### **Video Recommender System: Problem Statement**

The primary goal is to build a video recommendation system aimed at maximizing user engagement and introducing users to new types of content.

### **Metrics Design and Requirements**

#### **Metrics**

1. **Offline Metrics:**
   * **Precision:** Measures the accuracy of the recommended videos.
   * **Recall:** Measures the ability of the system to capture relevant videos.
   * **Ranking Loss:** Evaluates the correctness of the order of the recommended videos.
   * **Logloss:** Measures the uncertainty of the model’s predictions.
2. **Online Metrics:**
   * **A/B Testing:** Used to compare different versions of the recommendation system.
   * **Click Through Rates (CTR):** The ratio of users who click on a recommended video.
   * **Watch Time:** Total time users spend watching the recommended videos.
   * **Conversion Rates:** The rate at which recommendations lead to desired actions (e.g., subscriptions, likes).

### **Requirements**

1. **Training:**
   * The system needs to account for the unpredictable nature of user behavior and the potential virality of videos.
   * Frequent retraining (multiple times a day) is necessary to capture temporal changes and trends.
2. **Inference:**
   * The system should recommend 100 videos to each user visiting the homepage.
   * Latency needs to be minimal, ideally under 200ms, and preferably under 100ms.
   * The system must balance exploration (introducing new videos) and exploitation (recommending known relevant content) to ensure both relevancy and freshness.

### **Summary**

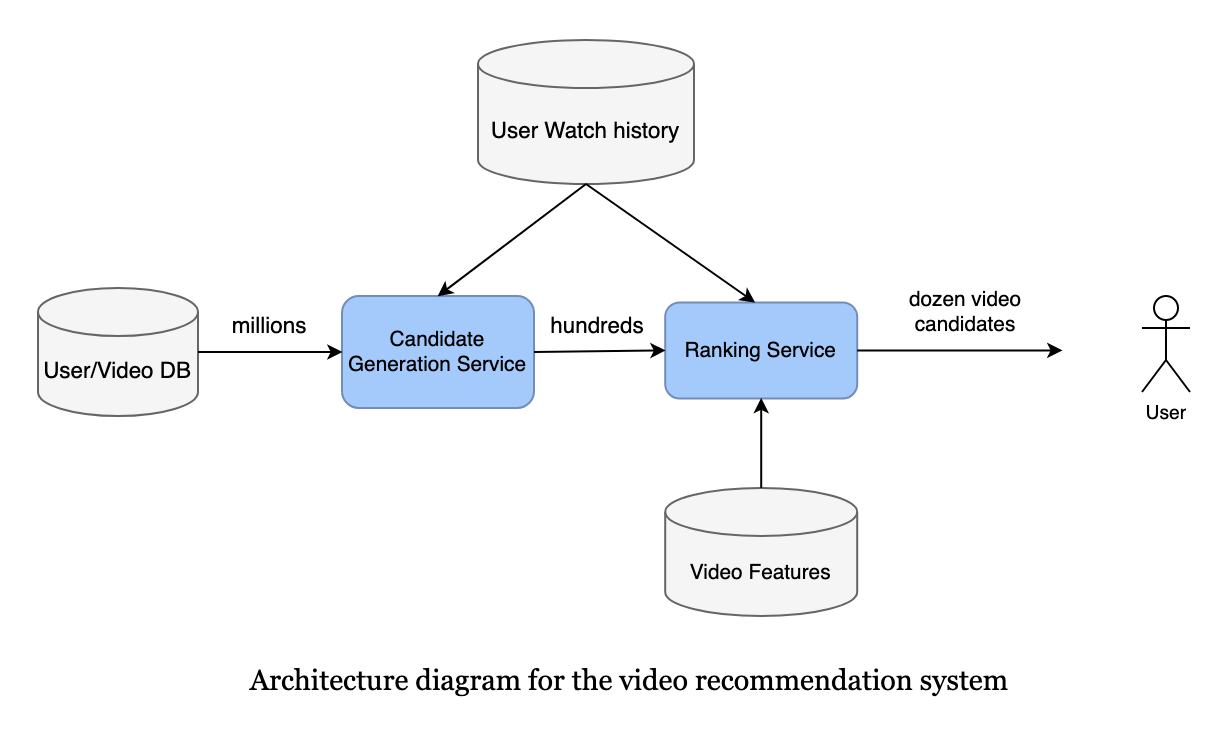
* **Desired Goals:**
  + **Metrics:** Achieve reasonable precision and high recall.
  + **Training:** Ensure high throughput with the capability for multiple daily retrainings.
  + **Inference:** Maintain low latency (100ms to 200ms) and allow for flexibility in managing exploration versus exploitation.

### **Video Recommender System: Candidate Generation and Ranking Model - Machine Learning System Design**

### **Overview**

The recommendation system is designed using a two-stage process: candidate generation and ranking. This design pattern is prevalent in many large-scale machine learning systems for scalability.

