Principal-Level Search System Design Cheat Sheet

Complete reference for Staff/Principal IC interviews at elite tech companies

1. Three-Layer Architecture

Core Principle: *"Retrieve with index, enrich with data, rank with features"*

Layer	Latency	Update	Purpose
Index	10-50ms	Hours	Recall, filter
Hydration	5-20ms	Seconds	Enrich top-K
Feature Store	1-5ms	Real-time	ML features

Field Placement Rule

- **Index:** IDs, titles, categories, price_bucket, availability_flag
- **Hydration:** Full descriptions, images, exact inventory, reviews, PII
- **Feature Store:** CTR, CVR, trending scores, user engagement

Decision Tree:

```
If (needed for recall/filter) → Index
Else if (large or PII) → Hydration
Else if (updated frequently) → Feature Store
Else → Hydration (lazy load)
```

2. Query Understanding (5-20ms)

Pipeline Stages:

1. **Normalization:** Lowercase, Unicode, special chars
2. **Spell Correction:** Levenshtein, Soundex, neural models
3. **Tokenization:** Language-specific (jieba, MeCab)
4. **Expansion:** Synonyms (WordNet), embeddings
5. **Intent Classification:** Rules (top 20%) + BERT (tail)

Key Tradeoff: Precision vs Recall

- Aggressive expansion → High recall (tail queries)
- Conservative → High precision (head queries)

3. Retrieval Layer

Index Design Decisions

Strategy	When to Use	Scale
Single index	Similar schemas	$<$1M docs
Separate indices	Different update freq	$>$10M docs
Federated	Multi-entity	Any

Vector vs Keyword Search

Metric	ES	Milvus
Latency p99	50-200ms	1-5ms
Recall@100	70-80%	85-95%
Use Case	Keyword	Semantic

Hybrid Approach (LinkedIn, Airbnb):

```
Candidates = Union(
  BM25(top-500),
  ANN(top-500)
) → Rank top-100
```

Inventory Freshness (30s SLA)

Two Approaches:

1. **Live Ingestion:** SQS → ES real-time update
2. **Hydration Layer:** Redis cache (10s TTL) + RDBMS

Hydration Architecture @ 10K QPS:

- 500K product lookups/sec (50 items/page)
- Redis Cluster: 3-5 shards, 2-3 replicas each
- Cache hit rate: 92-97% (hot products)
- CDC: PostgreSQL → Kafka → Redis (5-10s latency)

Fallback: Hydration down → Serve stale from index

4. Ranking System (40% Interview Time)

A. Learning-to-Rank Algorithms

Type	Algorithm	Use Case
Pointwise	Linear, NN	Cold start
Pairwise	RankNet	Moderate data
Listwise	**LambdaMART**	**Production**

Industry Standard: LambdaMART (XGBoost)

```
params = {
  'objective': 'rank:ndcg',
  'eval_metric': 'ndcg@10'
}
```

B. Feature Engineering

Category	Examples	Source
Query-Doc	BM25, TF-IDF, cosine	Index
Doc Static	Category, price, rating	Index
Doc Dynamic	Inventory, CTR, CVR	Feature Store
User	Location, cohort, history	Feature Store
Context	Time, device, season	Runtime

Critical: Train with *historical* features (point-in-time), serve with *current* features

C. Multi-Objective Optimization

Problem: Optimize relevance + revenue + diversity

Three Approaches:

1. Guardrail Method ✓ (Most common)

```
Step 1: Rank by relevance (NDCG > 0.75)
Step 2: Re-rank top-K by revenue
Step 3: Apply diversity (max 2/brand)
```

Pros: Interpretable, safe — Cons: Suboptimal

2. Weighted Sum

```
Score = a*Relevance + b*Revenue + c*Diversity
```

Pros: Simple — Cons: Hard to tune

3. Multi-Task Learning (Advanced)

```
Shared layers → Task heads (rel, CTR, CVR)
Loss = w1·L_rel + w2·L_CTR + w3·L_CVR
```

