🎯 ML SYSTEM DESIGN

▼ 🕵️FRAUD DETECTION

1. Problem Definition

“Imamo transakcije korisnika i želimo da predvidimo da li je neka transakcija fraud ili legitimate.”

✔️ Data je obično **visoko nebalansirana**

✔️ Fraud se stalno menja → **concept drift**

✔️ False negatives su najopasniji (propusteni fraud)

---

2. Data Sources i Ingestion

Objašnjavaš kao junior ali stručno:

Tipični atributi:

- amount
- time of day
- merchant category
- geo-location
- user history: avg spending, transaction velocity
- device ID, IP address
- distance between billed address i transaction location
- historijski fraud flag

Ingestion pipeline:

- batch ingestion (Kafka / EventHub)

- validation: missing values, outliers, schema check
- store u feature store (Databricks, Feast, Snowflake)

---

3. Feature Engineering

Ovo je deo gde Microsoft testira tvoje razmišljanje.

✔️ Transaction-level features

- amount normalized per user
- merchant risk score
- time since last transaction

✔️ User-behavior features

- avg daily spend
- standard deviation user spending
- transaction velocity (broj transakcija u 1h/24h)
- device reputation

✔️ Risk encoding

- frequency encoding merchant categories
- high-risk IP range indicator

✔️ Handling imbalance

- class weights
- focal loss (odličan plus za intervju)
- stratified sampling

---

4. Model Selection

Za fraud detection koristiš:

⭐ 1. Gradient boosting modeli (XGBoost, LightGBM)

Najbolji za tabular data.

⭐ 2. Neural networks (ako ima puno podataka)

- simple MLP
- wide & deep

⭐ 3. Autoencoders (unsupervised anomaly detection)

Koriste se kada nema dovoljno fraud labela.

---

5. Training Pipeline

- train/validation split (stratified)
- hyperparameter tuning (random search / grid search)
- cross-validation (important for imbalanced data)

Loss funkcija:

- binary crossentropy + class weights
- focal loss (napomeni → Microsoft voli ovo)

---

6. Evaluation Metrics

Ovde se testira tvoje razumevanje **precision/recall tradeoff-a**:

⭐⭐ Najbitnije metrike:

- **Recall** (true positive rate) – najbitnije
- **Precision** – da se smanji broj lažnih alarmiranja
- **F1** – balans
- **ROC-AUC** – globalna separacija
- **PR-AUC** – bitno kod nebalansiranih podataka

---

7. Threshold Optimization

Fraud detection nikad ne radi sa “0.5 threshold”.

Umesto toga:

- plot precision-recall curve
- pronađeš optimalni threshold koji minimizuje “fraud leakage”

Ovo je **ogroman plus** ako spomeneš u intervjuu.

---

8. Deployment Strategy

Microsoft često pita **“Kako bi ovo stavila u produkciju?”**

Tvoj odgovor:

Online scoring:

- realtime fraud scoring API
- low-latency model (boosting ili lightweight NN)
- caching user features

Batch scoring:

- dnevno re-skorovanje transakcija
- monitoring drift-a

---

9. Monitoring & Drift Detection

Fraud se menja → model se stalno degradira.

Praćenje kroz:

- feature drift (Kolmogorov-Smirnov test)
- model drift (pad precision/recall)
- data pipeline errors

- latency monitor

Microsoft ovo obožava da čuje → pokazuje “end-to-end ownership”.

---

10. Continuous Retraining

- automatski retrain kada padne performance
- novi fraud patterni → novi features
- offline evaluation → canary rollout

▼ 🧠 RAG Document Search Assistant

Problem Statement

„Kako bi napravila sistem koji korisniku omogućava da postavi pitanje i dobije odgovor iz sopstvenih dokumenata (PDF, text, web scraping), koristeći embedding modele, bez korišćenja FAISS?“

Intervju fokus:

- kako radi pipeline
- gde se čuva embedding
- kako se računa similarity
- kako optimizuješ upite
- kako rešavaš latencu
- kako radiš fallback ako nema rezultata

2️⃣ High-Level Arhitektura

Model itekako vole da vide da ovako objasniš:

1. **Document ingestion**
2. **Chunking**
3. **Embedding (SentenceTransformer)**

4. **Storage (pickle + numpy matrice)**
5. **Retrieval (cosine similarity u Pythonu)**
6. **Context assembly**
7. **LLM (Groq/OpenAI/HF)**
8. **Final answer to user**

To je tvoj realni sistem — i to je apsolutno korektno za Junior Applied Scientist.

