# **Ads Recommendation System Design - Machine Learning System Design**

### **Problem Statement**

The primary goal is to develop a machine-learning model that predicts whether an ad will be clicked. The document simplifies the problem by not delving into the complex cascade of classifiers typically used in AdTech. It begins by explaining the ad serving process, where an ad request is handled using a waterfall model. Publishers try to sell ad inventory directly with high CPM (Cost Per Thousand Impressions). If unsuccessful, the impression is passed to other networks until it is sold.

### **Metrics Design and Requirements**

#### **Metrics**

The document outlines metrics crucial during the training phase, focusing on machine learning metrics rather than revenue metrics or CTR metrics:

* **Offline Metrics:**
  + **Normalized Cross-Entropy (NCE):** Measures the predictive log-loss divided by the cross-entropy of the background CTR, making it insensitive to the background CTR. The NCE formula is provided for clarity.
* **Online Metrics:**
  + **Revenue Lift:** Percentage change in revenue over time. A new model is tested on a small traffic percentage, balancing between traffic percentage and A/B testing duration.

#### **Requirements**

The requirements for the system are divided into training and inference phases:

* **Training:**
  + **Imbalanced Data:** Since the CTR is usually very low (1%-2%), handling highly imbalanced data is essential for supervised training.
  + **Retraining Frequency:** The model should be retrainable multiple times a day to adapt to data distribution shifts in the production environment.
  + **Train/Validation Data Split:** Data should be partitioned by time to simulate a production system accurately.
* **Inference:**
  + **Serving:** The system needs to maintain low latency (50ms - 100ms) for ad prediction.
  + **Latency:** The recommendation latency must be minimal due to the waterfall model.
  + **Overspending:** The model should avoid repeatedly serving the same ads to prevent overspending the campaign budget, which could lead to financial losses for publishers.

### **Summary**

The document summarizes the desired goals for metrics, training, and inference:

* **Metrics:** Achieve reasonable NCE and CTR.
* **Training:** Handle imbalanced data and support high retraining frequency.
* **Inference:** Maintain low latency and control overspending.

### **1. Model Architecture**

The architecture section introduces the overall design of the ad click prediction model. This involves multiple stages from data collection to feature engineering, model selection, and evaluation.

### **2. Feature Engineering**

Feature engineering is a critical step in building the prediction model, involving the extraction and transformation of raw data into meaningful features. Key features mentioned include:

* **AdvertiserID:** Uses embedding or feature hashing due to the large number of advertisers.
* **User’s Historical Behavior:** Normalized data capturing the user's interaction with ads over time.
* **Temporal Features:** One-hot encoding for features such as time of day and day of the week.
* **Cross Features:** Combining multiple features to capture complex interactions.

### **3. Training Data**

Before training the model, it is essential to collect relevant data. This involves gathering data from different types of posts while improving user experience. Key points include:

* **Data Collection Period:** Data can be collected over different time frames (e.g., last month, last six months) to balance training time and model accuracy.
* **Handling Imbalanced Data:** Downsampling negative data (non-clicked ads) to address the common issue of imbalanced data in ad click prediction.

### **4. Model Selection**

The selection of the appropriate model is crucial. The document recommends starting with deep learning models, specifically fully connected layers with Sigmoid activation functions. Key considerations include:

* **Resampling Training Data:** To address the imbalance in the data, resampling is necessary to ensure the model can learn effectively.
* **Maintaining Validation and Test Sets:** Keeping these sets intact is essential for accurate performance estimation.

### **5. Model Evaluation**

Evaluating the model involves splitting the data into training and validation sets or using replay evaluation to avoid biased offline evaluation. Key aspects include:

* **Replay Evaluation:** Using test data from a future time period and reordering rankings based on the model's predictions. Accurate click predictions are recorded as matches, with the total matches considered as total clicks.
* **Training Data Size and Retraining Frequency:** Evaluating the optimal size of the training dataset and determining how frequently the model needs retraining to maintain performance.

