

ML System Design Templates

Building Production ML Systems for Staff/Principal Interviews

ML SYSTEM DESIGN FRAMEWORK

The MADE Framework

Model - Training & Architecture
API - Inference & Serving
Data - Pipelines & Features
Evaluation - Metrics & Monitoring
For each ML system design:

1. **Clarify Requirements** (5 min)
 - Use cases, scale, latency constraints
 - Online vs offline, batch vs real-time
 - Data availability, labeling budget
2. **Model Selection** (10 min)
 - Algorithm choice & justification
 - Training infrastructure needs
 - Model complexity vs latency trade-off
3. **Data Pipeline** (10 min)
 - Feature engineering strategy
 - Data collection & labeling
 - Feature store architecture
4. **Serving Architecture** (10 min)
 - Real-time vs batch predictions
 - Scaling & latency optimizations
 - Model deployment strategy
5. **Evaluation & Monitoring** (10 min)
 - Offline metrics, online A/B testing
 - Model drift detection
 - Feedback loops & retraining

COMMON ML SYSTEM PATTERNS

1. Search & Ranking Systems

Examples: Google Search, Amazon product ranking, YouTube recommendations
Architecture:

```
Query → Candidate Generation (Fast, Broad)
      ↓
      Ranker (Slow, Precise)
      ↓
      Re-ranker (Personalization)
      ↓
      Results
```

Candidate Generation:

- **Goal:** Reduce 1B items → 10K candidates (99.999% reduction)
- **Methods:** ANN (FAISS), inverted index, embedding similarity
- **Latency:** < 10ms

Ranking:

- **Goal:** Rank 10K candidates → Top 100 results
- **Model:** XGBoost, LightGBM, or two-tower neural network
- **Features:** Query-doc relevance, user history, CTR, engagement
- **Latency:** < 100ms

Re-ranking:

- **Goal:** Personalization, diversity, business rules
- **Model:** Lightweight NN or rule-based
- **Latency:** < 10ms

Key Metrics:

- **Offline:** NDCG@10, MRR, Precision@K
- **Online:** CTR, time-to-click, bounce rate, revenue

Scaling:

- **Candidate generation:** Sharded by item ID, ANN index distributed
- **Ranking:** Model replicas behind load balancer
- **Caching:** Query cache (Redis), result cache

2. Recommendation Systems

Examples: Netflix, Spotify, TikTok, Amazon "You May Also Like"
Two-Stage Architecture:

```
User → Candidate Generation
      ↓
      Ranking (Predicted Rating/CTR)
      ↓
      Top-N Recommendations
```

Candidate Generation Approaches:

A. Collaborative Filtering

- **Matrix Factorization:** Users × Items → User embeddings × Item embeddings
- **Pros:** Simple, works with implicit feedback
- **Cons:** Cold start problem

```
# User-Item matrix factorization
R = U × V^T
where U: (n_users, k), V: (n_items, k)

# ALS (Alternating Least Squares)
min ||R - UV^T||^2 + (||U||^2 + ||V||^2)
```

B. Two-Tower Neural Network

```
User Features → User Tower (NN) → User Embedding
Item Features → Item Tower (NN) → Item Embedding
                      ↓
                Dot Product → Score
```

C. Content-Based

- **Method:** TF-IDF, BERT embeddings for item features
- **Similarity:** Cosine similarity between user profile & items

Ranking Model:

- **Input:** User features, item features, context (time, device)
- **Model:** Deep & Wide, DeepFM, xDeepFM
- **Objective:** Predict CTR, watch time, or rating

Cold Start Solutions:

- **New users:** Popular items, demographic-based, onboarding survey
- **New items:** Content-based, similar to trending items

Key Metrics:

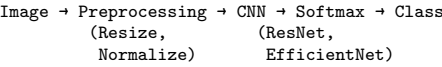
- **Offline:** RMSE, Precision@K, Recall@K, NDCG
- **Online:** CTR, engagement time, conversion rate
- **Diversity:** Intra-list similarity (avoid filter bubble)

Scaling:

- **Embeddings:** Store in vector DB (Pinecone, Milvus, FAISS)
- **ANN search:** HNSW, ScaNN for nearest neighbor retrieval
- **Batch updates:** Recompute embeddings nightly
- **Real-time:** Stream processing (Kafka + Flink) for user events

3. Computer Vision Systems

A. Image Classification Example: Content moderation, medical diagnosis
Architecture:



Model Selection:

- **High accuracy:** EfficientNet, Vision Transformer (ViT)
- **Low latency:** MobileNet, SqueezeNet
- **Transfer learning:** Pretrain on ImageNet, fine-tune on domain

Data Pipeline:

- **Augmentation:** Random crop, flip, rotation, color jitter
- **Labeling:** Mechanical Turk, active learning for hard examples
- **Class imbalance:** Weighted loss, oversampling minority class

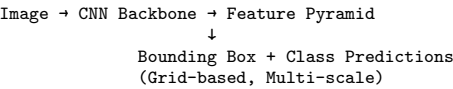
Serving:

- **Batch:** Process uploaded images async (S3 → SQS → Lambda)
- **Real-time:** TensorFlow Serving, TorchServe on GPU instances
- **Edge:** Model quantization (INT8), TFLite for mobile

B. Object Detection Example: Self-driving cars, surveillance
Model Options:

- **Two-stage:** Faster R-CNN (high accuracy, slow)
- **One-stage:** YOLO, SSD (fast, real-time)
- **Anchor-free:** FCOS, CenterNet

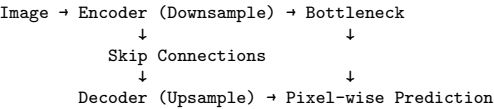
Architecture (YOLO):



Post-processing:

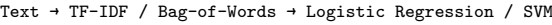
- **NMS:** Non-max suppression (remove duplicate boxes)
- **Threshold:** Confidence score filtering

C. Image Segmentation Example: Medical imaging, autonomous driving
Architecture (U-Net):

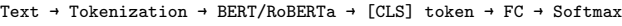


4. Natural Language Processing

A. Text Classification Examples: Sentiment analysis, spam detection, content categorization
Approaches:
Classical (Small data, low latency):



Deep Learning (Large data, high accuracy):



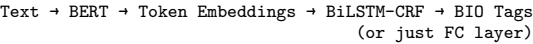
Model Selection:

- **High accuracy:** BERT-large, RoBERTa, DeBERTa
- **Low latency:** DistilBERT (40% faster, 97% accuracy)
- **Very low latency:** TF-IDF + Logistic Regression

Serving:

- **Real-time:** TorchServe, TensorFlow Serving
- **Batch:** Spark for large-scale processing
- **Optimization:** ONNX, TensorRT, quantization

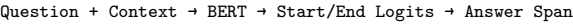
B. Named Entity Recognition (NER) Example: Extract names, dates, locations from text
Architecture:



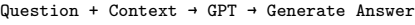
Output Format:

- **BIO tags:** B-PER, I-PER, B-ORG, I-ORG, B-LOC, O
- **Example:** "Barack Obama visited Paris"
- **Tags:** B-PER I-PER O B-LOC

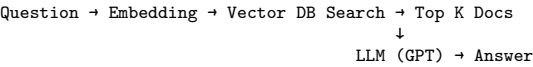
C. Question Answering Example: Chatbots, search engines
Extractive QA (BERT-based):



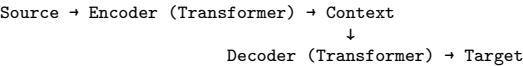
Generative QA (GPT-based):



Retrieval-Augmented (RAG):



D. Machine Translation Architecture:

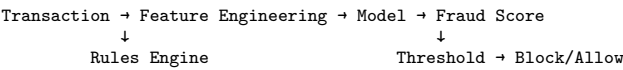


Key Components:

- **Encoder:** Self-attention on source sentence
- **Decoder:** Self-attention + cross-attention to encoder
- **Tokenization:** Byte-Pair Encoding (BPE), SentencePiece
- **Beam Search:** Generate top-K translations