### **Multi-stage Models**



#### **Candidate Generation Model**

**Objective:**

* Identify relevant video candidates based on user watch history and preferences.

**Feature Engineering:**

* Utilize user’s watch history (e.g., video IDs, minutes watched).

**Training Data:**

* Construct a user-video watch space using historical data (e.g., last month or last six months) to balance training time and model accuracy.

**Model:**

* **Matrix Factorization:** Commonly used for candidate generation due to its ability to capture user preferences and its fast inference time. It helps generate a list of “somewhat” relevant content.
* **Collaborative Filtering:** Captures similarities in user tastes but may be less preferred for very large-scale systems.
* **Alternative Methods:** In large-scale systems like those at Facebook and Google, more efficient methods like Inverted Index, FAISS (Facebook AI Similarity Search), or Google’s ScaNN (Scalable Nearest Neighbors) are used for their low latency.

#### **Ranking Model**

**Objective:**

* Rank the generated candidate videos based on the likelihood of being watched by the user.

**Feature Engineering:**

* **Watched video IDs:** Video embedding.
* **Historical search queries:** Text embedding.
* **Location:** Geolocation embedding.
* **User-associated features (age, gender):** Normalization or standardization.
* **Previous impressions:** Normalization or standardization.
* **Time-related features:** Embedding for month, week\_of\_year, holiday, day\_of\_week, and hour\_of\_day.

**Training Data:**

* Use user watch history. Typically, the ratio of watched to not-watched videos is 2/98, indicating that most recommended videos are not watched.

**Model:**

* **Fully Connected Neural Network:** A simple yet effective model for capturing non-linear relationships in large datasets. Uses sigmoid activation in the last layer to estimate probabilities (range [0, 1]).
* **Activation Function:** ReLU (Rectified Linear Unit) is used for hidden layers due to its practical effectiveness.
* **Loss Function:** Cross-entropy loss is used to measure the performance of the model.

**Inference:**

* The ranking model processes the list of video candidates generated by the candidate generation model, estimating the probability of each video being watched and sorting them accordingly.

### **Summary**

1. **Candidate Generation Model:**
   * Generates a broad list of potentially relevant videos using matrix factorization or other low-latency methods.
   * Ensures scalability and quick inference times.
2. **Ranking Model:**
   * Ranks the candidate videos based on their likelihood of being watched, using a neural network with features like embeddings and standardizations.
3. **Feature Engineering:**
   * Employs various embedding and normalization techniques to enhance the accuracy and relevance of recommendations.
4. **Training and Inference:**
   * Training data is carefully selected to balance accuracy and training time.
   * Models are designed to handle large-scale data with minimal latency, ensuring a seamless user experience.

This detailed approach ensures that the recommendation system is both scalable and effective, balancing the need for quick, relevant suggestions with the ability to adapt to new and trending content.

### **Video Recommender System: Candidate Generation and Ranking Model - Machine Learning System Design**

The document titled "Video Recommendation System Design - Machine Learning System Design" details the architecture, challenges, and scaling considerations of a video recommendation system. Here’s a detailed summary:

### **Overview**

The document outlines the design of a video recommendation system, including calculations, system architecture, data challenges, and scaling strategies.

### **Calculation & Estimation**

#### **Assumptions**

* **Video views:** 150 billion per month.
* **Recommendations:** 10% of videos watched are from recommendations, totaling 15 billion.
* **User behavior:** On average, a user watches 2 out of 100 recommended videos.
* **Missed recommendations:** Videos not clicked or watched within 10 minutes.
* **User base:** 1.3 billion users.

#### **Data Size**

* **Positive labels:** 15 billion positive labels (videos watched) per month.
* **Negative labels:** 750 billion negative labels (videos not watched) per month.
* **Storage:** Each data point (row) takes 500 bytes. For 800 billion rows, this results in 0.4 petabytes per month.
* **Data retention:** Last six months or one year of data in the data lake, older data in cold storage.

