ML SYSTEM DESIGH GUIDE

Timur Bikmukhametov

Table of Contents

Demand Forecasting Systems

1. Business Problem

Demand forecasting is about predicting how much of something people will need in the future, like products, staff, or delivery capacity.

Here are some typical problems it helps solve:

- **How many items will we sell?** → So we don't run out or overstock.
- **How many staff members do we need next week?** → For retail stores or warehouses.
- **How much should we produce?** → To avoid under- or over-manufacturing.
- **How many trucks or deliveries will we need?** → So logistics teams can plan ahead.

In short, demand forecasting helps businesses **make smarter decisions in advance** based on predicted future needs.

---

2. Industries

Demand forecasting is used in a wide variety of industries, including:

- Retail (e.g., grocery, fashion, electronics)
- Manufacturing (FMCG, auto, pharma)
- Logistics and Supply Chain (3PLs, last-mile delivery, warehouse ops)
- Consumer Goods and E-commerce

---

3. Data Sources and Database Types

The core inputs to demand forecasting models are time-stamped sales and event data. These include:

- **Historical Sales Data:** Captures past unit sales or revenue at the SKU-store-date level. Typically stored in cloud data warehouses such as BigQuery, Snowflake, or Redshift. Smaller teams may use PostgreSQL or MySQL.
- **Product Metadata:** Includes category, price, brand, size, packaging, and launch date. Often stored in relational databases or extracted from ERP systems.
- **Calendar Events:** Public holidays, promotional periods, weekends, and seasonal markers. Can be embedded in static tables, pulled from APIs like Google Calendar, or

---

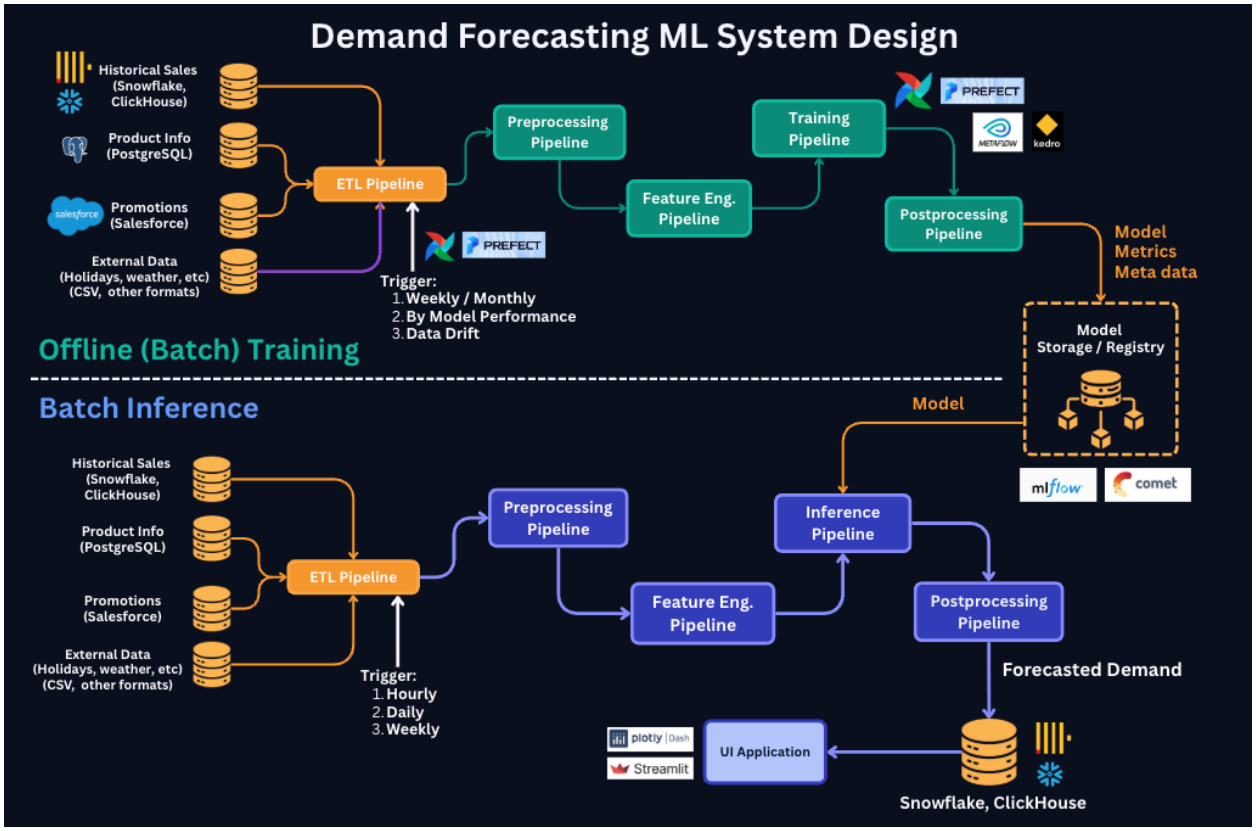
constructed using date-based logic in dbt.