5. Fraud Detection

Examples: Credit card fraud, fake accounts, click fraud
Architecture:



Feature Engineering:

- **Transaction features:** Amount, time, location, device
- **User features:** Account age, past behavior, velocity (txns/hour)
- **Graph features:** Social network, entity connections
- **Aggregations:** Rolling windows (1hr, 24hr, 7d)

Model Selection:

- **Traditional:** XGBoost, Random Forest (interpretable)
- **Deep Learning:** Autoencoders for anomaly detection
- **Graph:** Graph Neural Networks (GNN) for network fraud

Handling Imbalance:

- **Sampling:** SMOTE, undersampling majority class
- **Loss:** Focal loss, weighted cross-entropy
- **Metrics:** Precision-Recall curve (not accuracy!)

Real-time Requirements:

- **Latency:** < 100ms for payment approval
- **Serving:** Model in-memory, feature cache (Redis)
- **Fallback:** Rule-based system if model fails

Key Metrics:

- **Precision:** % of flagged transactions that are actually fraud
- **Recall:** % of fraud caught
- **F1-score:** Harmonic mean of precision/recall
- **Business:** False positive cost vs fraud loss prevented

Training Infrastructure

Small Models (⩽ 1GB):

- **Single GPU:** NVIDIA V100, A100
- **Framework:** PyTorch, TensorFlow
- **Training time:** Hours to days

Large Models (> 10GB):

- **Distributed training:** Data parallelism (multiple GPUs)
- **Model parallelism:** For models larger than single GPU memory
- **Tools:** Horovod, PyTorch DDP, DeepSpeed

Architecture:

Parameter Server (PS) Architecture:

Workers → Compute Gradients → PS → Aggregate → Update Model
← Fetch Parameters ←

Ring-AllReduce (Horovod):

Worker 1 Worker 2 Worker 3 Worker 4

(All workers communicate in ring, no PS bottleneck)

Hyperparameter Tuning:

- **Grid search:** Exhaustive, expensive
- **Random search:** Better than grid for high-dimensional
- **Bayesian optimization:** Optuna, Ray Tune
- **Early stopping:** Prune bad trials (ASHA, Hyperband)

Model Serving

Online Serving (Real-time):

Client → Load Balancer → Model Server (GPU/CPU)
↓
Response (< 100ms)

Serving Options:

- **TensorFlow Serving:** gRPC/REST, GPU support, batching
- **TorchServe:** PyTorch models, multi-model serving
- **ONNX Runtime:** Cross-framework, optimized inference
- **Custom:** Flask/FastAPI + model.predict()

Optimization:

- **Batching:** Group requests for GPU efficiency (trade latency for throughput)
- **Quantization:** FP32 → INT8 (4× smaller, 2-4× faster)
- **Model pruning:** Remove low-importance weights
- **Knowledge distillation:** Train small model to mimic large model

Batch Serving (Offline):

Scheduled Job (Airflow) → Spark → Model.predict(large_df)
↓
Write to DB/S3

Use cases:

- **Precompute:** Daily recommendations, email digests
- **Batch scoring:** Risk scores for all users

Model Deployment Strategies

Blue-Green Deployment:

Production Traffic → Blue (Current Model v1)
↓
Switch after validation

Production Traffic → Green (New Model v2)

Canary Deployment:

95% Traffic → Model v1 (Stable)
5% Traffic → Model v2 (Canary)
↓
Monitor metrics
↓
Gradually increase to 100%

Shadow Deployment:

Production Traffic → Model v1 (Serve responses)
↓
Model v2 (Log predictions, no serving)
↓
Compare predictions offline

A/B Testing:

50% Users → Model A (Control)
50% Users → Model B (Treatment)
↓
Compare metrics (CTR, revenue)

MONITORING & EVALUATION

Offline Evaluation

Train/Val/Test Split:

- **Random split:** 70/15/15 (IID data)
- **Temporal split:** Train on past, test on future (time series)
- **Stratified:** Preserve class distribution (imbalanced data)

Cross-Validation:

- **K-fold:** 5-fold or 10-fold (small datasets)
- **Stratified K-fold:** Preserve class ratios
- **Time-series:** Expanding window (no future leakage)

Metrics by Task:

Classification:

- **Binary:** Precision, Recall, F1, AUC-ROC, AUC-PR
- **Multi-class:** Accuracy, Macro/Micro F1, Confusion matrix
- **Imbalanced:** Precision-Recall curve (not ROC!)

