Machine Learning System Design

Video Recommendation

Problem Statement & Metrics

1. Problem Statement

Build a video recommendation system for YouTube users. We want to maximize users' engagement and recommend new types of content to users.

2. Metrics design and requirements

Metrics :

Offline metrics

- Use precision (the fraction of relevant instances among the retrieved instances), recall (the fraction of the total amount of relevant instances that were actually retrieved), ranking loss and logloss.

Online metrics

- Use A/B testing to compare click through rates, watch time and conversion rates.

Requirements :

Training

- User behavior is generally unpredictable, and videos can become viral during the day. Ideally, we want to train many times during the day to capture temporal changes.

Inference

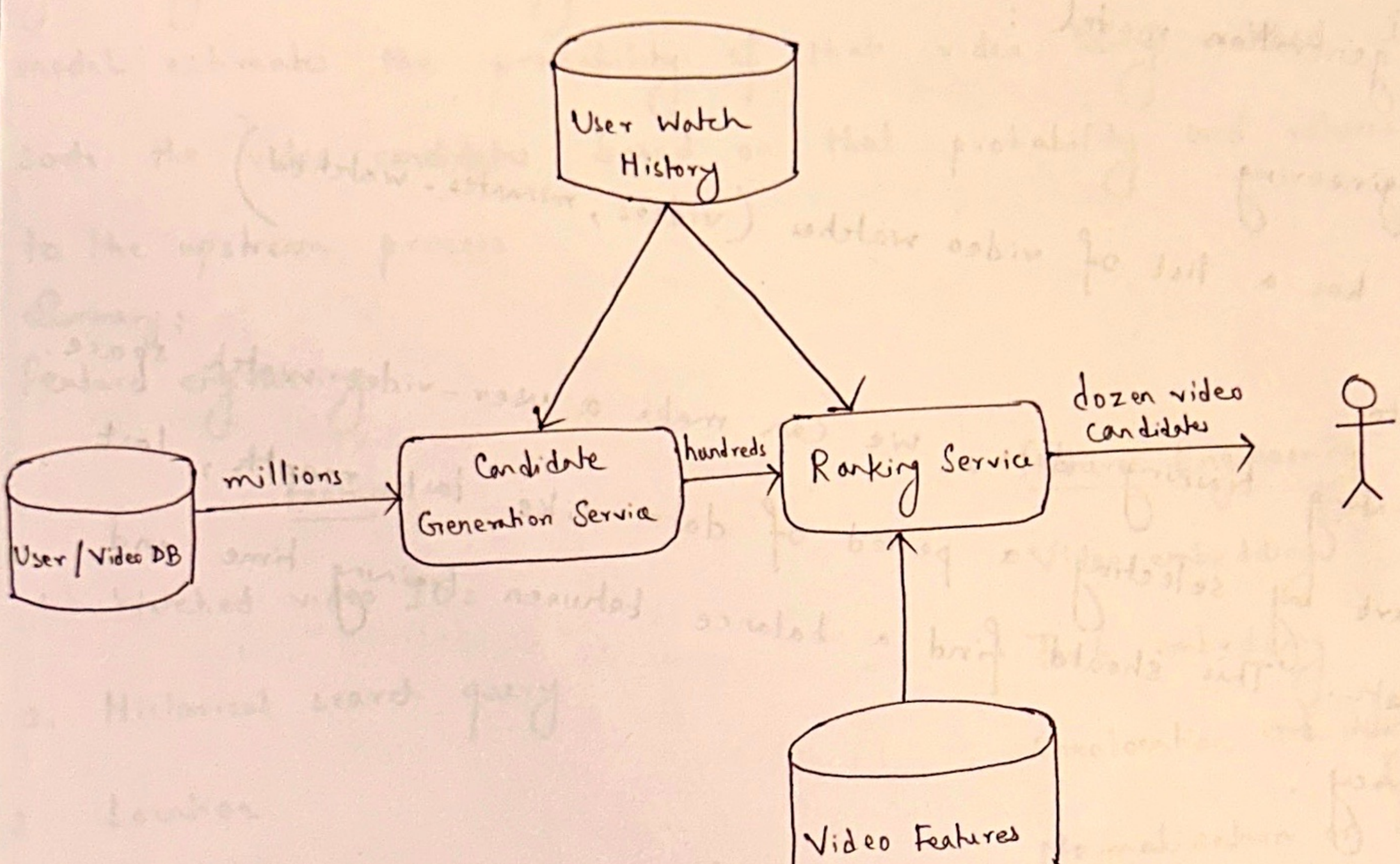
- For every user to visit the homepage, the system will have to recommend 100 videos for them. The latency needs to be under 200 ms, ideally sub 100 ms.
- for online recommendations, it's important to find the balance between exploration vs exploitation. If the model over-exploits historical data, new videos might not get exposed to users. We want to balance betwe relevancy and fresh new content.

