Silal LTD

Research and Development Team

Accelerated DLRM-based E-commerce Recommendation System Using Nvidia Merlin

Prepared By:

Ibraheem Alyan (CTO)

Mohammad Abu-Shelbaia (Developer)

Nidal Zabade (Developer)

A project report describing the research and development associated with the development of a smart recommendation system for Silal's e-commerce platform. This project was supported by Nvidia through its Inception program.

Jerusalem

July 15, 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 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.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

ANN Approximate Nearest Neighbor

API Application Programming Interface

AUC Area Under the ROC Curve

AWS Amazon Web Services

BCE Binary Cross Entropy

CCP Click and Conversion Prediction

CRUD Create, Read, Update, Delete

CTR Click Through Rate

DCN Deep and Cross Network

DLRM Deep Learning Recommendation Model

DNN Deep Neural Network

EC2 Elastic Compute Cloud

GLM Generalized Linear Model

GPU Graphics Processing Unit

LSTM Long Short-Term Memory

ML Machine Learning

MLP Multi-Layer Perceptron

NDCG Normalized Discounted Cumulative Gain

NGC NVIDIA GPU Cloud

NLP Natural Language Processing

RNN Recurrent Neural Network

ROC Receiver Operating Characteristic

S3 Simple Storage Service

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 an experiment that evaluates the proposed solution 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 ML, 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 as shown in Figure 2.1.

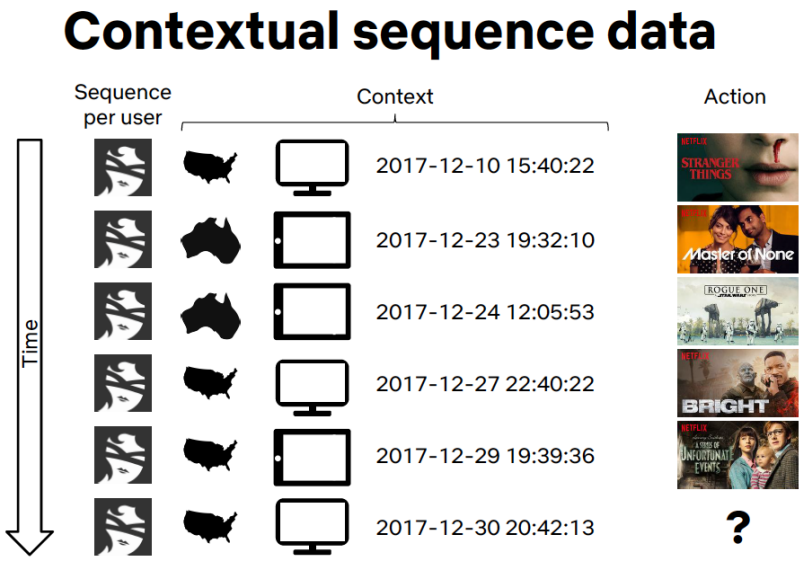


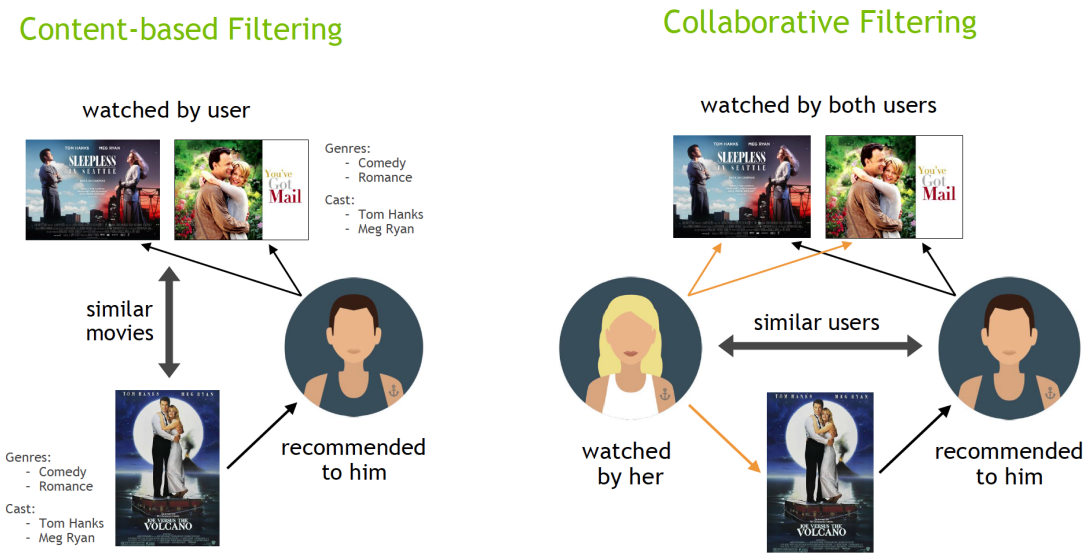
Figure 2.1: Context Filtering Diagram [2]

2.2.2 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. Figure 2.2b shows a diagram of the collaborative filtering technique.

2.2.3 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 with similar features. Figure 2.2a shows a diagram of the content filtering technique.



(a) Content Filtering (b) Collaborative Filtering Diagram

Figure 2.2: Nvidia Glossary Diagram [2]

2.2.4 Hybrid Recommendation Systems

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

2.2.5 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. As shown in Figure 2.3.

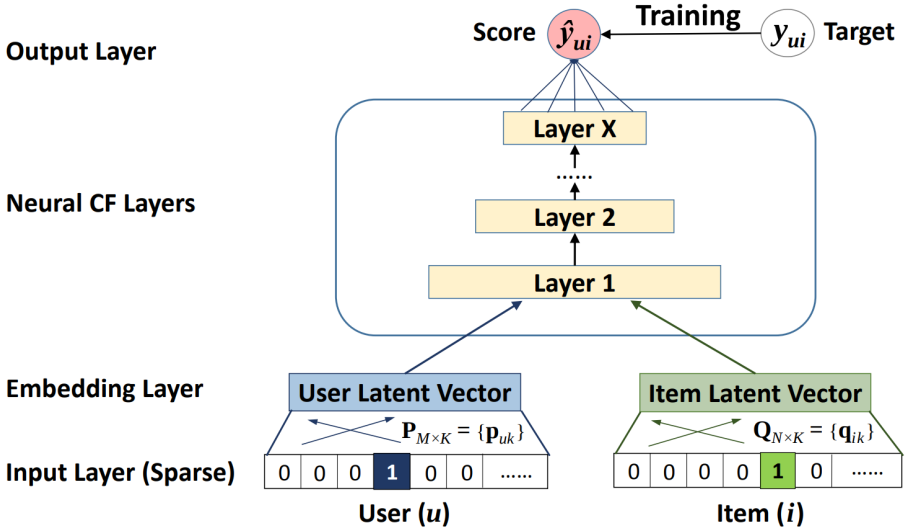


Figure 2.3: Neural Collaborative Filtering [2]

2.2.6 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 RNN. An example use case is session-based recommendations, RNNs predict the next items based on user event sequences, mirroring word embedding in NLP.

2.2.7 Wide And Deep

Wide & Deep is a technique that uses a wide neural network to learn the preferences of the customer and utilizes another DNN to learn the products's features. The wide model is a GLM and the deep model is built from a dense neural Network. The two models are then combined to create a recommendation. As shown in Figure 2.4.

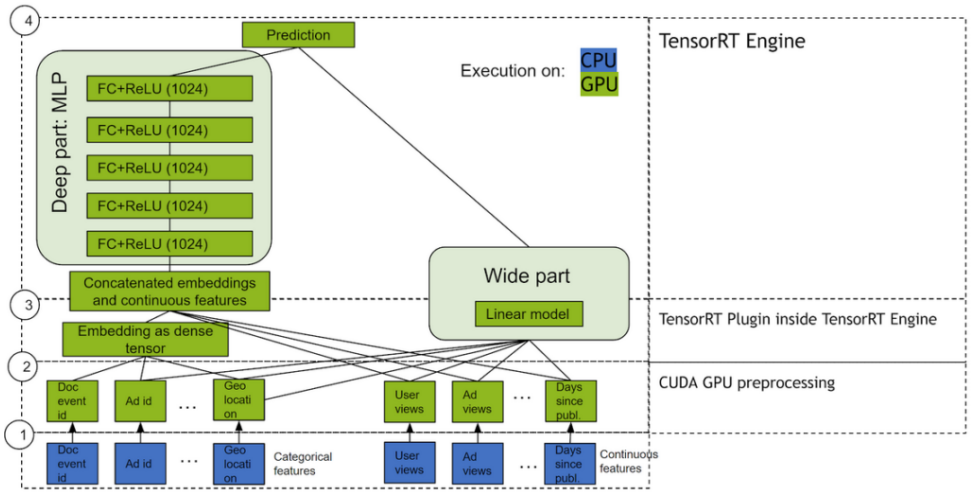
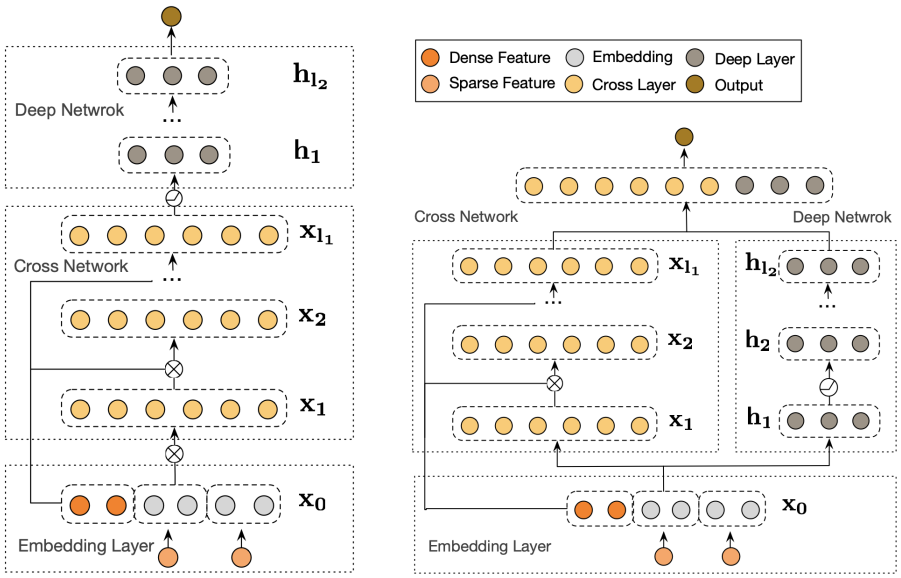


Figure 2.4: Wide Deep Structure [2]

2.2.8 Deep and Cross Network-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. Figure 2.5 shows the structure of the Deep Cross Network. 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 2.5: Deep Cross Network Structure [9]

2.2.9 Multi-layer perceptron

MLP is a technique that uses a DNN 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. As shown in Figure 2.6.