#### **Bandwidth**

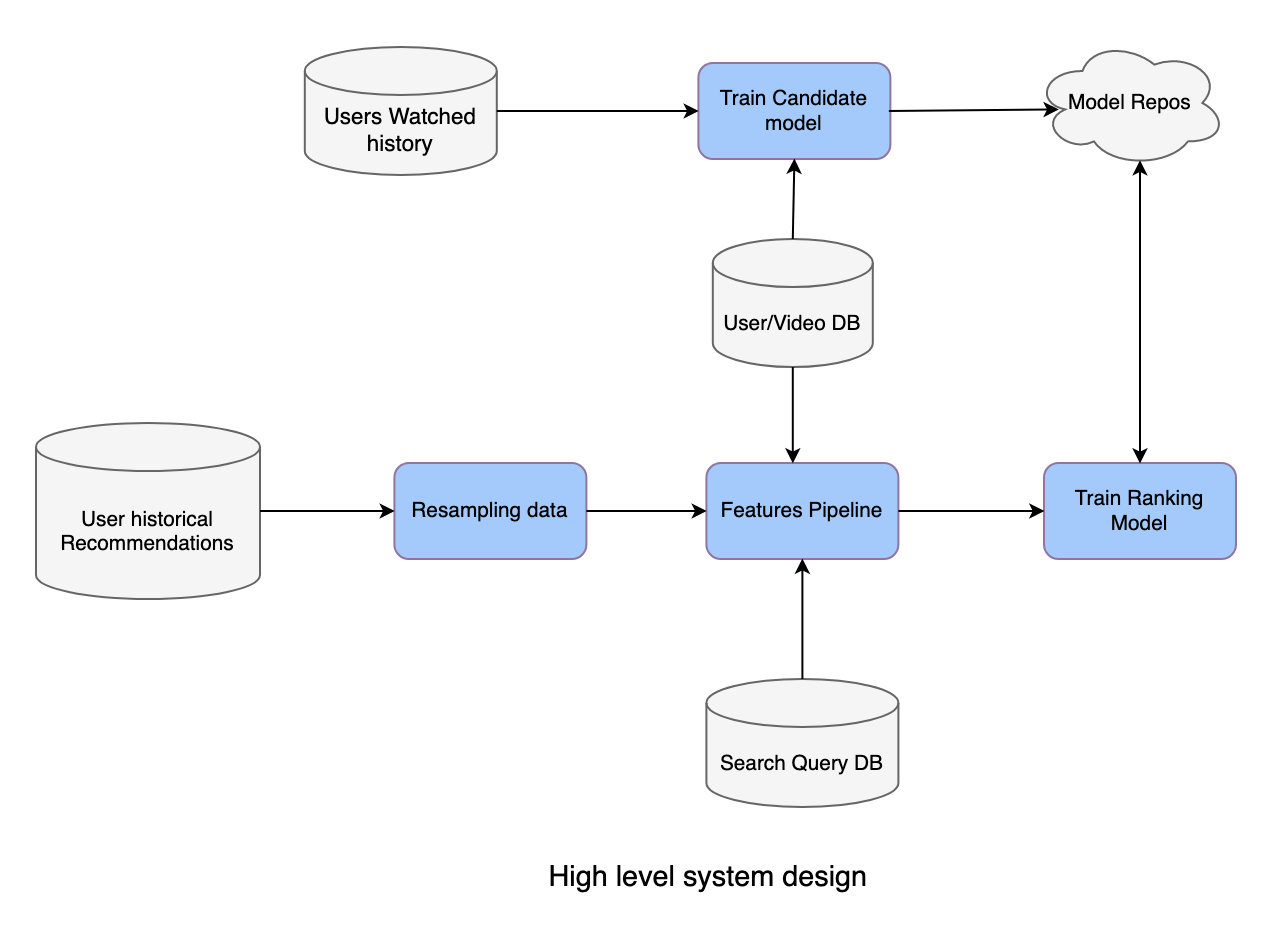
* **Requests:** Generate recommendations for 10 million users every second, with each request generating ranks for 1,000 to 10,000 videos.

#### **Scale**

* **User support:** 1.3 billion users.