Ranking:

- **NDCG@K:** Normalized Discounted Cumulative Gain
- **MRR:** Mean Reciprocal Rank
- **Precision@K, Recall@K**

Regression:

- **MSE, RMSE:** Sensitive to outliers
- **MAE:** Robust to outliers
- **R²:** Proportion of variance explained
- **MAPE:** Mean Absolute Percentage Error

Online Evaluation (A/B Testing)

North Star Metrics:

- **Search:** CTR, time-to-click, query reformulation rate
- **Recommendations:** CTR, watch time, conversion rate
- **Ads:** CTR, conversion rate, revenue per user
- **Social:** Engagement (likes, comments), DAU, session time

Statistical Significance:

- **Sample size:** Calculate required users for statistical power
- **Confidence:** 95% confidence interval
- **P-value:** ⩽ 0.05 for significance
- **Duration:** Run for at least 1-2 weeks (account for weekly patterns)

Guardrail Metrics:

- **Latency:** p95, p99 latency should not increase
- **Error rate:** Should not increase
- **User complaints:** Monitor feedback, reports

Model Monitoring

Data Drift:

- **Feature distribution shift:** Track mean, std, quantiles over time
- **Detection:** KL divergence, Kolmogorov-Smirnov test
- **Action:** Retrain model on recent data

Concept Drift:

- **Definition:** Relationship between features and label changes
- **Example:** COVID changed travel patterns, broke travel models
- **Detection:** Monitor online metrics (CTR, accuracy) degradation
- **Action:** Retrain with new labels, add new features

Prediction Drift:

- **Monitor:** Distribution of predicted scores
- **Alert:** If predicted CTR drops from 5% → 1%

Monitoring Dashboard:

Grafana + Prometheus:

- Prediction latency (p50, p95, p99)
- QPS (queries per second)
- Model accuracy over time
- Feature distribution plots
- Error rate, null predictions

Retraining Strategy

Periodic Retraining:

- **Schedule:** Daily, weekly, or monthly
- **Trigger:** Cron job (Airflow, Kubernetes CronJob)
- **Use case:** Slowly changing data (e.g., content recommendations)

Trigger-based Retraining:

- **Metric degradation:** If AUC drops by \geq 5%
- **Data drift detected:** Feature distribution change
- **Manual trigger:** New product launch, seasonal event

Online Learning:

- **Incremental updates:** Update model with new data (no full retrain)
- **Use case:** High-velocity data (ads, fraud detection)
- **Challenges:** Catastrophic forgetting, stability

CAPACITY ESTIMATION

Training Cost

GPU Hours Estimation:

$$\text{Training time} = (\text{Dataset size} \times \text{Epochs} \times \text{FLOPs per sample}) / (\text{GPU throughput} \times \text{Batch size})$$

Example: ResNet-50 on ImageNet

- **Dataset:** 1.2M images
- **Epochs:** 100
- **FLOPs:** 4 billion per image
- **GPU:** V100 (125 TFLOPS FP16)
- **Batch size:** 256
- **Time:** 10 days on 8 V100s

Cost:

- **AWS p3.8xlarge:** $4 \times \text{V100}$, \$12.24/hour
- **10 days:** $240 \text{ hours} \times \$12.24 = \mathbf{\$2,938}$

Serving Cost

QPS to Instances:

$$\text{Instances} = (\text{Target QPS} \times \text{Latency}) / (\text{Batch size} \times \text{Parallelism})$$

Example: Image Classification Service

- **Target QPS:** 1,000 requests/sec
- **Latency:** 50ms per image (with batching)
- **Batch size:** 32 images
- **GPU throughput:** 640 images/sec ($32 \times 20 \text{ batches/sec}$)
- **Instances needed:** $1000 / 640 = \mathbf{2 \text{ GPUs}}$