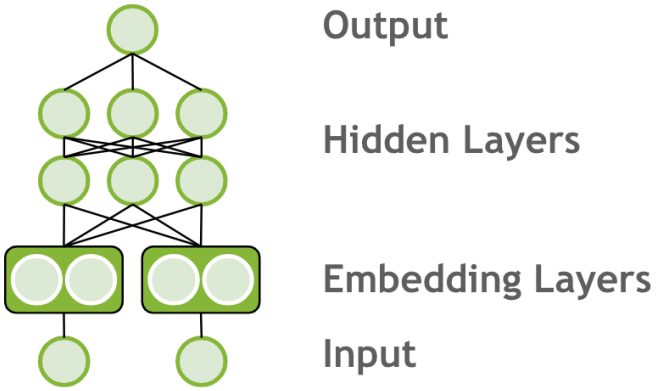


Figure 2.6: MLP Structure [10]

2.2.10 DLRM

DLRM is a technique that uses a DNN 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 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 2.7 shows the structure of the DLRM model.

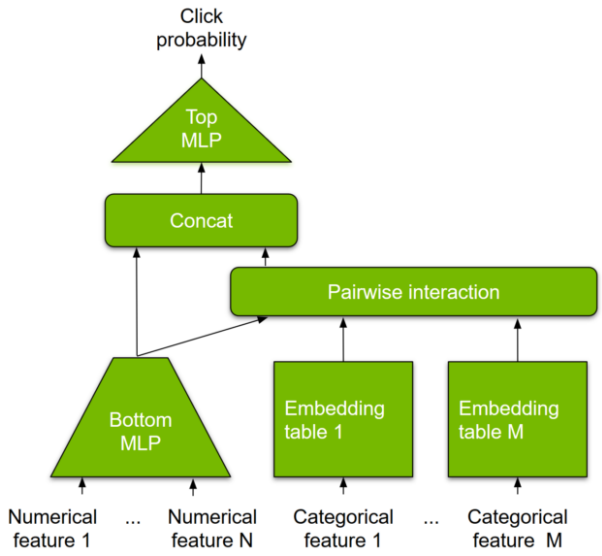


Figure 2.7: DLRM Structure [2]

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 as shown in figure 2.8: 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 ML models such as matrix factorization or deep learning models [11].

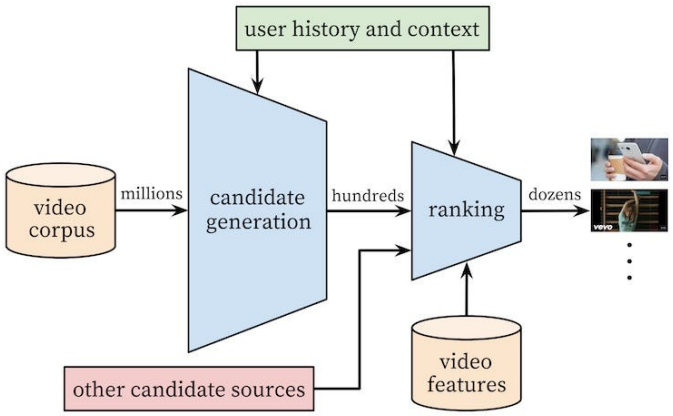


Figure 2.8: Two-stage Recommender System[11]

2.3.2 Four-stage Recommender Systems

Four-stage recommender systems as shown in figure 2.9 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 ML models such as matrix factorization or deep learning models, and the scoring and ordering stages are usually based on ML models such as deep learning models [12].

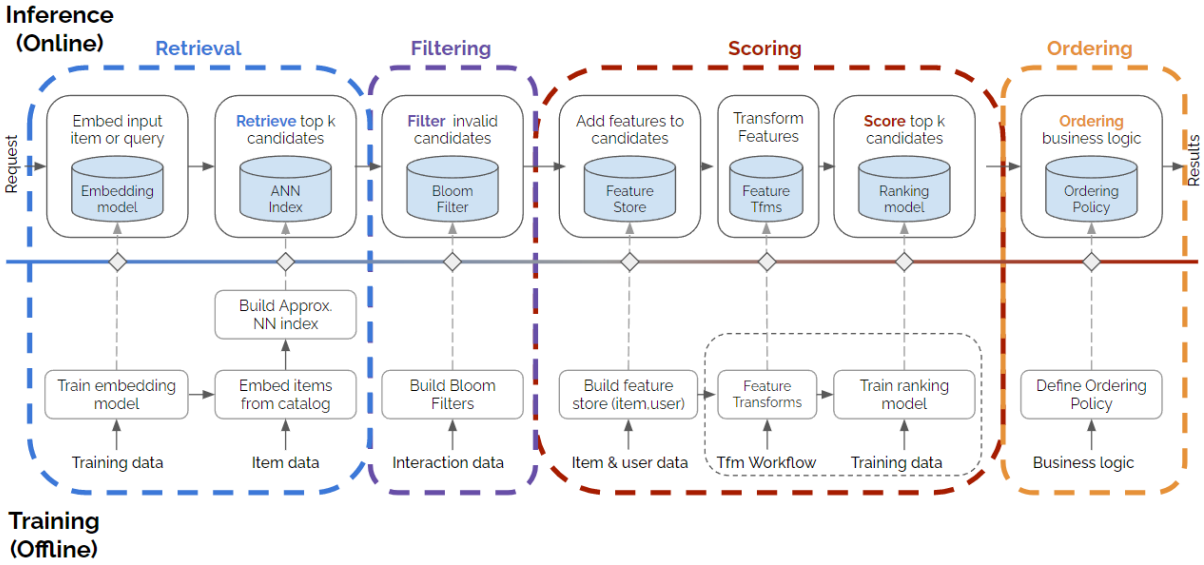


Figure 2.9: Four-stage Recommender System[12]

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 ML 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 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 ML 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 2.10 shows a pipeline suggested by Nvidia[15] for a two-stage online recommendation system.

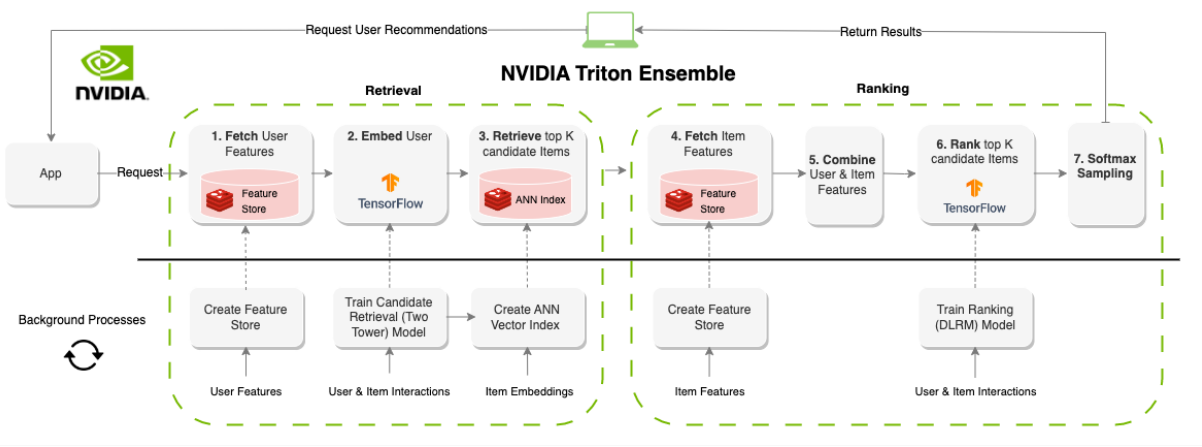


Figure 2.10: Two-stage Online Recommendation System [15]

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 2.11 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.

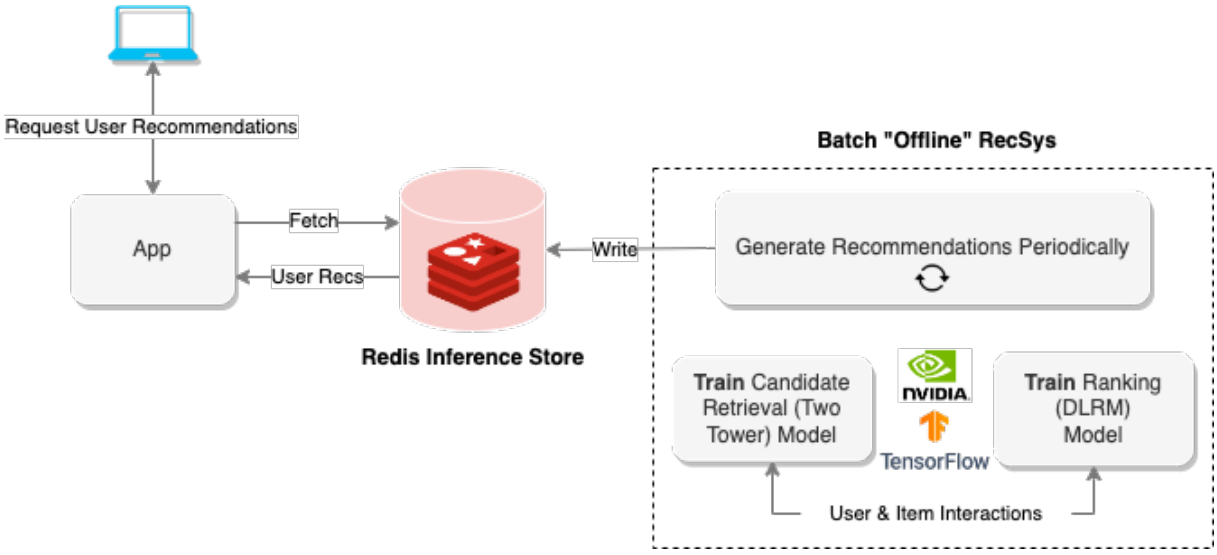


Figure 2.11: Batch Recommendation System [15]

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 ANN search to retrieve the top candidates for each user and use 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 2.12 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 DNN.

