

Faire Technical Interview Preparation Guide

Complete Prep for Software Engineer Roles

Interview Focus: Algorithms, System Design (Search), ML Basics

October 21, 2025

Contents

1 Executive Summary: About Faire

1.1 Company Overview

Faire is an online wholesale marketplace that connects independent retailers with unique brands. Founded in 2017 and based in San Francisco, Faire uses machine learning to match local retailers with brands and products that uniquely fit their stores.

Key Value Propositions:

- **For Retailers:** Discover unique products, net 60 payment terms, free returns on opening orders
- **For Brands:** Access to 700,000+ retailers globally, data-driven insights, streamlined operations
- **Two-Sided Marketplace:** Balance retailer needs (relevance, conversion, personalization) with brand needs (order volume, exposure)

1.2 Core Technical Challenges

Based on Faire's product and engineering blog, their main technical challenges include:

1. **Search & Discovery:** Help retailers find relevant products among millions of SKUs
2. **Recommendations:** Inspire retailers in "browsing mode" with personalized suggestions
3. **Ranking:** Balance multiple stakeholders (retailers, brands, platform) in ranking algorithms
4. **Two-Sided Optimization:** Optimize for both retailer satisfaction and brand success
5. **Cold Start:** New retailers and new brands need quality recommendations immediately
6. **Real-Time Features:** Low-latency feature serving for ranking models

2 Faire Tech Stack & Architecture

Understanding Faire's actual tech stack will help you tailor your answers and ask informed questions.

2.1 Backend & Languages

Primary Language: Kotlin

- Entire backend monolith written in Kotlin (migrated from Java)
- Handles 1,000+ requests per second
- Android app also in Kotlin
- Allows for concise, type-safe code with null safety

Why Kotlin?

- Interoperable with existing Java libraries (Hibernate for ORM, Guice for DI)
- Modern language features (coroutines for async, data classes)
- Strong community and tooling support

2.2 Core Infrastructure

Data Storage:

- **MySQL:** Primary relational database
- **Redis:** Caching layer and feature store for ML models
- **NoSQL:** For specific use cases

Search:

- **ElasticSearch:** Product search with custom scoring functions
- **Lucene:** Underlying search library

Message Queues & Streaming:

- **Amazon SQS:** Asynchronous job processing
- **Kinesis Firehose:** Event streaming to data warehouse

Compute & Orchestration:

- **Kubernetes:** Container orchestration
- **Nginx:** Load balancing and routing
- **AWS ELB:** External load balancing

2.3 Machine Learning Stack

Models:

- **XGBoost:** Primary model for ranking (search and recommendations)
- **Embedding Models:** Learn representations from user behavior (clicks, add-to-cart, orders)

Feature Engineering:

- **Redis Feature Store:** Hundreds of features retrieved in real-time per product
- Features include: retailer-product similarity, engagement signals, brand metadata, product attributes

Data Infrastructure:

- **DBT:** Data transformation and modeling
- **Tableau:** Business intelligence and analytics

2.4 Two-Stage Ranking Architecture

Faire uses a classic retrieval + ranking pipeline:

Stage 1: Retrieval (Candidate Generation)

- ElasticSearch retrieves top 1,000 candidate products
- Uses custom scoring function balancing precision/recall
- Fast: optimized for low latency

Stage 2: Ranking (Re-ranking)

- Retrieve hundreds of features per product from Redis
- Score 1,000 candidates with XGBoost model
- Rank products by predicted engagement/conversion

3 OFFICIAL Interview Structure (From Recruiter)

3.1 Virtual Onsite - 5 Parts Over 2 Days

This is the OFFICIAL interview structure provided by Faire's recruiting team for the Back-end Engineer - Search role.

3.1.1 Day 1: Technical Deep Dive (3 Hours Total)

Part 1: Programming Exercise [1 HOUR]

- **Format:** Paired programming on CoderPad
- **Goal:** Write a working solution to a coding problem
- **Evaluation Criteria:**
 1. *Approach and Planning:* Ask clarifying questions before coding
 2. *Code Structure & Cleanliness:* Write clean, readable code
 3. *Coachability:* Modify solution based on interviewer feedback
 4. *Debugging:* Account for edge cases and fix bugs
 5. *Testing:* Write test cases with good coverage

Part 2: Systems Design - Search [1 HOUR]

- **Focus:** Major components of an e-commerce search system
- **Topics:**
 - How data gets into the system, how it's processed, and how it's queried
 - Life of a product record from creation to search results
 - Life of a search query from user input to displayed results
 - Design tradeoffs and performance optimizations

- Scaling considerations
- High-level search algorithm concepts

- **Important:** This is **NOT** about super-detailed ML knowledge
- **Focus:** Architecture, data flows, and system components

Part 3: Algorithm Interview [1 HOUR]

- **CRITICAL:** This is **NOT** a LeetCode-style coding interview!
- **Format:** 1-2 questions with sub-questions
- **NO CODE REQUIRED** (emphasized in official document)
- **Goal:** Evaluate practical design knowledge for ML-powered systems
- **What You'll Do:**
 - Identify performance bottlenecks in EXISTING systems
 - Propose incremental improvements
 - Suggest architectural changes
 - Think about ML system improvements (NOT design from scratch)
- **Example:** "Here's our current search ranking system. It has latency issues. What would you investigate and how would you improve it?"

3.1.2 Day 2: Values & Leadership (1.5 Hours Total)

Part 4: Values Interview [30 MINUTES]

- **Format:** Behavioral questions
- **Focus:** Alignment with Faire's 5 core values
- **Example Topics:**
 - Tell me about a complex problem you solved
 - Describe a positive/challenging working relationship
 - When did you think outside the box?
 - Why Faire? Why our mission?
 - Examples of impact, influence, ownership
 - How do you handle feedback and growth?

Part 5: Levelling Deep Dive [1 HOUR]

- **Format:** Technical leadership discussion
- **Focus:** Career history, technical depth, seniority calibration
- **Topics:**
 - Most complex system you've designed

- Technical decisions and tradeoffs
- How you’ve led projects or mentored others
- Scaling challenges you’ve faced
- Your technical growth trajectory

3.2 Faire’s Core Values (OFFICIAL)

Understanding these values is CRITICAL for the Values Interview (Part 4).

3.2.1 1. Be an Owner

- **Resourceful:** Find creative solutions with available resources
- **Raise the Bar:** Set and maintain high standards
- **Share Our Legos:** Build reusable systems, share knowledge

3.2.2 2. Be Kind

- **Collaboration:** Work effectively with others
- **Coachability:** Accept feedback and grow
- **Inclusive:** Create welcoming environment for all

3.2.3 3. Embrace the Adventure

- **Agility:** Adapt quickly to change
- **Bias Towards Action:** Move fast, iterate, learn

3.2.4 4. Seek the Truth

- **Decision-Making:** Make data-driven decisions
- **Analytical Thinking:** Dig deep to understand root causes
- **Communication:** Communicate clearly and transparently

3.2.5 5. Serve Our Community

- **Customer Centricity:** Put retailers and brands first
- **Prioritization:** Focus on highest-impact work
- **Motivation:** Inspired by Faire’s mission

3.3 Company Mission & Scale

Mission: Empower entrepreneurs everywhere to chase their dreams.

Scale:

- 350,000+ independent retailers on the platform
- Independent retailers collectively larger than Amazon + Walmart combined
- 40% of wholesale buying moved online during COVID

Growth Flywheel:

1. Brand onboards → Refers other retailers
2. Retailer onboards → Refers other brands
3. Network effects create virtuous cycle

Data Flywheel:

1. Better recommendations → Higher sell-through for retailers
2. Higher sell-through → More orders placed
3. More orders → More data for ML models
4. Better data → Better recommendations (cycle continues)

4 Programming Exercise Prep (Interview Part 1)

Context: This section covers the Programming Exercise (Part 1 of the interview), which is a 1-hour paired programming session on CoderPad where you write actual code.

DO NOT CONFUSE with: The "Algorithm Interview" (Part 3), which does NOT require code!

4.1 Expected Difficulty

Format: Medium LeetCode-level problem, 45-60 minutes

Focus Areas: Arrays, hashmaps, two pointers, trees, graphs, dynamic programming

Evaluation: Approach, code cleanliness, coachability, debugging, testing

4.2 Common LeetCode Patterns for E-Commerce

Based on e-commerce/marketplace context, expect questions related to:

4.2.1 Pattern 1: Arrays & Hashing

Sample Problem: Product Inventory Management

Problem: Given an array of product IDs representing items in a retailer's cart, find the most frequently purchased product. If there's a tie, return the product with the smallest ID.

```

1 def most_frequent_product(cart):
2     """
3     Time: O(n), Space: O(n)
4     """
5     from collections import Counter
6
7     if not cart:
8         return -1
9
10    # Count frequencies
11    freq = Counter(cart)
12
13    # Find max frequency
14    max_freq = max(freq.values())
15
16    # Among products with max freq, return smallest ID
17    candidates = [pid for pid, count in freq.items()
18                  if count == max_freq]
19
20    return min(candidates)
21
22 # Example
23 cart = [101, 203, 101, 305, 203, 101, 203]
24 print(most_frequent_product(cart)) # 101 (freq=3, but smaller ID)

```

Follow-up: What if we need to return top K products? Use a heap.

