Faculty of Engineering & Technology

Department of Electrical & Computer Engineering

Accelerated DLRM-based E-commerce Recommendation System

Prepared By:

Ibraheem Alyan 1201180

Mohammad Abu-Shelbaia 1200198

Nidal Zabade 1200153

Supervised By:

Dr. Ahmed Shawahna

A graduation project report submitted to the Electrical and Computer Engineering Department as part of the B.Sc. in Computer Engineering degree requirements fulfillment.

Birzeit

February 12, 2024

Abstract

The project aims to design and develop a cutting-edge accelerated e-commerce deep learning recommendation system. The goal is to deliver a production-ready solution, with automated data injection and training pipelines, and a simple RESTful application programming interface (API) as a final interface. The project will have a special focus on scalability and performance.

This report discusses different types of recommendation systems and then compares them to deep learning recommendation model (DLRM) based systems in terms of different metrics and features, such as their accuracy, scalability, and performance. Furthermore, it compares existing solutions and their aspects, in addition to discussing possible technolgies and architectures to use in the system.

Arabic Abstract

يهدف المشروع إلى تصميم وتطوير نظام توصية لمنصات التجارة الإلكترونية باستعمال التعلم الآلي العميق. الهدف النهائي هو تقديم حل صالحة لبيئة التشغيل، تتم فيه أتمتة عمليات إدخال البيانات و تدريب نماذج التعلم اللآلي وواجهة برمجة تطبيقات RESTful API كواجهة نهائية. سيركز المشروع بشكل خاص على قابلية التوسع والأداء.

يناقش هذا التقرير أنواعًا مختلفة من أنظمة التوصيات ويقارنها بالأنظمة القائمة على نماذج التوصية بالتعلم اللآلي العميق (DLRM) من حيث المقاييس والميزات المختلفة. علاوة على ذلك، فهو يقارن الحلول المتوفرة حالياً و مزاياها، كما ويناقش التقنيات والبنى الممكن استعمالها في تطوير النظام وتشغيله.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Contents

1.1 Motivation

The exponential growth of e-commerce has introduced an enormous amount of choice, where consumers face overwhelming product options. To address this challenge, personalized recommendation systems [1] have become essential for enhancing the shopping experience and increasing the conversion rate for any e-commerce platform.

In contrast to conventional collaborative filtering [2], content-based [3], or popularity-based recommendation systems, our AI-based solution offers distinct advantages. Firstly, AI makes it possible to provide per-user personalized recommendations, which are tailored to their unique preferences and behaviors, enhancing user engagement and satisfaction. AI systems can also intelligently recommend comparable or complementary products or content to increase revenue through cross-selling. Furthermore, AI takes into account the impressions and interactions of users with items, allowing for a more dynamic and accurate understanding of user preferences. Using AI leads to improved recommendation accuracy and relevancy, leading to increased conversion rates and business growth.

Statistics from different use cases of recommendation systems:

- An intelligent recommender system delivers on average a 22.66% lift in conversions rates [4] for web products.
- IKEA experienced a 30% increase in click-through rate, 2% surge in average order value [5] using Google Recommendations AI [6].
- Lotte Mart experienced a 1.7x increase in new product purchases [7] using Amazon Personalize [8].

In summary, the project's motivation is elevating the e-commerce experience, driving

business success, and harnessing cutting-edge AI technologies to create a recommendation system that is both high-performing and scalable.

1.2 Problem Statement

The process of building the solution is mainly two parts:

- First, designing a personalized recommendation system that covers what traditional collaborative filtering, content-based, or popularity-based systems cannot achieve.
- Second, deploying and automating the solution, including, data cleaning, data storage, and model deployment processes, and ensuring a production-ready and scalable system.

1.3 Report Organization

The rest of the report is organized as follows.

Chapter 2 delves into recommendation systems and their components, and reviews related work. Chapter 3 outlines functional and system requirements alongside a literature review of the existing recommendation systems and libraries. Chapter 4 unveils the proposed solution, its components, and the recommendation pipeline with its stages. It also discusses its deployment and infrastructure. Chapter 5 presents a Proof-Of-Concept and its results, and analyzes their implications. Finally, Chapter 6 concludes with key findings and outlines promising avenues for future exploration.

Chapter 2

Background

Contents

2.1 Recommendation Systems

A recommendation system is an artificial intelligence or AI algorithm, usually associated with machine learning, that uses Big Data to suggest or recommend additional products to consumers. These can be based on various criteria, including past purchases, search history, demographic information, and other factors. [2]

Recommender systems undergo training to understand the preferences, earlier decisions, and attributes of the user and products using their past interactions which include impressions, clicks, purchases, and ratings. Recommender systems are usually used by content and product providers to suggest items to users that they may like based on their profiles and preferences.

2.2 Types of Recommendation Systems

2.2.1 Context Filtering

Context Filtering is a technique that uses the contextual information of the user by framing the recommendation problem as a contextual multi-armed bandit problem and using the contextual information to learn the user's preferences.

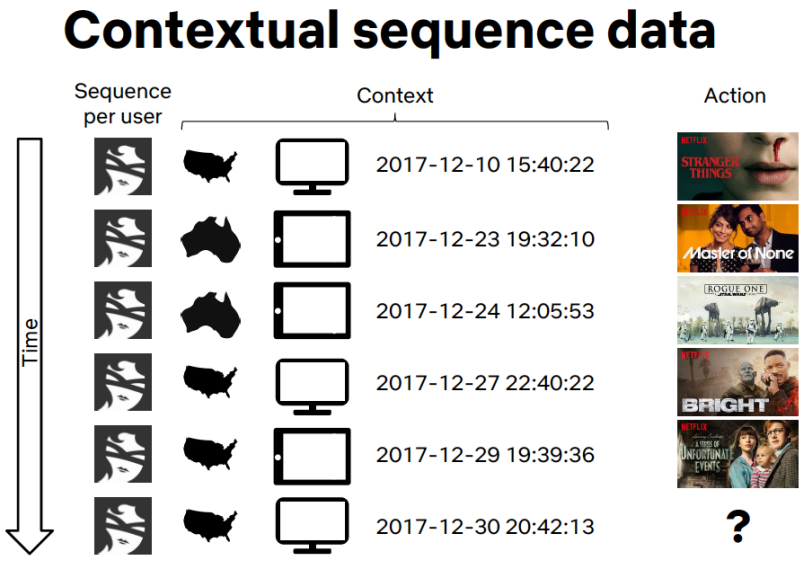


Figure 2.1: Context Filtering Diagram [2]

2.2.2 Variational Autoencoder for Collaborative Filtering

This model consists of two parts: an encoder and a decoder. The encoder takes the user's preferences as input and encodes them into a latent space. The decoder takes the latent space as input and decodes it into the item's features.

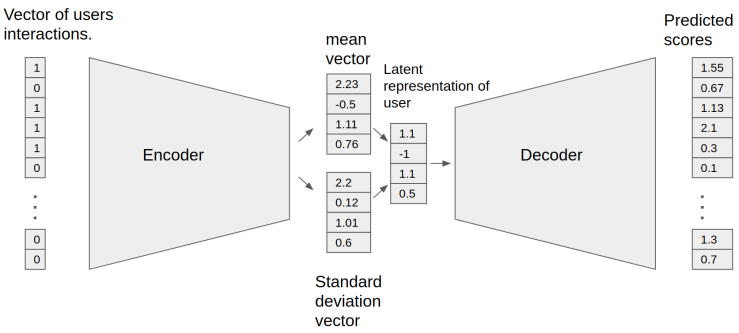


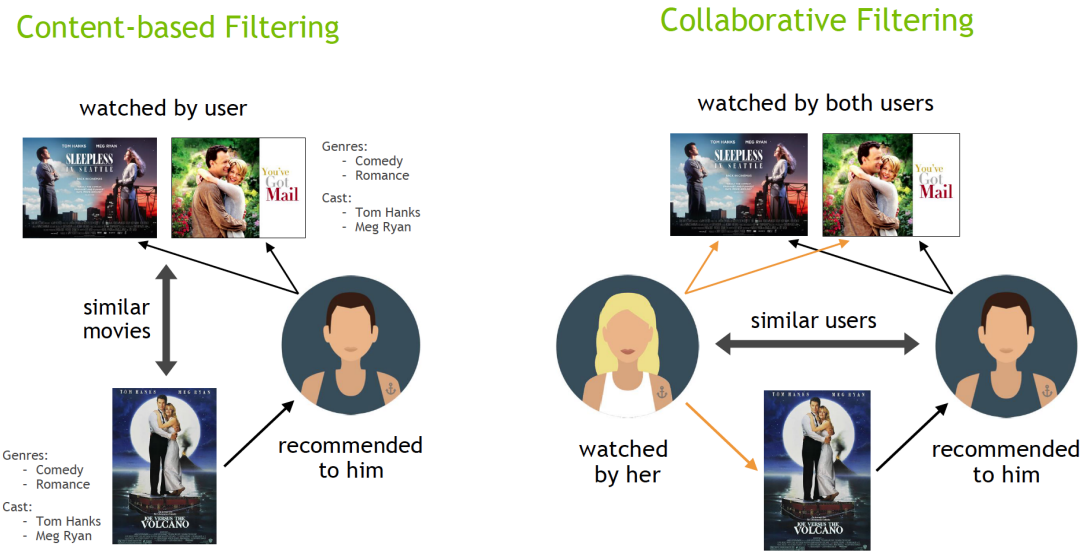
Figure 2.2: Variational Autoencoder for Collaborative Filtering Structure [2]

2.2.3 Collaborative Filtering

A technique that filters out products that a customer might like based on reactions by similar users. It functions by clustering customers into smaller sets with similar interests. Then it uses the items they show interest in to create a ranked suggestions list. The idea behind this technique is that individuals who have previously agreed will continue to do so in the future.