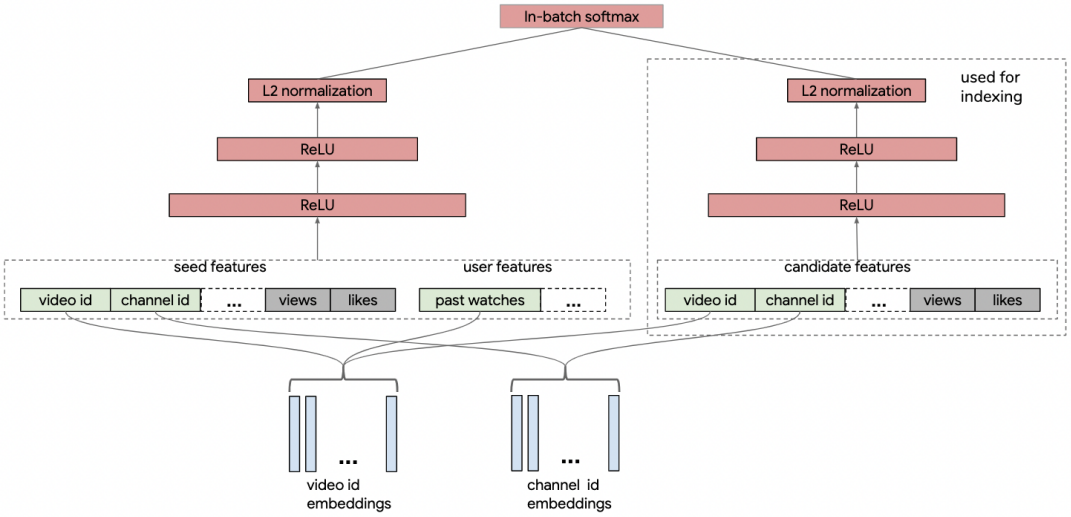


Figure 2: Illustration of the Neural Retrieval Model for YouTube.

Figure 2.12: 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.

Approximate Nearest Neighbor

One popular approach to retrieval uses embedding models. These models create a dense vector space where users and items are represented as points. 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 2.13 illustrates the concept of ANN search

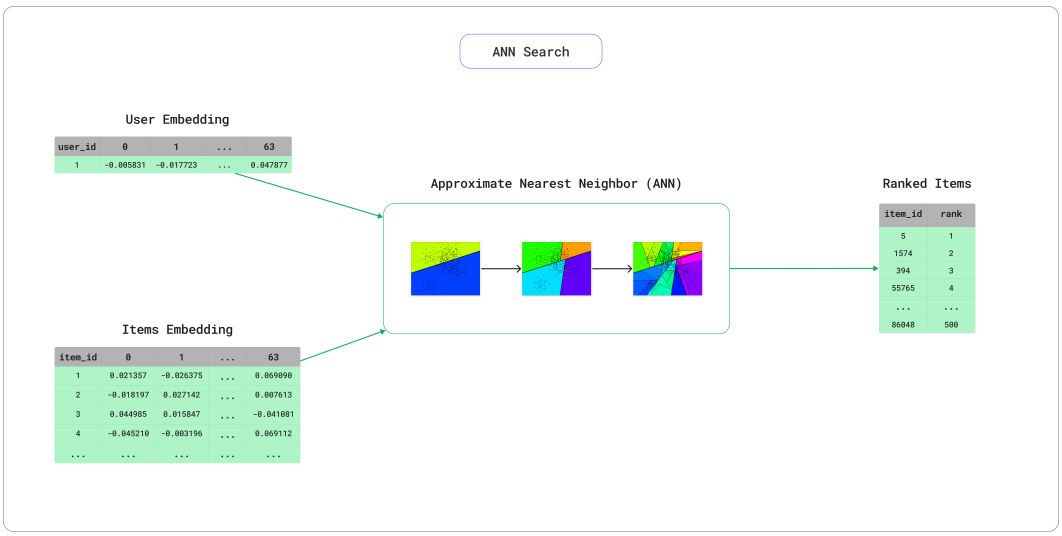


Figure 2.13: Approximate Nearest Neighbor

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 DNN [17].

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 [17].

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].

2.4 Softmax Function

The softmax function is a mathematical function that converts a vector of real numbers into a vector of probabilities that sum to one. The softmax function is used in machine learning models to convert the output of a neural network into a probability distribution over multiple classes. The softmax function is defined as follows:

$$\sigma(\vec{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \tag{2.1}$$

Where:

- $\vec{z}$ is a vector of real numbers.
- z_j is the j^{th} element of the vector $\vec{z}$.
- K is the number of classes.
- $\sigma(z)_j$ is the j^{th} element of the output vector.

Figure 2.14 illustrates the softmax function for a vector of real numbers. Each color

represents an element in the input vector, where the height of the intersection of the line and the curve represents the percentage of the output vector.

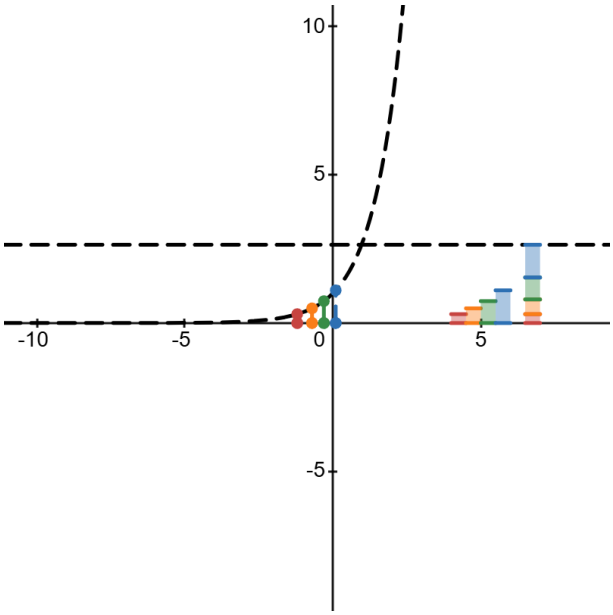


Figure 2.14: Softmax Function

2.4.1 Temperature Parameter

The softmax function can be modified by introducing a temperature parameter τ. The temperature parameter controls the entropy of the output distribution. A higher temperature parameter results in a more uniform distribution, while a lower temperature parameter results in a sharper distribution. The softmax function with a temperature parameter is defined as follows:

$$\sigma(\vec{z})_j = \frac{e^{z_j/\tau}}{\sum_{k=1}^{K} e^{z_k/\tau}} \tag{2.2}$$

Where:

- τ is the temperature parameter.
- $\vec{z}$ is a vector of real numbers.
- z_j is the j^{th} element of the vector $\vec{z}$.
- K is the number of classes.

Figure 2.15 illustrates the difference between the softmax function with a temperature parameter and the softmax function without a temperature parameter. The figure shows that the softmax function with a temperature parameter results in a more uniform distribution.

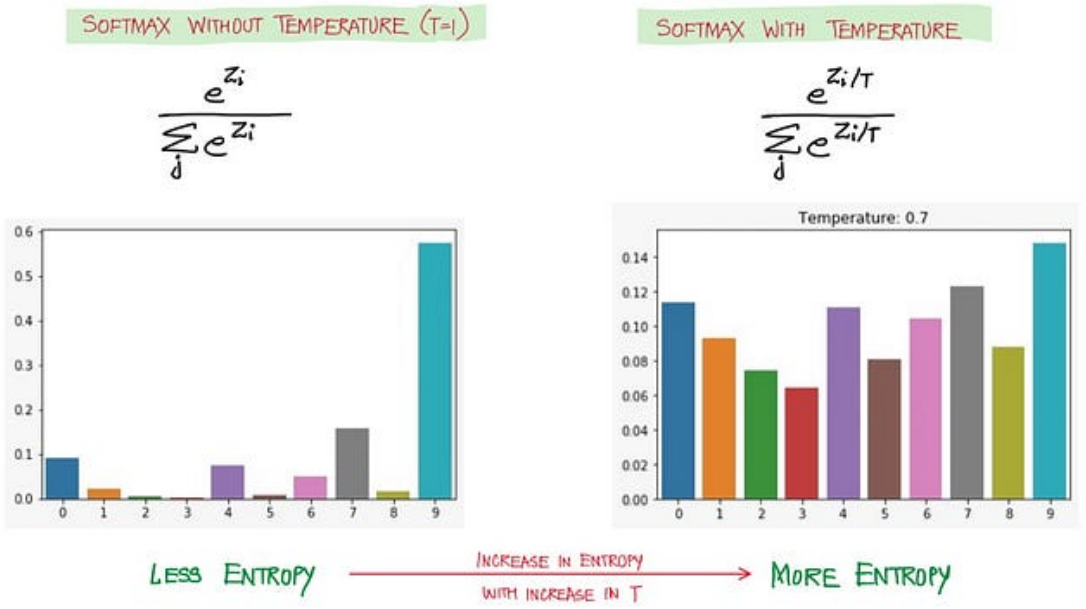


Figure 2.15: Softmax Function with Temperature Parameter [18]

2.5 Faiss

Faiss is a powerful library designed for efficient similarity search and clustering of dense vectors. It provides algorithms that can search through sets of vectors of any size, even those that may not fit in memory. Additionally, Faiss includes supporting code for evaluation and parameter tuning. The library is implemented in C++ and comes with complete wrappers for Python and `numpy`. Faiss is developed and maintained by Facebook AI Research [19].