4.2.2 Pattern 2: Two Pointers

Sample Problem: Merge Brand Catalogs

Problem: Two brands have sorted arrays of product prices. Merge them into one sorted array.

```

1 def merge_catalogs(prices1, prices2):
2     """
3     Two pointers to merge sorted arrays.
4     Time: O(m + n), Space: O(m + n)
5     """
6     result = []
7     i, j = 0, 0
8
9     while i < len(prices1) and j < len(prices2):
10        if prices1[i] <= prices2[j]:
11            result.append(prices1[i])
12            i += 1
13        else:
14            result.append(prices2[j])
15            j += 1
16
17    # Append remaining
18    result.extend(prices1[i:])
19    result.extend(prices2[j:])
20
21    return result

```


4.2.3 Pattern 3: Binary Search

Sample Problem: Find Minimum Price in Rotated Catalog

Problem: A sorted price list was rotated. Find the minimum price.

```

1 def find_min_price(prices):
2     """
3     Binary search on rotated sorted array.
4     Time: O(log n), Space: O(1)
5     """
6     left, right = 0, len(prices) - 1
7
8     while left < right:
9         mid = (left + right) // 2
10
11         # Right half is unsorted, min is in right half
12         if prices[mid] > prices[right]:
13             left = mid + 1
14         else:
15             # Left half is unsorted or mid is min
16             right = mid
17
18     return prices[left]
19
20 # Example
21 prices = [4.99, 5.99, 6.99, 1.99, 2.99, 3.99]
22 print(find_min_price(prices)) # 1.99

```

4.2.4 Pattern 4: Trees (Product Taxonomy)

Sample Problem: Category Tree Traversal

Problem: Faire has product categories organized in a tree. Count total products in a category and all subcategories.

```

1 class CategoryNode:
2     def __init__(self, name, product_count):
3         self.name = name
4         self.product_count = product_count
5         self.children = []
6
7 def count_products(root):
8     """
9     DFS to count products in category tree.
10    Time: O(n), Space: O(h) for recursion stack
11    """
12    if not root:
13        return 0
14
15    total = root.product_count
16
17    for child in root.children:
18        total += count_products(child)
19
20    return total

```

```

21
22 # Example tree: Home Decor -> (Furniture, Lighting)
23 root = CategoryNode("Home Decor", 100)
24 furniture = CategoryNode("Furniture", 50)
25 lighting = CategoryNode("Lighting", 30)
26 root.children = [furniture, lighting]
27
28 print(count_products(root)) # 180

```

4.2.5 Pattern 5: Graphs (Retailer-Brand Network)

Sample Problem: Find Connected Retailers

Problem: Given a graph where nodes are retailers and edges connect retailers who order from the same brands, find the number of connected components (retailer communities).

```

1 def count_communities(n, edges):
2     """
3     Union-Find to count connected components.
4     Time: O(n + e * alpha(n)), Space: O(n)
5     """
6     parent = list(range(n))
7
8     def find(x):
9         if parent[x] != x:
10             parent[x] = find(parent[x]) # Path compression
11         return parent[x]
12
13     def union(x, y):
14         root_x, root_y = find(x), find(y)
15         if root_x != root_y:
16             parent[root_x] = root_y
17
18     # Build connections
19     for u, v in edges:
20         union(u, v)
21
22     # Count unique roots
23     return len(set(find(i) for i in range(n)))
24
25 # Example: 5 retailers, edges: [(0,1), (1,2), (3,4)]
26 print(count_communities(5, [(0,1), (1,2), (3,4)])) # 2 communities

```

4.2.6 Pattern 6: Dynamic Programming

Sample Problem: Maximum Order Value (Knapsack Variant)

Problem: A retailer has a budget. Given products with prices and profit margins, maximize total profit without exceeding budget.

```

1 def max_profit(budget, prices, profits):
2     """
3     0/1 Knapsack problem.
4     Time: O(n * budget), Space: O(budget)
5     """

```

```

6     n = len(prices)
7     dp = [0] * (budget + 1)
8
9     for i in range(n):
10        price = prices[i]
11        profit = profits[i]
12
13        # Traverse backwards to avoid using same item twice
14        for b in range(budget, price - 1, -1):
15            dp[b] = max(dp[b], dp[b - price] + profit)
16
17    return dp[budget]
18
19 # Example
20 budget = 100
21 prices = [30, 40, 50, 60]
22 profits = [10, 20, 25, 30]
23 print(max_profit(budget, prices, profits)) # 50 (items 1 and 2)

```

4.3 E-commerce Specific Coding Questions

These are deduced based on Faire's product:

4.3.1 Question 1: Product Search Ranking

Problem: Given a search query and a list of products (each with name, description, price), return the top K most relevant products. Relevance is based on keyword matches (more matches = higher relevance). If tied, prioritize lower price.

```

1 def search_products(query, products, k):
2     """
3     Time: O(n * m + n log n), where n = products, m = avg words
4     Space: O(n)
5     """
6     query_words = set(query.lower().split())
7
8     scored_products = []
9     for product in products:
10        text = (product['name'] + ' ' + product['description']).lower()
11        words = set(text.split())
12
13        # Count keyword matches
14        matches = len(query_words & words)
15
16        # Score: (matches, -price) for sorting
17        scored_products.append((matches, -product['price'], product))
18
19    # Sort by matches (desc), then price (asc)
20    scored_products.sort(key=lambda x: (x[0], -x[1]), reverse=True)
21
22    return [p[2] for p in scored_products[:k]]
23
24 # Example

```

```

25 products = [
26     {"name": "Vase", "description": "ceramic blue vase", "price": 25},
27     {"name": "Blue Pot", "description": "ceramic pot", "price": 20},
28     {"name": "Ceramic Bowl", "description": "handmade ceramic", "price":
        30}
29 ]
30 query = "blue ceramic"
31 print(search_products(query, products, 2))
32 # Returns: Blue Pot (2 matches, $20), Vase (2 matches, $25)

```

4.3.2 Question 2: Order Fulfillment

Problem: Faire processes orders from multiple retailers. Given an array of order timestamps (Unix time), find the maximum number of orders processed in any 1-hour window.

```

1 def max_orders_in_hour(timestamps):
2     """
3     Sliding window approach.
4     Time: O(n log n + n), Space: O(1)
5     """
6     if not timestamps:
7         return 0
8
9     timestamps.sort()
10    max_count = 0
11
12    for i in range(len(timestamps)):
13        window_end = timestamps[i] + 3600 # 1 hour in seconds
14
15        # Binary search for end of window
16        left, right = i, len(timestamps) - 1
17        while left < right:
18            mid = (left + right + 1) // 2
19            if timestamps[mid] <= window_end:
20                left = mid
21            else:
22                right = mid - 1
23
24        count = left - i + 1
25        max_count = max(max_count, count)
26
27    return max_count
28
29 # Example
30 timestamps = [1000, 2000, 3000, 4500, 5000, 6000]
31 print(max_orders_in_hour(timestamps)) # 4 orders

```

4.4 SQL Questions

Faire emphasizes SQL and data modeling. Expect questions like:

4.4.1 Question 1: Top Brands by Revenue

Problem: Given tables orders(order_id, brand_id, amount, date) and brands(brand_id, name), find the top 5 brands by total revenue in the last 30 days.

```
1 SELECT
2     b.name,
3     SUM(o.amount) AS total_revenue
4 FROM orders o
5 JOIN brands b ON o.brand_id = b.brand_id
6 WHERE o.date >= CURRENT_DATE - INTERVAL 30 DAY
7 GROUP BY b.brand_id, b.name
8 ORDER BY total_revenue DESC
9 LIMIT 5;
```

4.4.2 Question 2: Retailer Churn Rate

Problem: Calculate monthly churn rate. Tables: retailers(retailer_id, signup_date), orders(order_id, retailer_id, order_date).