3️⃣ Detailed Pipeline (kao u intervjuu, savrseno jasno)

3.1 Document ingestion

- korisnik uploaduje PDF / TXT
- PDF se pretvara u tekst (pdfplumber / PyPDF2)
- vrši se basic cleaning:
  - uklanjaš newline
  - spajaš tekst
  - deliš po paragrafima

Intervjuer gleda: razmišljanje o kvalitetu teksta i normalizaciji.

3.2 Chunking

Tvoj stil:

- fiksni chunk size (npr. 500–800 karaktera)
- overlap izmedju chunkova (npr. 50–150 karaktera)

Zašto?

- smanjuje gubitak konteksta
- bolje preklapanje informacija

Ovo Microsoft voli.

3.3 Embedding

Ti koristiš:

`sentence-transformers/all-MiniLM-L6-v2`

Proces:

- svaki chunk se pretvara u embedding dimenzije 384
- embeddings se čuvaju u numpy matrici `embeddings.npy`

3.4 Storage bez FAISS

Ti koristiš:

- `chunks.pkl` — lista svih chunkova
- `embeddings.npy` — vektori
- sve držiš u RAM tokom rada
- pretraga je linearna (`numpy.dot + norm`)

Za juniore potpuno prihvatljivo.

Niko ne očekuje skaliranje na milijarde dokumenata.

3.5 Retrieval (bez FAISS)

Tvoj kod radi:

1. Embeduj korisnički upit
2. Izračunaj cosine similarity između query vektora i svih chunk vektora
3. Sortiraj rezultate
4. Uzmi top K chunkova

Ovo je 100% validan retrieval algoritam za manje datasete.

3.6 Context assembly

- spajaš top chunkove u jedan “context window”
- ograničiš veličinu (da ne pređe token limit)

3.7 LLM Answering

Ti koristiš:

- Groq Llama
- OpenAI fallback

Prompter radi ovako:

System: “Ti si dokument asistent. Koristi samo dati context.”

User: pitanje

Context: top chunkovi

3.8 Final Answer Formatting

Daješ korisniku:

- odgovor
- relevantne delove dokumenta (chunkovi)

Microsoft voli ovo → zove se **answer grounding**.

4️⃣ Kako ovo objasniti na intervjuu? (idealno)

„Napravila sam RAG pipeline koji koristi SentenceTransformer embedding model.

Dokument se chunkuje, embeddinguje i čuva u pickle + numpy formatima.

Retrieval vršim pomoću cosine similarity pretrage bez FAISS-a, što je optimalno za manje datasete i serverless okruženja poput Render-a.

LLM koristi retrieved chunkove kao context za generisanje sigurnih odgovora.“

→ kratko, jasno, savršeno za Microsoft.

5️⃣ Prednosti tvog dizajna

- jednostavan
- nema spoljne zavisnosti
- idealno za besplatne hostinge
- lako se debuguje
- odlično za diplomski rad

6️⃣ Moguća unapređenja (koja Microsoft voli da čuje)

- dodavanje FAISS ili ChromaDB u budućnosti
- embedding caching
- async retrieval
- thread-safe loading
- monitoring preko logging-a
- sigurnosni filteri (PII removal)

▼ 🏞️ IMAGE CLASSIFICATION PIPELINE

1️⃣ Problem Definition

“Zadatak je napraviti sistem koji prima sliku i predvidja kojoj klasi pripada (npr. mačka/pas ili 100 razlicitih kategorija).”

Potrebno je definisati:

- **ulaz:** slika (JPEG, PNG)
- **izlaz:** klasa ili verovatnoce klasa
- **latency zahtevi:** npr. <100ms

- **skaliranje:** veliki broj zahteva?
- **offline ili online:** zavisi od aplikacije

2️⃣ Data Pipeline

2.1 Data Collection

- Slike mogu biti iz dataset-a (CIFAR, ImageNet) ili interno prikupljene.
- Mora postojati balans klasa.

2.2 Data Preprocessing

- Resize svih slika (npr. na 224×224)
- Normalizacija piksela (mean/std)
- Pretvaranje u tensors

2.3 Data Augmentation

Obavezno za smanjenje overfittinga:

- random crop
- flip
- rotation
- color jitter
- random erasing

Augmentacije se rade **u runtime-u**, ne unapred.