2.6 Triton Inference Server

The Triton Inference Server is an open-source model-serving software that simplifies the deployment of AI models at scale in production environments. It is designed to support model parallelism, which allows the server to scale to multiple GPUs and multiple models. Triton supports TensorFlow, PyTorch, ONNX, and TensorRT models. It also provides a RESTful API and gRPC endpoints for model inference. Triton is developed and maintained by NVIDIA [20].

2.7 Apache Parquet

Parquet is an open-source binary columnar storage file format that is more efficient than CSV for reading and writing data. It supports complex data structures and is designed to work with big data systems like Apache Hadoop. Parquet represents nested structures using the record shredding and assembly technique from the Dremel [21] paper. Parquet is designed to be efficient for both read and write operations. Parquet is developed and maintained by the Apache Software Foundation [22].

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 Near 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 hundred 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 [23], integrated with **Amazon SageMaker** model deployment [24] as infrastructure.

3.2.3 Continuous Training and Deployment

This implies continuous training and deployment of the model which requires the automation of training steps 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 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) [25].

Its approach is described with more depth in the LightFM paper [26].

3.3.2 Rexy

Rexy [27] 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 [27].

[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 [23]

3.3.3 Gorse

Gorse [28] 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 [27].

3.3.4 AWS Personalize

According to AWS, developers can utilize Amazon Personalize [8] to rapidly create and implement customized recommendations and sophisticated user segmentation on a large scale through 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 [30] 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 [32]

  > "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 [33]

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

- Merlin HugeCTR [35]
- Merlin Transformers4Rec [36]
- Merlin SOK (SparseOperationsKit)
- Merlin Distributed Embeddings (DE)

[2]AWS description of the service [8]
[3]Google Cloud Marketplace Recommendations AI Page[29]
[4]Nvidia Merlin Repository [31]
[5]NVTabular[32]
[6]Merlin Models Documentation[34]

- Merlin Systems [37]

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 might be referred to as customers or buyers (representing individuals interacting with the platform which can be through a website or an app), while products might be 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, Figure 4.1 shows a suggested system design by Nvidia.

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 (Offline)
- Inference Ensemble (Online)
- Caching Layer
- Automation and DevOps

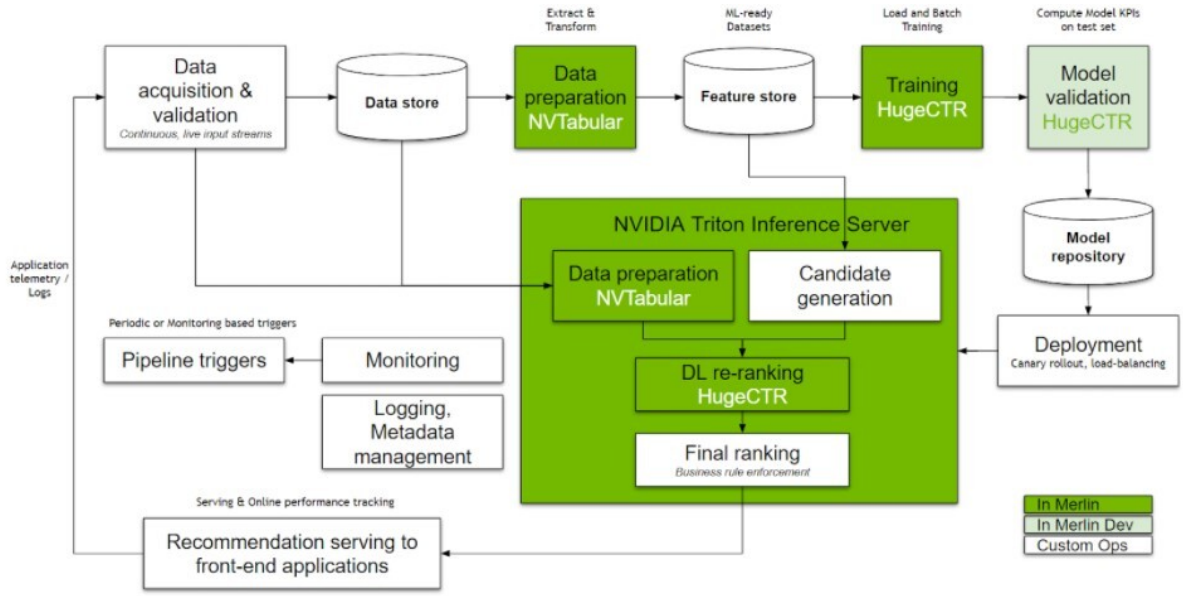


Figure 4.1: System Components [12]

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:

- CRUD customers
- CRUD products
- Adding interactions between customers and products

Such endpoints interact mainly with the storage components to store the data, those endpoints are platform-dependent, hence they will not be discussed in this project.

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 pipelines (ensembles) 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 first 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 a different ensemble that uses the item embedding as a query instead of the user embedding for the retrieval stage.

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 [38] 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 [40] or Google BigQuery [41] can be used.

- Online Feature Store

[1] AWS Relational Database Service [39]

The feature store is a storage system that stores data in a format optimized for ML models. [15] It also stores the embedding tables of the customers and products, in addition to any other categorical features. This project uses FEAST [42] as the feature store, which supports using a Redis [43] cluster as an Online store, the Redis cluster is deployed using AWS ElastiCache [44].

- Item Embedding Store

  A vector database that stores the embeddings of the products while allowing them to be queried using an ANN search algorithm. In this project, the item embedding store is a Faiss Index[19] that is stored in an S3 bucket and is materialized, loaded in memory, on inference server startup.

- Model Repository

  After training the models, their parameters are stored in the model repository. In this project, the model repository is an AWS S3 bucket [45].

- 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[43] cluster deployed using AWS ElastiCache [44].

4.3 Recommendation Pipeline (Offline)

The recommendation pipeline is the core of the recommendation system. Deploying a recommendation training 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 Preprocessing Workflow

The first step in the recommendation pipeline is the preprocessing of the data. In this stage, the raw data is cleaned, transformed, and tagged using NVTabular [32]. The main purpose of this stage is to generate embeddings for the categorical features, NVTabular allows saving the workflow and its intermediate results such as the embeddings, this ensures reproducibility and consistency between the training and inference stages. Once the workflow is fitted to the data, NVTabular generates the embeddings and saves them along with the workflow in the model repository. Then it transforms the dataset using the fitted workflow to prepare it for model training.

4.3.2 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 Training

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 can be done using a distributed training framework like Merlin HugeCTR [46] which can be deployed on AWS SageMaker [47] inside a Merlin HugeCTR container [48].

Also, it can be done using frameworks like TensorFlow [49] since Merlin has a TensorFlow implementation of the two-tower model.

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 products, 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 DLRM [50] proposed by Facebook. Instead of implementing the DLRM from scratch, the system uses Merlin Models [33] implementation of the DLRM, which is optimized to run accelerated on Nvidia GPUs using the HugeCTR [46] or TensorFlow frameworks.

4.4 Customer Inference Ensemble

This section describes the inference ensemble, which is responsible for generating recommendations for a given user in real-time. The next section describes a different ensemble that generates similar products to a given product.

The deployment of the recommendation system in inference mode involves setting up the Triton Ensemble and the external components of the system as shown below in Figure 4.2.

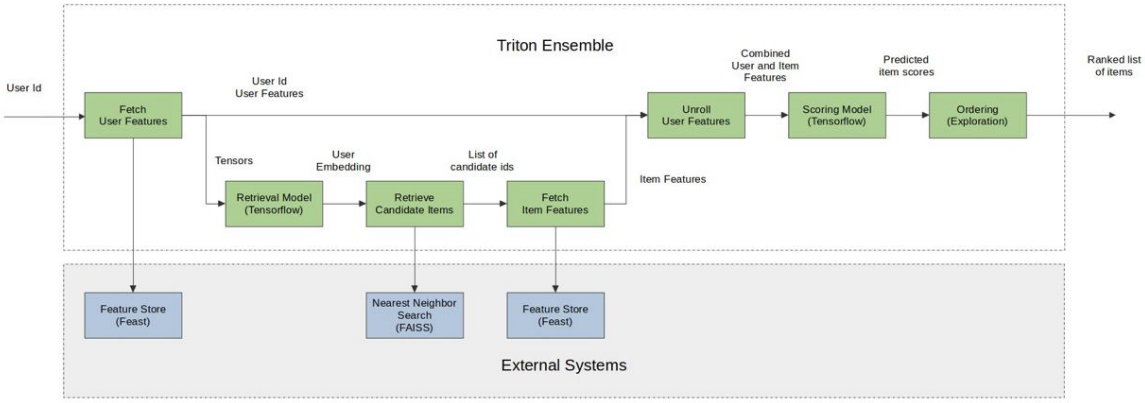


Figure 4.2: Inference Ensemble Diagram

Initially, the ensemble receives a user ID as input. Once received, Triton starts executing the ensemble pipeline step by step, which consists of the following steps:

4.4.1 Fetch User Features

In this step, the server fetches the user features from the feature store (Feast). Then those features are processed by the workflow that was defined previously and saved in the model repository.

4.4.2 Generate User Embedding

Once the server has the user features, it uses them to generate a user embedding. It loads the query tower of the retrieval model, which is responsible for generating the user embedding.

4.4.3 Retrieve Candidate Item IDs

During inference (online), using two separate towers to retrieve the candidate items for each customer is not efficient. [15]

ANN (Online Retrieval)

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 by doing an ANN search using the customer embedding, in other words, calculates the similarity between the customer embedding and item embeddings then retrieves the top hundred or so nearest items.

The server uses FAISS [19] to perform the nearest neighbor search computation on a GPU. Faiss returns a list of candidate item IDs based on their embedding similarity to the user embedding.