- **Promotional Calendars:** Discounts, campaigns, and marketing events. These are usually managed in CRM or campaign management platforms and synced to data warehouses via API or ETL tools.

- **External Signals:** Weather, economic indicators (e.g. inflation), commodity prices, or event schedules. These are typically sourced from external APIs such as OpenWeatherMap, Quandl, or Google Trends, and stored in flat files or cloud object storage.

---

4. ML System Design



---

5. Common Features

- Lag features: Sales over the past 7, 14, 30 days
- Rolling means/medians (e.g., 7-day avg sales)

- Price elasticity: sales vs price changes
- Promotion indicators (is_promo_active)
- Time-based flags (weekend, holiday, season)
- Weather impact indicators (temperature bin, rain flag)

---

6. Common Targets

- Number of sold units in the next X days (sum / mean / multistep values per day)
- Forecasted demand (sum / mean / multistep values per day)
- Forecasted revenue (sum / mean / multistep values per day)

---

7. Common ML Models Used

- Gradient Boosting (XGBoost, Catboost, LightGBM)
- Prophet / NeuralProphet
- DeepAR, Temporal Fusion Transformer (TFT), N-BEATS
- ARIMA/SARIMA/Holt-Winters for univariate classical forecasting

---

8. Training Frequency

Most demand forecasting systems are trained periodically (not continuously). This is because demand patterns evolve over time, but not necessarily in real-time.

Common training intervals:

- Weekly: Suitable for products with high sales velocity and frequent promotions.
- Monthly: Common for slower-moving items or higher-level forecasts (e.g., by category or region).

Training is usually triggered by one or more of the following:

- A scheduled retraining routine (e.g., every Monday at 2 AM).
- The availability of new data (e.g., 7 or more new days of sales data).
- A performance threshold being exceeded (e.g., if MAPE or RMSE exceeds acceptable limits).
- A business event, such as the start of a new promotion period.

---

9. Inference Frequency

Inference refers to generating forecasts for future periods. Most commonly, forecasts are produced as batch jobs with the following frequencies:

- Daily: The most common setup. Every day, the system generates a new rolling forecast for the next 7, 14, or 30 days per SKU, store, or product category.

- Weekly: Often used for long-term planning in manufacturing or corporate-level dashboards.

Forecasts are scheduled after the ETL pipeline completes and new sales + promotional features are available. Inference results are written to a table or file for downstream applications.

---

10. Feature Store Consideration

Most demand forecasting systems do not require a formal feature store.

A feature store becomes relevant when:

- You need consistent, reusable feature logic across hundreds of time series models.
- Multiple teams are developing models that depend on the same lag/rolling features.
- You plan to support real-time inference (rare for forecasting).

In typical batch setups, reproducible feature generation using tools like dbt, partitioned fact tables, and scheduled workflows (Airflow or Prefect) is sufficient.

---

11. Output and Actioning

Forecasted data is stored as time-series tables in cloud data warehouses and used by downstream tools such as BI dashboards (Looker, Power BI), ERP systems, and inventory planning platforms.