2.2.4 Content Filtering

A technique that uses the features of items a user has interacted with to recommend more items with similar features. This technique is based on the idea that if a customer shows interest in a particular product, he will also be interested in a similar product.



(a) Content Filtering

(b) Collaborative Filtering Diagram

Figure 2.3: Nvidia Glossary Diagram [2]

2.2.5 Hybrid Recommendation Systems

Combine the advantages of multiple types of recommendation algorithms to create a more comprehensive recommending system.

2.2.6 Neural Collaborative Filtering

NCF is a technique that uses neural networks to learn the customer's preferences and recommend items. It uses one neural network that learns the customer's preferences and another neural network that learns the item's features. The two networks are then combined to create a recommendation.

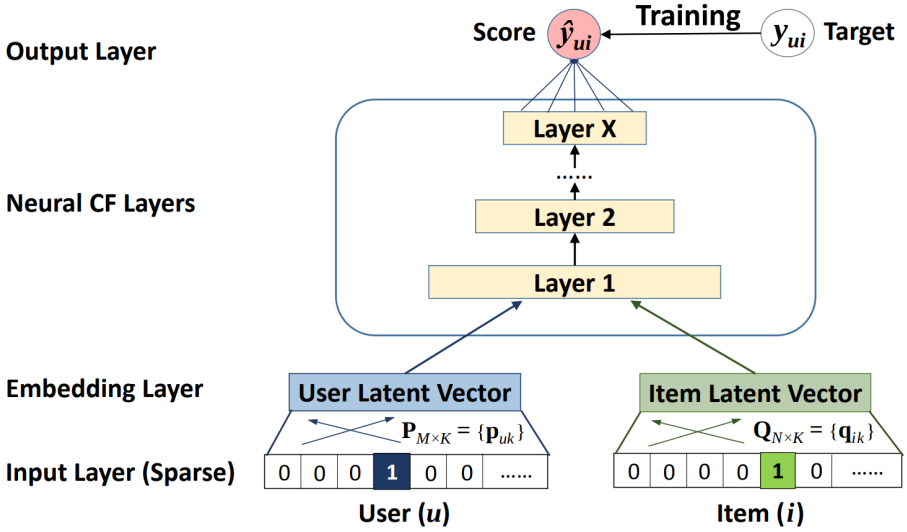


Figure 2.4: Neural Collaborative Filtering [2]

2.2.7 Contextual Sequence Learning

Contextual Sequence Learning is a technique that takes into account the context and sequence of the user's action, it usually uses a recurrent neural network (RNN). An example use case is session-based recommendations, RNNs predict the next items based on user event sequences, mirroring word embedding in NLP.

2.2.8 Wide And Deep

Wide & Deep is a technique that uses a wide neural network to learn the preferences of the customer and utilizes another deep neural network to learn the products's features. The wide model is a generalized linear model (GLM) and the deep model is built from a dense neural network (DNN).

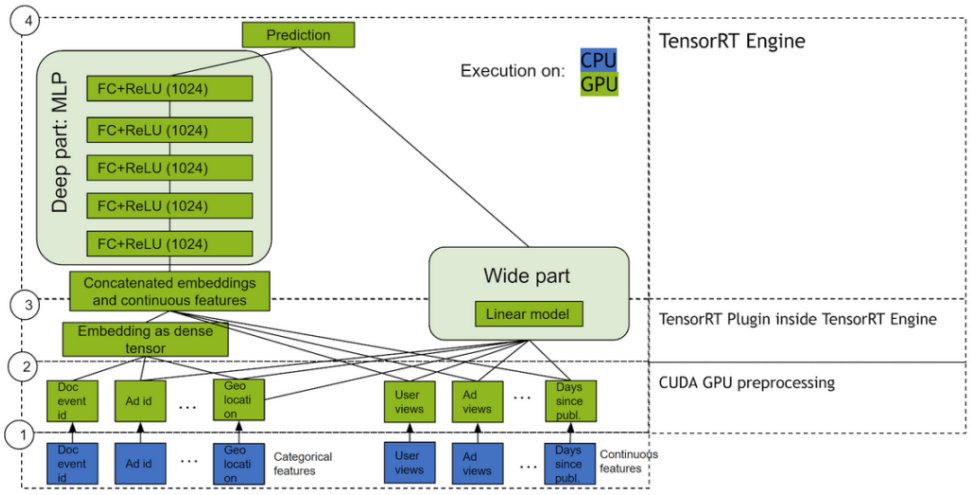
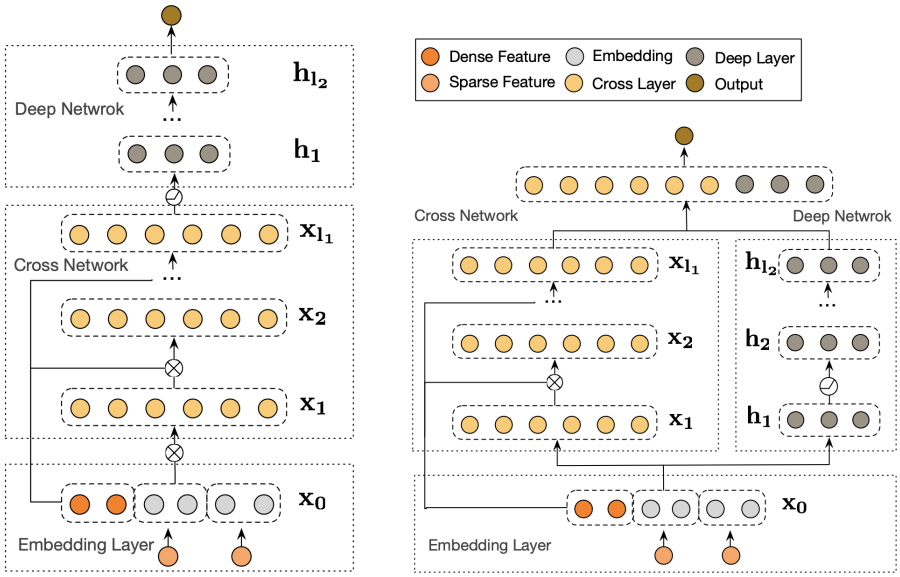


Figure 2.5: Wide Deep Structure [2]

2.2.9 Deep & Cross Network-V2 (DCN-V2)

DCN-V2 builds on the DCN model to excel at ranking tasks. It tackles this by analyzing both direct connections between features and more complex interactions. Unlike DCN, it effectively captures these intricate relationships. This makes it especially good at understanding how individual features influence each other to impact the final ranking.[9]



(a) Stacked **(b) Parallel**

Figure 1: Visualization of DCN-V2. $\otimes$ represents the cross operation in Eq. (1), *i.e.*, $x_{l+1} = x_0 \odot (W_l x_l + b_l) + x_l$.

Figure 2.6: Deep Cross Network Structure [9]

2.2.10 Multi-layer perceptron (MLP)

Multi-layer perceptron (MLP) is a technique that uses a deep neural network to learn the customer's preferences and recommend items. It uses an embedding layer to learn the customer's preferences and another embedding layer to learn the item's features. The two embedding layers are then combined to create a recommendation.

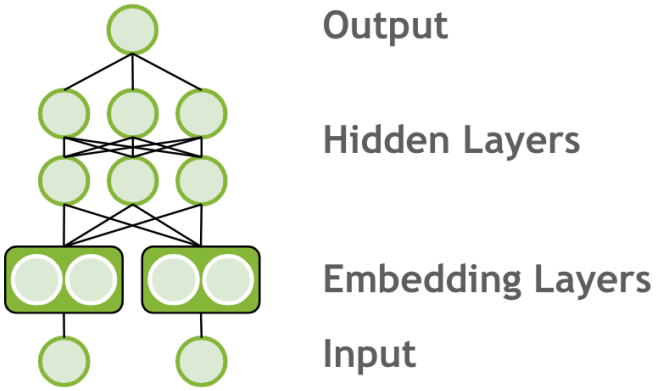


Figure 2.7: Multi-Layer Perceptron Structure [10]

2.2.11 DLRM

Deep Learning Recommendation Model (DLRM) is a technique that uses a deep neural network to handle categorical and numerical features. Each categorical feature is represented as a dense vector produced by an embedding model, and each numerical feature is represented as a dense vector, both fed into multi-layer perceptron (MLP) layers. The output of the MLP layers is then fed into a dot product layer to compute the inner product of the feature vectors. Finally, the output of the dot product layer is then fed into a sigmoid layer to compute the probability of the user liking the item.