Summary :

	Desired goals
Type	Reasonable precision, high recall
Metrics	High Throughput with the ability to retrain many times per day
Training	Latency from 100 ms to 200 ms
Inference	Flexible to control exploration vs exploitation

andidate Generation & Ranking Model

3. Multi-stage models.



Architecture digram for the video recommendation system

There are two stages, candidate generation and ranking. The reason for two stages is to make the system scale.

- The candidate model will find the relevant videos based on user watch history and the type of videos the user has watched.
- The ranking model will optimize for the view likelihood, i.e., videos tha

have a high watch possibility should be ranked high. It natural fit for the logistic regression algorithm.

Candidate generation model :

feature engineering

- Each user has a list of video watches (videos, minutes-watched)

Training data

- For generating training data, we can make a user-video watch We can start by selecting a period of data like last month, 6 months, etc. This should find a balance between training time a model accuracy.

Model

- The candidate generation can be done by matrix factorization. The purpose of candidate generation is to generate somewhat relevant conten to users based on their watched history. The candidate list needs to big enough to capture potential matches for the model to perform well with desired latency.

The ideal choice is to use collaborative algorithms because the inference time fast, and it can capture the similarity between user taste in the user-video space.

ing model :

During inference, the ranking model receives a list of video candidates given by the candidate generation model. For each candidate, the ranking model estimates the probability of that video being watched. It then sorts the video candidates based on that probability and returns the list to the upstream process.

Summary:

feature engineering

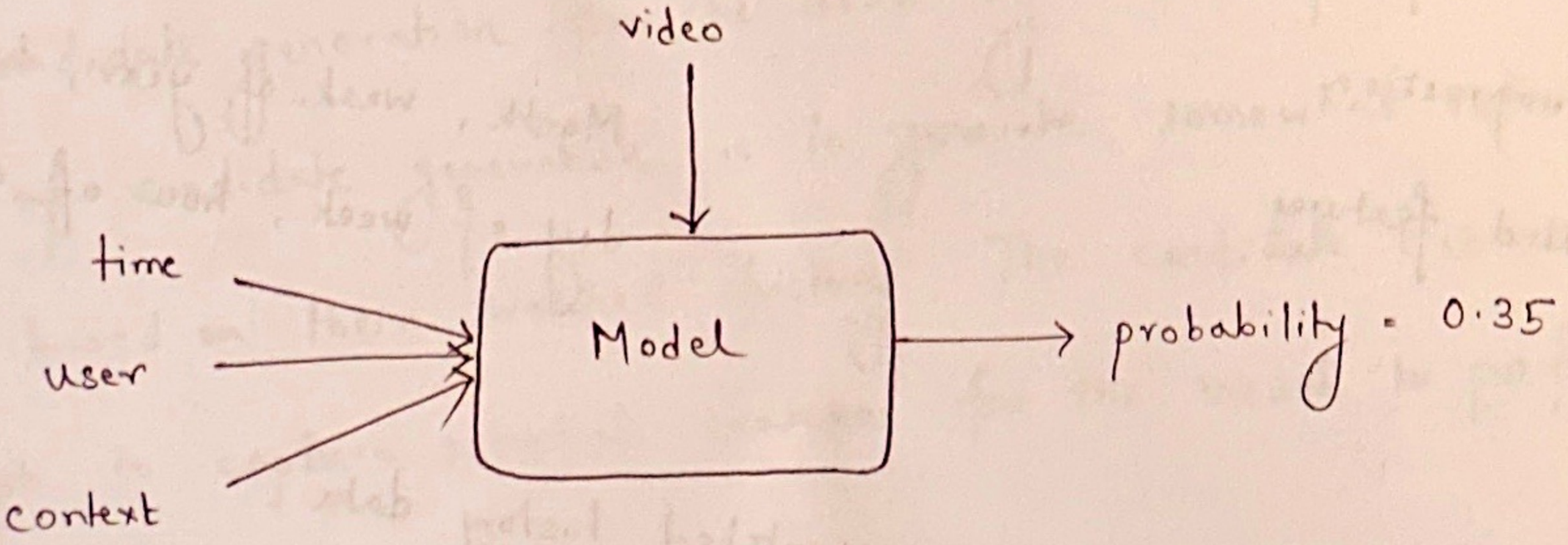
Features	Feature Engineering
1. Watched video IDs	Video embedding
2. Historical search query	Text embedding
3. Location	Geolocation embedding
4. User associated features : age, gender	Normalization or Standardization
5. Previous impression	Normalization or Standardization
6. Time related features	Month, week of year, holiday, day of week, hour of day.