Cost:

- **AWS g4dn.xlarge:** $1 \times \text{T4 GPU}$, \$0.526/hour
- **Monthly (2 instances):** $2 \times 730 \times \$0.526 = \mathbf{\$768/month}$

Storage

Feature Store:

$$\text{Storage} = \text{Num users} \times \text{Features per user} \times \text{Bytes per feature}$$

Example (100M users, 500 features, 4 bytes each):
$$= 100\text{M} \times 500 \times 4 \text{ bytes} = 200 \text{ GB}$$

Model Storage:

- **BERT-base:** 440MB ($110\text{M parameters} \times 4 \text{ bytes}$)
- **GPT-3:** 700GB ($175\text{B parameters} \times 4 \text{ bytes}$)
- **ResNet-50:** 98MB ($25\text{M parameters} \times 4 \text{ bytes}$)

Embedding Tables:

$$\text{Size} = \text{Num items} \times \text{Embedding dim} \times 4 \text{ bytes}$$

Example (1B items, 128-dim embeddings):
$$= 1\text{B} \times 128 \times 4 = 512 \text{ GB}$$

STAFF-LEVEL EXPECTATIONS

Trade-off Analysis

Always discuss:

1. **Accuracy vs Latency**
 - **High accuracy:** BERT-large (slower)
 - **Low latency:** DistilBERT, quantization
 - **Decision:** Depends on use case (search ranking vs content moderation)
2. **Model Complexity vs Data Size**
 - **Small data:** Simple model (avoid overfitting)
 - **Large data:** Complex model (capture patterns)
3. **Real-time vs Batch**
 - **Real-time:** Higher cost, lower latency (ad serving)
 - **Batch:** Lower cost, higher latency (email recommendations)
4. **Precision vs Recall**
 - **High precision:** Minimize false positives (fraud flagging)
 - **High recall:** Catch all positives (medical diagnosis)

Failure Modes & Mitigations

Training Failures:

- **Overfitting:** Add regularization, early stopping, more data
- **Underfitting:** Larger model, more features, less regularization
- **Class imbalance:** Weighted loss, SMOTE, focal loss
- **Data leakage:** Careful train/test split, temporal validation

Serving Failures:

- **Model server down:** Load balancer + multiple replicas
- **High latency spike:** Circuit breaker, fallback to simple model
- **OOM errors:** Batch size tuning, model sharding
- **Stale predictions:** Cache invalidation strategy

Data Failures:

- **Missing features:** Default values, imputation, robust model
- **Data drift:** Monitoring + auto-retrain pipeline
- **Label noise:** Confident learning, multi-annotator consensus

Real-World Considerations

- **Cold start:** How to handle new users/items?
- **Fairness:** Avoid bias in gender, race (use fairness constraints)
- **Privacy:** Federated learning, differential privacy
- **Explainability:** SHAP, LIME for model interpretability
- **Feedback loops:** Positive feedback (recommendations) can create filter bubbles
- **Multi-objective:** Balance engagement, diversity, revenue

INTERVIEW TIPS

How to Approach ML System Design

1. **Clarify (5 min)**
 - Scale: 1M or 1B users?
 - Latency: Real-time or batch?
 - Data: Labeled? How much?
2. **High-level Design (10 min)**
 - Draw 3-stage pipeline: Data \rightarrow Model \rightarrow Serving
 - Identify key components
3. **Deep Dive (20 min)**
 - Model selection + justification
 - Feature engineering
 - Serving architecture
 - Metrics & monitoring
4. **Trade-offs (10 min)**
 - Discuss alternatives
 - Justify your choices
 - Address edge cases

Common Mistakes

- **Jumping to model too fast:** Clarify requirements first!
- **Ignoring data pipeline:** Most time is spent on data, not modeling
- **Over-engineering:** Start simple, add complexity if needed
- **Not discussing metrics:** How do you know if model is good?
- **Forgetting monitoring:** Production models degrade over time