Fetch Item Features

Once the candidate item IDs are retrieved, the server fetches the item features from the feature store (Feast) based on the candidate item IDs. Then, loads the preprocessing workflow from the model repository and feeds those features into the workflow to process and tag them.

4.4.4 Unroll User Features

After fetching the item features, the server combines them with the user features using an operation called "Unrolling", where the user features are repeated for each item in the candidate list as shown in Figure 4.3.

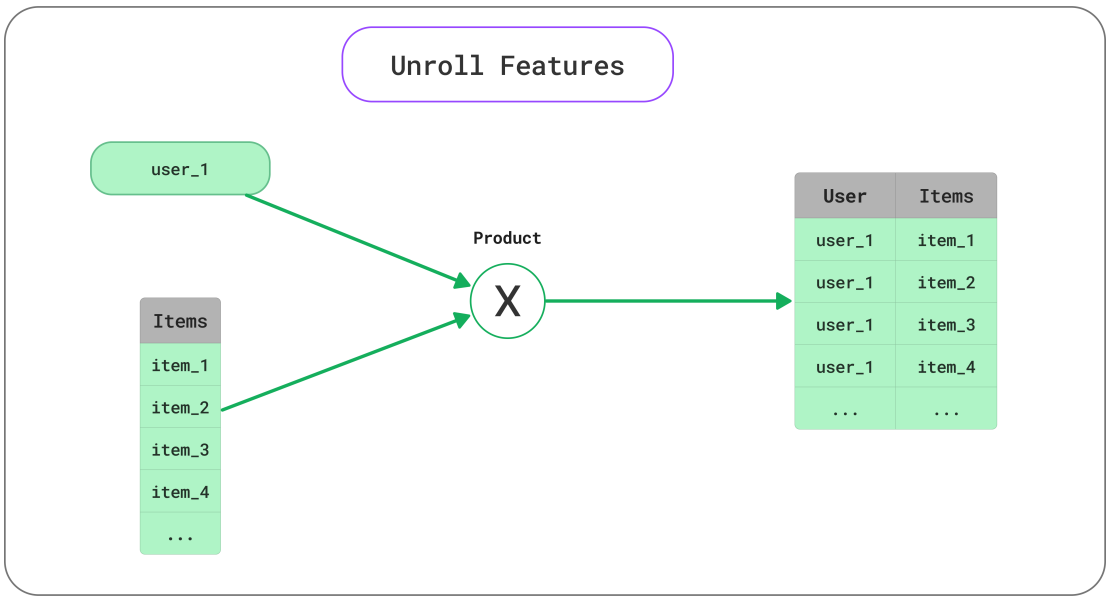


Figure 4.3: Unroll User/Item Features

4.4.5 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 [32], 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.

4.4.6 Score Items

The server loads the DLRM model from the model repository and then feeds the combined user and item features into the model to get the relevance scores.

The DLRM is deployed using Nvidia Triton Inference Server [23] on a Merlin Inference container [51] from Nvidia's NGC [52], Which is deployed on AWS SageMaker. AWS SageMaker [47] automatically scales the number of instances of the Triton Inference Server based on the load.

This step outputs a list of item IDs and their corresponding relevance scores (click probability).

4.4.7 Order Items and Softmax Sampling

Finally, the server uses the relevance scores to rank the items in the candidate list.

Softmax Sampling

Instead of returning the top N items, the server uses a softmax sampling technique to sample items from different positions in the ranked list. The sampling is important to ensure diversity in the recommendations and to avoid recommending the same items to all users or the same items to the same user in different sessions. In addition, sampling helps the user explore new categories of items and interact with them which tells the system more about the user preferences.

This step is the last step in the ensemble pipeline, and the server returns the sampled items to the client.

Business Rules Ordering

In most e-commerce platforms, the business rules require the system to promote some products, promoting products means putting them 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

This step is implemented in the API gateway since the number of products to be ordered after the ranking is very small and doesn't need an accelerated infrastructure.

4.5 Similar Products Inference Ensemble

This section describes the ensemble that generates similar products to a given product. The ensemble should receive a product ID as input and return a list of similar products to the client.

The ensemble pipeline is similar to the customer recommendation ensemble pipeline, but instead of using the user features, the server uses the item features to generate the embedding query and it retrieves the candidate items based on the similarity between the wanted item embedding and other item embeddings in the item embedding store.

After the candidate items IDs are retrieved, the ensemble follows the same steps as the customer recommendation ensemble to rank, filter, score, and order the items.

4.6 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.

4.7 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 [53] 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 AliCCP and it is a public dataset that contains user-item interactions from an e-commerce platform. The dataset gathers traffic logs of recommendation systems in Taobao, a Chinese online shopping website. The dataset is collected from the real-world recommendation system of Taobao.[54]. The dataset contains 4 types of features: user features, item features, combination features and context features, each feature is represented by a unique ID, and they are 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

The AliCCP dataset contains 615,599 unique users and 5,142,585 unique products. The products are divided into

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 [55].

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.

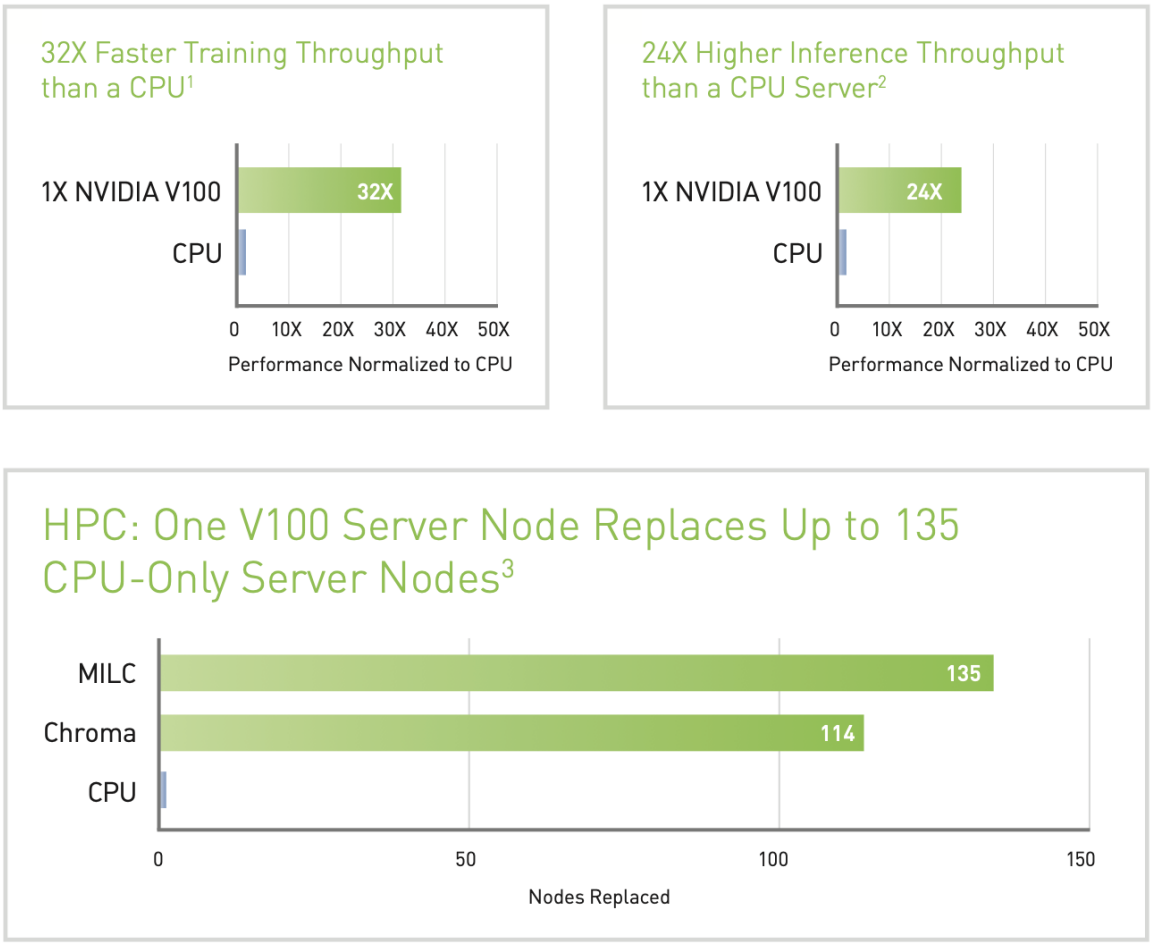


Figure 5.1: Nvidia Tesla V100 GPU vs CPU [56]

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** [57], which includes access to NVIDIA AI Enterprise [58] with NGC [52].

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 [49] was used as a runtime for the experiment Jupyter notebook.

5.3 Expirement Offline Pipeline

5.3.1 Data Preprocessing

Converting the Data from CSV to Parquet

The raw AliCCP dataset is stored in CSV format. However, to leverage GPU-accelerated libraries such as cuDF, the data is converted into the Apache Parquet [22].

The original data in CSV format was about 11GB in size. After converting it to Parquet format, the data size was reduced to 1.1GB. The training data consists of 42,299,771 rows, while the testing data consists of 42,999,775 rows.

Eliminating Rows with Excessive Missing Data

Null values can introduce biases or inaccuracies into the analysis, so removing rows with many missing data points helps to mitigate this issue. Rows with more than five null features were removed to ensure data completeness. The five nulls threshold ensures that the remaining rows have enough complete information to be useful for training and testing the recommender system. This resulted in a training dataset with 33,088,764 rows and a testing dataset with 33,678,677 rows.

Resplitting Data

Originally the data was split with a 1:1 ratio into training and testing data. To balance the training and testing data, 60% of the testing data was moved to the training data. The resulting datasets contain 53,299,746 training rows and 13,467,695 testing rows. The ratio of the training data to the testing data is approximately 80:20.

The following table shows the changes made to the data during the preprocessing phase:

Data Set	Original Rows	Data After Cleaning	Data After Resplitting
Training Data	42,299,771	33,088,764	53,299,746
Testing Data	42,999,775	33,678,677	13,467,695

Table 5.2: Data Rows Before and After Preprocessing

Saving customer and products parquets

In this step, the user features as well as the item features are extracted from the training data and saved as parquet files. Those files will later be used as an offline data store by Feast, the feature store that will be used at inference time.