Training data - We can use user watched history data.

Model

At the beginning, it's important that we start with a simple model, we can add complexity later.

- A fully connected neural network is a simple yet powerful for representing non-linear relationships and it can handle a lot of data.
- We start with a fully connected neural network with sigmoid activation at the last layer. The reason for this is that the sigmoid function returns values in the range $[0, 1]$; therefore it's a natural fit for estimating probability.
- For deep learning architecture, we can use relu as an activation function for hidden layers. It's very effective in practice.
- The loss function can be cross-entropy loss.



Model Video Recommendation System Design

4. Calculation and estimation

Assumptions :

- For the sake of simplicity, we can make these assumptions - Video views per month are 150 billion.
- 10% of videos watched are from recommendations, a total of 15 billion videos.
- On the homepage, a user sees 100 video recommendations.
- On average, a user watches two videos out of 100 video recommendations.
- If users do not click or watch some video within a given time frame, i.e., 10 minutes, then it is a missed recommendation.
- The total number of users is 1.3 billion.

Data size :

- for 1 month, we collected 15 billion positive labels and 750 billion negative labels.
- Generally, we can assume that for every data point we collect, we also collect hundreds of features. For simplicity, each row takes 500 bytes to store. In one month, we need 800 billion rows.
- Total size : $500 \times 800 \times 10^9 = 4 \times 10^{15}$ bytes = 4 Petabytes. To save costs,

we can keep the last six months or one year of data in the data lake and archive old data in cold storage.

Bandwidth:

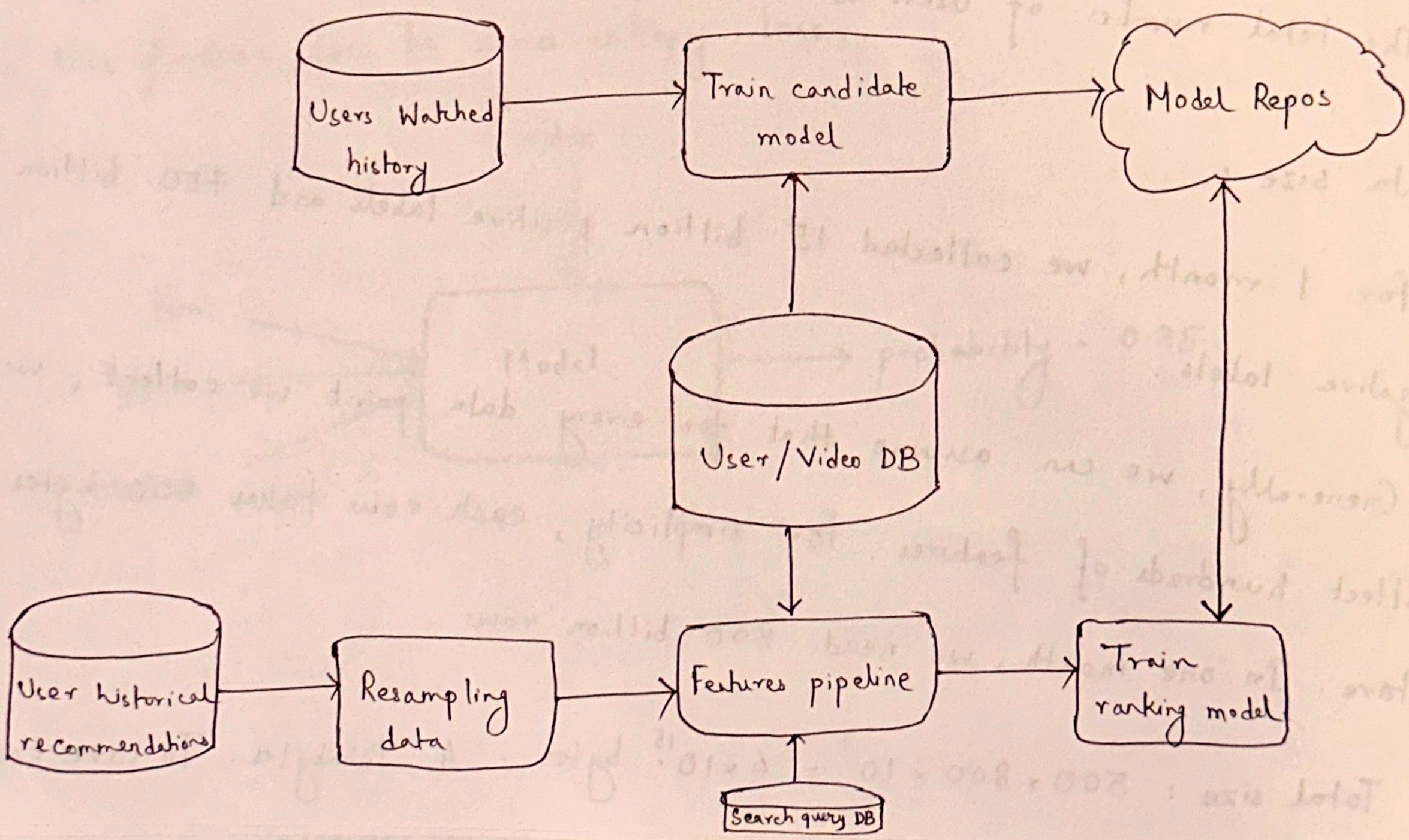
- Assume that every second we have to generate a recommendation request 10 million users. Each request will generate ranks for 1k - 10k videos.

Scale:

- Support 1.3 billion users

5. System Design

High-level system design



Database

User Watched history stores which videos are watched by a particular user over time.

- Search Query DB stores historical queries that users have searched in the past.
- User historical recommendations stores past recommendations for a particular user.

- Resampling data: It's part of the pipeline to help scale the training process by downsampling negative samples.
- feature pipeline: A pipeline program to generate all required features for training a model. It's important for feature pipelines to provide high throughput, as we require this to retrain models multiple times. We can use Spark or Elastic MapReduce or Google DataProc.
- Model Repos: Storage to store all models, using AWS S3 is a popular option.
- In practice, during inference, it's desirable to be able to get the latest model near real-time. One common pattern for the inference component is to frequently pull the latest models from Model Repos based on timestamp.