Definition: Churn = retailers who placed orders in month N but not in month N+1.

```
1 WITH monthly_active AS (
2     SELECT
3         retailer_id,
4         DATE_TRUNC('month', order_date) AS month
5     FROM orders
6     GROUP BY retailer_id, month
7 ),
8 churned AS (
9     SELECT
10        m1.month,
11        m1.retailer_id
12     FROM monthly_active m1
13     LEFT JOIN monthly_active m2
14         ON m1.retailer_id = m2.retailer_id
15         AND m2.month = m1.month + INTERVAL '1 month'
16     WHERE m2.retailer_id IS NULL
17 )
18 SELECT
19     month,
20     COUNT(DISTINCT retailer_id) AS churned_retailers
21 FROM churned
22 GROUP BY month
23 ORDER BY month;
```

5 System Design - Search Prep (Interview Part 2)

Context: This section covers the Systems Design - Search interview (Part 2 of the interview), which is a 1-hour system design discussion focused on e-commerce search architecture.

Key Guidance from Recruiter:

- Focus on major components of e-commerce search system

- NOT about super-detailed ML knowledge
- Emphasize: data flows, architecture, scaling, tradeoffs

5.1 Why Search is Critical for Faire

Search is the main system design focus. This aligns with Faire's core value proposition:

- **700K+ retailers** need to discover products from **100K+ brands**
- **Millions of SKUs** across diverse categories (home decor, apparel, food, etc.)
- **Two shopping modes:**
 - *Focused search:* "I need ceramic mugs for my coffee shop"
 - *Inspirational browsing:* "Show me trending home decor"

5.2 System Design Problem 1: Design Faire's Product Search

5.2.1 Problem Statement

Design a search system for Faire's wholesale marketplace. Retailers should be able to search for products across millions of SKUs from 100K+ brands. The system should support:

- Keyword search with autocomplete
- Filters (price, category, brand, location, minimum order, etc.)
- Relevance ranking
- Low latency (< 200ms p99)
- Personalization based on retailer's store type and past orders
- Handle 10,000 QPS at peak

5.2.2 Clarifying Questions

Ask the interviewer:

1. **Scale:** How many products? How many searches/day?
2. **Latency:** What's acceptable p50, p95, p99?
3. **Ranking:** Should we optimize for clicks? Add-to-cart? Orders?
4. **Personalization:** How much personalization vs. global relevance?
5. **Consistency:** Strong consistency needed or eventual consistency OK?
6. **Internationalization:** Multiple languages? Multiple currencies?

5.2.3 Requirements & Scale Estimation

Functional Requirements:

- Search products by keywords
- Filter by attributes (price, category, brand, etc.)
- Autocomplete search queries
- Rank results by relevance
- Personalize results per retailer

Non-Functional Requirements:

- Low latency: $\leq 200\text{ms p99}$
- High availability: 99.9% uptime
- Scalability: 10K QPS peak
- Consistency: Eventual consistency OK for search index

Scale Estimation:

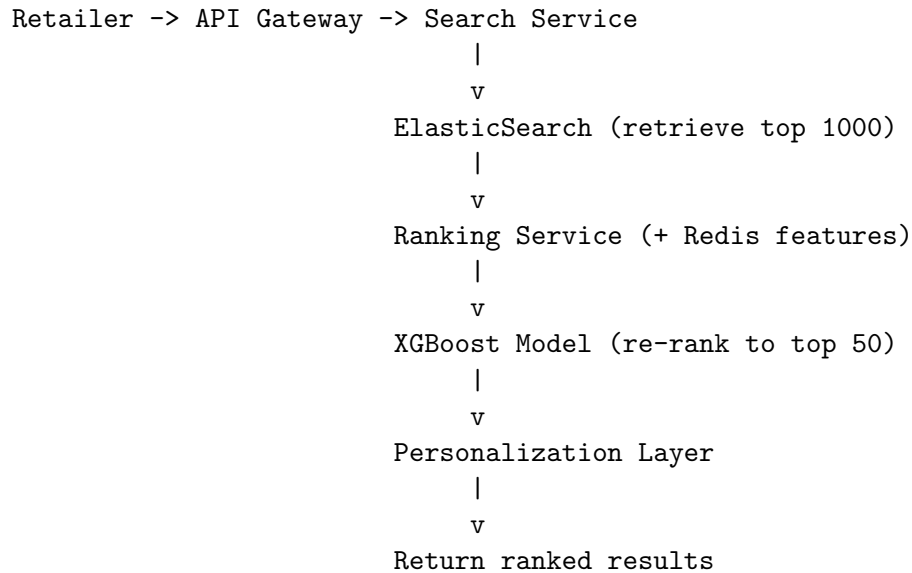
- **Products:** 10M SKUs
- **Brands:** 100K brands
- **Retailers:** 700K active retailers
- **Searches:** 1M searches/day = 12 QPS average, 120 QPS peak (10x)
- **Index size:** Assume 10KB per product \rightarrow 100GB index

5.2.4 High-Level Architecture

Components:

1. **API Gateway:** Load balancing, rate limiting, authentication
2. **Search Service:** Handles search requests, queries Elasticsearch
3. **ElasticSearch Cluster:** Inverted index for fast keyword search
4. **Ranking Service:** ML-based ranking (XGBoost)
5. **Feature Store (Redis):** Real-time features for ranking
6. **Autocomplete Service:** Trie-based prefix matching
7. **Personalization Service:** Retriever embeddings, user preferences
8. **Data Ingestion Pipeline:** Update search index from product catalog

Data Flow:



5.2.5 Detailed Component Design

1. ElasticSearch Cluster

Index Schema:

```

1 {
2   "product_id": "12345",
3   "name": "Ceramic Mug Set",
4   "description": "Handmade ceramic mugs, set of 4",
5   "category": "Home & Kitchen > Drinkware",
6   "brand_id": "brand_789",
7   "brand_name": "ArtisanCraft Co",
8   "price": 45.99,
9   "tags": ["ceramic", "handmade", "eco-friendly"],
10  "min_order_quantity": 6,
11  "inventory_count": 150,
12  "brand_location": "Portland, OR",
13  "created_at": "2024-01-15T10:00:00Z",
14  "popularity_score": 8.5
15 }
  
```

Scoring Function:

ElasticSearch uses BM25 for text relevance, but we can customize:

$$\text{score} = \alpha \cdot \text{BM25}(q, d) + \beta \cdot \text{popularity}(d) + \gamma \cdot \text{recency}(d) \quad (1)$$

Where:

- $\text{BM25}(q, d)$: keyword match score
- $\text{popularity}(d)$: click-through rate, order rate
- $\text{recency}(d)$: boost newer products
- α, β, γ : learned weights

Sharding Strategy:

- Shard by product ID hash (uniform distribution)
- 10 shards for 10M products = 1M products/shard
- 3 replicas per shard for high availability

2. Ranking Service with XGBoost

After ElasticSearch retrieves 1,000 candidates, re-rank with ML model.

Features (hundreds per product):

- **Text relevance:** BM25 score, title match, description match
- **Retailer-product similarity:** Embedding cosine similarity
- **Engagement signals:** CTR, add-to-cart rate, purchase rate
- **Product attributes:** Price, category, brand popularity, inventory
- **Personalization:** Retailer's past category preferences, price sensitivity

Model Training:

- **Objective:** Predict probability of order given search query
- **Training data:** Search queries + clicked/ordered products (positive), random products (negative)
- **Model:** XGBoost (gradient boosted trees)
- **Offline evaluation:** NDCG@10, Precision@10, Recall@50
- **Online evaluation:** A/B testing (CTR, conversion rate, revenue)

Feature Retrieval:

- Store features in Redis (key: product_id, value: feature vector)
- Batch fetch 1,000 products' features in parallel
- Latency: < 10ms for Redis lookup

3. Autocomplete Service**Data Structure: Trie (Prefix Tree)**

```

1 class TrieNode:
2     def __init__(self):
3         self.children = {}
4         self.is_end_of_word = False
5         self.suggestions = [] # Top 5 completions
6
7 class Autocomplete:
8     def __init__(self):
9         self.root = TrieNode()
10
11     def insert(self, query, frequency):

```

```

12     node = self.root
13     for char in query.lower():
14         if char not in node.children:
15             node.children[char] = TrieNode()
16             node = node.children[char]
17
18     node.is_end_of_word = True
19     # Update top suggestions at this node
20     self.update_suggestions(node, query, frequency)
21
22     def search(self, prefix):
23         node = self.root
24         for char in prefix.lower():
25             if char not in node.children:
26                 return []
27             node = node.children[char]
28
29     return node.suggestions[:5]

```

Populating Autocomplete:

- Mine frequent queries from search logs
- Use product names, brand names, category names
- Update daily from offline job
- Store in Redis for low-latency lookup

4. Personalization Layer

Use **embedding-based retrieval** to personalize results.