### **Summary**

The document provides a comprehensive guide to building an ad click prediction model, detailing each stage from feature engineering to evaluation. The focus is on handling imbalanced data, selecting the appropriate model architecture, and employing robust evaluation methods to ensure the model performs well in a production environment.

This structured approach ensures that the machine learning system for ad click prediction is both effective and efficient, capable of handling the complexities of real-world ad serving scenarios.

### **1. Calculation and Estimation**

#### **Assumptions**

* The system needs to handle 40,000 ad requests per second, equivalent to 100 billion ad requests per month.
* Each observation (record) has hundreds of features, with each record requiring approximately 500 bytes of storage.

#### **Data Size**

* Historical ad click data includes information about users, ads, and whether the ad was clicked or not. With an estimated 1% Click-Through Rate (CTR), this results in 1 billion clicked ads.
* For training and validation, starting with one month of data is suggested. This data translates to approximately 50 petabytes (PB) of storage.
* To manage the data size, downsampling to 1%-10% or using one week of data for training and the next day for validation is recommended.

### **2. High-Level Design**

#### **Data Lake**

* Stores data collected from multiple sources such as logs or event-driven data (e.g., Kafka).

#### **Batch Data Preparation**

* Involves ETL (Extract, Transform, Load) jobs to store data in a Training Data Store.

#### **Batch Training Jobs**

* Organizes both scheduled and on-demand jobs to retrain models based on the data in the Training Data Store.

#### **Model Store**

* Uses distributed storage systems like S3 to store the trained models.

#### **Ad Candidates**

* A set of ad candidates provided by upstream services, following the waterfall model of ad serving.

#### **Stream Data Preparation Pipeline**

* Processes online features and stores them in key-value storage for low-latency downstream processing.

#### **Model Serving**

* A standalone service that loads various models and provides ad click probability predictions.