Challenges

Huge data size

- Solution: Pick 1 month or 6 months of recent data

Imbalanced data

- Solution: Perform random negative down sampling

High availability

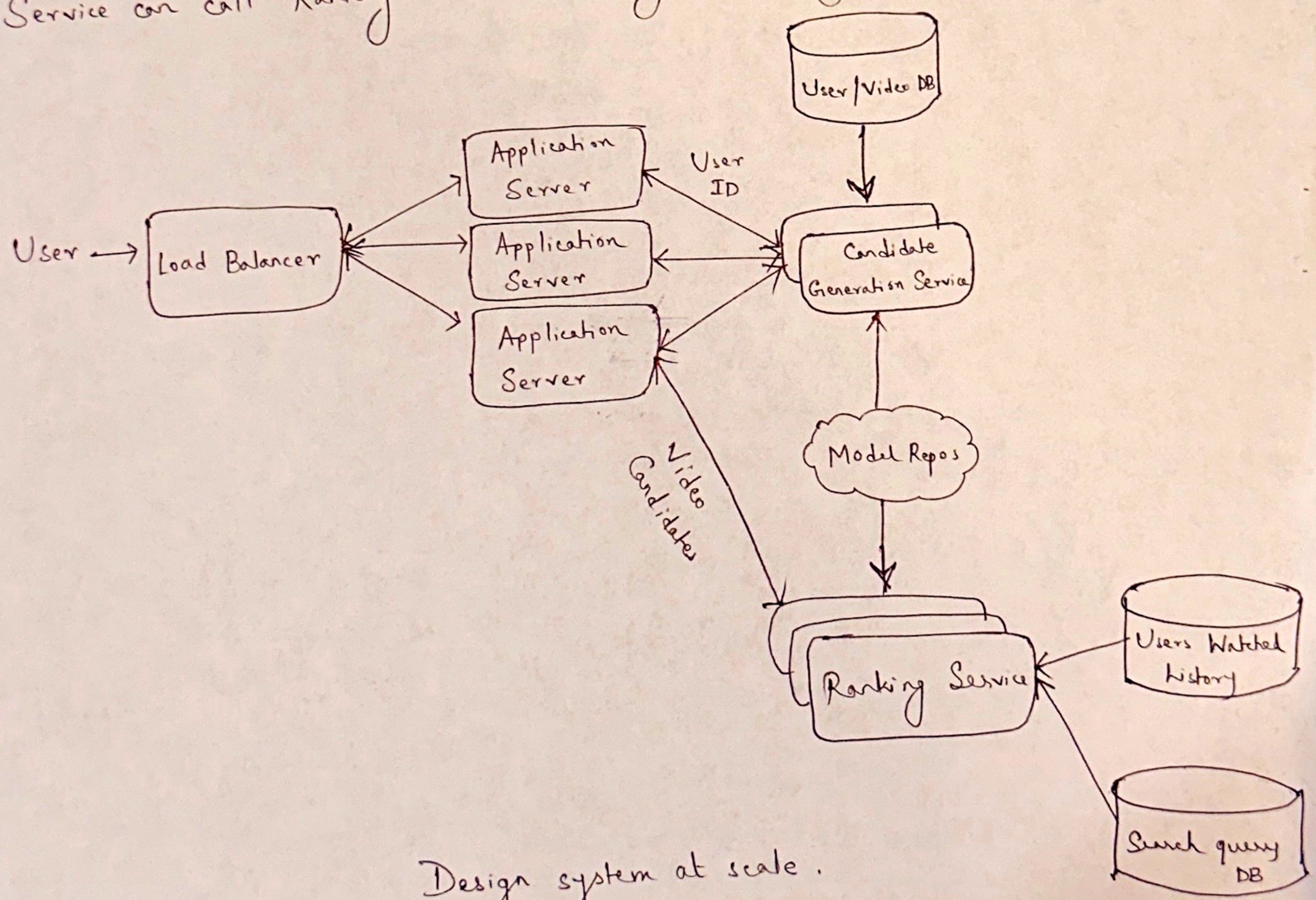
- Solution 1: Use model-as-a-service, each model will run in Docker containers.
- Solution 2: We can use Kubernetes to auto-scale the number of pods.

When a user requests a video recommendation, the Application Server requests video candidates from the Candidate Generation Model. Once it receives the candidates, it then passes the candidate list to the ranking model to get the sorting order. The ranking model estimates the watch probability and returns the sorted list to the Application Server. The Application Server then returns the top videos that the user should watch.

Scale the Design

Scale out (horizontal) multiple Application Servers and use Load Balancers to balance loads.

- Scale out (horizontal) multiple Candidate Generation Services and Ranking Services.
- It's common to deploy these services in a Kubernetes Pod and take advantage of the Kubernetes Pod Autoscaler to scale out these services automatically.
- In practice, we can also use Kube-proxy so the Candidate Generation Service can call Ranking Service directly, reducing latency even further.



Design system at scale.

7. Follow up questions

1. How do we adapt to user behavior changing over time ?

(a) Read more about Multi-armed bandit

(b) Use the Bayesian Logistic Regression model so we can update prior data

(c) Use different loss functions to be less sensitive with click through rates, etc

2. How do we handle the ranking model being under-explored ?

We can introduce randomization in the ranking service. For example, 2% of requests will get random candidates, and 98% will get sorted candidates from the ranking service.