Demand Forecasting and Optimization System Design

Use Case: Uber Demand and ETR Forecasting at Airports

Purpose:

To predict demand and estimated time of arrival (ETR) to optimize resource allocation and improve service efficiency.

Data Sources:

1. **Flight Schedules:** Arrival times, delays, cancellations.
2. **Historical Demand Data:** Past ride requests, peak times.
3. **Weather Data:** Temperature, precipitation, severe weather alerts.
4. **App Engagement Metrics:** Active users, session times.
5. **Temporal Features:** Time of day, day of the week, seasonality patterns.

Feature Engineering:

1. **Temporal Features:** Hour of day, day of week, month, holiday indicators.
2. **Weather Features:** Temperature, precipitation, weather conditions.
3. **Flight-Related Features:** Arrival times, delays, cancellations.
4. **Historical Demand Patterns:** Trends, peaks, and troughs.
5. **User Engagement:** Number of active users, app session lengths.

Model Choice:

1. **Time Series Models:**
    - **ARIMA:** Autoregressive Integrated Moving Average for baseline time series forecasting.
    - **Prophet:** Facebook's forecasting tool for handling seasonality and holidays.
2. **Machine Learning Models:**
    - **Gradient Boosting Machines (GBMs):** XGBoost, LightGBM for robust handling of non-linear relationships.
    - **Random Forests:** For ensemble-based predictions with feature importance.
3. **Deep Learning Models:**
    - **LSTM (Long Short-Term Memory):** For capturing sequential dependencies in time series data.
    - **GRU (Gated Recurrent Unit):** Similar to LSTM but more computationally efficient.
4. **Ensemble Methods:**
    - **Combining Models:** Use ensemble methods to aggregate predictions from multiple models for improved accuracy.

System Design:

1. **Data Pipeline:**
   - **Ingestion:** Real-time data ingestion using Apache Kafka or AWS Kinesis.
   - **Preprocessing:** Data cleaning, normalization, and feature generation using Apache Spark or AWS Glue.
   - **Storage:** Use a data lake (e.g., AWS S3) for raw data and a data warehouse (e.g., Amazon Redshift) for structured data.
2. **Feature Store:**
   - **Management:** Centralized feature store (e.g., Feast) for consistent feature engineering and retrieval.
3. **Model Training:**
   - **Batch Processing:** Use distributed computing frameworks (e.g., Apache Spark) for scalable training.
   - **Cross-Validation:** K-fold cross-validation to ensure model robustness.
   - **Hyperparameter Tuning:** Bayesian optimization (e.g., Hyperopt, Optuna) for efficient hyperparameter search.
   - **Class Imbalance Handling:** Techniques like SMOTE or weighted loss functions for handling class imbalance.
4. **Real-Time Forecasting:**
   - **Deployment:** Deploy models as microservices using Docker and Kubernetes for scalability.
   - **Inference Engine:** Use frameworks like TensorFlow Serving or NVIDIA Triton Inference Server for low-latency predictions.
5. **Resource Allocation Engine:**
   - **Integration:** Use the predictions to optimize driver dispatch, surge pricing, and resource allocation.

Scalability:

1. **Cloud Infrastructure:** AWS EC2 for compute, Lambda for serverless functions, S3 for storage.
2. **Distributed Processing:** Kubernetes for orchestrating containerized applications and managing distributed workloads.

Monitoring and Feedback Loop:

1. **Monitoring:** Real-time monitoring of model performance and system health using Prometheus and Grafana.
2. **Feedback Loop:** Continuous learning pipelines to retrain models with new data and adapt to changing conditions.

Evaluation Metrics:

1. **Mean Absolute Error (MAE):** Measure of prediction accuracy.
2. **Root Mean Squared Error (RMSE):** Evaluate the magnitude of prediction errors.

3. **Mean Absolute Percentage Error (MAPE):** Understand relative accuracy.
4. **Business Impact Metrics:** Service efficiency, customer satisfaction, resource utilization.
5. **Latency:** Ensure real-time prediction capabilities within acceptable limits.