A typical schema might include:

- sku_id: Identifier for product
- store_id: Identifier for location
- forecast_date: Date the forecast is for
- forecast_value: Point estimate
- generated_at: Timestamp when forecast was made

Churn Prediction Systems

1. Business Problem

Churn prediction systems identify users or customers likely to cancel a subscription, stop purchasing, or disengage from a service.

Common business goals include:

- Improving customer retention by identifying high-risk users early.
- Reducing churn-driven revenue loss.
- Supporting targeted interventions such as discounts, reactivation offers, or proactive outreach.

Churn prediction can be modeled as binary classification (will churn / will not churn) or as risk scoring (likelihood to churn).

---

2. Industries

Churn prediction is widely used in any subscription or usage-based business:

- SaaS companies (B2B and B2C)
- Subscription-based mobile apps (streaming, learning, fitness)
- Telecom providers
- Fintech and digital banking
- Gaming and media platforms

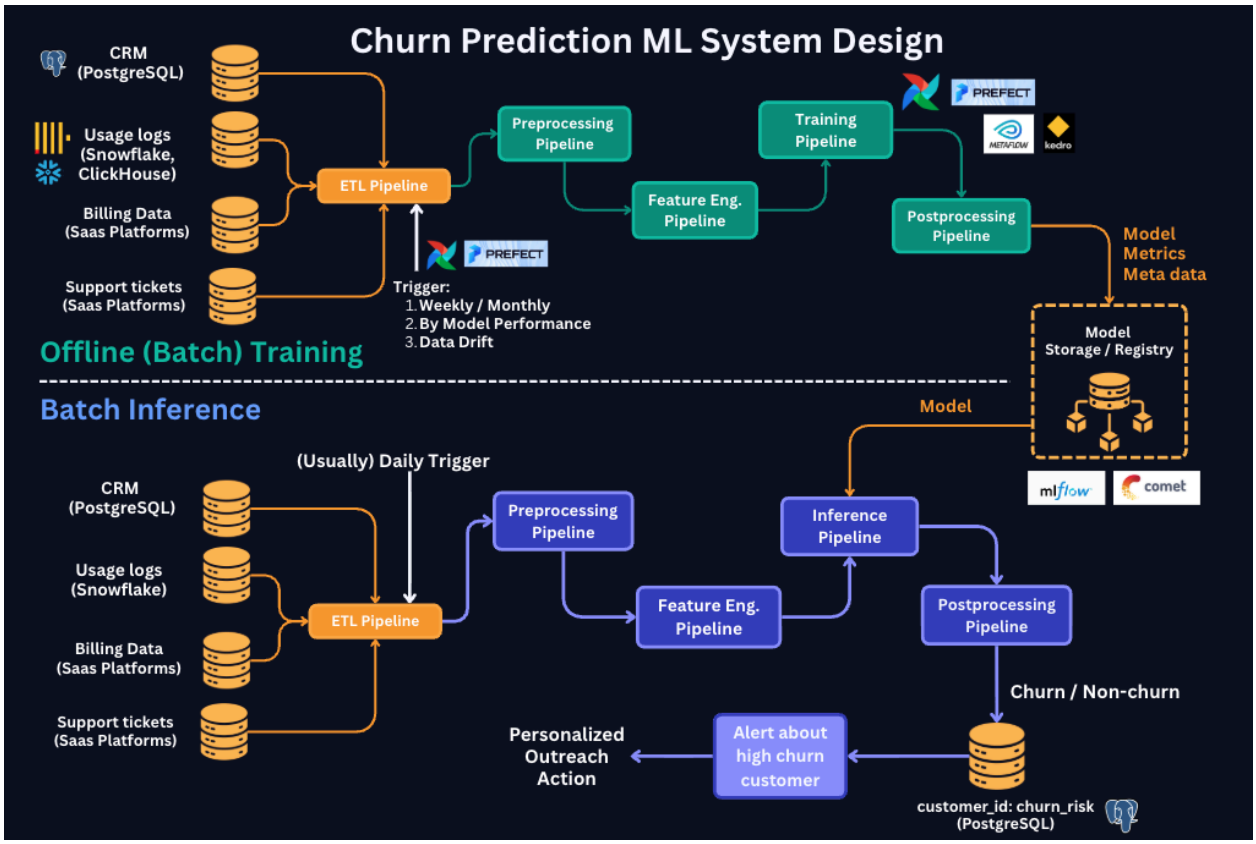
---

3. Data Sources and Database Types

- **CRM Data**: Customer profiles, sign-up dates, plan type, and region. Stored in Salesforce, HubSpot, or internal PostgreSQL databases.
- **Usage Logs**: Session counts, feature interactions, last activity, frequency. Stored in cloud warehouses (e.g., Snowflake) or event DBs (ClickHouse).
- **Support History**: Number of support tickets, open/closed status, resolution times. Pulled from helpdesk platforms like Zendesk or Intercom via API.
- **Payment Records**: Invoices, subscription renewals, and billing failures. Accessed from Stripe, Chargebee, or similar billing systems.
- **Marketing Touchpoints**: Email campaign data, click-throughs, discount redemptions, survey feedback. Often exported from Braze, Iterable, or internal tools.

---

4. ML System Design



---

5. Common Features

- **Recency:** Days since last login or activity
- **Frequency:** Number of logins, sessions, or purchases in the last X days
- **Engagement duration:** Average session length over the last week or month
- **Feature usage counts:** How often key features were used (e.g., reports generated, messages sent)
- **Plan or tier info:** Subscription type, feature access
- **Support activity:** Number of support tickets filed, time to resolution
- **Billing patterns:** Failed payments, payment method changes, recent upgrades/downgrades