Pros: Learns relationships — Cons: Complex

D. Fast-Moving Features

Problem: Training/serving skew (inventory changes)

Method	Pros	Cons
Raw value	Complex patterns	Out-of-dist
Post-filter	Simple	Ignores signal
Bucketing ✓	**Stable**	**Less granular**

Buckets: 0, 1-10, 11-100, 100+ (reduces feature space)

5. Personalization at Scale

Two-Tier Strategy

Tier 1: Coarse (All users, $<$1ms)

- User cohort (new vs returning)
- Location, device, time-of-day

Tier 2: Fine (Top 20%, 1-2ms)

- User embedding (50-100 dims) in Redis
- Async update every 5-15 min
- Pre-compute: dot(user_emb, item_embs)

Serving Flow:

```
Query → Retrieve (1000 items)
      → Redis: user_embedding (1ms)
      → Batch dot products (2ms)
      → Merge with base rank (3ms)
```

Key: Never compute user embedding at query time!

6. Multi-Entity Blending

Problem: Blend products + brands + collections

Two Strategies

A. Widget-Level Ranking ✓ (Simpler)

```
Rank: [Products, Brands, Collections]
Each shows top-3 items
```

Training: User-widget affinity

B. Item-Level Interleaving (Advanced)

```
Merge all → Single ranked list
[Product1, Brand1, Product2, ...]
```

Training: Bootstrap → Collect feedback

Cold Start: Round-robin for 2-4 weeks to collect data

7. Evaluation Metrics

Offline Metrics

NDCG@K (Most common):

$$\text{NDCG@K} = \frac{\sum_{i=1}^{K} \frac{\text{rel}_i}{\log_2(i+1)}}{\text{IDCG@K}}$$

Range: [0, 1] — Use: Graded relevance

MRR (Navigational):

$$\text{MRR} = \frac{1}{|Q|} \sum_{q=1}^{|Q|} \frac{1}{\text{rank}_q}$$

Use: First result matters (brand search)

MAP (Rare): All relevant results matter

Online Metrics

Metric	Definition	When
CTR	Clicks / Impressions	Early funnel
Conversion	Purchases / Clicks	Revenue
GMV	Total $ value	Business
Zero-result	% no results	Coverage

North Star: Conversion Rate (relevance + revenue)

8. Training Pipeline

Batch vs Online Learning

Type	Frequency	Use Case
Batch	Weekly	Search (stable)
Mini-batch	4-6 hours	Ads (shifts)
True online	Per request	Too unstable

Standard Pipeline:

```
Day 0: Logs → Label → Feature eng
     → Train XGBoost → Eval (NDCG)
     → Deploy model.jar
Day 1-7: Serve (fixed model + RT features)
Day 7: Retrain with new data
```

Key: Real-time features fetched at inference, not retrained

9. Failure Modes

Failure	Impact	Mitigation
Index down	No results	Cache queries, fallback
Hydration down	Incomplete	Serve index data
Feature Store down	Bad ranking	Static ranking
Model timeout	High latency	Circuit breaker

SLA: 99.9% → 43 min downtime/month

10. Interview Checklist

Opening Statement Template

"Let me clarify [latency/scale/cost]. I see 3 approaches: [A/B/C]. I recommend [X] because [tradeoff]. Let me walk through..."

