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:

```
Image → Preprocessing → CNN → Softmax → Class
         (Resize,         (ResNet,
          Normalize)       EfficientNet)
```

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):

```
Image → CNN Backbone → Feature Pyramid
                           ↓
              Bounding Box + Class Predictions
              (Grid-based, Multi-scale)
```

Post-processing:

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

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

```
Image → Encoder (Downsample) → Bottleneck
              ↓                    ↓
           Skip Connections
              ↓                    ↓
        Decoder (Upsample) → Pixel-wise Prediction
```

4. Natural Language Processing

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

```
Text → TF-IDF / Bag-of-Words → Logistic Regression / SVM
```

Deep Learning (Large data, high accuracy):

```
Text → Tokenization → BERT/RoBERTa → [CLS] token → FC → Softmax
```

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:

```
Text → BERT → Token Embeddings → BiLSTM-CRF → BIO Tags
                                   (or just FC layer)
```

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):

```
Question + Context → BERT → Start/End Logits → Answer Span
```

Generative QA (GPT-based):

```
Question + Context → GPT → Generate Answer
```

Retrieval-Augmented (RAG):

```
Question → Embedding → Vector DB Search → Top K Docs
                                            ↓
                                       LLM (GPT) → Answer
```

D. Machine Translation Architecture:

```
Source → Encoder (Transformer) → Context
                                    ↓
                      Decoder (Transformer) → Target
```

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:

```
Transaction → Feature Engineering → Model → Fraud Score
                ↓                                ↓
           Rules Engine                    Threshold → Block/Allow
```

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

6. Ad Click Prediction (CTR)

Examples: Google Ads, Facebook Ads
Architecture:

```
User + Ad + Context → Feature Engineering → CTR Model → pCTR
                                                       ↓
                                              Auction (Bid × pCTR)
```

Feature Engineering:

- **User**: Demographics, browsing history, interests
- **Ad**: Creative type, landing page, advertiser
- **Context**: Time, device, location, query
- **Cross-features**: User×Ad interactions (critical!)

Model Architectures:
A. Logistic Regression (Baseline)

```
Features → One-hot encoding → Logistic Regression → pCTR
```

- **Pros**: Fast, interpretable, easy to debug
- **Cons**: Manual feature engineering, limited capacity

B. Factorization Machines (FM)

```
y = w0 +  wi*xi +   <vi, vj> xi*xj
         (linear)   (pairwise interactions)
```

- **Pros**: Captures feature interactions, sparse data
- **Cons**: Still limited to 2-way interactions

C. Deep & Wide

```
Wide (Memorization): Cross-features → Linear
                                         ↓
                                      Combine → Output
                                         ↓
Deep (Generalization): Embeddings → DNN
```

D. DeepFM (Facebook)

```
Features → Embeddings → FM Component (2-way)
                         ↓
                         DNN Component (high-order)
                         ↓
                       Sigmoid → pCTR
```

Training:

- **Positive samples**: Clicked ads (rare, 1-5%)
- **Negative sampling**: Down-sample non-clicks (10:1 ratio)
- **Loss**: Weighted cross-entropy
- **Calibration**: Isotonic regression (adjust predicted probabilities)

Serving:

- **Latency**: ¡ 10ms (ad auction is time-sensitive)
- **Throughput**: 100K+ QPS
- **Caching**: User embeddings, popular ad features
- **Model size**: Embedding tables can be 100GB+ (use parameter server)

Key Metrics:

- **Offline**: AUC-ROC, Log Loss, Calibration error
- **Online**: CTR, conversion rate, revenue per mille (RPM)

7. Feed Ranking (News/Social)

Examples: Facebook News Feed, Twitter Timeline, LinkedIn Feed
Objectives:

- Maximize user engagement (likes, comments, shares, time spent)
- Balance relevance, recency, diversity, connections

Architecture:

```
User → Candidate Selection (1000s of posts)
     ↓
     Ranking Model (Engagement prediction)
     ↓
     Re-ranking (Diversity, Freshness)
     ↓
     Top N posts in feed
```