Embeddings:

- **Retailer embeddings:** Learned from order history, clicks, store category
- **Product embeddings:** Learned from co-purchase patterns, attributes
- **Similarity:** Cosine similarity in embedding space

Model: Two-tower neural network (similar to YouTube's DNN)

- **Retailer tower:** Encode retailer features → retailer embedding
- **Product tower:** Encode product features → product embedding
- **Training:** Maximize dot product for positive pairs (retailer ordered product)

Personalization at Serving:

```

1 def personalize(ranked_products, retailer_embedding):
2     """
3     Re-weight top 50 products by retailer-product similarity.
4     """
5     for product in ranked_products:
6         similarity = cosine_similarity(
7             retailer_embedding,

```

```

8         product.embedding
9     )
10    product.score = 0.7 * product.score + 0.3 * similarity
11
12    return sorted(ranked_products, key=lambda p: p.score, reverse=True)

```

5.2.6 Database Schema

Products Table:

```

1 CREATE TABLE products (
2     product_id BIGINT PRIMARY KEY,
3     brand_id BIGINT,
4     name VARCHAR(255),
5     description TEXT,
6     category_id INT,
7     price DECIMAL(10, 2),
8     min_order_qty INT,
9     inventory_count INT,
10    created_at TIMESTAMP,
11    updated_at TIMESTAMP,
12    INDEX (brand_id),
13    INDEX (category_id)
14 );

```

Search Logs Table:

```

1 CREATE TABLE search_logs (
2     log_id BIGINT PRIMARY KEY AUTO_INCREMENT,
3     retailer_id BIGINT,
4     query VARCHAR(500),
5     clicked_product_id BIGINT,
6     position INT,
7     timestamp TIMESTAMP,
8     INDEX (retailer_id, timestamp),
9     INDEX (query)
10 );

```

5.2.7 API Design

Search API:

```

1 GET /api/v1/search?q=ceramic+mugs&filters={"category":"drinkware","
   price_max":50}&page=1&size=20
2
3 Response:
4 {
5     "query": "ceramic mugs",
6     "total_results": 1523,
7     "page": 1,
8     "size": 20,
9     "products": [
10        {
11            "product_id": 12345,

```

```

12     "name": "Ceramic Mug Set",
13     "brand": "ArtisanCraft Co",
14     "price": 45.99,
15     "image_url": "https://...",
16     "min_order": 6
17 },
18 ...
19 ],
20 "facets": {
21     "categories": {"drinkware": 800, "home_decor": 300},
22     "price_ranges": {"0-25": 500, "25-50": 700, "50-100": 323}
23 }
24 }

```

Autocomplete API:

```

1 GET /api/v1/autocomplete?q=cera
2
3 Response:
4 {
5     "suggestions": [
6         "ceramic mugs",
7         "ceramic vases",
8         "ceramic plates",
9         "ceramic bowls",
10        "ceramic decor"
11    ]
12 }

```

5.2.8 Optimizations & Tradeoffs

Caching Strategy:

- **CDN:** Cache popular search results (top 1000 queries)
- **Redis:** Cache recent searches per retailer
- **TTL:** 5 minutes for search results (balance freshness vs. performance)

Cold Start Problem:

- **New retailers:** Use store category to initialize preferences
- **New products:** Boost in ranking for visibility, collect data quickly

Handling Spikes:

- Auto-scaling Elasticsearch cluster
- Rate limiting per retailer (prevent abuse)
- Degrade gracefully: skip personalization if Redis is slow

Consistency Tradeoffs:

- Product updates: eventual consistency OK (few minutes delay)
- Critical updates (out-of-stock): real-time update via pub/sub

5.2.9 Metrics & Monitoring

System Metrics:

- Latency: p50, p95, p99
- QPS, error rate, cache hit rate
- Elasticsearch shard health

Business Metrics:

- CTR (click-through rate) per search
- Conversion rate (search \rightarrow order)
- Revenue per search
- Zero-result rate (searches with no results)

ML Metrics:

- NDCG@10 (offline)
- Precision@5, Recall@20 (offline)
- Online A/B testing (CTR, conversion)

5.3 System Design Problem 2: Design Faire's Recommendation System

5.3.1 Problem Statement

Design a recommendation system for Faire's homepage and product pages. When a retailer logs in, show personalized product recommendations. Also show "You may also like" on product detail pages.

Requirements:

- Personalized homepage recommendations (50 products)
- "Similar products" on product pages (10 products)
- Handle cold start (new retailers, new products)
- Low latency (\leq 100ms)
- Optimize for order conversion, not just clicks

5.3.2 Approach: Two-Stage Retrieval + Ranking

This mirrors Faire's actual architecture (based on their engineering blog).

Stage 1: Candidate Generation

- Retrieve 1,000 candidate products from multiple sources:
 1. **Collaborative filtering:** Retailers similar to you ordered these
 2. **Content-based:** Products similar to what you've ordered

3. **Trending:** Popular products in your category
4. **New arrivals:** Recently added products

Stage 2: Ranking

- Score 1,000 candidates with XGBoost
- Features: retailer-product similarity, engagement, diversity
- Return top 50

5.3.3 Collaborative Filtering with Embeddings

Faire uses **embedding-based CF** learned from implicit feedback.

Training Data:

- **Positive:** (retailer, product) pairs from orders, add-to-cart, favorites
- **Negative:** Random products (or hard negatives: clicked but not ordered)

Model: Two-Tower DNN

```

1 import torch
2 import torch.nn as nn
3
4 class TwoTowerModel(nn.Module):
5     def __init__(self, num_retailers, num_products, embedding_dim=128):
6         super().__init__()
7
8         # Retailer tower
9         self.retailer_embedding = nn.Embedding(num_retailers,
10                                                embedding_dim)
11         self.retailer_fc = nn.Sequential(
12             nn.Linear(embedding_dim, 256),
13             nn.ReLU(),
14             nn.Linear(256, 128)
15         )
16
17         # Product tower
18         self.product_embedding = nn.Embedding(num_products, embedding_dim)
19         self.product_fc = nn.Sequential(
20             nn.Linear(embedding_dim, 256),
21             nn.ReLU(),
22             nn.Linear(256, 128)
23         )
24
25     def forward(self, retailer_ids, product_ids):
26         # Encode retailer
27         retailer_emb = self.retailer_embedding(retailer_ids)
28         retailer_vec = self.retailer_fc(retailer_emb)
29
30         # Encode product
31         product_emb = self.product_embedding(product_ids)
32         product_vec = self.product_fc(product_emb)

```

```

33     # Dot product (similarity)
34     scores = (retailer_vec * product_vec).sum(dim=1)
35
36     return torch.sigmoid(scores)

```

Loss Function:

$$\mathcal{L} = - \sum_{(r,p) \in \text{positive}} \log \sigma(e_r \cdot e_p) - \sum_{(r,p') \in \text{negative}} \log(1 - \sigma(e_r \cdot e_{p'})) \quad (2)$$

Serving:

- Store product embeddings in FAISS (Approximate Nearest Neighbors)
- Given retailer embedding, find top K nearest products
- Latency: ~ 10ms for ANN search

5.3.4 Product-to-Product Recommendations

For "You may also like" on product pages:

Approach 1: Co-Purchase Matrix

- Build matrix: product A \times product B = count of co-purchases
- Use collaborative filtering (item-item similarity)
- Cosine similarity in co-purchase space

Approach 2: Embedding Similarity

- Use product embeddings from two-tower model
- For product P, find K nearest neighbors in embedding space
- Pre-compute and cache in Redis

5.3.5 Diversity & Exploration

Avoid showing only similar products (filter bubble).

Diversity Technique: MMR (Maximal Marginal Relevance)

$$\text{MMR}(p) = \lambda \cdot \text{relevance}(p) - (1 - \lambda) \cdot \max_{p' \in S} \text{similarity}(p, p') \quad (3)$$

Where:

- S : already selected products
- λ : tradeoff between relevance and diversity (e.g., 0.7)

Exploration: Thompson Sampling

- Occasionally show new/underexplored products
- Use multi-armed bandit to balance exploration vs. exploitation

5.3.6 Two-Sided Optimization

Faire must balance:

- **Retailer utility:** Relevant, high-quality products
- **Brand utility:** Sufficient exposure, especially for new brands

Approach:

- Add **brand diversity** constraint: limit products per brand
- Boost **new brands** in ranking (time decay)
- Monitor brand-level metrics: impression share, order share

5.4 Follow-Up Questions

Be prepared for these follow-ups:

1. How would you handle multi-language search?

- Use language-specific analyzers in Elasticsearch
- Cross-lingual embeddings for semantic search
- Translate queries to English, then search

2. How to prevent spam/fake products in search results?

- ML classifier to detect spam (based on title, description patterns)
- Human review queue for flagged products
- Downrank products with low engagement or high return rate

3. How to A/B test ranking algorithms?

- Split traffic by retailer ID hash
- Track metrics: CTR, conversion, revenue per search
- Use statistical significance testing (t-test, chi-square)
- Gradual rollout: 5% → 10% → 50% → 100%

4. What if Elasticsearch goes down?

- Fallback to cached popular queries
- Degrade to simpler search (MySQL full-text search)
- Show trending products as fallback
- Alert on-call engineer, auto-restart nodes

5. How to scale to 100K QPS?

- Horizontal scaling: more Elasticsearch nodes
- Read replicas for feature store (Redis)
- CDN for static content and popular searches
- Database sharding by geography or retailer segment

6 Algorithm Interview Prep (Interview Part 3)

CRITICAL: This is **NOT** a LeetCode coding interview! Read this section carefully.