### **System Design**

#### **High-level System Design**



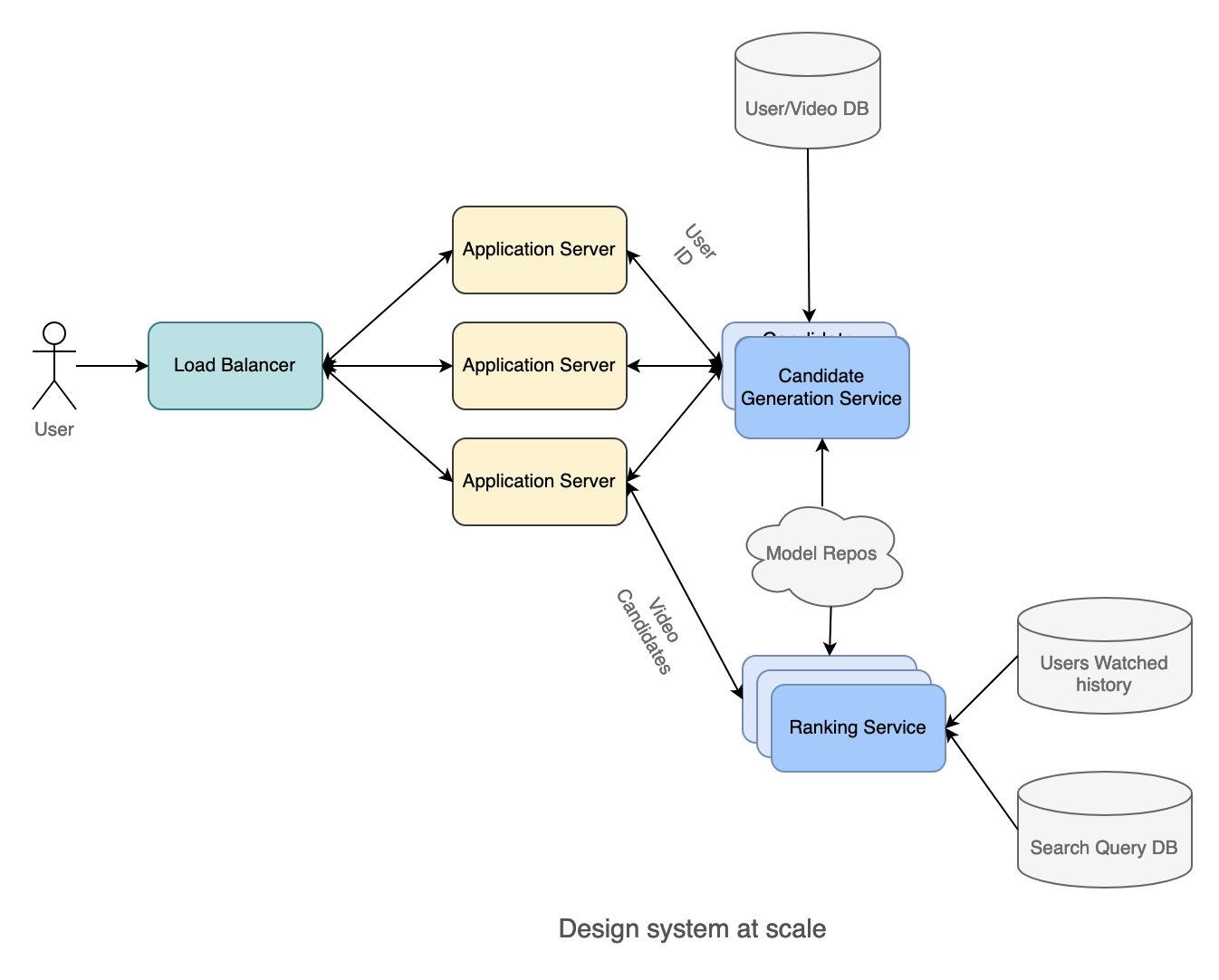
1. **Databases:**
   * **User Watched History:** Stores user watch history over time.
   * **Search Query DB:** Stores historical search queries.
   * **User/Video DB:** Stores user profiles and video metadata.
   * **User Historical Recommendations:** Stores past recommendations for users.
2. **Resampling Data:**
   * Part of the pipeline to down-sample negative samples for scalability.
3. **Feature Pipeline:**
   * Generates all required features for training models. High throughput is necessary for frequent retraining. Tools like Spark, Elastic MapReduce, or Google DataProc can be used.
4. **Model Repositories:**
   * Stores all models, with AWS S3 being a popular choice. Models need to be accessible in near real-time for inference.

#### **Challenges**

1. **Huge Data Size:**
   * Solution: Use the most recent 1 to 6 months of data.
2. **Imbalance Data:**
   * Solution: Perform random negative down-sampling.
3. **High Availability:**
   * Solution 1: Use model-as-a-service with models running in Docker containers.
   * Solution 2: Use Kubernetes to auto-scale the number of pods.

### **System Flow**

1. **User Request:**
   * User sends a video recommendation request to the Application Server.
2. **Candidate Generation:**
   * The Application Server requests video candidates from the Candidate Generation Model.
3. **Ranking:**
   * The candidate list is passed to the Ranking Model to get the sorted order based on watch probability.
4. **Response:**
   * The Application Server returns the top videos to the user.



### **Scaling the Design**

1. **Horizontal Scaling:**
   * Scale out multiple Application Servers and use Load Balancers.
   * Scale out multiple Candidate Generation Services and Ranking Services using Kubernetes Pod Autoscaler.
2. **Direct Communication:**
   * Use Kube-proxy to allow Candidate Generation Service to call Ranking Service directly, reducing latency.

### **Follow-up Questions**

1. **Adapting to Changing User Behavior:**
   * Use Multi-arm Bandit strategies.
   * Bayesian Logistic Regression to update prior data.
   * Use different loss functions to be less sensitive to click-through rates.
2. **Handling Under-explored Ranking Model:**
   * Introduce randomization in the Ranking Service (e.g., 2% of requests get random candidates).

### **Summary**

* The recommendation system is divided into two main services: Candidate Generation Service and Ranking Service.
* Initial models use deep learning with fully connected layers and handle feature engineering.
* Scaling involves using Kubernetes and load balancers, with direct service communication to minimize latency.