3️⃣ Modeling

Tri opcije:

🔷 Opcija 1: CNN from scratch (za male datasete loše)

- sporije treniranje
- slabije performanse
- koristi se samo za ucenje ML osnova

🔷 Opcija 2: Transfer Learning (realno rešenje za industriju)

Najbolja opcija.

Pretrained modeli:

- ResNet50
- MobileNetV2
- EfficientNet
- VGG16

Strategija:

- zamrznuti backbone
- trenirati poslednji classifier sloj
- eventualno fine-tuning ako ima dosta podataka

🔷 Opcija 3: Large Vision Transformers (ViT, CLIP)

Za veliki industrijski sistem, ali preteško za junior intervju.

4️⃣ Training Pipeline

1. Train/validation/test split
2. Loss funkcija
    - Cross entropy

3. Optimizer
    - Adam ili SGD
4. Learning rate scheduling
    - cosine annealing
    - step decay
5. Monitoring metrika
    - accuracy
    - confusion matrix
    - per-class precision/recall

5️⃣ Evaluation

Šta obavezno kazati u intervjuu:

- overall accuracy
- per-class accuracy
- F1 score
- confusion matrix analitika
- overfitting analiza (train vs val loss)
- misclassification analysis
- interpretability: Grad-CAM

6️⃣ Deployment Pipeline

6.1 Model Export

- ONNX ili TorchScript
- TensorFlow SavedModel

6.2 Serving

Opcije:

(A) Batch Mode

- procesira slike u serijama
- koristi se kada nema real-time zahteva

(B) Real-Time API

- FastAPI / Flask
- model ucitan u RAM-u
- GPU ako je skupo u CPU

7️⃣ System Architecture (Microsoft style)

Client → API Gateway → Preprocessing → Model Inference → Postprocessing → Response

Scaling:

- autoscaling GPU worker-a
- model caching
- batching inference poziva
- CDN za ulazne slike

8️⃣ Monitoring

Microsoft uvek voli ovo:

- inference latency
- accuracy drift

- model drift
- data drift (slike promenjene u bojama/kvalitetu)
- watchdog za degradaciju performansi

9️⃣ Fail Modes & Risk Mitigation

- pretreniran model → resiti augmentacijom + regularizacijom
- problem sa out-of-distribution slikama → OOD detection
- bias u datasetu
- preveliki latency → kvantizacija modela

1️⃣0️⃣ Kako bi ti ovo uradila u praksi (tvoj odgovor)

U intervjuu mozes reci:

“Napravila bih end-to-end image classification sistem koristeci transfer learning, npr. ResNet50.
Podatke bih normalizovala i augmentirala u realnom vremenu.
Model bih trenirala uz early stopping i learning rate scheduler.
Za evaluaciju bih koristila accuracy i confusion matrix.
Za deploy bih izvezla model u ONNX i servirala ga preko FastAPI-ja, sa monitoringom latency-ja i accuracy drift-a.”

▼ 💻 Recommender System

Problem:

Imamo platformu (npr. video, muzika, e-commerce) i hocemo da svakom korisniku preporucimo relevantne stavke (filmove, klipove, proizvode).

1️⃣ Ciljevi i zahtevi

Functionali ciljevi:

- personalizovane preporuke po korisniku
- brze preporuke (latencija niska)
- sposobnost da se preporuke osveze kako dolazi novi sadrzaj i novi user signali

Ne-funkcionalni ciljevi:

- skalabilnost (mnogo korisnika i mnogo item-a)
- nizak response time (npr. <100ms za serving)
- mogucnost online/AB evaluacije

Metrici:

- CTR (click-through rate)
- Watch time / view duration
- Conversion rate (kupovina)
- Retention (koliko se korisnik vraca)

2️⃣ Podaci

Vrste podataka:

- **User features**
  - ID, lokacija (ako postoji), device, vreme registracije
  - istorija gledanja/kupovine
  - eksplicitni feedback (rating, like/dislike)
- **Item features**
  - ID, kategorija, tagovi, opis, cena, slika, zanr, trajanje
  - popularnost (broj pregleda/kupovina)
- **Interaction logs**
  - user_id, item_id, timestamp
  - action: view, click, add_to_cart, purchase, like, share

- trajanje pregleda (watch time)

3️⃣ Labeling – kako pravimo labelu za trening?