What They Look For

✓ Structured thinking (break into components)
✓ Tradeoffs with numbers
✓ Real experience ("At X, we chose...")
✓ Business impact ("% improvement")
✓ Scale estimates (QPS, latency)

Red Flags to Avoid

✗ No clarifying questions
✗ Single approach only
✗ Vague tradeoffs ("faster")
✗ Over-engineering

Latency Breakdown Example

```
200ms total budget:
  20ms: Retrieval (index query)
  100ms: Ranking (ML model)
  50ms: Blending + hydration
  30ms: Network overhead
```

Scale Calculations

At 10K QPS with 50 items/page:

- 500K product lookups/sec
- Redis: 3-5 shards × 100K ops/sec
- Cache size: 50K hot products × 1KB = 50MB
- Expected hit rate: 95%+

Quick Reference Formulas

Cache Hit Rate Estimation

Zipf's Law:

$$\text{hit\_rate} = 1 - \left(\frac{\text{long\_tail}}{\text{total}}\right)^{\alpha}$$

where $\alpha = 0.8\text{-}1.2$ for e-commerce

Feature Serving Cost

Per-query cost:

$$\text{Cost} = N_{\text{items}} \times N_{\text{features}} \times \text{lookup\_time}$$

Example: $50 \times 100 \times 0.1\text{ms} = 500\text{ms}$ (too slow!)

Solution: Batch lookups, pre-compute, cache

Key Architectural Patterns

Two-Pass Ranking

```
Pass 1: Retrieve 10K candidates (cheap)
Pass 2: Rank top-1K (expensive ML)
Return: Top-100
```

Cascade Architecture

```
Stage 1: Keyword (BM25) → 10K
Stage 2: Lightweight NN → 1K
Stage 3: Heavy BERT → 100
```

Trade latency for quality progressively

Lambda Architecture

```
Batch Layer: Historical data → Weekly model
Speed Layer: Recent data → Hourly adjust
Serving: Combine both at inference
```

Common Interview Questions

Retrieval

- How to handle typos at scale?
- When to use vector vs keyword search?
- How to keep inventory fresh?

Ranking

- Explain pointwise vs pairwise vs listwise
- How to handle cold start products?
- Feature engineering for new user?
- Multi-objective: relevance vs revenue?

Scale

- 10K QPS with 200ms p99 → architecture?
- Index vs hydration tradeoff?
- Caching strategy for hot products?

System Design

- Design e-commerce search end-to-end
- How to A/B test ranking changes?
- Monitoring: what metrics to track?

Real-World Benchmarks

Latency Targets

- Google: 100-200ms end-to-end
- Amazon: 150-250ms
- Airbnb: 200-300ms (complex ranking)

QPS Scale

- Small: 100 QPS (startup)
- Medium: 1K-10K QPS (growth stage)
- Large: 100K+ QPS (FAANG)

Conversion Impact

- 10ms latency → 1% conversion drop
- Personalization → 5-15% lift
- Better ranking → 2-10% lift

Ranking Signal Categories

Production System: 83 Total Signals

Category	Count	Key Examples
Text Relevance	18	BM25, TF-IDF, unigram+bigram
Document Quality	9	Rating, sales, price, reviews
Query-Doc Match	25	Brand, category, NER
Engagement (Doc)	3	Total clicks, sales volume
Engagement (Q-D)	3	NavBoost (position-norm CTR)
ML Predictions	4	Click, purchase, cart prob
Business Logic	31	GPPU (profit), delivery, promo
Demotion	6	Out-of-stock, adult content
Query Expansion	6	Rewrite, relaxation
Experimental	2	A/B test variants

Signal Combination:

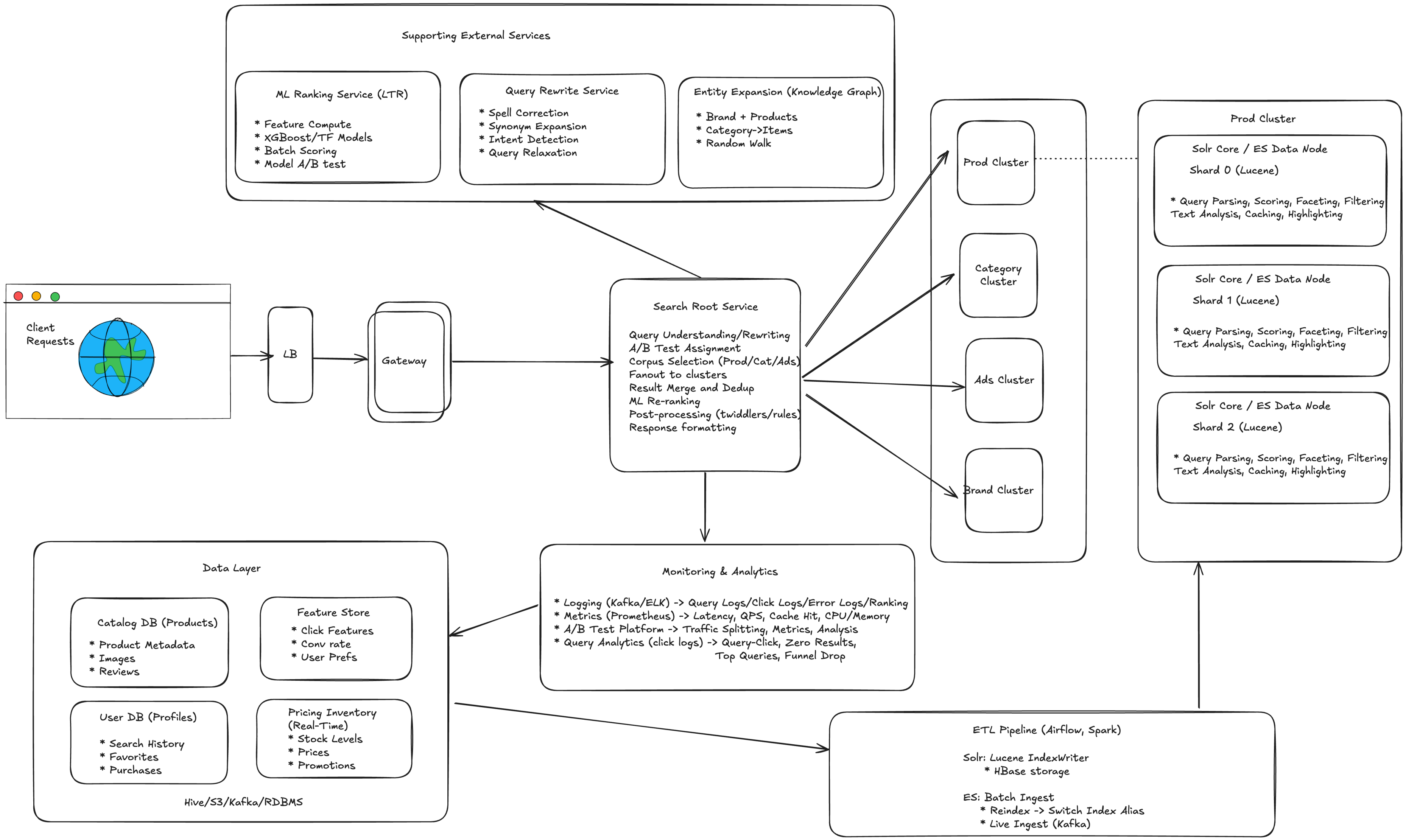
```
Score = product(
  TextRelevance(BM25),
  CategoryMatch(1.0-1.4x),
  NavBoost(CTR),
  ML_ClickProb(1.0-2.0x),
  GPPU_Profit(1.0-1.5x),
  Quality(rating, sales)
)
```

Key Insight: Multiplicative combination balances relevance, engagement, and business metrics

---

Final Wisdom: "Trust your 20 years of experience. Speak from what you've built. Make decisions like the Principal engineer you are."

Complete Search System Architecture



End-to-end architecture showing: Query flow → Search Root Service → Cluster fanout → Solr shards with supporting services (ML Ranking, Query Rewrite, Entity Expansion) and data layer (Catalog DB, User DB, Feature Store, Pricing/Inventory)