6.1 What Makes This Different

From Official Recruiter Document:

- **NO CODE REQUIRED** - This is a discussion, not a coding exercise
- **Format:** 1-2 questions with sub-questions
- **Goal:** Evaluate practical design knowledge for ML-powered systems
- **Focus:** Improve EXISTING systems, not design from scratch
- **Skills Tested:**
 - Identify performance bottlenecks
 - Propose incremental improvements
 - Suggest architectural changes
 - Think critically about ML system tradeoffs

6.2 Example Question Format

Scenario: "Here's our current search ranking system. It retrieves 1000 candidates from ElasticSearch, then re-ranks with XGBoost using 200 features from Redis. The system has the following issues:

- P95 latency is 800ms (target: 300ms)
- Cold-start retailers get poor recommendations
- Feature drift - model trained 3 months ago, performance degrading

Questions:

1. What would you investigate first to reduce latency?
2. How would you approach the cold-start problem?
3. What's your strategy for detecting and handling feature drift?
4. What incremental changes would you make vs. major architectural changes?

6.3 How to Approach This Interview

6.3.1 Step 1: Understand the Existing System

Ask clarifying questions:

- What are the main components? (retrieval, ranking, serving)
- What's the current data flow?

- What metrics are being optimized? (latency, relevance, etc.)
- What's the scale? (QPS, data volume, model size)
- What infrastructure is in place? (caching, load balancing, etc.)

6.3.2 Step 2: Identify Bottlenecks

Think systematically:

- **Data bottlenecks:** Slow feature retrieval, database queries
- **Compute bottlenecks:** Model inference time, CPU/GPU utilization
- **Network bottlenecks:** Inter-service latency, data transfer
- **ML-specific:** Model size, feature dimensionality, batch size

6.3.3 Step 3: Propose Solutions (Incremental + Architectural)

Incremental Improvements (quick wins):

- Add caching layers (Redis for features, CDN for results)
- Optimize database queries (indexes, denormalization)
- Feature selection (reduce 200 features to 50 most important)
- Model compression (quantization, pruning)
- Batch processing where possible

Architectural Changes (longer-term):

- Move from synchronous to asynchronous feature retrieval
- Implement multi-stage ranking (fast initial filter, expensive final ranking)
- Build feature store for offline/online consistency
- Add monitoring and alerting for drift detection
- Implement online learning or periodic retraining pipeline

6.4 Practice Problems

6.4.1 Problem 1: High Latency in Search Ranking

System: Elasticsearch retrieves 1000 candidates, XGBoost re-ranks using 300 features from Redis.

Issue: P95 latency 1.2 seconds (target: 400ms)

Your task:

1. Where would you look for bottlenecks? (network, compute, data access)
2. What quick wins could reduce latency by 30%?

3. What architectural changes for 3x improvement?

Example Answer:

Step 1: Identify bottlenecks

- Profile the request: measure time for retrieval vs. ranking vs. feature fetching
- Likely culprit: fetching 300 features \times 1000 candidates = 300K Redis lookups
- Secondary issue: XGBoost inference on 1000 candidates might be slow

Step 2: Quick wins (30% improvement)

- **Feature caching:** Cache popular product features (80/20 rule)
- **Batch Redis reads:** Use pipelining to fetch features in parallel
- **Reduce candidates:** Re-rank only top 500 instead of 1000
- **Feature selection:** Use only top 100 features instead of 300

Step 3: Architectural changes (3x improvement)

- **Multi-stage ranking:**
 1. Stage 1: Fast linear model on 20 features (1000 \rightarrow 200 candidates)
 2. Stage 2: Full XGBoost on 300 features (200 \rightarrow final ranking)
- **Feature store:** Pre-compute and batch-update features (hourly instead of real-time)
- **Model optimization:** Quantize XGBoost model (FP32 \rightarrow FP16 or INT8)
- **Async processing:** Return fast results, refine in background

6.4.2 Problem 2: Cold-Start Recommendations

System: Collaborative filtering model recommends products based on retailer's purchase history.

Issue: New retailers with 0-2 orders get irrelevant recommendations.

Your task:

1. Why does collaborative filtering fail for new users?
2. What signals can you use when there's no purchase history?
3. How would you blend multiple strategies?

Example Answer:

Step 1: Why it fails

- Collaborative filtering needs user-item interactions
- New retailers have sparse or zero interaction data
- Model can't find similar retailers to learn from

Step 2: Alternative signals

- **Content-based:** Use retailer profile (category, location, store size)
- **Popularity:** Show trending/best-selling products in retailer's category
- **Geographic:** Recommend products popular in retailer's region
- **Onboarding data:** Ask preferences during signup (style, price range)

Step 3: Blended strategy

- Use a **multi-armed bandit** or **ensemble model**
- Weight strategies by data availability:
 - 0-5 orders: 70% content + 30% popularity
 - 5-20 orders: 50% content + 30% collaborative + 20% popularity
 - 20+ orders: 80% collaborative + 20% content
- **Exploration:** Inject 10-20% random/diverse items to learn preferences faster

6.4.3 Problem 3: Feature Drift Detection

System: XGBoost ranking model trained 6 months ago, performance degrading.

Issue: Click-through rate dropped from 12% to 9% over 3 months.

Your task:

1. What causes feature drift?
2. How would you detect it?
3. What's your retraining strategy?

Example Answer:

Step 1: Causes of drift

- **Data distribution shift:** New products, seasonal changes, user behavior changes
- **Feature distribution shift:** Price ranges changed, new brands added
- **Concept drift:** Relationship between features and labels changed
- Example: During COVID, "home office" products surged

Step 2: Detection strategies

- **Monitor feature distributions:** Compare train vs. live (KL divergence, PSI)
- **Track model performance:** Daily CTR, NDCG, precision@K
- **A/B test against random:** If model barely beats random, it's drifted
- **Shadow models:** Train weekly model, compare performance to production

Step 3: Retraining strategy

- **Schedule:** Retrain monthly (or when drift detected)

- **Data:** Use last 3-6 months of data (balance recency vs. volume)
- **Validation:** A/B test new model before full rollout
- **Gradual rollout:** 5% \rightarrow 25% \rightarrow 50% \rightarrow 100% traffic
- **Rollback plan:** Keep previous model ready for instant rollback
- **Online learning:** Consider incremental updates between full retrains

6.5 Key Concepts to Know

6.5.1 ML System Performance Issues

- **Latency:** Model inference time, feature retrieval, batching
- **Throughput:** QPS limits, model serving capacity
- **Cost:** Compute costs (GPU), storage costs (feature store)
- **Accuracy:** Drift, cold-start, data quality

6.5.2 Common Optimization Techniques

- **Caching:** Feature caching, result caching, model caching
- **Model compression:** Quantization, pruning, distillation
- **Feature selection:** Reduce dimensionality, remove correlated features
- **Multi-stage ranking:** Fast filter + expensive rerank
- **Asynchronous processing:** Return fast results, refine later

6.5.3 System Design Patterns for ML

- **Feature store:** Centralized feature management (offline + online)
- **Model registry:** Versioning, A/B testing, rollback
- **Monitoring:** Drift detection, performance tracking, alerting
- **Feedback loops:** Log predictions, retrain with outcomes

6.6 How to Prepare

1. **Study Faire's architecture:** Read their engineering blog (craft.faire.com)
2. **Practice system analysis:** Take existing ML systems (Netflix, Spotify), identify bottlenecks
3. **Learn optimization techniques:** Caching, model compression, multi-stage ranking
4. **Understand tradeoffs:** Latency vs. accuracy, cost vs. performance, complexity vs. maintainability
5. **Think incrementally:** Always propose quick wins before major rewrites

7 Machine Learning Evaluation Metrics (For All Interviews)

7.1 Why ML Metrics Matter for Faire

Faire's search and recommendations are ML-powered. You need to:

- Understand common ML evaluation metrics
- Know when to use each metric
- Articulate tradeoffs (precision vs. recall, etc.)
- Relate metrics to business goals

7.2 Classification Metrics

7.2.1 Confusion Matrix

	Predicted Positive	Predicted Negative
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

7.2.2 Core Metrics

1. Accuracy

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \quad (4)$$

When to use: Balanced classes

Limitation: Misleading for imbalanced datasets (e.g., fraud detection: 99% non-fraud)

2. Precision

$$\text{Precision} = \frac{TP}{TP + FP} \quad (5)$$

Interpretation: Of all predicted positives, how many are correct?