- **Marketing interaction:** Click-through rate on emails, offers redeemed
- **Net promoter score (NPS)** or survey feedback, when available
- **Account age:** Days since sign-up or subscription
- **Inactivity streak:** Longest stretch of inactivity in the recent time window

---

6. Common targets

In churn prediction, the goal is to identify users who are likely to disengage or cancel their subscription soon. This is done by analyzing recent user behavior and assigning a binary label based on future inactivity.

Definition of Target (`churn_label`):

- A user is labeled as **churned (1)** if they become inactive (no login, usage, or payment) during a defined **churn window** after the observation date.
- A user is labeled as **retained (0)** if they remain active during that same churn window.

Key Concepts:

- **Observation Window**: A period (e.g., past 30 days) used to extract behavioral features (e.g., sessions, tickets).
- **Churn Window**: A forward-looking window (e.g., next 14 days) used to determine whether the user churned.

Example:
If the model observes user behavior up to August 1 and the churn window is 14 days, then:

- If the user does not log in or take action from August 2 to August 15 → `churn_label = 1`
- If the user logs in or is active during that period → `churn_label = 0`

---

7. Common ML Models Used

- Logistic Regression for interpretable binary classification
- Random Forest and Gradient Boosting (XGBoost, LightGBM)
- Neural Networks for behavioral and event sequence modeling
- Autoencoders or Isolation Forests for anomaly-based churn risk

---

8. Training Frequency

Churn prediction models are retrained frequently to adapt to changes in user behavior and platform engagement trends.

Typical retraining intervals:

- Weekly (most common): Especially when models rely on fast-moving engagement signals.
- Bi-weekly: For platforms with more stable user patterns.

Triggering Conditions:

- Time-based (e.g., retrain every Monday)
- Model performance drop (e.g., drop in precision/recall)
- New cohort added (e.g., launch in a new region)
- Drift detected in key features (e.g., usage patterns or product changes)

---

9. Inference Frequency

- Daily batch inference is the industry standard. Each day, the system scores all active customers and outputs churn probabilities.
- Results are used by marketing systems, CRMs, or customer success teams to drive retention campaigns.