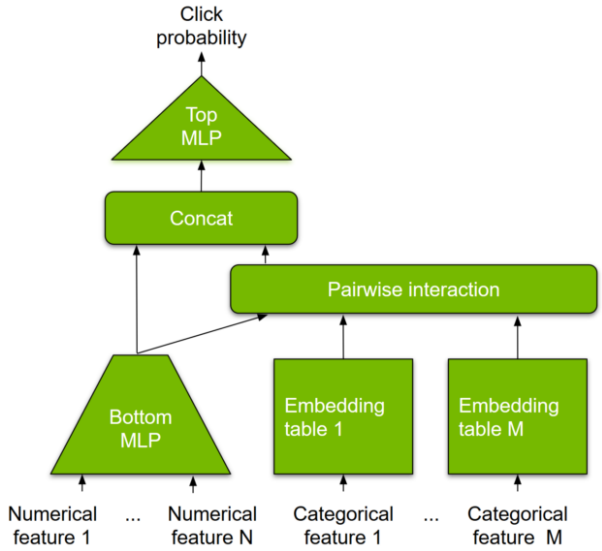


Figure 2.8: DLRM Structure [2]

2.3 DLRM Recommendation System Architecture

To build a recommendation system based on a deep learning ranking model, there are several stages that the system should go through to provide the final recommendations.

Any system is a group of components that work together to achieve a goal according to a set of rules. The recommendation system is no different, it is a group of components that work together to provide personalized suggestions to users. There are two recommendation system types based on the number of stages they have:

2.3.1 Two-stage Recommender Systems

Two-stage recommender systems are systems that have two main stages: candidate generation and ranking. The candidate generation stage is responsible for generating a set of candidate items for each user, while the ranking stage is responsible for ranking the candidate items and selecting the top items to be recommended to the user. The candidate generation stage is usually based on collaborative filtering or content-based filtering, while the ranking stage is usually based on machine learning models such as matrix factorization or deep learning models.[11]

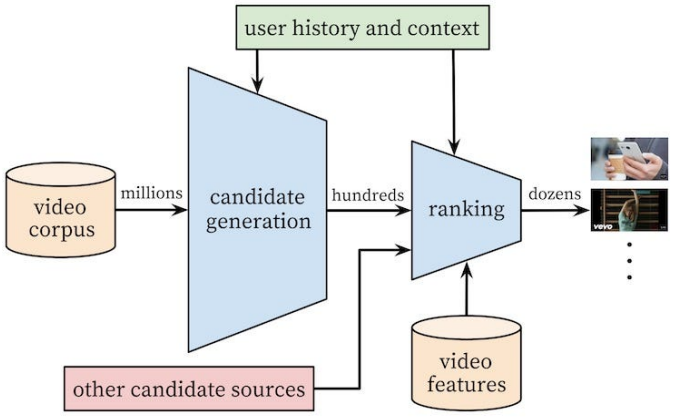


Figure 2.9: Two-stage Recommender System[11]

2.3.2 Four-stage Recommender Systems

Four-stage recommender systems are systems that have four main stages: retrieval, filtering, scoring, and ordering. The retrieval stage is responsible for retrieving a set of candidate items for each user, the filtering stage is responsible for filtering the candidate items according to business rules and constraints, and the ordering stage is responsible for ordering the scored items and selecting the top items to be recommended to the user. The retrieval stage is usually based on collaborative filtering or content-based filtering, while the filtering stage is usually based on machine learning models such as matrix factorization or deep learning models, and the scoring and ordering stages are usually based on machine learning models such as deep learning models.[12]

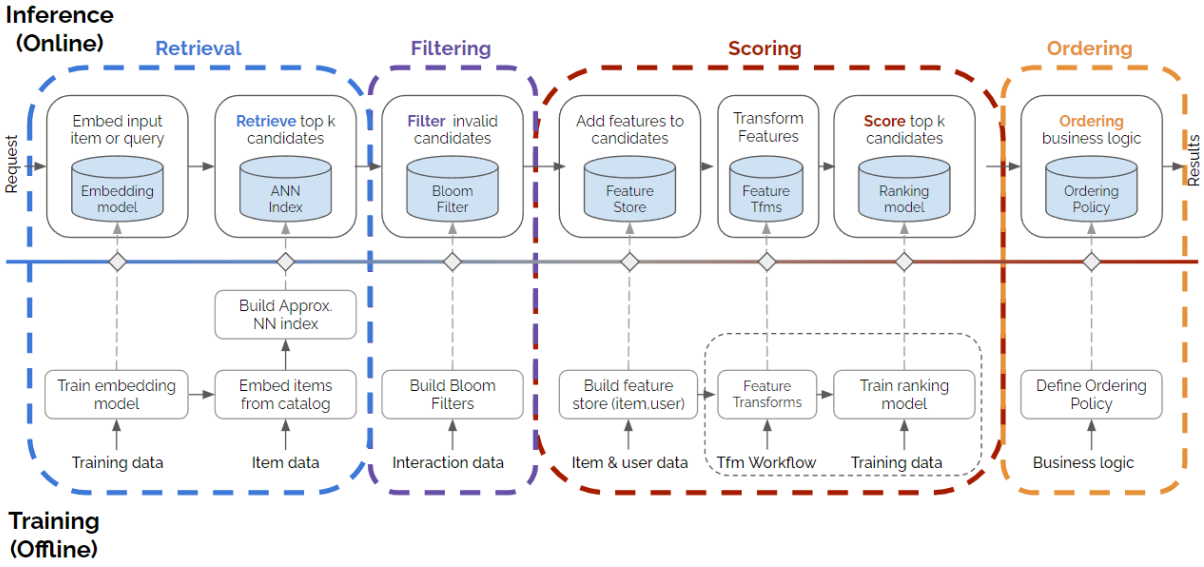


Figure 2.10: Four-stage Recommender System[12]

each stage utilizes a set of components and algorithms in training and inference processes to achieve its goal.

2.3.3 Dealing with categorical features

One of the main challenges in building a recommendation system is dealing with high cardinality categorical features, which are features that have a large number of unique values. The high cardinality of categorical features makes it difficult to use traditional machine learning models such as linear regression and decision trees, as these models require the categorical features to be one-hot encoded, which results in a large number of features and a large model size. To deal with high cardinality categorical features, the deep learning recommendation model (DLRM) uses several techniques such as:

Features embedding

Feature embedding is a technique used to represent categorical features as dense vectors of real numbers, where each unique value of the categorical feature is represented by a unique vector. The feature embedding technique is used to reduce the dimensionality of the categorical features and to learn the relationships between the unique values of the categorical features.[13]

Targaet encoding

The target encoding technique is employed to encode categorical features with high cardinality by using the mean of the target variable for each unique value of the categorical feature. This technique helps reduce the dimensionality of the categorical features while encoding them based on their relationship with the target variable.[14]

2.3.4 Training (Offline)

Like any machine learning system, the process of training the models occurs offline, where the system is trained on a set of historical data, aka batched data, to optimize the model's

parameters. Also in the training phase, the system computes the feature embeddings for all users and items in addition to any other categorical features.

2.3.5 Real-Time Inference (Online)

The inference phase is when the system makes the predictions and suggestions for the application. To be useful in any website or application, the system should be able to provide real-time predictions, and suggestions, with a few hundred milliseconds of latency.

In online inference, the system computes the recommendation in Real-time based on the user's interactions and preferences, usually with user actions being the trigger for the recommendation generation process.

Figure 2.11 shows a pipeline suggested by Nvidia[15] for a two-stage online recommendation system.

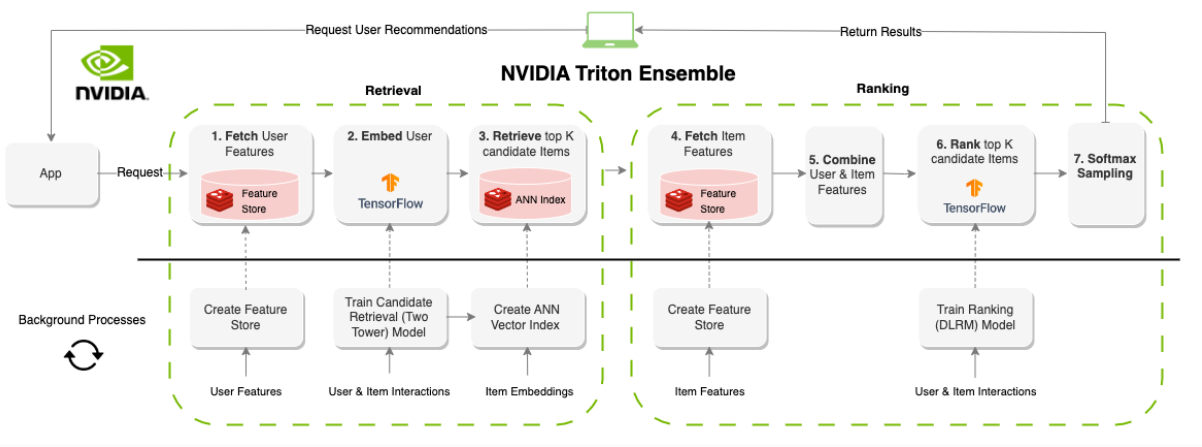


Figure 2.11: Two Stage Online Recommendation System [15]

2.3.6 Batch Inference (Offline)

Unlike online inference, batch inference is the process of generating recommendations in advance and storing them, where the system computes the recommendations for all users and items, the process is repeated on a periodical basis.[15] Figure 2.12 illustrates from a high perspective how a completely offline recommendation system operates.

This process allows much shorter response times and is useful for systems with a large number of users and items and a high volume of traffic.