When to use: Cost of FP is high (e.g., spam detection - don't mark legitimate emails as spam)

Faire Example: Predicting "retailer will order this product"

- High precision: most recommended products are relevant
- Low precision: many irrelevant products shown

3. Recall (Sensitivity, True Positive Rate)

$$\text{Recall} = \frac{TP}{TP + FN} \quad (6)$$

Interpretation: Of all actual positives, how many did we catch?

When to use: Cost of FN is high (e.g., disease detection - don't miss sick patients)

Faire Example: Search results

- High recall: retrieve all relevant products
- Low recall: miss many relevant products

4. F1 Score

$$F1 = 2 \cdot \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} = \frac{2TP}{2TP + FP + FN} \quad (7)$$

When to use: Balance between precision and recall, imbalanced classes

Limitation: Treats precision and recall equally (sometimes one matters more)

5. ROC-AUC

ROC Curve: Plot of True Positive Rate (Recall) vs. False Positive Rate at various thresholds

$$\text{FPR} = \frac{FP}{FP + TN} \quad (8)$$

AUC (Area Under Curve): Probability that model ranks random positive higher than random negative

When to use: Compare models, threshold-agnostic evaluation

Limitation: Optimistic for imbalanced datasets (use PR-AUC instead)

7.2.3 Example Interview Question

Question: Faire wants to predict which products a retailer will order. You build a model with 90% accuracy. Is this good?

Answer:

- **Need more context:** What's the class distribution?
- If only 5% of shown products are ordered, a naive model (always predict "no order") has 95% accuracy but is useless.
- **Better metrics:** Precision@K, Recall@K (see ranking metrics below)
- **Business metric:** Revenue per recommended product

7.3 Ranking Metrics

For search and recommendations, we care about **order** of results, not just classification.

7.3.1 Precision@K and Recall@K

Precision@K:

$$\text{Precision@K} = \frac{\# \text{ relevant items in top } K}{K} \quad (9)$$

Recall@K:

$$\text{Recall@K} = \frac{\# \text{ relevant items in top } K}{\text{total relevant items}} \quad (10)$$

Example: Search for "ceramic mugs", 100 total relevant products

Top 10 results: 7 relevant, 3 irrelevant

- $\text{Precision@10} = 7/10 = 0.7$
- $\text{Recall@10} = 7/100 = 0.07$

Faire Use Case: Precision@10 for search (top 10 results matter most)

7.3.2 Mean Average Precision (MAP)

Average Precision (AP): Average of precision values at each relevant position

$$AP = \frac{1}{|\text{relevant}|} \sum_{k=1}^n \text{Precision}@k \times \text{rel}(k) \quad (11)$$

Where $\text{rel}(k) = 1$ if item at position k is relevant, else 0.

MAP: Average AP across multiple queries

Example:

Ranked list: [R, NR, R, R, NR, R] (R = relevant, NR = not relevant)

Relevant positions: 1, 3, 4, 6

Precision at each:

- $P@1 = 1/1 = 1.0$
- $P@3 = 2/3 = 0.67$
- $P@4 = 3/4 = 0.75$
- $P@6 = 4/6 = 0.67$

$$AP = \frac{1}{4}(1.0 + 0.67 + 0.75 + 0.67) = 0.77$$

Limitation: Doesn't consider position (item at rank 1 vs. rank 10 weighted equally)

7.3.3 Normalized Discounted Cumulative Gain (NDCG)

Why NDCG? Addresses two issues:

1. **Graded relevance:** Items can be highly relevant, somewhat relevant, or not relevant (not just binary)
2. **Position matters:** Higher-ranked items should be more relevant

Cumulative Gain (CG):

$$CG@K = \sum_{i=1}^K \text{rel}_i \quad (12)$$

Where rel_i is relevance score of item at position i (e.g., 0, 1, 2, 3)

Discounted Cumulative Gain (DCG):

$$DCG@K = \sum_{i=1}^K \frac{\text{rel}_i}{\log_2(i+1)} \quad (13)$$

Position discount: items at lower ranks contribute less

Ideal DCG (IDCG):

DCG if items were perfectly ranked by relevance

Normalized DCG (NDCG):

$$NDCG@K = \frac{DCG@K}{IDCG@K} \quad (14)$$

Range: $[0, 1]$, where 1 = perfect ranking

Example:

Ranked list with relevance scores: [3, 2, 0, 1, 2]

$$DCG@5 = \frac{3}{\log_2 2} + \frac{2}{\log_2 3} + \frac{0}{\log_2 4} + \frac{1}{\log_2 5} + \frac{2}{\log_2 6}$$

$$DCG@5 = 3.0 + 1.26 + 0 + 0.43 + 0.77 = 5.46$$

Ideal ranking: [3, 2, 2, 1, 0]

$$IDCG@5 = 3.0 + 1.26 + 0.77 + 0.43 + 0 = 5.46$$

$$NDCG@5 = 5.46/5.46 = 1.0 \text{ (if we had perfect ranking)}$$

For actual ranking: $NDCG@5 = 5.46/5.46 = 1.0$ (happens to be perfect in this case)

Faire Use Case: NDCG@10 for evaluating search ranking offline

7.3.4 Hit Rate (Recall@K for Recommendations)

Definition: Proportion of users who found at least one relevant item in top K recommendations

$$\text{Hit Rate@K} = \frac{\# \text{ users with } \geq 1 \text{ relevant item in top K}}{\text{total users}} \quad (15)$$

Faire Example:

100 retailers, show top 10 recommendations

- 80 retailers ordered at least one recommended product
- Hit Rate@10 = 80/100 = 0.8

7.4 Regression Metrics

If predicting continuous values (e.g., estimated order value):

7.4.1 Mean Absolute Error (MAE)

$$MAE = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i| \quad (16)$$

Pros: Easy to interpret (average error in same units as target)

Cons: Treats all errors equally (large errors not penalized more)

7.4.2 Mean Squared Error (MSE)

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2 \quad (17)$$

Pros: Penalizes large errors more heavily

Cons: Units are squared (harder to interpret)

7.4.3 Root Mean Squared Error (RMSE)

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2} \quad (18)$$

Pros: Same units as target, penalizes large errors

7.4.4 R-Squared (R^2)

$$R^2 = 1 - \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{\sum_{i=1}^n (y_i - \bar{y})^2} \quad (19)$$

Interpretation: Proportion of variance explained by model

Range: $(-\infty, 1]$, where 1 = perfect predictions, 0 = model is as good as mean baseline

7.5 A/B Testing & Online Metrics

Offline metrics (NDCG, Precision@K) don't always correlate with business outcomes. Must A/B test.

7.5.1 Online Metrics for Search/Recommendations

Engagement Metrics:

- **Click-Through Rate (CTR):** clicks / impressions
- **Add-to-Cart Rate:** adds / impressions
- **Conversion Rate:** orders / impressions

Business Metrics:

- **Revenue per search**
- **Average order value (AOV)**
- **Customer lifetime value (LTV)**

User Experience:

- **Session duration**
- **Bounce rate**
- **Zero-result rate** (searches with no results)

7.5.2 A/B Test Design

Hypothesis: New ranking model increases conversion rate

Setup:

- **Control group:** 50% of retailers see current ranking
- **Treatment group:** 50% see new ranking
- **Randomization:** Hash retailer ID to assign group

Duration: Run for 2 weeks (capture weekly patterns)

Sample Size: Calculate needed sample for statistical power

$$n = \frac{2(Z_{\alpha/2} + Z_{\beta})^2 \sigma^2}{(\mu_1 - \mu_0)^2} \quad (20)$$

Statistical Test:

- **Metric:** Conversion rate (proportion)
- **Test:** Two-proportion z-test
- **Significance level:** $\alpha = 0.05$

Decision:

- If $p < 0.05$ and treatment is better: ship new model
- If not significant: keep investigating (longer test, larger sample)
- Monitor for novelty effect (users like new things initially)

7.6 Common ML Interview Questions

7.6.1 Question 1: Precision vs. Recall Tradeoff

You're building a spam product detector for Faire. Would you optimize for precision or recall? Why?