Key output table schema:

- user_id: Identifier of customer or user
- churn_score: Probability of churn (0.0 to 1.0)
- scored_at: Timestamp of inference
- model_version: Model version used for scoring

---

10. Feature Store Consideration

In most churn systems, a dedicated feature store is not necessary.

You may consider it if:

- Multiple models across segments (e.g., per country or tier) require reusable features.
- You plan to support real-time churn scoring (e.g., in-app modals).
- You want strict version control of shared engagement metrics.

Otherwise, a combination of dbt models, Airflow DAGs, and versioned training datasets is sufficient for most use cases.

---

11. Output and Actioning

Scored churn data is stored in cloud data warehouses or CRM-linked databases for downstream use:

- Trigger automated emails or SMS offers
- Surface high-risk users in dashboards
- Feed into customer support workflows

Downstream tools include BI dashboards, Retention tools (Customer.io, Iterable), and CRM automation (Salesforce flows, HubSpot workflows)

Fraud Detection Systems

1. Business Problem

Fraud detection systems aim to identify and prevent fraudulent activity in real time or near real time, reducing financial losses, minimizing reputational damage, and ensuring compliance with regulations.

Common use cases include:

- Detecting unauthorized or suspicious transactions.
- Identifying account takeovers or identity theft.
- Flagging new account fraud.
- Monitoring unusual patterns in user behavior or transactions.

These systems are often used to trigger alerts, hold transactions, or escalate to human review.

---

2. Industries

Fraud detection is a critical use case in sectors such as:

- Fintech and digital wallets
- Traditional banking and credit unions
- E-commerce and marketplaces
- Insurance
- Online gaming and gambling platforms

---

3. Data Sources and Database Types

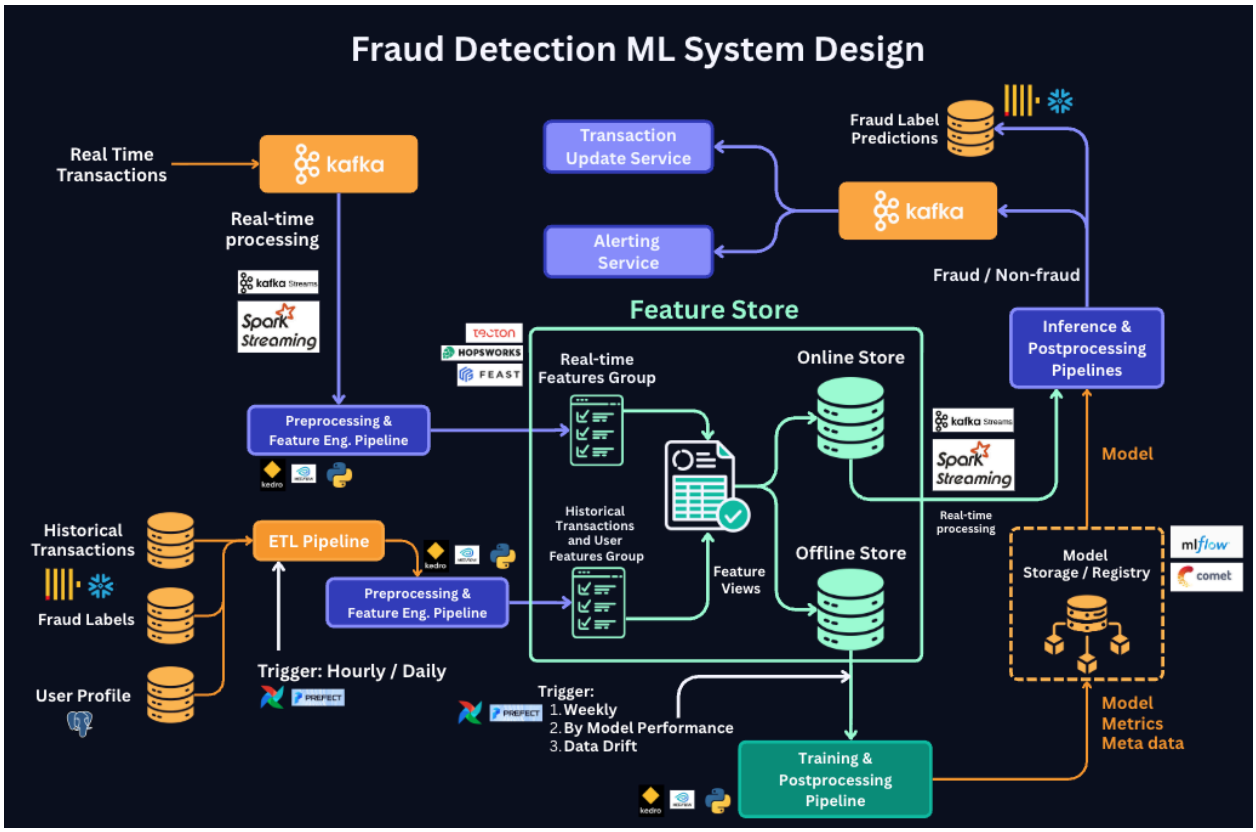
Fraud detection relies on both real-time and historical data:

- **Transaction Data:** Amount, time, location, channel, and merchant. Usually comes from payment gateways or internal transaction logs stored in relational databases or time-series DBs.
- **Device Data:** IP address, user-agent, device fingerprint, and geolocation. Captured via frontend or backend SDKs and often processed through stream ingestion systems like Kafka.
- **Behavioral Biometrics:** Mouse movement, typing speed, swipe patterns, login sequences. Stored in NoSQL databases or real-time processing queues.

- **Account History:** Previous fraud cases, account flags, and historical risk scores. Stored in operational databases and sometimes cached in Redis for fast retrieval.

---

4. ML System Design



---

5. Common Features

Transactional Patterns:

- Amount, frequency, and recency of transactions
- Velocity: spend rate or number of actions in short windows (e.g., 10 mins)

Device & Location Signals:

- New IP, device, or location
- Distance between recent locations
- Proxy/VPN usage flags

Behavioral Signals:

- Time-of-day anomalies
- Login success/failure ratios
- Mouse/typing patterns (if used)

Historical & Graph Features:

- Past fraud flags
- Account age
- Shared devices, cards, or addresses
- Risky connections in account graphs

6. Common ML Models Used

Supervised Classification:

- Logistic Regression
- Random Forest
- XGBoost / LightGBM

Unsupervised Anomaly Detection:

- Isolation Forest
- Autoencoders
- One-Class SVM

---

7. Common Targets

Type: Binary classification target

- `1` = fraudulent transaction
- `0` = legitimate transaction

Granularity:
Usually at the **transaction level** (each transaction is one data point).

Labeling Window and Source

The label is often assigned **retrospectively**, once a transaction has been confirmed as fraud through:

- Manual investigation (e.g., flagged and reviewed by an analyst)

- Chargeback reports (from payment networks)
- External fraud services

---

8. Training Frequency

- Typically retrained every 1–2 weeks for classification models.
- Anomaly detection systems may be unsupervised and updated continuously.
- Retraining is triggered by observed model drift, new fraud patterns, or increased false positives/negatives.

---

9. Inference Frequency

- Real-time inference is the norm. As soon as a transaction or login occurs, it is evaluated for fraud risk.
- Latency must be extremely low (under 100ms), especially in payment workflows.
- For batch risk scoring (e.g., overnight audits), predictions are generated once daily.

---

10. Feature Store Consideration

A feature store is strongly recommended for fraud detection systems, particularly when:

- Features must be served in real-time with low latency.
- Features require time-window aggregation (e.g., transaction count in the past 10 minutes).
- You must ensure consistency between training and live scoring environments.

Popular feature stores: Feast, Tecton, or custom Redis-based feature engines.

11. Output and Actioning

- Risk scores and fraud flags are stored in OLAP or analytical DBs.
- Alerts are published to fraud monitoring dashboards.
- Integration with alerting systems (e.g., PagerDuty) or ticketing (e.g., Jira, Zendesk).
- High-risk transactions may trigger automated blocking, additional verification (e.g., 2FA), or be sent to manual review queues.