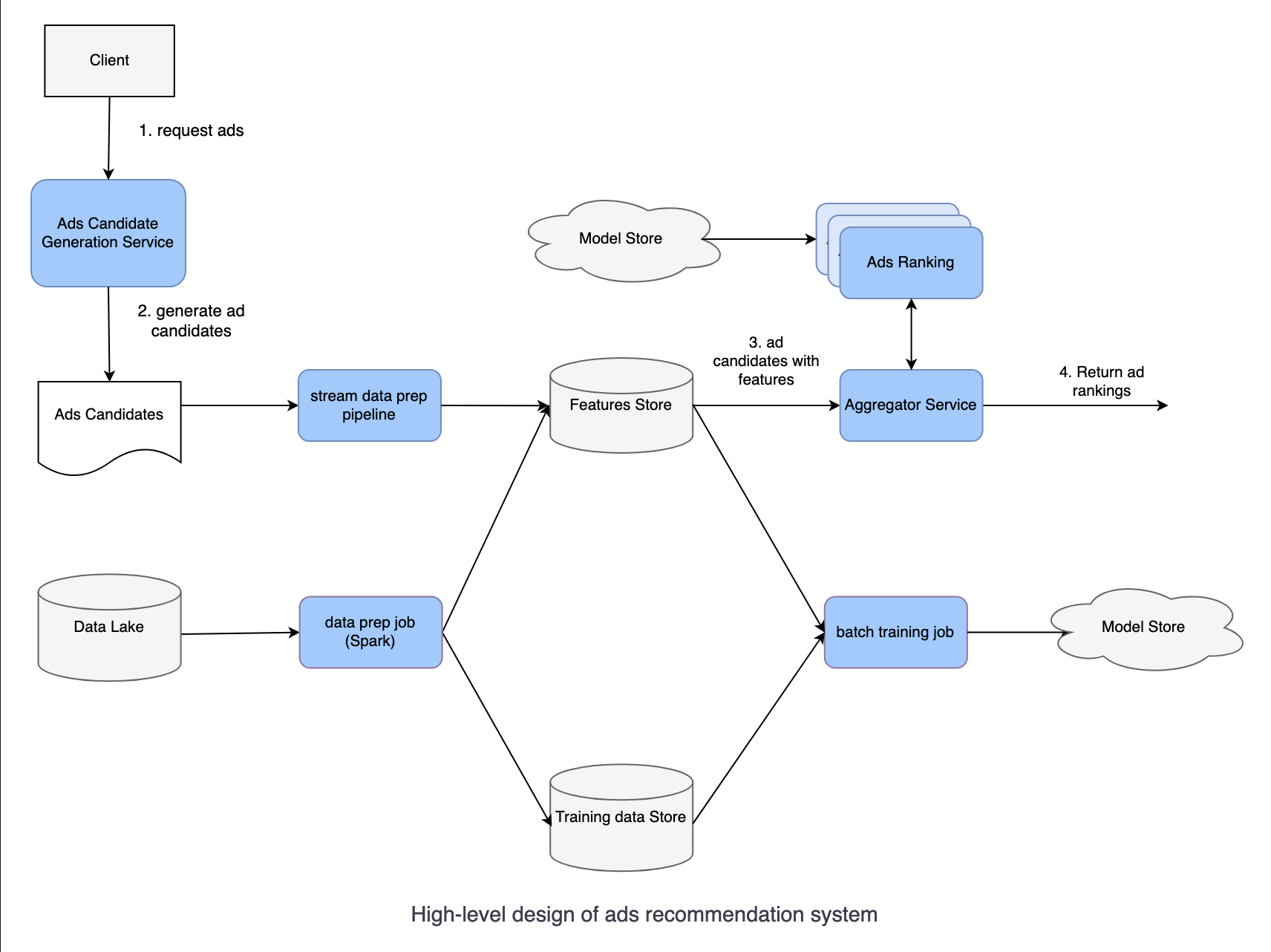
### **3. System Flow**

1. **Client Request**: A user visits the homepage and sends an ad request to the Application Server.
2. **Candidate Generation**: The Candidate Generation Service generates a list of ad candidates and sends them to the Aggregator Service.
3. **Ad Ranking**: The Aggregator Service splits the list of candidates and sends them to the Ad Ranking workers for scoring.
4. **Model Loading and Scoring**: The Ad Ranking Service retrieves the latest model from the Model Repository, obtains the relevant features from the Feature Store, computes ad scores, and returns the scored ads to the Aggregator Service.
5. **Top Ads Selection**: The Aggregator Service selects the top K ads (e.g., K=10, 100) and returns them to the upstream services.

### **4. Scaling the Design**

#### **Latency Requirements**

* Given a latency requirement of 50ms-100ms for a large volume of ad candidates (50,000-100,000), scaling is necessary to meet the Service Level Agreement (SLA).
* The Model Serving component must be scaled out, and the Aggregator Service must distribute the load across multiple serving instances.



### **5. Follow-Up Questions**

#### **Adapting to User Behavior Changes**

* Retrain the model as frequently as possible, ideally every few hours, using new data collected from user interactions.

#### **Handling Under-Explored Ad Ranking Models**

* Introduce randomization in the Ranking Service by allowing a small percentage (e.g., 2%) of requests to receive random candidates, while the majority (98%) get sorted candidates from the Ad Ranking Service.

### **6. Summary**

* The document emphasizes using Normalized Cross-Entropy as the metric for the Ad Click Prediction Model.
* It highlights the use of the Aggregator Service to achieve low latency and balance workloads.
* To scale the system and reduce latency, tools like Kubeflow can be employed, allowing Ad Generation services to communicate directly with Ad Ranking services.

This comprehensive guide outlines the necessary steps and considerations for designing a robust ads recommendation system, ensuring scalability, efficiency, and low latency in handling a large volume of ad requests.