Candidate Selection:

- **Sources**: Friends' posts, followed pages, ads, suggested content
- **Filtering**: Recent (last 7 days), not seen before
- **Reduce**: 100K candidates → 10K

Ranking Model: Inputs:

- **User features**: Demographics, interests, engagement history
- **Post features**: Type (photo/video/text), creator, topic, recency
- **User-Post**: Past interactions with creator, similar posts

Multi-task Learning:

```
Features → Shared Layers → Task-specific Heads:
                              - Predict Like (binary)
                              - Predict Comment (binary)
                              - Predict Share (binary)
                              - Predict Time Spent (regression)
                              ↓
                        Weighted Score =  wi * pi
```

Re-ranking Considerations:

- **Diversity**: Avoid consecutive posts from same source
- **Freshness**: Boost recent posts (decay function)
- **Explore/Exploit**: 10% random posts to discover new interests

Key Metrics:

- **Engagement**: Likes, comments, shares per user
- **Retention**: Daily active users (DAU), session time
- **Negative**: Hide/report rate (minimize)

ML INFRASTRUCTURE COMPONENTS

Feature Store

Purpose: Centralized repository for feature engineering
Architecture:

```
Raw Data → Feature Engineering → Feature Store
                                   ↓
                         Offline (Training): S3/Hive
                         Online (Serving): Redis/DynamoDB
```

Key Properties:

- **Consistency**: Same features for training & serving (avoid skew)
- **Versioning**: Track feature evolution
- **Monitoring**: Detect data drift, missing values

Example (Feast, Tecton):

```
# Define feature
@feature_view(
    entities=["user_id"],
    ttl=timedelta(days=1)
)
def user_features(df):
    return df.groupby("user_id").agg({
        "purchase_count": "sum",
        "avg_spend": "mean"
    })

# Fetch for training
features = store.get_historical_features(
    entity_df=entity_df,
    features=["user_features:purchase_count"]
)

# Fetch for serving
features = store.get_online_features(
    features=["user_features:purchase_count"],
    entity_rows=[{"user_id": "123"}]
)
```

Model Registry

Purpose: Version control for ML models
Components:

- **Storage**: S3, GCS for model artifacts (.pkl, .h5, .onnx)
- **Metadata**: Model version, training params, metrics, dependencies
- **Lineage**: Track data → features → model → deployment

Example (MLflow):

```
import mlflow

# Log model
with mlflow.start_run():
    mlflow.log_param("learning_rate", 0.01)
    mlflow.log_metric("accuracy", 0.95)
    mlflow.sklearn.log_model(model, "model")

# Load model
model = mlflow.sklearn.load_model("models:/my_model/production")
```

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
- **R^2**: 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 ¿ 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:

```
Training time = (Dataset size × Epochs × FLOPs per sample)
                / (GPU throughput × 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× V100, $12.24/hour
- **10 days**: 240 hours × $12.24 = **$2,938**

Serving Cost

QPS to Instances:

```
Instances = (Target QPS × Latency) / (Batch size × 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 × 20 batches/sec)
- **Instances needed**: 1000 / 640 = **2 GPUs**

Cost:

- **AWS g4dn.xlarge**: 1× T4 GPU, $0.526/hour
- **Monthly (2 instances)**: 2 × 730 × $0.526 = **$768/month**

Storage

Feature Store:

```
Storage = Num users × Features per user × Bytes per feature

Example (100M users, 500 features, 4 bytes each):
= 100M × 500 × 4 bytes = 200 GB
```

Model Storage:

- **BERT-base**: 440MB (110M parameters × 4 bytes)
- **GPT-3**: 700GB (175B parameters × 4 bytes)
- **ResNet-50**: 98MB (25M parameters × 4 bytes)

Embedding Tables:

```
Size = Num items × Embedding dim × 4 bytes

Example (1B items, 128-dim embeddings):
= 1B × 128 × 4 = 512 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 → Model → 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