Recommendation System

(This example focuses on a video recommendation system)

1. Business Problem

Video recommendation systems aim to deliver personalized content suggestions to maximize user engagement, watch time, and satisfaction. Key business goals include:

- Increasing total watch time per user session.
- Enhancing user retention and return visits.
- Promoting high-value or trending content.
- Reducing churn by keeping users engaged.

These systems address questions like: **“What should we show next to this user to keep them watching/buying?”**

---

2. Industries

- Streaming platforms (Netflix, YouTube, Disney+)
- E-learning (Coursera, Udemy, Khan Academy)
- News and media apps
- Social media platforms with video (TikTok, Instagram, Facebook)

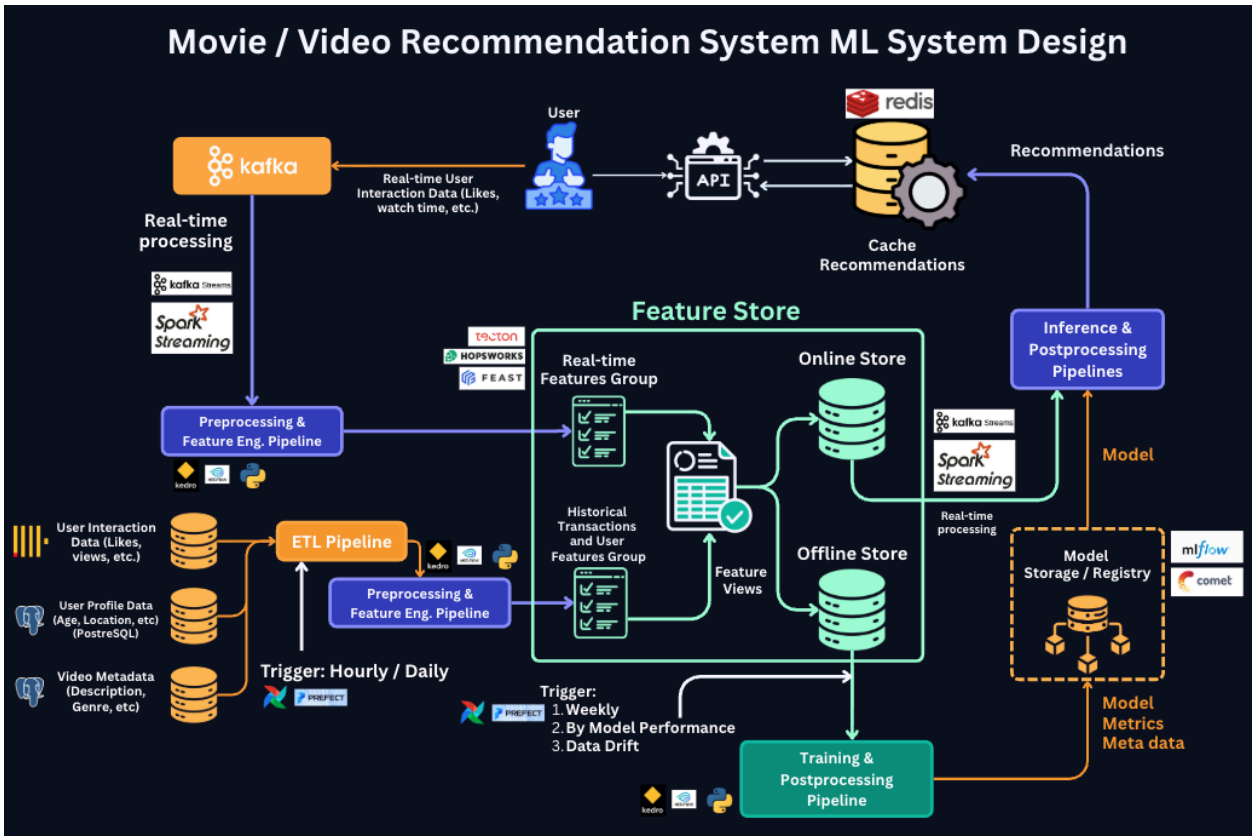
---

3. Data Sources and Database Types

- **User Behavior Logs**: Video views, likes, comments, shares, watch duration. Typically stored in event databases (ClickHouse, BigQuery).
- **User Profile Data**: Age, location, device type, subscription status. Stored in relational DBs or user profile stores.
- **Video Metadata**: Title, tags, category, length, upload time. Managed in relational or document-based databases.
- **Interaction Graphs**: Who watched what, co-watch networks, similar user patterns. Stored in graph DBs like Neo4j.
- **Session History**: Sequence of interactions per session. Stored in time-series DBs or derived in stream aggregators (Kafka + Flink).

---

4. ML System Design



---

5. Common Features

User-Centric Features

- Average watch time per session.
- Diversity score of watched content (genre/category spread).
- Time since last session.
- Ratio of completed vs skipped videos.
- Number of likes/dislikes given recently.
- Preferred content length (short-form vs long-form).
- Language preference inferred from past interactions.

Video-Centric Features

- Freshness score (time since upload).
- Engagement metrics: like ratio, comment rate, share rate.
- Creator popularity score (global or personalized).

- Tag/category encoding vectors.
- Visual/audio embeddings (from models like CLIP or Whisper).

Contextual Features

- Current network speed or bandwidth.
- Time of day and weekday vs weekend.
- Device battery level or screen size (mobile optimization).
- Location (urban vs rural), regional trends.

Interaction Graph Features

- Graph centrality of user/video in the watch graph.
- Number of mutual viewers between the current user and a given video.
- Personalized PageRank or DeepWalk-based scores.

