ML System Design (Meta L6) — 2-page print cheat sheet

Goal: avoid forgetting fundamentals. Drive with structure, explicit assumptions, and quantified tradeoffs.

1) 2-3 minute kickoff (use this every time)

- Restate problem as a product: users, surfaces, business value, constraints.
- Ask clarifying questions (scope): ranking vs retrieval vs generation, regions, on-device vs server, privacy policy, traffic, update cadence.
- Define success: **primary metric** + guardrails (latency, reliability, fairness, safety/integrity, cost).
- Get numbers: QPS, p95/p99 budgets, model size, feature freshness, storage/compute, label delay.
- Propose 2 approaches (baseline + advanced). Pick one to design end-to-end.
- Draw the diagram early: offline pipeline + online request path + monitoring/feedback.

2) Requirements checklist (functional + non-functional)

- **Functional:** input/output contract (what comes in; what you return: score/rank/label/embedding), business rules, dedupe/filtering, personalization.
- **Online path:** request -> candidate gen -> feature fetch -> model -> postprocess -> response.
- **Offline path:** events/logs -> labels -> training -> validation -> registry -> deploy.
- **Non-functional:** latency (p50/p95/p99), availability/SLO, scalability (peak/region), privacy/security, interpretability, fairness, cost, operability.

3) Data & labeling (say how you get trustworthy labels)

- **Sources:** impression/click/conversion events, content metadata, social graph, feedback/reports, human labels.
- **Label definition:** positive/negative? position bias? delayed rewards? session windows? counterfactual needs?
- **Sampling:** handle imbalance; hard negatives; debias by exposure (IPS) if needed; avoid train/serve skew.
- **Leakage checks:** time-based splits; prevent using future features; consistent joins (as-of).
- **Quality:** bot/spam filtering; dedupe; annotation guidelines + inter-rater agreement; outlier handling.
- **Freshness:** streaming vs batch; late events; backfill strategy; retention/TTL.
- **Privacy:** PII minimization; access controls; encryption; auditing; data retention; differential privacy where appropriate.

4) Feature engineering & feature store (production parity)

- **Types:** user/item/context, cross features, sequence/session, aggregates (windowed counts), embeddings.
- **Offline/online parity:** same transforms; avoid training-only joins; unit tests for parity.
- **Store design:** offline warehouse + online KV + registry/metadata; TTL; versioning; ownership.
- **Fresh features:** idempotent updates; watermarking; time-travel joins; null/default handling.
- **Feature monitoring:** missing rate, distribution drift, cardinality spikes, staleness, schema changes.

5) Modeling choices (start simple, then add sophistication)

- **Baseline:** logistic regression / GBDT / shallow DNN with robust features.
- **Common shape:** multi-stage retrieval + ranking: (retrieve -> rank -> rerank).
- **Sequence:** transformer/RNN for sessions; recency decay; last-N interactions.
- **Multi-task:** shared trunk + heads; label hierarchy; loss weighting.
- **Constraints:** model size vs latency; quantization/distillation; calibration; interpretability needs.

Diagram skeleton to draw fast (say it while drawing)

Offline: logs/events -> ETL/joins/dedupe -> labeler -> train/validate -> model registry -> deploy. Online: request -> auth -> candidate gen -> feature fetch -> model inference -> postprocess (business rules, calibration, diversity) -> response. Sidecars: caches, feature store, experiment config, monitoring/logging, rollback.

6) Training pipeline (show maturity: reproducibility + scale)

- **Data pipeline:** snapshot, join, dedupe, time split (avoid leakage), train/val/test; lineage.
- **Loss & objectives:** logloss, pairwise/listwise, sampled softmax, contrastive; regularization; class weights.
- **Distributed:** data/model parallel; mixed precision; checkpointing; sharding; handle stragglers.
- **Repro:** dataset version, code hash, feature schema, seeds, configs; deterministic eval.

7) Evaluation & validation (offline + online + bias awareness)

- **Offline metrics:** AUC/PR-AUC, logloss, NDCG@k, MRR, Recall@k, calibration (ECE), RMSE.
- **Slices:** device, locale, new vs heavy users, cold-start, content types; long-tail performance.
- **Biases:** position/exposure/selection bias; survivorship; use IPS/counterfactual methods if needed.
- **Stress tests:** adversarial/spam, missing features, distribution shift, extreme load, latency spikes.
- **Fairness:** disparity metrics where applicable; harm analysis; policy constraints.

8) Serving architecture (online inference)

- **Request path:** API -> candidate gen -> feature fetch -> model -> postprocess -> response.
- **Feature fetch:** parallel reads; local cache; timeouts; fallback defaults; batching.
- **Inference:** CPU vs GPU; micro-batching; model caching; warmup; thread pools.
- **Tail latency:** hedged requests, circuit breakers, partial results, early exit, load shedding.
- **Consistency:** model/feature version pinning; sticky routing during rollout; schema compatibility.

9) Scaling & performance knobs

- **Throughput:** batching, vectorization, async I/O, reduce fanout, approximate retrieval.
- **Caching:** embeddings/features/top-K results; invalidation rules; TTL; stampede control.
- **Sharding:** consistent hashing; hot-key mitigation; regionalization; multi-tenant isolation.
- **Cost:** right-size hardware; spot for training; quantize/distill; autoscale by QPS/latency.

10) Experimentation, rollout, and monitoring (production muscle)

- **Rollout ladder:** offline -> shadow -> canary -> 1% -> 10% -> 50% -> 100% (with holdout).
- **Metrics:** primary + guardrails + long-term; define stop/rollback conditions.
- **Stat rigor:** power, multiple testing, novelty/SRM checks; log configs; backtest.
- **Monitoring:** latency/errors/timeouts; feature missing/staleness; score distributions; drift.
- **Alerts & debug:** actionable thresholds; request tracing; privacy-safe feature dumps; replay.

11) Retraining, drift, and continuous improvement

- **Cadence:** periodic + triggered retrains (drift, new content, seasonality).
- **Drift detection:** PSI/KS on features; embedding drift; label delay handling; proxies.
- **Feedback loops:** mitigate exploitation/filter bubbles; exploration (epsilon/Thompson).
- **Human-in-loop:** active learning, triage queues, label audits, abuse review.

12) Reliability & safety failure modes (name them explicitly)

- **Cold start:** priors + similarity + onboarding; fast embedding init.
- **Spam/abuse:** adversarial content/bots; rate limit; anomaly detection; robust features.
- **Data outage:** stale/missing features -> cached/fallback model; degrade gracefully.
- **Model regression:** canary/shadow/diff tests; automatic rollback; audit trails.
- **Privacy incidents:** PII in features/logs -> minimization; redaction; incident playbook.

L6 signal checklist (what they are listening for)

- **Leadership:** you drive ambiguity to clarity; propose plan A/plan B; align on success criteria early.
- **Tradeoffs:** you quantify and choose (latency vs quality vs cost vs freshness); call out risks + mitigations.
- **End-to-end ownership:** data -> model -> serving -> experiments -> monitoring -> incident response.
- **Pragmatism:** pick a baseline, ship safely, iterate; avoid over-engineering.
- **Depth on demand:** zoom in (loss/caching/drift) or zoom out (product/system) smoothly.