Online and batched inference can be used together to provide real-time recommendations, where the system periodically pre-computes the recommendations for active users, but for new users, the system computes the recommendations in Real-time.

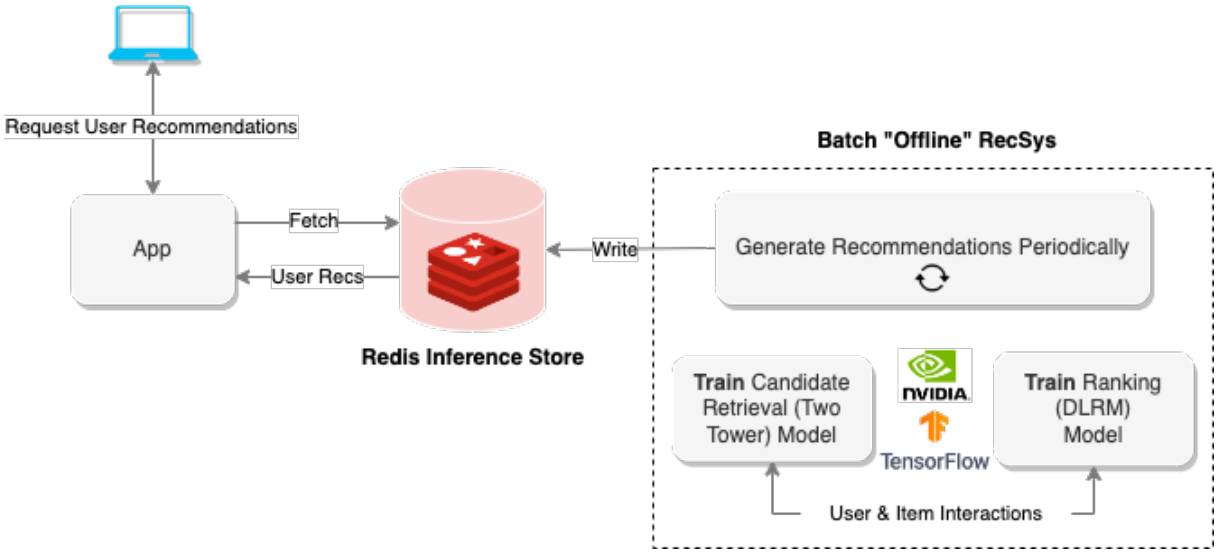


Figure 2.12: Batch Recommendation System [15]

2.3.7 Retrieval Stage

To save resources and make a recommendation system effective, the first stage of it is retrieval, where the system aims to narrow down the massive catalog of items (potentially millions) to a much smaller, manageable pool of candidates (around hundreds or thousands). This initial filtering is crucial because directly running the ranking model on every item for every user would be computationally expensive and unnecessary.

The recommended approach for retrieval according to Nvidia [15] is to use a two-tower model in the offline training phase to generate embeddings for users and items. Then in the online phase (serving the recommendations), an approximate nearest neighbor (ANN) search to retrieve the top candidates for each user and using the tower user alone.

Two-Tower Model

Another popular approach to retrieval is using a two-tower model, The concept for this model design is rather straightforward; as the diagram in Figure 2.13 illustrates, it consists of two completely independent towers, one for the user and one for the item. The model can learn high-level abstract representations for a user and an item based on previous user-item interactions thanks to deep neural networks.

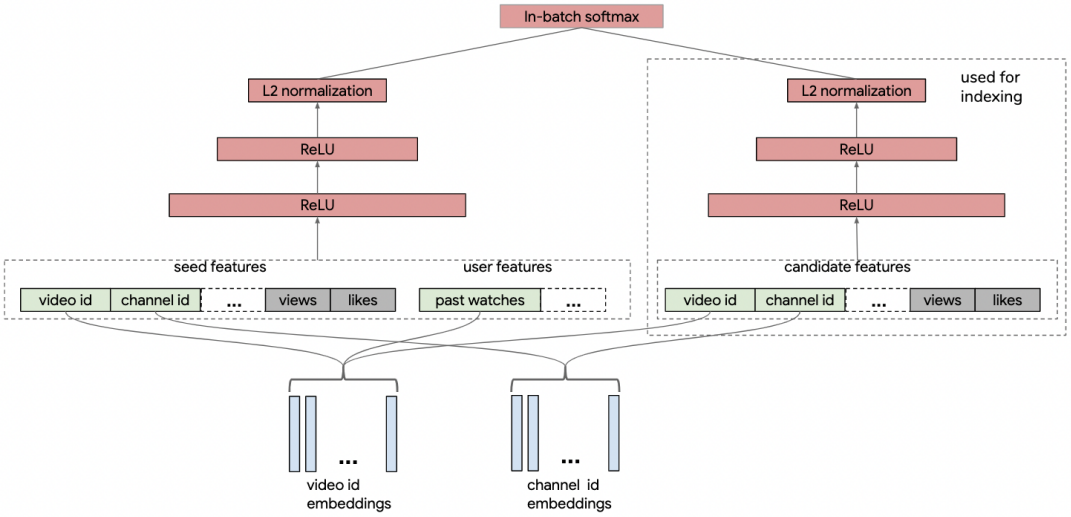


Figure 2: Illustration of the Neural Retrieval Model for YouTube.

Figure 2.13: Two-Tower Neural Retrieval Model for YouTube [16]

The output of the two-tower model is a vector representing the similarity between item embedding and user embedding, it indicates the user's level of interest in the specified item.

Similarity can be either Cosine Similarity, or Euclidean Distance, but cosine is more popular in recommendation systems because it is less sensitive to the magnitude of the vectors, as shown in equation 2.1.

$$\text{Cosine Similarity} = \frac{A \cdot B}{\|A\|\|B\|} \tag{2.1}$$

Approximate Nearest Neighbor (ANN)

One popular approach to retrieval uses embedding models. These models create a dense vector space where users and items are represented as points. Approximate Nearest Neighbor (ANN) search, which identifies items closest to the user's representation in the space, indicating potential relevance and making them strong candidates for recommendation.

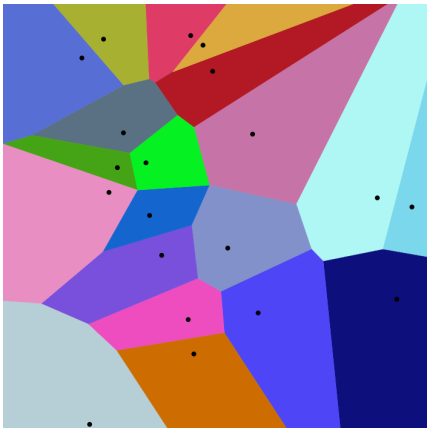


Figure 2.14: ANN Voronoi Diagram [17]

2.3.8 Filtering Stage

After retrieving a set of candidate items, the next stage is filtering them according to business rules and constraints. This stage ensures that only valid candidates are passed to the scoring stage. For example, products that are out of stock, or products that users already bought.

One possible approach to filtering is using a Bloom filter, which is a space-efficient probabilistic data structure that is used to test whether an element is a member of a set.

Another simpler approach is to use a rule-based system, where the system applies a set of rules to the candidate items to filter out the invalid ones, but it might be less performant than the Bloom filter.

2.3.9 Scoring Stage

In the scoring stage, valid candidates are scored based on their relevance to the user. Precision in this stage is crucial, more features including user, item, and session features are added that wouldn't have been possibly added in the retrieval stage due to the computational and latency constraints since the retrieval stage processes a large number of items. Moreover, a more expressive model can be used in this stage such as deep neural networks. [18]

Ranking items can be expressed as either a classification problem or a dedicated learning-to-rank problem. In deep learning applications, the final output layer either employs a softmax function to generate preference scores for a set of items or utilizes a sigmoid function to estimate the probability of user interaction (such as clicking or purchasing) for each possible user-item combination. [18]

2.3.10 Ordering Stage

In the ordering stage, We narrow down the choices to the best K options, then adjust their order based on specific business needs. For instance, a product category or manufacturer promotion might influence their ranking. [12]

Chapter 3

Requirements And Literature Review

Contents

3.1 Functional Requirements

The system should provide a RESTful API as the final interface to be used by the front-end application. The API provides endpoints that allow inserting customers, products, and interactions. In addition to endpoints for retrieving the recommendations for a given customer.

3.2 System Requirements

In order for the system to be useful it has to meet the following specifications:

3.2.1 Scalability

Scalability implies that it has to be cloud-native, the inference system should apply proper load balancing across multi-node, multi-model deployments.

3.2.2 Real-time Predictions

To be usable in any website or application, the system should be able to provide real-time predictions, and suggestions, with a few milliseconds latency.

To fulfill this requirement, trained models should run on optimized inference servers or services, the suggested deployment plan is to use **Nvidia Triton** [1]. inference server [19], integrated with **Amazon SageMaker** model deployment [20] as infrastructure.

3.2.3 Near Real-time Training

This implies continuous training and deployment of the model which requires the automation of training and deployment.

3.2.4 Elasticity And Optimization

Elasticity is vital for keeping up with traffic spikes and declines while optimizing infrastructure costs. To achieve this, the system should be able to scale up and down based on the traffic and load.

3.2.5 Security

Like any other system, the system has to be immune to security threats by implementing best practices at every level in the deployment and design.

For example, rate-limiting requests to interaction injection endpoints, using attestation when possible, and limiting access to user and product create, read, update, and delete (CRUD) operations.