Answer:

Optimize for precision (minimize false positives).

Reasoning:

- **FP (false positive):** Legitimate product marked as spam
 - Harm: Brand loses sales, may leave platform
 - High cost to business
- **FN (false negative):** Spam product not caught
 - Harm: Some low-quality products shown
 - Lower cost (can be caught by user reports)

Approach:

- Set high threshold for spam classification (e.g., 0.95 probability)
- Human review queue for borderline cases
- Monitor precision/recall in production

7.6.2 Question 2: Imbalanced Dataset

Only 2% of shown products are actually ordered. How do you handle this in training?

Answer:

1. **Resampling:**

- **Oversample** minority class (ordered products)
- **Undersample** majority class (not ordered)
- Use SMOTE (Synthetic Minority Over-sampling)

2. **Class Weights:**

- Assign higher weight to minority class in loss function
- XGBoost: `scale_pos_weight` parameter

3. Evaluation Metrics:

- Don't use accuracy (misleading)
- Use Precision@K, Recall@K, F1, AUC-PR

4. Hard Negative Mining:

- Sample negatives intelligently (products that were clicked but not ordered)
- Harder negatives improve model discrimination

7.6.3 Question 3: NDCG vs. MAP

When would you use NDCG instead of MAP for search evaluation?

Answer:

Use NDCG when:

1. **Graded relevance:** Products have different levels of relevance (highly relevant, somewhat relevant, not relevant)
2. **Position matters a lot:** Top results should be weighted much more than lower results
3. **Industry standard:** NDCG is more common in industry (Google, Bing, etc.)

Use MAP when:

1. **Binary relevance:** Items are either relevant or not (no grades)
2. **All relevant items matter:** Not just top K

For Fairst search: Use NDCG@10 (graded relevance, top 10 results critical)

7.7 ML System Design: Click Prediction Model

Design an ML system to predict click probability for search results at Fairst.

7.7.1 Problem Formulation

Input: (retailer, query, product)

Output: $P(\text{click} \mid \text{retailer, query, product})$

Training Data: Search logs (query, shown products, clicked products)

7.7.2 Feature Engineering

Query Features:

- Query length, word count
- Query category (intent classification)
- Historical CTR for this query

Product Features:

- Price, category, brand
- Popularity (orders, clicks, add-to-carts)
- Image quality score
- Inventory level

Retailer Features:

- Store category, location
- Past order history (categories, price range)
- Account age, activity level

Query-Product Features:

- Text match score (BM25)
- Embedding similarity (query embedding \cdot product embedding)
- Position in search results (strong predictor!)

Retailer-Product Features:

- Cosine similarity in embedding space
- Did retailer order this category before?
- Price relative to retailer's avg. order value

7.7.3 Model Choice**Option 1: XGBoost (Faire's choice)**

- **Pros:** Fast training, handles tabular features well, interpretable
- **Cons:** Doesn't capture complex interactions automatically

Option 2: Deep Neural Network

- **Pros:** Learns feature interactions, works well with embeddings
- **Cons:** Slower, harder to interpret, needs more data

Hybrid: Use embeddings from DNN as features in XGBoost

7.7.4 Training Pipeline

1. **Data Collection:** Stream search logs to data warehouse (Kinesis)
2. **Feature Engineering:** Daily batch job (Spark) to compute features
3. **Training:** Weekly re-train XGBoost on last 30 days of data
4. **Evaluation:** Offline: AUC, LogLoss; Online: A/B test CTR
5. **Deployment:** Export model to PMML, load in ranking service

7.7.5 Serving

```

1 def rank_search_results(retailer_id, query, candidates):
2     # Get retailer features from cache
3     retailer_features = redis.get(f"retailer:{retailer_id}")
4
5     # Compute query features
6     query_features = compute_query_features(query)
7
8     # For each candidate product
9     scored_products = []
10    for product in candidates:
11        # Get product features from cache
12        product_features = redis.get(f"product:{product.id}")
13
14        # Compute interaction features
15        interaction_features = compute_interaction_features(
16            retailer_features, query_features, product_features
17        )
18
19        # Combine all features
20        feature_vector = concatenate([
21            retailer_features,
22            query_features,
23            product_features,
24            interaction_features
25        ])
26
27        # Predict click probability
28        click_prob = xgboost_model.predict(feature_vector)
29
30        scored_products.append((product, click_prob))
31
32    # Sort by predicted click probability
33    scored_products.sort(key=lambda x: x[1], reverse=True)
34
35    return [p for p, score in scored_products]

```

7.7.6 Position Bias Problem

Issue: Items at top positions get more clicks simply due to position, not relevance.

Solutions:

1. **Position feature:** Include position as feature (naive, doesn't fix bias)
2. **Inverse Propensity Weighting:** Weight training examples by $1/P(\text{position})$
3. **Randomization:** Randomly shuffle top results for small % of traffic to collect unbiased data
4. **Examination hypothesis:** Model $P(\text{click}) = P(\text{relevant}) \times P(\text{examined})$

8 Values Interview Prep (Interview Part 4)

Context: This section covers the Values Interview (Part 4), a 30-minute behavioral interview focused on alignment with Faire's 5 core values.

8.1 Faire's OFFICIAL Core Values

These are the official values from the recruiter document. You MUST prepare stories for each value.

8.1.1 1. Be an Owner

Sub-values:

- **Resourceful:** Find creative solutions with available resources
- **Raise the Bar:** Set and maintain high standards
- **Share Our Legos:** Build reusable systems, share knowledge

Prepare a story about:

- Taking ownership of a complex problem end-to-end
- Being resourceful when faced with constraints
- Raising technical standards on your team
- Building reusable infrastructure or tools

8.1.2 2. Be Kind

Sub-values:

- **Collaboration:** Work effectively with others
- **Coachability:** Accept feedback and grow
- **Inclusive:** Create welcoming environment for all

Prepare a story about:

- A positive working relationship with a teammate
- A challenging working relationship you navigated
- Receiving feedback and how you improved
- Making your team more inclusive or collaborative

8.1.3 3. Embrace the Adventure

Sub-values:

- **Agility:** Adapt quickly to change
- **Bias Towards Action:** Move fast, iterate, learn

Prepare a story about:

- Thinking outside the box to solve a problem
- Adapting to a major change (requirements, tech stack, team)
- Taking action despite uncertainty
- Learning from failure and iterating quickly

8.1.4 4. Seek the Truth

Sub-values:

- **Decision-Making:** Make data-driven decisions
- **Analytical Thinking:** Dig deep to understand root causes
- **Communication:** Communicate clearly and transparently

Prepare a story about:

- Using data to make a critical decision
- Debugging a complex issue by digging into root causes
- Communicating transparently about a problem or failure
- Changing your mind based on evidence

8.1.5 5. Serve Our Community

Sub-values:

- **Customer Centricity:** Put retailers and brands first
- **Prioritization:** Focus on highest-impact work
- **Motivation:** Inspired by Faire's mission

Prepare a story about:

- Prioritizing customer needs over technical elegance
- Impact and influence - a high-impact project you led
- Why you care about Faire's mission (empowering entrepreneurs)

8.2 Faire's Mission & Scale (Know This Cold)

Mission: Empower entrepreneurs everywhere to chase their dreams.

Scale:

- 350,000+ independent retailers
- Independent retail sector larger than Amazon + Walmart combined
- 40% of wholesale buying moved online during COVID
- Faire is uniquely positioned to serve this massive market

Why it matters: Small businesses are the backbone of local economies. Faire helps them compete.

8.3 Common Behavioral Questions

8.3.1 Question 1: Why Faire?

Structure your answer:

1. **Mission alignment:** Why you care about supporting small businesses
2. **Technical challenge:** Excited about search/ML/two-sided marketplace problems
3. **Growth stage:** Opportunity to make impact at a scaling startup
4. **Personal connection:** Maybe you have friends who run small shops

Example Answer:

"I'm drawn to Faire because of the mission to empower independent retailers. My aunt runs a small boutique, and I've seen firsthand how challenging wholesale sourcing can be—long payment terms, high minimums, limited product discovery. Faire solves these pain points beautifully.

From a technical perspective, I'm excited about the search and recommendation challenges. Building ML systems that balance retailer preferences with brand visibility is a fascinating two-sided optimization problem. And as someone passionate about data-driven product development, Faire's focus on experimentation and metrics really resonates with me."