Before saving the parquet files, timestamps are added to the data to help Feast keep track of the data freshness and do incremental materialization.

5.3.2 Retrieval Stage

As mentioned earlier, the retrieval model consists of two towers, the user tower and the item tower. The user tower is responsible for processing the user features, while the item tower is responsible for processing the item features.

Retrieval Model Training

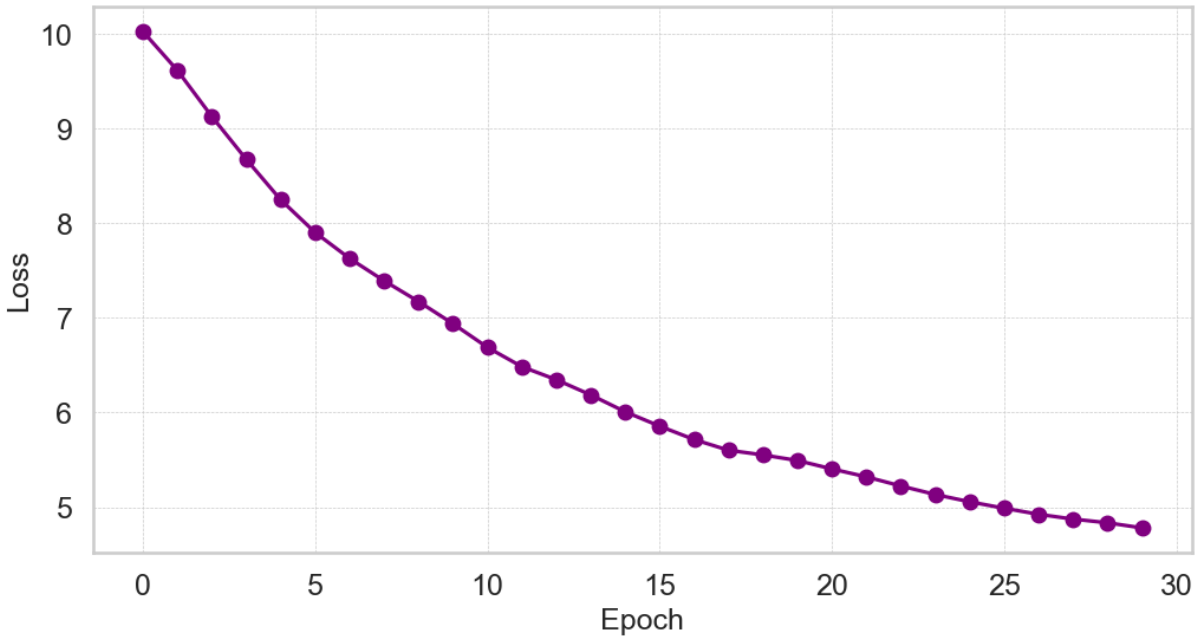
The model is trained using the following hyperparameters:

- Batch Size: 28K
- Number of Epochs: 30
- Optimizer: Adam
- Loss Function: Categorical Crossentropy
- Model: Two-Tower (v2 Implementation By Merlin Models)

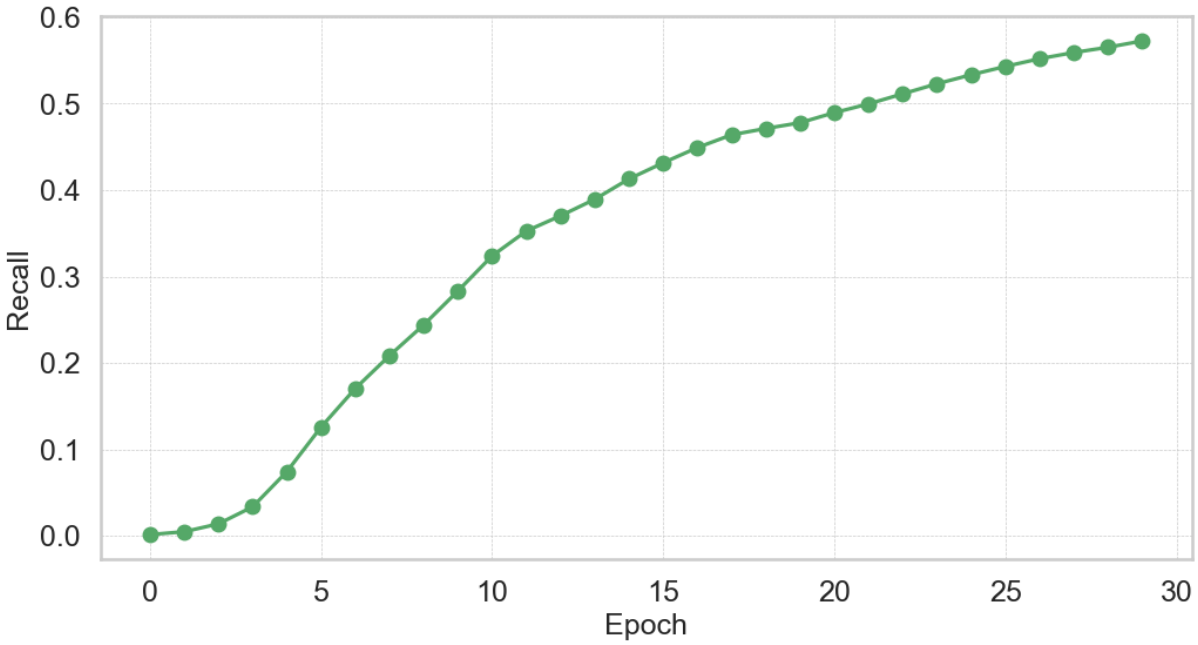
Figure 5.2 shows the training loss and recall of the retrieval model. It shows that the model achieves a recall at 20 of around 30% on the training data, but on the validation data, the recall is around 4% only.

This means that in the top 20 recommendations, 4% of the actual positive items in the validation set are listed. In production, not only 20 but the top hundreds of items are taken from the retrieval model and are fed into the ranking model.

So if the recall is 4% at 20, it is expected to be much higher at values like 500 which is used in this project. Unfortunately, due to technical limitations, the recall at 500 could not be calculated for each user of the dataset.



(a) Retrieval Model Loss Vs Epochs



(b) Retrieval Model Recall At 20 Vs Epochs

Figure 5.2: Retrieval Model Training Results

Saving Query Tower

Once the retrieval model is trained, the query tower is saved as a Keras model to the model repository, in this project, it should be an S3 bucket, but for the simplicity of this experiment, it is saved locally to an EBS GP3 SSD attached to the server. This model will be used to generate the user embeddings at inference time.

Generating Item Embeddings

the embeddings of each item in the dataset are calculated at training time using the trained retrieval model. Then those embeddings are saved in a parquet [22] file which will next be converted to a Faiss [19] index for fast retrieval at inference time.

Once the retrieval model finishes generating the item embedding parquet, the parquet is used to generate a Faiss [19] index. In this experiment, the index is saved to the model repository and later loaded to GPU memory on inference server start-up.

5.3.3 Filtering

In a four-stage recommender system, the filtering stage typically follows retrieval. During filtering, products are removed based on business rules, such as stock or age restrictions.

However, the AliCCP dataset does not include this type of information, so the filtering step was skipped in this experiment.

5.3.4 Ranking Model Training

A DLRM model is trained using the following hyperparameters:

- Batch Size: 204K
- Number of Epochs: 2
- Optimizer: Adam
- Binary Crossentropy Loss
- Model: DLRM

Figure 5.3 shows the training and validation area under the curve of the ranking model vs epochs. It shows that the model starts overfitting after the second epoch, as the training AUC keeps increasing while the validation AUC decreases. This is why the model was early stopped at two epochs.

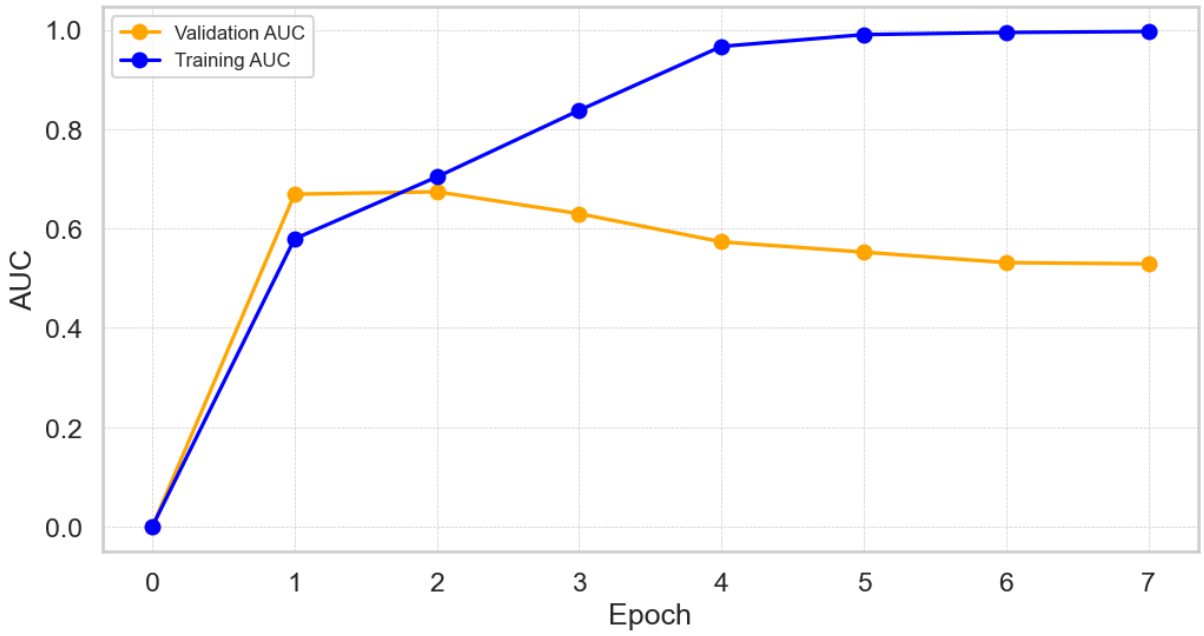


Figure 5.3: Ranking Model Training/Validation Set AUC

Different models were trained and tested, such as the DCN and NCF, but the DLRM model was chosen as it had slightly better performance and higher metrics. Figure 5.4 shows the metrics of the different models tested.