3.3 Related Work

There are many open-source and paid solutions that provide recommendation systems and libraries, this section discusses some of them.

3.3.1 LightFM

A Python library that enables the classic matrix factorization techniques to include metadata about both items and users, incorporating both content and collaborative information into the recommendation process (hybrid)[21].

Its approach is described with more depth in the LightFM paper [22].

3.3.2 Rexy

Rexy [23] is a Python library that provides a general-purpose recommendation system framework. It is flexible and can be adapted to a variety of data schemas [23].

[1]Nvidia Triton Inference Server, part of the Nvidia AI platform and available with Nvidia AI Enterprise, is open-source software that standardizes AI model deployment and execution across every workload [19]

3.3.3 Gorse

Gorse [24] is an open-source recommender system engine implemented in Go that provides a scalable and flexible recommendation system framework. It supports a variety of algorithms, including collaborative filtering, content-based filtering, and deep learning [23].

3.3.4 AWS Personalize

According to Amazon Web Services (AWS), developers can utilize Amazon Personalize [8] to rapidly create and implement customized recommendations and sophisticated user segmentation on a large scale through machine learning (ML). This service is adaptable to individual requirements, enabling the delivery of personalized customer experiences at the optimal moment and location[2].

3.3.5 Google Recommendations AI

As outlined by Google on their cloud marketplace[3], Recommendations AI [6] allows individuals to develop a fully personalized recommendation system utilizing state-of-the-art deep learning ML models, without requiring expertise in ML or recommendation systems.

3.3.6 Nvidia Merlin

Nidia defines Merlin [26] as "an open source library providing end-to-end GPU-accelerated recommender systems, from feature engineering and preprocessing to training deep learning models and running inference in production." [4].
The frameworks, which was used in this project, provide many components including:

- Merlin NVTabular[28]

  > "Merlin NVTabular is a feature engineering and preprocessing library designed to effectively manipulate terabytes of recommender system datasets and significantly reduce data preparation time."[5]

- Merlin Models[29]

  > "Merlin Models library provides standard models for recommender systems with an aim for high-quality implementations that range from classic machine learning models to highly advanced deep learning models."[6]

- Merlin HugeCTR [31]
- Merlin Transformers4Rec [32]

[2]AWS description of the service [8]
[3]Google Cloud Marketplace Recommendations AI Page[25]
[4]Nvidia Merlin Repository [27]
[5]NVTabular[28]
[6]Merlin Models Documentation[30]

- Merlin SOK (SparseOperationsKit)
- Merlin Distributed Embeddings (DE)
- Merlin Systems [33]

Making it a very customizable and extensible solution.

Table 3.1: Comparison of Recommendation Solutions

System	**LightFM**
License	Apache 2.0
Algorithm Type	Matrix Factorization
Hardware Utilization	CPU
Deployment Readiness	Library (Additional Components Needed)
Notes	-
System	**Rexy**
License	MIT
Algorithm Type	Matrix Factorization
Hardware Utilization	CPU
Deployment Readiness	Library (Additional Components Needed)
Notes	-
System	**Gorse**
License	Apache 2.0
Algorithm Type	Matrix Factorization
Hardware Utilization	CPU
Deployment Readiness	Single-node-learning multi-node-inference cluster
Notes	Unreliable and has many bugs
System	**AWS Personalize**
License	Proprietary
Algorithm Type	DLRM
Hardware Utilization	-
Deployment Readiness	A lot of customization required
Notes	High customization, predictions, and training fees
System	**Google Recommendations AI**
License	Proprietary
Algorithm Type	DLRM
Hardware Utilization	-
Deployment Readiness	End-to-End service
Notes	High predictions, and training fees
System	**Nvidia Merlin**
License	Apache 2.0
Algorithm Type	Multiple Options
Hardware Utilization	Optimized for Nvidia GPUs
Deployment Readiness	Recommendation pipelines components
Notes	Very customizable

Chapter 4

Solution

Contents

The main goal of this solution is to develop a recommendation system for an e-commerce platform. In this context, users are referred to as customers or buyers (representing individuals interacting with the platform which can be through a website or an app), while products are referred to as items (representing the goods offered for sale).

To operate all the stages of the recommendation pipeline and the system components, a lot of infrastructure and DevOps work is required, from data ingestion to the deployment of the components. The following sections will explain the system components and the deployment and technologies used in each stage of the recommendation pipeline.

The system components can be divided into the following categories:

- API gateway
- Storage Components
- Recommendation Pipeline
- Caching Layer
- Automation and DevOps

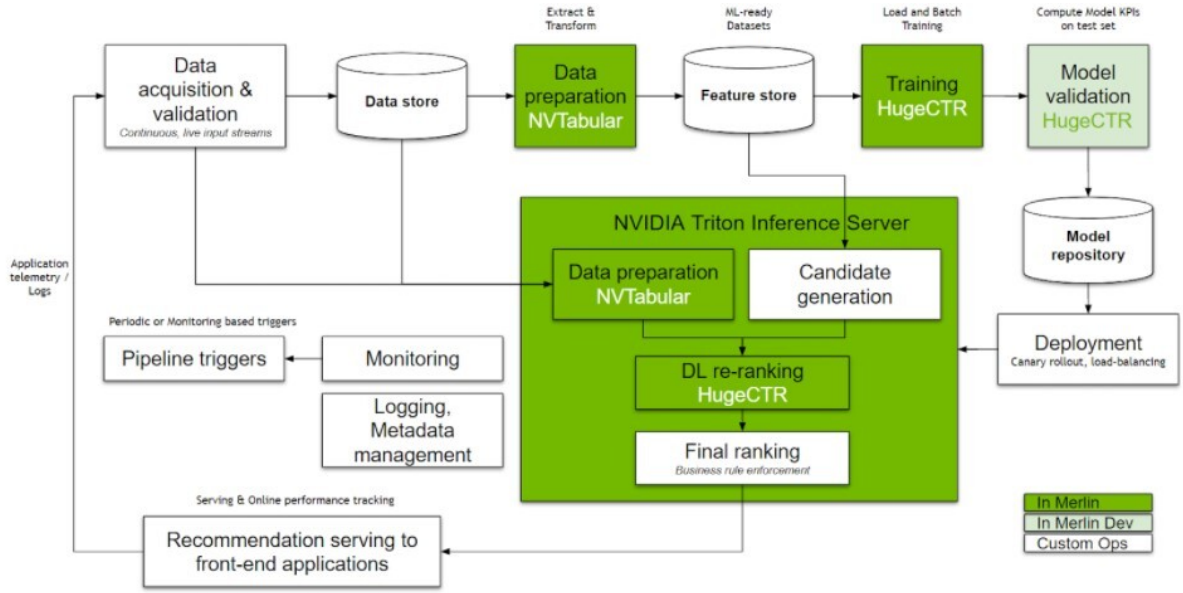


Figure 4.1: System Components [12]

Each component is explained in the following sections.

4.1 API Gateway

The API gateway, a RESTful API, is the entry point for the recommendation system. It is responsible for handling all the requests and responses from the customers. The main two functionalities of the API gateway are data ingestion and getting recommendations for a customer.

In addition to these functionalities, the API gateway is also responsible for authenticating and authorizing the requests.

4.1.1 Data Ingestion Endpoints

To handle the data ingestion, the API gateway exposes the following endpoints:

- Create, Read, Update, Delete customers
- Create, Read, Update, Delete products
- Adding interactions between customers and products

Such endpoints interact mainly with the storage components to store the data.

4.1.2 Recommendation Endpoints

To get recommendations for a customer, the API gateway exposes the following endpoint:

- Get recommendations for a customer
- Get similar products to a product

Those endpoints interact mainly with the recommendation pipeline and the caching layer to get the recommendations.

The first endpoint queries the caching layer to get the recommendations, and if the offline recommendations are not found in the caching layer, the API gateway queries the recommendation pipeline to get online recommendations, and then stores the recommendations in the caching layer, after getting the recommendations from the recommendation pipeline, the API gateway orders the recommendations using the output of the ranking stage and using business rules, then returns the top recommendations to the customer.

In the second endpoint, the API gateway queries the feature store to get the embeddings of the product and then executes an Approximate Nearest Neighbor (ANN) search to get similar products.

4.2 Storage Components

The storage components are responsible for storing the data in different formats and stages, in addition to the models used in the recommendation system. There are four main storage components in the system:

- Main Database

  The main database is an SQL database that stores the customers, products, and interactions data. In this project, a PostgreSQL [34] instance deployed using AWS RDS [1] is used as the main database. But in a bigger production environment, a distributed Data Warehouse such as Amazon Redshift [36] or Google BigQuery [37] can be used.

- Online Feature Store

  The feature store is a storage system that stores data in a format optimized for machine learning models. [15] It also stores the embedding tables of the customers and products, in addition to any other categorical features. This project uses FEAST [38] as the feature store, which supports using a Redis [39] cluster as an Online store, the Redis cluster is deployed using AWS ElastiCache [40].

- Item Embedding Store

  A vector database that stores the embeddings of the products while allowing them to be queried using an Approximate Nearest Neighbor (ANN) search algorithm. In this project, the item embedding store is a Redis [39] cluster with the RediSearch [41] module deployed using AWS ElastiCache [40].

- Model Repository

  After training the models, their parameters are stored in the model repository. In this project, the model repository is an AWS Simple Storage Service (S3) bucket [42].