Primer:

- definisemo pozitivan primer: user je kliknuo / pogledao / kupio item
- negativni primer: item je prikazan ali nije kliknut (impressions bez klika)

To postaje **binary klasifikacija**:

- y = 1 ako je interakcija "pozitivna" (click/purchase)
- y = 0 ako je bilo prikazano ali bez akcije

4️⃣ Feature engineering

User features:

- broj ukupnih pregleda
- broj pregleda po kategoriji
- poslednje aktivnosti (recent history)
- embedding korisnika (iz istorije interakcija)

Item features:

- kategorija, tagovi (one-hot ili embedding)
- broj ukupnih pregleda
- globalni CTR
- starost item-a

User-Item features (par):

- da li korisnik voli tu kategoriju (procenat pregleda iz te kategorije)
- slicnost izmedju item-a i poslednjih item-a koje je user gledao

---

5️⃣ Arhitektura modela (klasicna verzija)

Najcesce se radi u **dve faze**:

5.1 Candidate Generation (brzi, siroki filter)

Cilj: od npr. 1M item-a odabrati 500–1000 kandidata.

Modeli:

- simple: popularni item-i po kategorijama
- collaborative filtering (matrix factorization, embeddings)
- ANN pretraga (nearest neighbors po embedding prostoru)

5.2 Ranking model (detaljan, tezi model)

Cilj: od tih ~1000 kandidata napraviti top 10–20 preporuka.

Model:

- Gradient Boosted Trees (XGBoost / LightGBM)
- ili DNN koji uzima user + item + interaction feature-e

Output:

- score = verovatnoca da ce user kliknuti/pogledati/kupiti

6️⃣ Trening pipeline

1. **Data pipeline**
   - vezemo se na logove (clickstream)
   - kreiramo trening dataset: (user, item, features, label)
2. **Splitovanje podataka**
   - train = stari podaci
   - val/test = noviji period (time-based split)
3. **Trening ranking modela**
   - optimizujemo logloss / AUC / NDCG
   - hyperparameter tuning (grid/random)

4. **Offline evaluacija**
    - CTR@k, NDCG@k, MAP@k

7️⃣ Serving u produkciji

Koraci kada user udje na stranicu:

1. User request stigne (user_id, kontekst, device, vreme)
2. **Candidate generation service** vrati npr. 1000 kandidata
3. **Ranking service**:
    - spoji user + item features
    - pusti kroz ranking model
    - sortira po score-u
4. Vrati top N item-a frontendu (npr. 10–20 preporuka)

Cache:

- popularne liste za anonymous korisnike
- per-user cache za kratke periode

8️⃣ Monitoring i iteracije

Pratimo:

- realni CTR, watch time, conversion
- distribuciju scores kroz vreme
- data drift (npr. novi tip sadrzaja)

Radimo:

- AB test stari model vs novi model
- degradaciju ako novi model ne radi bolje

9️⃣ Kako bi ovo ispricala na intervjuu (kratka verzija za tebe)

Kada te pitaju npr:

“Kako bi dizajnirala recommendation sistem za video platformu?”

Ti mozes ovako (ukratko):

> Krenula bih od ciljeva – da sistem bude personalizovan, brz i skalabilan.
>
> Zatim bih definisala koje podatke koristimo: user istorija, item meta-podaci, logovi interakcija.
>
> Labela bi bila da li je user kliknuo/pogledao ili ne.
>
> Feature-i bi ukljucivali user profile, item karakteristike i user-item interakcione feature-e.
>
> Sistemi ovog tipa se tipicno rade u dve faze: candidate generation (brzi model ili embedding ANN) i ranking (npr. XGBoost ili DNN).
>
> Trening bih radio na istorijskim logovima, sa time-based split-om, a evaluaciju preko CTR@k, NDCG i slicnih metrika.
>
> U produkciji bih imao poseban servis za generisanje kandidata i servis za ranking, sa cache-om za popularne liste i AB testiranje novih modela.“

▼ 😬 NLP Sentiment Analysis (End-to-End System Design)

1️⃣ Problem Definition

Cilj:

Napraviti model koji klasifikuje tekst (recenziju, komentar, tvit) kao:

- pozitivno
- neutralno
- negativno

Tip problema:

Supervised multi-class classification.