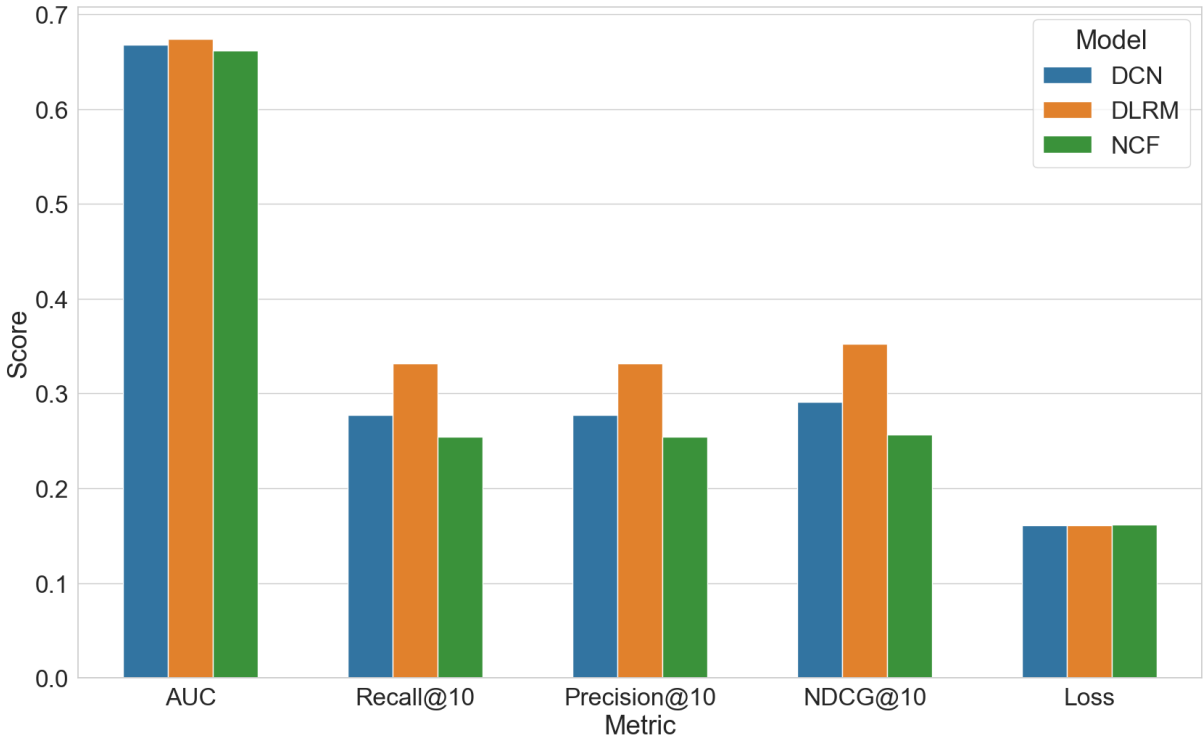


Figure 5.4: Ranking Models Validation Metrics Comparison

Once the model finished training, it was saved as a TensorFlow model to the model repository.

5.4 Deployment

The deployment pipeline of the recommendation system involves setting up the Triton Ensemble and the external components of the system as shown in Figure 4.2. To use a Triton inference server, a Triton Ensemble has to be compiled with the necessary models, workflows, and configurations.

5.4.1 Feast Materialization

In order to use Feast as the feature store, the user and item features have to be materialized, meaning that they have to be moved from Feast's offline store, which in this case is a Parquet file, to its online store, which is a Redis database.

5.4.2 Compiling Ensemble

After the user and item features are materialized, the Triton Ensemble is compiled with all the models, workflows, and configurations. Once compiled with all the components shown in Figure 4.2, it outputs a directory tree similar to the one shown in Figure 5.5.

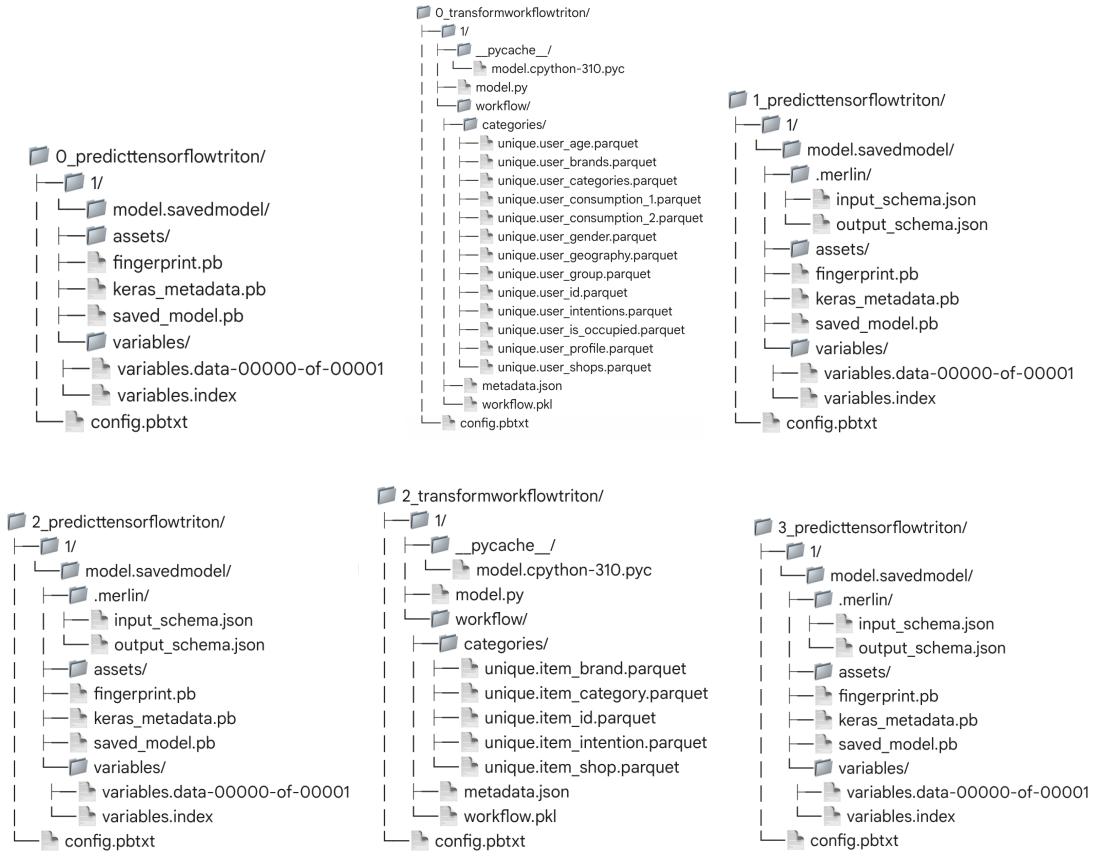


Figure 5.5: Ensemble Directory Tree

As shown in Figure 5.5, the Triton Ensemble directory tree contains model configurations, model weights, embedding categories, and other necessary files. Those files are saved to the model repository.

5.5 Demo

Once the Triton Ensemble is compiled, the system can be deployed and tested. When deploying a Triton Ensemble, by default, the Triton Inference Server is started on port 8001 for HTTP and 8002 for gRPC. The Triton Inference Server can be accessed through the Triton Client, which is a Python library that allows the user to interact with the server.

A demo web UI was created to demonstrate the recommender system's functionality by illustrating its inputs and outputs. To calculate metrics within the demo, a small subset of users with their positive clicks from the validation data was utilized.

The demo is a straightforward web application built using HTML, FastAPI [59], and Tailwind CSS. Figure 5.6 and Figure 5.7 show an illustrative example that represents a sample input and output of the recommendation system.

Figure 5.6 illustrates the recommendations generated for a specific user (ID #12001) who has interacted with 306 items, representing the positive instances in the validation set.

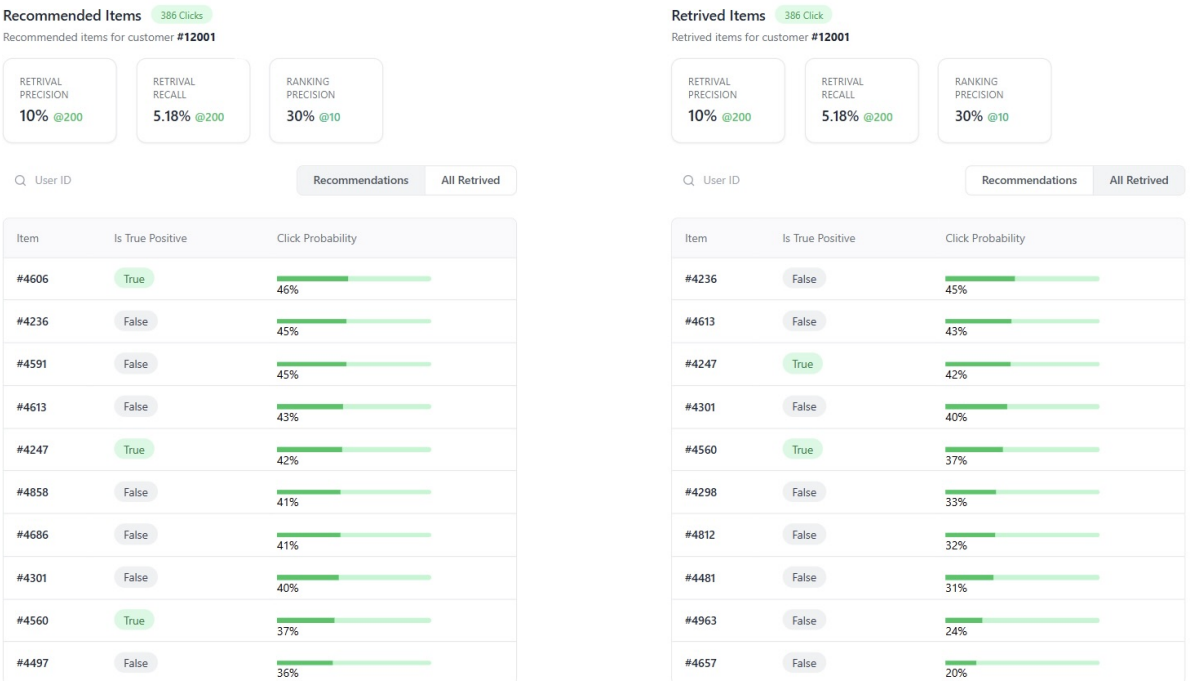


Figure 5.6: Demo Example 1

The left section displays the final 10 recommendations after Softmax sampling, each accompanied by its score. This list includes 3 true positives (items the user interacted with) and 7 false positives, resulting in a system precision of 30%.