[1] Amazon Web Services Relational Database Service [35]

- Results Store

  To implement offline batch recommendations, the results of running the recommendation pipeline periodically for all users are stored in the results store. In this project, the results store is a Redis cluster deployed using AWS ElastiCache [40].

4.3 Recommendation Pipeline

The recommendation pipeline is the core of the recommendation system. Deploying a recommendation pipeline requires an accelerated infrastructure optimized for deep learning and several MlOps workflows.

It starts with the candidate generation stage, where a set of thousands of candidate products for each customer is chosen among millions of products. Then the filtering stage filters the candidate items according to business rules and constraints, the scoring stage ranks the candidate items and finally the ordering stage orders the scored items and selects the top items to be recommended to the customer.

4.3.1 Candidate Generation (Retrieval)

To generate the candidate items for each customer, the system starts by retrieving the embeddings of the customer and the products from FEAST (the feature store).

Two Tower Model (Offline Retrieval)

During training (offline), the system uses a two-tower model, where the customer and product embeddings are produced through two separate towers, and the output of the two towers is used to compute the similarity between the customer and the products.

The products' embeddings are stored in a vector database, which is the item embedding store.

The training of the two-tower model is done using a distributed training framework, in this case, it was done using Merlin HugeCTR [43] deployed on AWS SageMaker [44] inside a Merlin HugeCTR container [45].

ANN (Online Retrieval)

During inference (online), using two separate towers to retrieve the candidate items for each customer is not efficient. [15] To reduce the latency of the responses, the system uses only the customer tower to generate the embeddings of the customers, after that, an ANN search algorithm is used to retrieve the candidate products from the item embedding store.

In production, the user tower is deployed using Nvidia Triton Inference Server [19] on AWS SageMaker [44] inside a Merlin Inference container [46] from Nvidia's NGC [47].

4.3.2 Rule Based Filtering (Filtering)

After retrieving the candidate products for each customer, the system filters the candidate products according to business rules and constraints. In an e-commerce platform,

the business rules and constraints usually include the availability of the products, the customer's location and shipping availability to it.

The filtering workflow is implemented using NVTabular [28], and the workflow is stored in the model repository.

In production, the filtering workflow is deployed with the deep learning ranking model in the same container.

4.3.3 Deep Learning Ranking Model (Scoring)

This stage is the core of the recommendation pipeline, where the system ranks the candidate and filtered products and selects the top products to be recommended to the customer.

To rank the model, a deep learning ranking model is trained using the historical data of the interactions between the customers and the products. Each categorical feature is replaced with its embedding and those embeddings are stored in the feature store during offline training. After training the model, the model's parameters are stored in the model repository.

The model's output is the ranking score of each product for each customer, which is the probability of the customer interacting with the product.

The ranking model that was used in this project is the Deep Learning Recommendation Model (DLRM) [48] proposed by Facebook.

Instead of implementing the DLRM from scratch, the system uses Merlin Models [29] implementation of the DLRM, which is optimized to run accelerated on Nvidia GPUs using the HugeCTR [43] framework.

The DLRM is trained using a distributed training framework, in this case, it was done using Merlin HugeCTR [43] deployed on AWS SageMaker [44] inside a Merlin HugeCTR container [45] from Nvidia's NGC [47].

In production, the DLRM is deployed using Nvidia Triton Inference Server [19] on a Merlin Inference container [46] from Nvidia's NGC [47]. AWS SageMaker [44] automatically scales the number of instances of the Triton Inference Server based on the load.

4.3.4 E-commerce ordering logic (Ordering)

After getting the ranking score of each product for the customer, the system orders the products using the ranking score and business rules, then selects the top products to be recommended to the customer.

In the context of an e-commerce platform, the ordering logic includes promoting products (sponsored products), where the sponsored products are put on top of the list if they exceed a certain threshold score.

The following equation represents the final score of the product which is used to order the products:

New score = 1.0 if (IsPromoted AND RankingScore >= Threshold) else RankingScore

The ordering stage is implemented in the API gateway since the number of products to

be ordered after the ranking is very small.

4.4 Storing Results

To enhance the performance of the recommendation system, the results of online recommendations are stored in an in-memory caching layer. In addition to that, the results of running the recommendation pipeline periodically for all users are stored in the results store.

This is done to provide the recommendations with ultra-low latency and to provide the recommendations in case of a failure in the recommendation pipeline or overload.

4.5 Automation and MlOps

To keep the recommendation system running and to keep the models up to date, a lot of automation and DevOps work is required. The main task that needs to be automated is the periodic training of the models and the deployment of the models and the workflows.

To run periodic tasks a cloud-native service similar to cron jobs is required, in this project, AWS Batch [49] was used. AWS Batch allows running computing workloads on the cloud, in an automated and scalable way.

The job should respawn the training environment and run the training workflow, then store the model's parameters and the workflow in the model repository, in addition to storing the embeddings in the feature store.

Chapter 5

Experiment And Results

Contents

5.1 Dataset

The dataset used in this experiment is called AliCPP and it is a public dataset that contains user-item interactions from an e-commerce platform. The dataset gather traffic logs of recommendation systems in Taobao, a Chinese online shopping website. The dataset is collected from the real-world recommendation system of Taobao.[50]. The dataset contains of 4 types of features: user features, item features, combination features and context features, each feature is represented by an unique ID, and they described as follows:

Feature Category	Feature ID	Feature Description
User Features	101	User ID
	109_14	User historical behaviors of category ID and count
	110_14	User historical behaviors of shop ID and count
	127_14	User historical behaviors of brand ID and count
	150_14	User historical behaviors of intention node ID and count
	121	Categorical ID of User Profile
	122	Categorical group ID of User Profile
	124	Users Gender ID
	125	Users Age ID
	126	Users Consumption Level Type I
	127	Users Consumption Level Type II
	128	Users Geography Informations
	129	Users Geography Informations
Item Features	205	Item ID
	206	Category ID to which the item belongs to
	207	Shop ID to which item belongs to
	210	Intention node ID which the item belongs to
	216	Brand ID of the item
Combination Features	508	The combination of features with 109_14 and 206
	509	The combination of features with 110_14 and 207
	702	The combination of features with 127_14 and 216
	853	The combination of features with 150_14 and 210
Context Features	301	A categorical expression of position

Table 5.1: Features Description

5.2 Experimental Setup

In order to evaluate the performance of the proposed solution with real-world data, the environment has to be accelerated with a GPU.

5.2.1 Hardware Environment

To achieve this, the experiment is conducted on a cloud-based environment, specifically on an AWS EC2 p3.2xlarge instance [51].

The instance is equipped with 8 vCPUs, 61 GiB of memory, and a single NVIDIA Tesla V100 GPU.

The V100 is a high-end GPU that is designed for AI workloads especially deep learning tasks. It is based on the Volta architecture and is equipped with 5120 CUDA cores, 640 Tensor cores, and 32GB of HBM2 memory with 1.1 TB/s. It can deliver 7 TFLOPS of double-precision floating-point performance and 116 TFLOPS of deep learning performance.

Figure 5.1 shows the performance comparison between the Tesla V100 and a CPU.

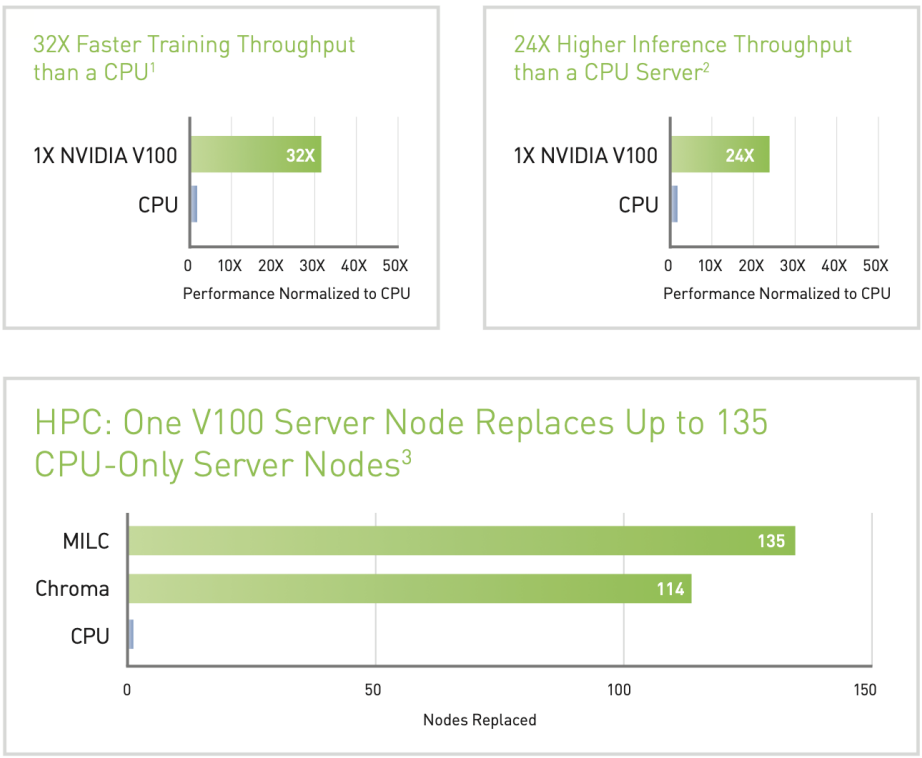


Figure 5.1: Tesla V100 vs CPU [52]

5.2.2 Software Environment

In addition to the hardware environment, the software environment is also important to be considered. Having access to **Nvidia Inception Program** [53], which includes access to NVIDIA AI Enterprise [54] with **Nvidia GPU Cloud (NGC) Catalog**[47].

