the current state of the art. On the Yelp datasets, on the other hand, models from 3 to 5 as well as Model CFCB-KNN all outperform collaborative filtering baselines, highlighting how, under certain circumstances, straightforward methods as well as complex neural methods succeed in obtaining results that improve the current state of the art, fully confirming the hypothesis underlying this thesis.

The promising results presented in this thesis pave the way for numerous potential directions in future research. Firstly, only one model for generating embeddings has been employed, making it worthwhile to showcase the outcomes achieved by employing other LLMs to compare the quality of the produced embeddings. At the moment, Google's PaLM model can be used through an API that is very similar to OpenAI's API, both in pricing and in model capabilities. Therefore, it could be easily used for comparison. Other directions include exploring alternative techniques for processing reviews through LLMs, beyond utilizing models specialized in embedding generation. One possibility could involve employing a chat model (such as ChatGPT) to obtain a summary of the review or a set number of adjectives that can encapsulate its essence. This approach would then provide a representation in a latent space of a different dimension. Given the impres-

sive results obtained by Model CFCB-KNN, another direction could be the exploration of more complex hybrid techniques, still without employing neural models, which may struggle to achieve comparable results, as demonstrated by the first 5 models. Examples can be some hybrid techniques described in Subsection 2.2.4, or the use of models like Factorization Machines[45]. Finally, another direction to expand the scope of this thesis is a more detailed exploration and analysis of the embeddings produced by the LLM. This involves spatial analyses to understand which dimensions are most significant and potentially extract them. Given the demonstrated quality of the generated embeddings, they could be utilized not only to enhance the results of tasks like top-k recommendations but also for more intricate and delicate tasks such as explainable recommendations.

To conclude, every day new applications of large language models are being released, showcasing the limitless potential of these models. Consequently, endless directions can be explored to harness reviews and enhance recommender systems. This thesis represents just the initial step, demonstrating that the path forward is promising.

Bibliography

[1] Common crawl dataset. URL https://commoncrawl.org/.

[2] Yelp dataset. URL https://www.yelp.com/dataset.

[3] Acm recsys 2023 evaluation frameworks, 2023. URL https://github.com/ACMRecSys/recsys-evaluation-frameworks.

[4] S. Aciar, D. Zhang, S. Simoff, and J. Debenham. Informed recommender: Basing recommendations on consumer product reviews. *IEEE Intelligent systems*, 22(3): 39–47, 2007.

[5] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

[6] P. Barham, A. Chowdhery, J. Dean, S. Ghemawat, S. Hand, D. Hurt, M. Isard, H. Lim, R. Pang, S. Roy, et al. Pathways: Asynchronous distributed dataflow for ml. *Proceedings of Machine Learning and Systems*, 4:430–449, 2022.

[7] D. Blei, A. Ng, and M. Jordan. Latent dirichlet allocation. In T. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems*, volume 14. MIT Press, 2001. URL https://proceedings.neurips.cc/paper_files/paper/2001/file/296472c9542ad4d4788d543508116cbc-Paper.pdf.

[8] P. F. Brown, V. J. Della Pietra, P. V. Desouza, J. C. Lai, and R. L. Mercer. Class-based n-gram models of natural language. *Computational linguistics*, 18(4):467–480, 1992.

[9] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[10] C. Chen, M. Zhang, Y. Liu, and S. Ma. Neural attentional rating regression with review-level explanations. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web*, pages 1583–1592, 2018.

[11] L. Chen, G. Chen, and F. Wang. Recommender systems based on user reviews: the state of the art. *User Modeling and User-Adapted Interaction*, 25:99–154, 2015.

[12] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

[13] P. F. Christiano, J. Leike, T. Brown, M. Martic, S. Legg, and D. Amodei. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30, 2017.

[14] F. Christoffel, B. Paudel, C. Newell, and A. Bernstein. Blockbusters and wallflowers: Accurate, diverse, and scalable recommendations with random walks. In *Proceedings of the 9th ACM Conference on Recommender Systems*, pages 163–170, 2015.