The right section lists all 200 items retrieved by the retrieval model, of which 20 are true positives and 180 are false positives. Based on these figures, the recall of the retrieval stage is calculated to be 5.18% in this example.

Figure 5.7 shows a different example, where the user (ID #5042) has interacted with 57 items.

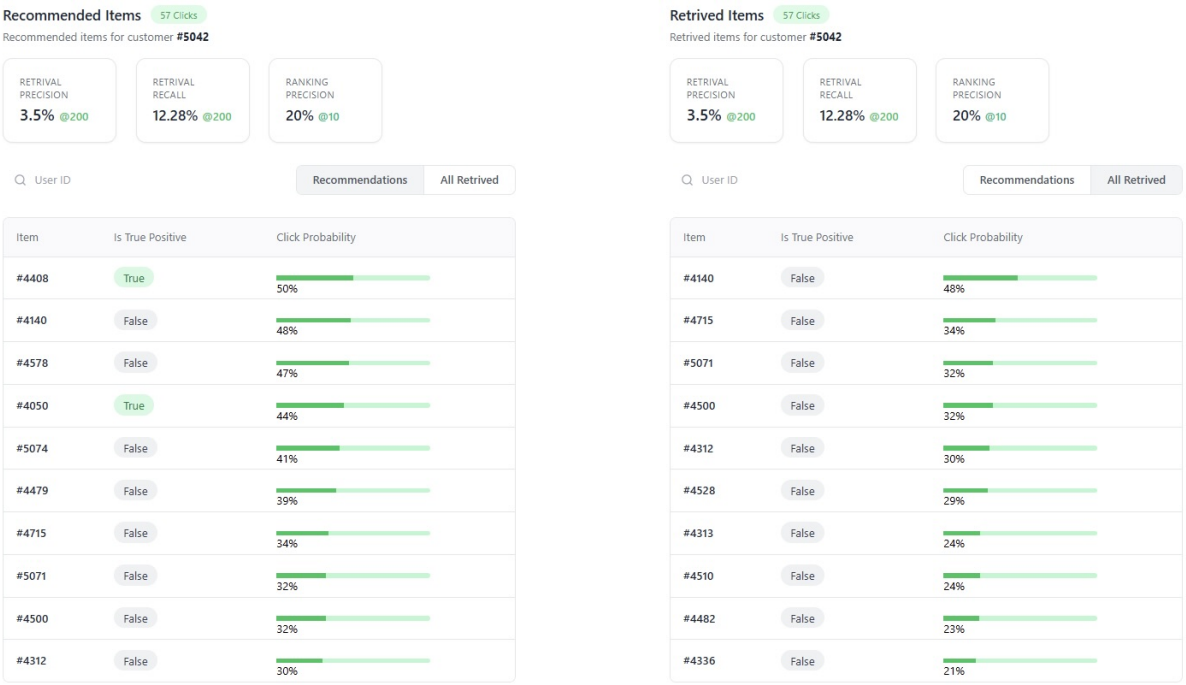


Figure 5.7: Demo Example 2

In the second example, the system was able to recommend 2 true positives in the final 10 recommendations, resulting in a precision of 20%. The retrieval model retrieved 200 items, of which 7 were true positives, resulting in a retrieval recall of 12.28%.

5.6 Performance Evaluation

To evaluate the performance of the recommendation system, a simple benchmark was conducted using the Triton Inference Client, 100 requests were sent to the server, and the average latency was calculated using different values for retrieval TopK items limit.

Figure 5.8 shows the average latency of the system with different values for the TopK items limit.

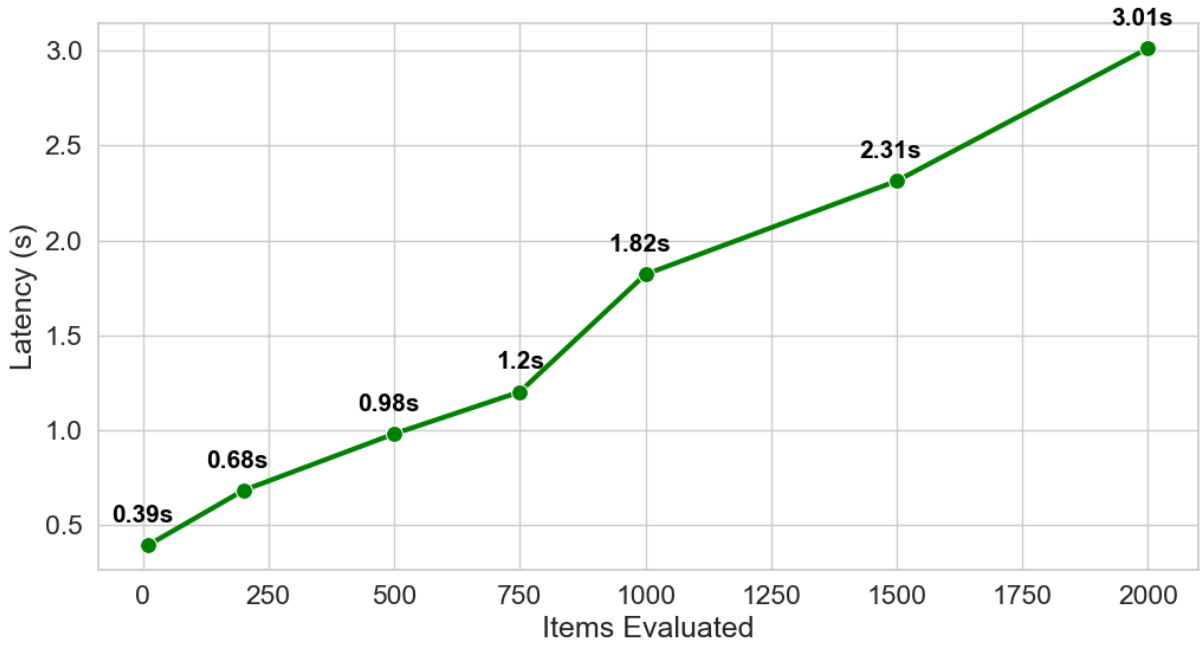


Figure 5.8: Performance Evaluation

As shown in Figure 5.8, the average latency of the system increases as the TopK items limit increases, this is because the bottleneck of the system is loading item features from the feature store, the more items are loaded the more time it takes to retrieve them. Meanwhile, the time required to evaluate 100 or 1000 items in the recommendation model is negligible compared to the time required to load the item features.

An approach to improve the performance of the system is to use a more efficient feature store, probably a better offline store for Feast instead of the Parquet files that are used in this experiment.

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 DLRMs offer promising potential in terms of scalability and performance. However, evaluation of the retrieval component reveals that the current Two-Tower model requires further optimization or replacement to enhance the selection of relevant items from vast product databases. Additionally, the system's latency increases with the number of items retrieved, indicating a need for improvements in the feature retrieval process.

Moving forward, research will prioritize optimizing the retrieval stage, focusing on improving accuracy (especially recall) and reducing latency. A more effective retrieval model helps significantly in enhancing the user experience by presenting more relevant product recommendations, leading to increased customer engagement and potentially higher sales. Furthermore, by addressing latency issues, the system can better handle real-time recommendation requests, ensuring a seamless browsing experience for e-commerce users. Such optimizations have the potential to not only improve the performance of the systems but also contribute to advancements in the field of e-commerce personalization.

6.2 Future Technical Work

Build and implement the system's components, deploy and automate the recommendation pipeline and required infrastructure In addition to optimizing the ranking and retrieval models 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 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 RNN with 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 [60], which is a flexible and efficient library for sequential and session-based recommendation [60].

6.3 Economic Implications On Silal

This cutting-edge recommendation system, once optimized, holds immense potential to transform Silal's e-commerce businesses. By significantly improving the accuracy and relevance of product recommendations, it can drive increased customer engagement, higher click-through rates, and ultimately, increased sales. This translates to a direct boost in revenue for Silal. Additionally, the ability to provide real-time personalized recommendations can enhance the user experience, leading to improved customer satisfaction and loyalty, further contributing to long-term revenue growth.

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

[18] H. Gupta, "Softmax Temperature." https://medium.com/@harshit158/softmax-temperature-5492e4007f71, accessed: 2024-6-15.

[19] Facebook Research, "Faiss." https://github.com/facebookresearch/faiss/tree/main, accessed: 2024-6-16.

[20] Nvidia, "Nvidia Triton Inference Server." https://github.com/triton-inference-server, accessed: 2024-6-16.

[21] S. Melnik, A. Gubarev, J. J. Long, G. Romer, S. Shivakumar, M. Tolton, and T. Vassilakis, "Dremel: Interactive analysis of web-scale datasets," in *Proc. of the 36th Int'l Conf on Very Large Data Bases*, pp. 330–339, 2010.

[22] Apache, "Apache Parquet." https://parquet.apache.org/, Accessed on June 30, 2024.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[50] 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.

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

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

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

[54] 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.

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

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

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

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

[59] S. Ramírez, "FastAPI."

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