Instead of setting up the drivers, tools, libraries and frameworks manually, the Nvidia AI Enterprise provides a pre-configured flavor of Linux Ubuntu that includes all necessities for deep learning tasks.

In addition to that, instead of compiling the Merlin TensorFlow from the source code, the Nvidia Merlin TensorFlow Container [55] was used as a runtime for the experiment Jupyter notebook.

5.3 Evaluation Results

solution is conducted using the AliCPP dataset with various models such as Multi-Layer Perceptron (MLP), Deep Learning Recommendation Model (DLRM), Deep & Cross Network (DCN), Wide&Deep, and the Two-Tower model. The evaluation metric used is Area Under the ROC Curve (AUC). Furthermore, we studied the impact of the number of epochs on the training and validation loss and Area Under the ROC Curve (AUC), using Binary Cross-Entropy (BCE) as a loss function.

5.3.1 AUC Comparison

The Area Under the Curve (AUC) metric describes a model's ability to separate positive and negative instances. A higher AUC indicates stronger distinction between the two classes. Where the AUC value ranges from 0 to 1, where 0.5 is the random guess and 1 is the perfect model.

The AUC metric is used to evaluate the performance of the models on the AliCPP dataset. The results are shown in Figure 5.2.

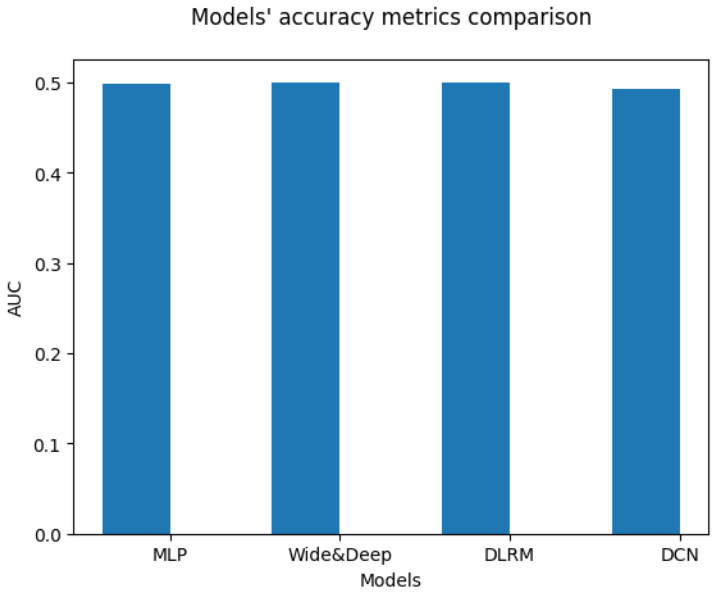


Figure 5.2: AUC Comparison

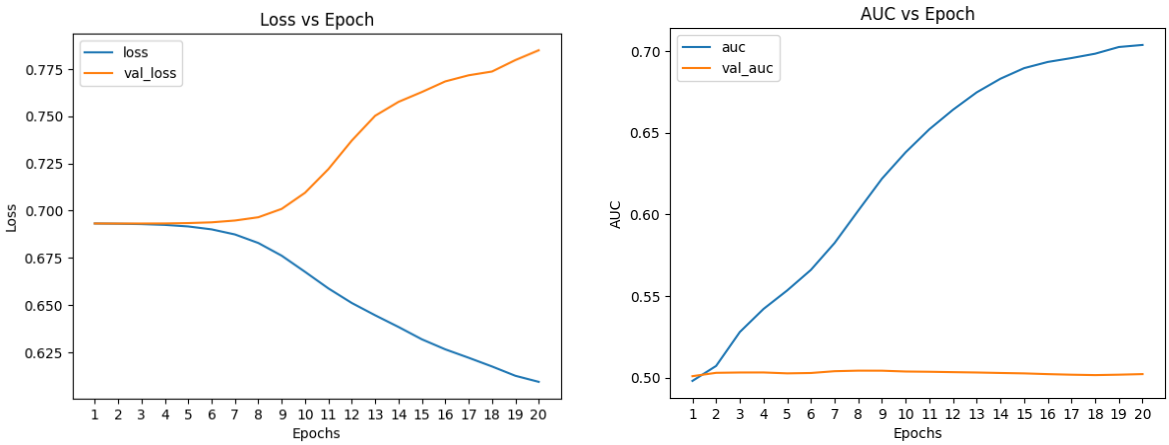
The results show that all the models had a similar performance with AUC values around 0.5, which indicates that the models performed similarly.

Recommending relevant items from millions of possibilities is a significant challenge. Although an AUC score of 0.5 might seem underwhelming, considering it translates to a 50% chance of the user clicking on a recommended item becomes more impressive. Since systems typically show dozens of items, even a single click signifies the success of the system.

5.3.2 Training and Validation

To study the effect of the number of epochs on the training and validation loss and AUC, we trained the models using BCE as a loss function and plotted the AUC, BCE loss for both training and validation datasets over the number of epochs. The results are shown in Figure 5.3a and Figure 5.3b.

The results show that the training loss kept decreasing over the number of epochs, while the validation loss kept increasing, which indicates that the model is overfitting the training data.



(a) Loss Comparison

(b) AUC Over Epochs

Figure 5.3: Measures Over Epoches

Chapter 6

Conclusion and Future Work

Contents

6.1 Conclusion

In conclusion, the development and deployment of a cutting-edge accelerated e-commerce deep learning recommendation system present both challenges and opportunities. Through the exploration of various recommendation system types, it is evident that deep learning recommendation models (DLRMs) offer promising potential in terms of scalability and performance. However, the evaluation on the AliCPP dataset suggests that while these models perform similarly in terms of AUC metrics, they still face the challenge of effectively recommending relevant items from vast databases.

Despite the seemingly modest AUC scores around 0.5, it's crucial to contextualize these results within the practical implications of recommendation systems. A 50% chance of user interaction with recommended items demonstrates considerable success, especially considering the volume of choices users encounter. Even a single click on a recommended item signifies the system's effectiveness in guiding user engagement.

Moving forward, the focus lies on the efficient deployment and automation of the recommendation pipeline, including data injection, training, and serving recommendations through a simple RESTful API interface. Optimization of ranking and retrieval models remains paramount to enhance the system's accuracy and user satisfaction.

In essence, while there are challenges ahead, the project's dedication to leveraging state-of-the-art technologies and methodologies promises to deliver a robust and scalable e-commerce recommendation system poised to meet the evolving needs of users and businesses alike.

6.2 Future Work

There are many possible improvements and future work that can be done to the system, some of them are related to the model and data, and others are related to the system architecture and infrastructure.

The following sections will discuss some of the possible future work and improvements that can be made to improve the system.

6.2.1 Deeper Product Features

One of the main limitations of the current system is the lack of product features related to the product's content, such as the product's description, title, and images. Adding these features to the model can improve the recommendation system's accuracy and performance. But to do that, the system needs to be able to process and analyze the product's content, which requires additional layers of data processing and feature extraction.

As an example, to extract features from the product's name and description, the system can use natural language processing (NLP) techniques to extract embeddings representing the product's content, and then use these embeddings as input features to the model.

For images, the system can use computer vision techniques, such as convolutional networks, to extract features from the product's images, and then use these features as input to the model.

6.2.2 Increasing Interactions Data

The current system recommends products solely based on a single type of interaction, which is the user's click history. Adding more types of interactions, such as the user's purchase history, the user's rating history, and the user's browsing history, might improve the recommendation system's accuracy and performance.

Moreover, some interaction types might be converted to numeric features for further analysis, for example, a user opens the product page, and stays for a certain amount of time, that time can be used as a feature to represent the user's interest in the product.

6.2.3 Integrating with Ecommerce Platforms

The current system is a standalone system that provides an API for the client to interact with. To make the system more accessible and easier to use, it can be integrated with popular e-commerce platforms, such as Shopify, Magento, and WooCommerce. Such integration can be done by providing plugins or extensions that can be installed on the e-commerce platform, and then the plugin can communicate with the system's API to get recommendations for the platform's users.

6.2.4 Session-based Recommendations

The current system requires re-training to update the model with new data, and it does not take into account the user's current session or context. So the customer's interactions do not affect the recommendations until the next re-training cycle.

A major improvement to the system is to add a session-based recommendation model that

can provide real-time recommendations based on the user's current session and context. Such a model can utilize Recurrent Neural Networks (RNN) with Long Short-Term Memory (LSTM) to capture the user's session and context without requiring prior knowledge of the interactions during training.

One possible library to use for this purpose is Merlin Transformer4Rec [56], which is a flexible and efficient library for sequential and session-based recommendation.[56]

Bibliography

[1] Raghavendra C K, Srikantaiah K.C, Venugopal K. R, "Personalized Recommendation Systems (PRES): A Comprehensive Study and Research Issues," *International Journal of Modern Education and Computer Science (IJMECS)*, vol. 10, no. 10, pp. 11–21, 2018. DOI: 10.5815/ijmecs.2018.10.02.