Input: nečist tekst (sleng, emoji, greške, varijacije)

Output: labela + verovatnoće klasa

Microsoft voli da prvo definišeš **“What is the business impact?”**:

- analiza korisničkog zadovoljstva
- monitoring proizvoda
- automatsko sortiranje recenzija
- podrška korisnicima

2️⃣ Data Pipeline (Ingestion + Cleaning)

2.1. Data Sources

- recenzije sa aplikacija
- korisnički komentari
- social media
- anketni odgovori
- interni logovi

2.2. Data Cleaning

Najčešći koraci:

- uklanjanje HTML tagova
- uklanjanje URL-ova
- uklanjanje emoji (ili pretvaranje u text tokens)
- lowercasing
- normalizacija whitespace-a
- skidanje stop words (opciono — zavisi od modela)

2.3. Labeling

Ako imaš ručno labelovani dataset, super.

Ako ne, koristi:

- heuristike
- weak supervision
- active learning
- već labelovane recenzije (npr. IMDb)

3️⃣ Feature Engineering (Traditional vs Modern)

Microsoft voli da kažeš da postoje **dva pristupa**:

A) Tradicionalni (ako je dataset mali)

- bag-of-words
- TF-IDF
- n-grams
- SVD/PCA redukcija

B) Moderni (preporučeno — Transformers)

- BERT embeddings
- DistilBERT kao brža verzija
- finetuning pretreniranog modela

Intervju savet:

“I would prefer a transformer-based architecture because sentiment often depends on context, negation, sarcasm and word order.”

4️⃣ Model Architecture

Model 1 (baseline): Logistic Regression + TF-IDF

Za poređenje i proveru pipeline-a.

Model 2 (production candidate): Finetuned BERT

Koraci:

1. Tokenizacija (WordPiece)
2. Padding/truncation
3. Masking
4. Fine-tuning poslednjeg sloja
5. Softmax na kraju

5️⃣ Training Pipeline

5.1. Train/Val/Test Split

Najčešće: 80/10/10

5.2. Handling Class Imbalance

- class weights
- upsampling
- focal loss

5.3. Hyperparameter Tuning

- learning rate (1e-5, 2e-5, 3e-5)
- batch size (16, 32)
- sequence length
- epochs (2–4 za BERT)

6️⃣ Evaluation Metrics

Microsoft često pita:

- Accuracy (samo ako su klase uravnotežene)
- F1 score (najbitniji!)
- Precision / Recall
- Confusion matrix
- ROC/AUC (opciono)

Napomena koju intervjueri obožavaju:

“Recall is important when detecting harmful/negative comments.”

7️⃣ Error Analysis (Critical Step)

Ovo **uvek** kažeš!

Analiziramo:

- sarkazam
- negaciju (“I don’t like this phone at all”)
- duge recenice
- kratke fraze (“meh”, “ok”, “wow”)
- domenski specifičan jezik

Ako želiš da impresioniraš:

“I would collect hard examples from misclassified samples and retrain the model using hard-negative mining.”

8️⃣ Deployment Architecture

Batch scoring

Ako korisnici ostavljaju mnogo komentara → koristi se Spark job koji jednom dnevno procesira podatke.

Real-time API

Najčešće REST endpoint:

- input: JSON sa tekstom
- output: label + probability
- latency < 50 ms (koristi DistilBERT ili quantization)

Preporuke:

- TorchScript ili ONNX za brzu inferencu
- quantization (8-bit)
- autoscaling kroz Kubernetes / Azure ML

9️⃣ Monitoring After Deployment

Šta se prati:

- data drift (rečnik se menja)
- sentiment drift (korisnici menjaju način pisanja)
- prediction drift
- latency
- memory usage

Ako nešto odskače → automatski retraining pipeline.

🔟 How Would You Improve the System? (Obavezno reći)

- dodatni domenski pretraining (Domain Adaptive Pretraining – DAPT)
- koriscenje RoBERTa / DeBERTa modela

- cross-lingual modeli (ako podržavaš više jezika)
- uklanjanje sarkazma korišćenjem posebnih datasetova
- prompt engineering za LLM modele

🎯 Kratak rezime odgovora koji treba da izgovoriš na intervjuu:

> “For sentiment analysis I would build a BERT-based pipeline that starts with data cleaning, tokenization and balanced train/val/test split.
>
> I would finetune a pretrained transformer such as DistilBERT, evaluate with F1 score, run detailed error analysis focusing on sarcasm and negation, deploy the model as a low-latency API using ONNX optimization, and monitor data drift for automatic retraining.”