[15] Z. Cui, J. Ma, C. Zhou, J. Zhou, and H. Yang. M6-rec: Generative pretrained language models are open-ended recommender systems. *arXiv preprint arXiv:2205.08084*, 2022.

[16] S. N. Dorogovtsev, A. V. Goltsev, and J. F. F. Mendes. K-core organization of complex networks. *Physical review letters*, 96(4):040601, 2006.

[17] S. G. Esparza, M. P. O'Mahony, and B. Smyth. Effective product recommendation using the real-time web. In *International Conference on Innovative Techniques and Applications of Artificial Intelligence*, pages 5–18. Springer, 2010.

[18] M. Ferrari Dacrema, P. Cremonesi, and D. Jannach. Are we really making much progress? a worrying analysis of recent neural recommendation approaches. In *Proceedings of the 13th ACM conference on recommender systems*, pages 101–109, 2019.

[19] M. Ferrari Dacrema, S. Boglio, P. Cremonesi, and D. Jannach. A troubling analysis of reproducibility and progress in recommender systems research. *ACM Trans. Inf. Syst.*, 39(2), jan 2021. ISSN 1046-8188. doi: 10.1145/3434185. URL `https://doi.org/10.1145/3434185`.

[20] M. Ganapathibhotla and B. Liu. Mining opinions in comparative sentences. In *Proceedings of the 22nd international conference on computational linguistics (Coling 2008)*, pages 241–248, 2008.

[21] S. Geng, S. Liu, Z. Fu, Y. Ge, and Y. Zhang. Recommendation as language processing (rlp): A unified pretrain, personalized prompt & predict paradigm (p5). In *Proceedings of the 16th ACM Conference on Recommender Systems*, pages 299–315, 2022.

[22] A. Graves, G. Wayne, and I. Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

[23] R. Greene, T. Sanders, L. Weng, and A. Neelakantan. New and improved embedding model, 2022. URL `https://openai.com/blog/new-and-improved-embedding-model`.

[24] N. Hariri, B. Mobasher, R. Burke, and Y. Zheng. Context-aware recommendation based on review mining. In *ITWP@ IJCAI*, 2011.

[25] R. He and J. McAuley. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In *proceedings of the 25th international conference on world wide web*, pages 507–517, 2016.

[26] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*, pages 173–182, 2017.

[27] S. N. Ian MacKenzie, Chris Meyer. How retailers can keep up with consumers, 2013. URL `https://www.mckinsey.com/industries/retail/our-insights/how-retailers-can-keep-up-with-consumers#/`.

[28] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[29] K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Transactions on Information Systems (TOIS)*, 20(4):422–446, 2002.

[30] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization, 2017.

[31] Y. Koren and R. Bell. *Advances in Collaborative Filtering*, pages 77–118. Springer US, Boston, MA, 2015. ISBN 978-1-4899-7637-6. doi: 10.1007/978-1-4899-7637-6_3. URL `https://doi.org/10.1007/978-1-4899-7637-6_3`.

[32] D. Lee and H. S. Seung. Algorithms for non-negative matrix factorization. *Advances in neural information processing systems*, 13, 2000.

[33] H. Liu, Y. Wang, Q. Peng, F. Wu, L. Gan, L. Pan, and P. Jiao. Hybrid neural recommendation with joint deep representation learning of ratings and reviews. *Neurocomputing*, 374:77–85, 2020. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2019.09.052. URL `https://www.sciencedirect.com/science/article/pii/S0925231219313207`.

[34] J. McAuley and J. Leskovec. Hidden factors and hidden topics: Understanding rating

dimensions with review text. In *Proceedings of the 7th ACM Conference on Recommender Systems*, RecSys '13, page 165–172, New York, NY, USA, 2013. Association for Computing Machinery. ISBN 9781450324090. doi: 10.1145/2507157.2507163. URL https://doi.org/10.1145/2507157.2507163.

[35] Y. Moshfeghi, B. Piwowarski, and J. M. Jose. Handling data sparsity in collaborative filtering using emotion and semantic based features. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, pages 625–634, 2011.

[36] C.-C. Musat, Y. Liang, and B. Faltings. Recommendation using textual opinions. In *IJCAI International Joint Conference on Artificial Intelligence*, number CONF, pages 2684–2690, 2013.