Session Features

- Session position (e.g., first video vs fifth).
- Engagement trajectory (e.g., increasing or decreasing watch times).
- Recent feedback (e.g., user skipped the last 3 videos).

---

6. Common targets

- **Next Video Clicked**: Classification — Did the user click on video X?
- **Watch Time Prediction**: Regression — How many seconds will they watch?
- **Skip Indicator**: Binary classification — Will the user skip this video?
- **Session Completion**: Classification — Will the user complete the session?

---

7. Common ML Models Used

- Matrix Factorization (SVD) for collaborative filtering.
- Nearest-neighbor models (FAISS, Annoy) for similarity retrieval.
- Gradient Boosting Trees (LightGBM, CatBoost) for ranking.
- Deep Learning Recommenders (Two-tower models, DLRM).
- Recurrent Neural Networks or Transformers for session-based recommendations.
- Hybrid models combining collaborative + content + context.

---

8. Training Frequency

- Daily or hourly retraining for user embeddings and trending content.
- Weekly full model retraining with recent sessions.
- Real-time model updates for personalization in session.

---

9. Inference Frequency

- On interaction (e.g., user watches or finishes a video): Predict the next set of recommendations.
- On app open: Pre-load recommended list.
- During the session: Update rankings based on user feedback in real time.

---

10. Feature Store Consideration

- A feature store is often used to serve static features (user profile, video metadata).
- Real-time embeddings updated through online stores or cache (Redis, Faiss).
- Tools: Feast, custom-built stores, or online vector databases.

---

11. Output and Actioning

Recommendation List Generation: The model outputs a ranked list of recommended video IDs for each user, typically limited to top-N (e.g., Top 10, Top 50) items.

Personalized Home Feed: The ranked list is merged with editorial, sponsored, or policy-driven content to construct a final feed.

Real-Time UI Integration: The recommendations are fetched by the front-end application via API and displayed in the user interface—on the homepage, video sidebar, or “next up” section.

A/B Testing Frameworks: Different recommendation models or ranking strategies are tested in production through real-time experimentation to measure KPIs like watch time or click-through rate.

Logging and Feedback Loop: User interactions (clicks, skips, watch time) on recommended items are logged for use in future training and model evaluation.