[2] Nvidia Glossary, "Recommendation System." `https://www.nvidia.com/en-us/glossary/data-science/recommendation-system/`. Accessed: 2023-12-19.

[3] M. J. Pazzani and D. Billsus, "Content-based recommendation systems," in *The adaptive web: methods and strategies of web personalization*, pp. 325–341, Springer, 2007.

[4] Salesforce Marketing Cloud, "Predictive intelligence benchmark report." `https://brandcdn.exacttarget.com/sites/exacttarget/files/deliverables/etmc-predictiveintelligencebenchmarkreport.pdf`, 2014.

[5] Google Cloud Summit, "Ikea's approach to building a powerful recommendations engine." by Google Cloud Events `https://www.youtube.com/watch?v=PyjC0wRRtBg`, 2021.

[6] Google Cloud, "Google Recommendations AI." `https://cloud.google.com/recommendations`. Accessed: 2023-11-9.

[7] Sungoh Park and Kyoungtae Hwang, "Increasing customer engagement and loyalty with personalized coupon recommendations using Amazon Personalize," *AWS Machine Learning Blog*, 2020.

[8] AWS, "AWS Personalize." `https://aws.amazon.com/personalize/`, accessed: 2023-10-4.

[9] R. Wang, R. Shivanna, D. Z. Cheng, S. Jain, D. Lin, L. Hong, and E. H. Chi, "Dcn v2: Improved deep & cross network and practical lessons for web-scale learning to rank systems," *arXiv preprint arXiv:2008.13535*, 2020.

[10] Nvidia, "Exploring Different Models." `https://github.com/NVIDIA-Merlin/models/blob/main/examples/03-Exploring-different-models.ipynb`, accessed: 2024-2-1.

[11] M. AI, "Two stage Recommendation System." `https://medium.com/mlearning-ai/building-a-multi-stage-recommendation-system-part-1-1-95961ccf3dd8`, accessed: 2024-2-5.

[12] Nvidia, "Recommendation Systems Best Practices." https://docs.nvidia.com/deeplearning/performance/recsys-best-practices/index.html, accessed: 2024-2-5.

[13] Saturn Cloud, "Feature Embedding." https://saturncloud.io/glossary/feature-embedding/#:~:text=What%20is%20Feature%20Embedding%3F,to%20vectors%20of%20real%20numbers, accessed: 2024-2-7.

[14] Rahul Agarwal, "Dealing with Categorical Variables by using Target Encoder." https://towardsdatascience.com/dealing-with-categorical-variables-by-using-target-encoder-a0f1733a4c69, accessed: 2024-2-7.

[15] S. Partee, T. Hutcherson, and N. Stephens, "Offline to online: Feature storage for real-time recommendation systems with nvidia merlin," 2024.

[16] X. Yi, J. Yang, L. Hong, D. Z. Cheng, L. Heldt, A. Kumthekar, Z. Zhao, L. Wei, and E. Chi, "Sampling-bias-corrected neural modeling for large corpus item recommendations," in *Proceedings of the 13th ACM Conference on Recommender Systems*, RecSys '19, (New York, NY, USA), p. 269–277, Association for Computing Machinery, 2019.

[17] Balu Ertl, "A Voronoi diagram." https://en.wikipedia.org/wiki/Proximity{_}analysis#/media/File:Euclidean{_}Voronoi{_}diagram.svg, accessed: 2024-2-7.

[18] Z. Yan, "System design for recommendations and search," *eugeneyan.com*, Jun 2021.

[19] Nvidia, "Nvidia Triton Inference Server." https://developer.nvidia.com/triton-inference-server, accessed: 2023-10-4.

[20] AWS, "Amazon SageMaker Model Deployment." https://aws.amazon.com/sagemaker/deploy/, accessed: 2023-10-4.

[21] "LightFM." https://making.lyst.com/lightfm/docs/home.html, accessed: 2023-10-8.

[22] M. Kula, "Metadata embeddings for user and item cold-start recommendations," 2015.

[23] "Rexy." https://github.com/kasraavand/Rexy, accessed: 2023-10-8.

[24] "Gorse." https://github.com/gorse-io/gorse, accessed: 2023-10-8.

[25] "Google Cloud Marketplace - Recommendation AI." https://console.cloud.google.com/marketplace/product/google/recommendationengine.googleapis.com, accessed: 2023-12-20.

[26] Nvidia, "Merlin." https://developer.nvidia.com/merlin, accessed: 2023-10-6.

[27] Nvidia, "Merlin Repository." https://github.com/NVIDIA-Merlin/Merlin, accessed: 2023-10-6.

[28] Nvidia, "Merlin NVTabular." https://developer.nvidia.com/nvidia-merlin/nvtabular, accessed: 2023-10-6.

[29] Nvidia, "Merlin Models Repository." https://github.com/NVIDIA-Merlin/models, accessed: 2023-10-6.

[30] Nvidia, "Merlin Models Documentation." https://nvidia-merlin.github.io/models/stable/README.html, accessed: 2024-1-23.

[31] Nvidia, "Merlin HugeCTR." https://developer.nvidia.com/nvidia-merlin/hugectr, accessed: 2023-10-6.

[32] Nvidia, "Merlin Transformers4Rec." https://github.com/NVIDIA-Merlin/Transformers4Rec, accessed: 2023-10-6.

[33] Nvidia, "Merlin Systems Repository." https://github.com/NVIDIA-Merlin/systems, accessed: 2023-10-6.

[34] PostgreSQL, "PostgreSQL." https://www.postgresql.org/, accessed: 2024-2-1.

[35] Amazon Web Services, "Amazon RDS." https://aws.amazon.com/rds/, accessed: 2024-2-1.

[36] Amazon Web Services, "Amazon Redshift." https://aws.amazon.com/redshift/, accessed: 2024-2-1.

[37] Google Cloud, "Google BigQuery." https://cloud.google.com/bigquery, accessed: 2024-2-1.

[38] Feast, "Feast." https://feast.dev, accessed: 2024-2-1.

[39] Redis, "Redis." https://redis.io/, accessed: 2024-2-1.

[40] Amazon Web Services, "Amazon ElastiCache." https://aws.amazon.com/elasticache/, accessed: 2024-2-1.

[41] Redis, "RediSearch Vector Search." https://redis.io/docs/interact/search-and-query/, accessed: 2024-2-1.

[42] Amazon Web Services, "Amazon S3." https://aws.amazon.com/s3/, accessed: 2024-2-4.

[43] Nvidia, "Nvidia Merlin HugeCTR." https://developer.nvidia.com/nvidia-merlin/hugectr, accessed: 2024-2-4.

[44] Amazon Web Services, "Amazon SageMaker." https://aws.amazon.com/sagemaker/, accessed: 2024-2-4.

[45] Nvidia, "Nvidia Merlin HugeCTR Container." https://catalog.ngc.nvidia.com/orgs/nvidia/teams/merlin/containers/merlin-hugectr, accessed: 2024-2-4.

[46] Nvidia, "Nvidia Merlin Inference Container." https://catalog.ngc.nvidia.com/orgs/nvidia/teams/merlin/containers/merlin-inference, accessed: 2024-2-4.

[47] Nvidia, "Nvidia NGC Catalog." https://ngc.nvidia.com/, accessed: 2024-2-5.

[48] M. Naumov, D. Mudigere, H.-J. M. Shi, J. Huang, N. Sundaraman, J. Park, X. Wang, U. Gupta, C.-J. Wu, A. G. Azzolini, D. Dzhulgakov, A. Mallevich, I. Cherniavskii, Y. Lu, R. Krishnamoorthi, A. Yu, V. Kondratenko, S. Pereira, X. Chen, W. Chen, V. Rao, B. Jia, L. Xiong, and M. Smelyanskiy, "Deep learning recommendation model for personalization and recommendation systems," 2019.

[49] Amazon Web Services, "Amazon Batch." `https://aws.amazon.com/batch/`, accessed: 2024-2-4.

[50] X. Ma, L. Zhao, G. Huang, Z. Wang, Z. Hu, X. Zhu, and K. Gai, "Entire space multi-task model: An effective approach for estimating post-click conversion rate," in *SIGIR 2018-Proceedings of the 41th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 8–12, ACM, July 2018.

[51] AWS, "AWS EC2 P3 Instances." `https://aws.amazon.com/ec2/instance-types/p3/`, accessed: 2024-1-4.

[52] Nvidia, "Nvidia V100 Datasheet." `https://images.nvidia.com/content/technologies/volta/pdf/volta-v100-datasheet-update-us-1165301-r5.pdf`, accessed: 2024-1-4.

[53] Nvidia, "Nvidia Inception Program For Startups." `https://www.nvidia.com/en-us/startups/`, accessed: 2024-1-4.

[54] Nvidia, "Nvidia Ai Enterprise." `https://www.nvidia.com/en-eu/data-center/products/ai-enterprise/`, accessed: 2024-2-5.

[55] Nvidia, "Nvidia Merlin TensorFlow Container." `https://catalog.ngc.nvidia.com/orgs/nvidia/teams/merlin/containers/merlin-tensorflow/`, accessed: 2024-2-5.

[56] Nvidia, "Nvidia Merlin Transformers4Rec Introduction." `https://nvidia-merlin.github.io/Transformers4Rec/main/README.html`, accessed: 2024-2-4.