[37] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[38] A. Neelakantan, T. Xu, R. Puri, A. Radford, J. M. Han, J. Tworek, Q. Yuan, N. Tezak, J. W. Kim, C. Hallacy, et al. Text and code embeddings by contrastive pre-training. *arXiv preprint arXiv:2201.10005*, 2022.

[39] OpenAI. Gpt-4 technical report. Please cite this work as "OpenAI (2023)", 2023.

[40] E. Parzen. On estimation of a probability density function and mode. *The annals of mathematical statistics*, 33(3):1065–1076, 1962.

[41] K. Pearson. Mathematical contributions to the theory of evolution. iii. regression, heredity, and panmixia. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, 187:253–318, 1896. ISSN 02643952. URL http://www.jstor.org/stable/90707.

[42] D. Poirier, F. Fessant, and I. Tellier. Reducing the cold-start problem in content recommendation through opinion classification. In *2010 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, volume 1, pages 204–207. IEEE, 2010.

[43] PyTorch. Pytorch documentation, 2023. URL https://pytorch.org/docs/stable/nn.html.

[44] S. Raghavan, S. Gunasekar, and J. Ghosh. Review quality aware collaborative filtering. In *Proceedings of the sixth ACM conference on Recommender systems*, pages 123–130, 2012.

[45] S. Rendle. Factorization machines. In *2010 IEEE International conference on data mining*, pages 995–1000. IEEE, 2010.

[46] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. *arXiv preprint arXiv:1205.2618*, 2012.

[47] N. Sachdeva and J. McAuley. How useful are reviews for recommendation? a critical review and potential improvements. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '20, page 1845–1848, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450380164. doi: 10.1145/3397271.3401281. URL https://doi.org/10.1145/3397271.3401281.

[48] A. Salah, Q.-T. Truong, and H. W. Lauw. Cornac: A comparative framework for multimodal recommender systems. *Journal of Machine Learning Research*, 21(95): 1–5, 2020.

[49] G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Information processing & management*, 24(5):513–523, 1988.

[50] J. B. Schafer, D. Frankowski, J. Herlocker, and S. Sen. *Collaborative Filtering Recommender Systems*, pages 291–324. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007. ISBN 978-3-540-72079-9. doi: 10.1007/978-3-540-72079-9_9. URL https://doi.org/10.1007/978-3-540-72079-9_9.

[51] S. Smith, M. Patwary, B. Norick, P. LeGresley, S. Rajbhandari, J. Casper, Z. Liu, S. Prabhumoye, G. Zerveas, V. Korthikanti, et al. Using deepspeed and megatron to train megatron-turing nlg 530b, a large-scale generative language model. *arXiv preprint arXiv:2201.11990*, 2022.

[52] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, et al. Llama: Open and efficient foundation language models. corr, abs/2302.13971, 2023. doi: 10.48550. *arXiv preprint arXiv.2302.13971*, 2023.

[53] A. Töscher and M. Jahrer. The bigchaos solution to the netflix grand prize. 01 2009.

[54] L. Van der Maaten and G. Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

[55] R. van Meteren. Using content-based filtering for recommendation. 2000. URL https://api.semanticscholar.org/CorpusID:2088490.

[56] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[57] L. Wang and E.-P. Lim. Zero-shot next-item recommendation using large pretrained language models. *arXiv preprint arXiv:2304.03153*, 2023.

[58] W. Zhang, G. Ding, L. Chen, C. Li, and C. Zhang. Generating virtual ratings from chinese reviews to augment online recommendations. *ACM Transactions on intelligent systems and technology (TIST)*, 4(1):1–17, 2013.

[59] Y. Zhang, H. Ding, Z. Shui, Y. Ma, J. Zou, A. Deoras, and H. Wang. Language models as recommender systems: Evaluations and limitations. 2021.

[60] L. Zheng, V. Noroozi, and P. S. Yu. Joint deep modeling of users and items using reviews for recommendation. *CoRR*, abs/1701.04783, 2017. URL `http://arxiv.org/abs/1701.04783`.

List of Figures

List of Tables

Acknowledgements