8.3.2 Question 2: Tell me about a time you led a project

Use STAR method:

- **Situation:** Set the context
- **Task:** What needed to be done?
- **Action:** What did YOU do? (focus on your contributions)
- **Result:** Quantifiable impact

Prepare examples for:

- Leading a technical project
- Solving a challenging bug

- Collaborating with non-engineers (PM, design)
- Making a data-driven decision
- Handling ambiguity or changing requirements
- Dealing with a difficult teammate or situation

8.3.3 Question 3: What metrics would you track for Faire?

This was mentioned as a common behavioral question. Show business acumen.

Retailer Metrics:

- **Acquisition:** New retailer signups, activation rate
- **Engagement:** Monthly active retailers, searches per session
- **Conversion:** Order conversion rate, avg. order value
- **Retention:** Repeat purchase rate, churn rate, LTV

Brand Metrics:

- **Acquisition:** New brand signups, catalog size
- **Engagement:** Order volume, revenue
- **Retention:** Brand churn, % brands with repeat orders

Platform Metrics:

- **GMV:** Gross Merchandise Value
- **Take rate:** Platform revenue as % of GMV
- **Search quality:** Zero-result rate, CTR, conversion

Two-Sided Health:

- **Liquidity:** % retailers who find products to order
- **Balance:** New brands getting orders vs. established brands

8.4 Questions to Ask Interviewers

About the Role:

- What does success look like in the first 3/6/12 months?
- What are the biggest technical challenges the team is facing?
- How does the team balance new features vs. tech debt?

About the Team:

- How is the search team structured? (Frontend, backend, ML, data science?)

- How do you collaborate with PM, design, and data science?
- What's the on-call rotation like?

About the Company:

- How has Faire's strategy evolved with the market?
- What are the company's top priorities for the next year?
- How do you balance retailer needs vs. brand needs in product decisions?

Technical:

- What ML models are you currently using for search ranking?
- How do you evaluate new ranking algorithms? (A/B testing process?)
- What's the data infrastructure like? (Streaming vs. batch?)
- How do you handle cold start for new retailers/brands?

9 Part 5: Preparation Strategy

9.1 4-Week Study Plan

9.1.1 Week 1: Algorithms & Data Structures

Goal: Solve 30 LeetCode problems (focus on medium difficulty)

Daily: 2 hours

- Day 1-2: Arrays & Hashing (10 problems)
- Day 3-4: Two Pointers, Sliding Window (10 problems)
- Day 5: Binary Search (5 problems)
- Day 6: Linked Lists (5 problems)
- Day 7: Review & mock interview

Resources:

- LeetCode Blind 75
- NeetCode (YouTube + roadmap)
- Tech Interview Handbook

9.1.2 Week 2: More Algorithms + SQL

Goal: 25 more problems + 10 SQL problems

Daily: 2 hours

- Day 1-2: Trees (BFS, DFS, BST) - 10 problems
- Day 3: Graphs (BFS, DFS, Union-Find) - 5 problems
- Day 4: Dynamic Programming (5 problems)
- Day 5: SQL (10 problems on LeetCode)
- Day 6: Heaps, Stacks, Queues (5 problems)
- Day 7: Review & mock interview

9.1.3 Week 3: System Design

Goal: Master 4 system design problems + read about Faire's stack

Daily: 2-3 hours

- Day 1: Read Faire's engineering blog (craft.faire.com)
- Day 2: Design search system (practice this guide's Problem 1)
- Day 3: Design recommendation system (practice Problem 2)
- Day 4: Design URL shortener (classic problem)
- Day 5: Design Instagram/Twitter feed
- Day 6: Review all designs, focus on tradeoffs
- Day 7: Mock system design interview with friend

Resources:

- Grokking the System Design Interview
- System Design Primer (GitHub)
- ByteByteGo (YouTube)

9.1.4 Week 4: ML + Behavioral + Mock Interviews

Goal: Solidify ML metrics, prepare stories, simulate interviews

Daily: 2-3 hours

- Day 1: Study ML metrics (this guide's Part 3)
- Day 2: Practice ML system design (click prediction, ranking)
- Day 3: Prepare behavioral stories (STAR method, 5 stories)
- Day 4: Mock coding interview (Pramp or friend)
- Day 5: Mock system design interview
- Day 6: Review all weak areas
- Day 7: Rest, light review, prepare questions to ask

9.2 Day Before Interview

- **Light review:** Skim your notes, don't cram
- **Re-read this guide:** Sections 1, 2, 3 summaries
- **Prepare questions:** Write down 5 questions to ask
- **Logistics:** Test video/audio, charge laptop, have pen & paper ready
- **Mindset:** Get good sleep, stay confident!

9.3 During the Interview

Coding Interview:

1. **Clarify:** Ask questions, confirm input/output, edge cases
2. **Think aloud:** Explain your approach before coding
3. **Start simple:** Brute force first, then optimize
4. **Test:** Walk through examples, catch bugs
5. **Communicate:** If stuck, ask for hints (shows collaboration)

System Design Interview:

1. **Clarify requirements:** Functional, non-functional, scale
2. **High-level design first:** Draw boxes (API, DB, cache, etc.)
3. **Deep dive:** Pick 2-3 components to detail (interviewer will guide)
4. **Discuss tradeoffs:** SQL vs. NoSQL, sync vs. async, etc.
5. **Time management:** Don't spend 40 min on API design

Behavioral Interview:

1. **STAR method:** Structure answers clearly
2. **Quantify impact:** "Reduced latency by 40%", "Increased CTR by 15%"
3. **Show learning:** "This taught me to...", "I'd do X differently next time"
4. **Ask questions:** Show genuine interest in the role

10 Appendix A: Faire-Specific Questions (To Be Added)

This section will be populated with actual interview questions from Glassdoor, Prepfully, and InterviewQuery after you provide the content.

10.1 Placeholder for Glassdoor Questions

[To be filled in after user provides Glassdoor content]

10.2 Placeholder for Prepfully Questions

[To be filled in after user provides Prepfully content]

10.3 Placeholder for InterviewQuery Content

[To be filled in after user provides InterviewQuery content]

11 Appendix B: Faire Engineering Blog Insights

11.1 Key Articles to Read

1. "How data and machine learning shape Faire's marketplace"

- URL: craft.faire.com/how-data-and-machine-learning-shape-faires-marketplace-510855c4a9bc
- Key takeaways:
 - Recommendation embeddings from clicks, add-to-cart, orders
 - Hybrid recommender system for cold start
 - XGBoost models for ranking
 - Redis feature store for real-time features
 - Balancing retailer and brand stakeholder needs

2. "Real-time ranking at Faire part 2: the feature store"

- URL: craft.faire.com/real-time-ranking-at-faire-part-2-the-feature-store-3f1013d3fe5d
- Key takeaways:
 - Feature store architecture with Redis
 - Handling hundreds of features per product
 - Low-latency feature retrieval (< 10ms)

11.2 Tech Stack Summary

Languages: Kotlin (backend), JavaScript/TypeScript (frontend)

Data: MySQL, Redis, NoSQL, ElasticSearch, Kinesis, DBT

ML: XGBoost, embeddings, collaborative filtering

Infrastructure: Kubernetes, AWS (ELB, SQS), Nginx

12 Appendix C: Additional Resources

12.1 Coding Practice

- **LeetCode:** leetcode.com (Blind 75, NeetCode 150)
- **NeetCode:** neetcode.io (video solutions + roadmap)
- **AlgoExpert:** algoexpert.io (paid, high quality)
- **HackerRank:** hackerrank.com (SQL practice)

12.2 System Design

- **Grokking the System Design Interview:** educative.io
- **System Design Primer:** github.com/donnemartin/system-design-primer
- **ByteByteGo:** bytebytego.com (Alex Xu's course)
- **Designing Data-Intensive Applications:** Book by Martin Kleppmann

12.3 Machine Learning

- **StatQuest:** youtube.com/@statquest (ML concepts)
- **Google ML Crash Course:** developers.google.com/machine-learning/crash-course
- **Chip Huyen's ML Interviews Book:** huyenchip.com/ml-interviews-book

12.4 Behavioral

- **Tech Interview Handbook:** techinterviewhandbook.org/behavioral-interview
- **Grokking the Behavioral Interview:** educative.io

13 Summary & Final Tips

13.1 Key Takeaways

1. **Faire is a two-sided marketplace** connecting retailers and brands via search and recommendations
2. **Tech stack:** Kotlin, Elasticsearch, XGBoost, Redis, MySQL, Kubernetes
3. **Interview focus:**
 - Algorithms: Medium LeetCode, 2 rounds DSA
 - System Design: Search systems (your main focus)
 - ML: Evaluation metrics (precision, recall, NDCG)
 - Behavioral: Culture fit, why Faire
4. **Search system design** is critical - know two-stage retrieval + ranking
5. **ML metrics:** Master NDCG, Precision@K, Recall@K for ranking problems
6. **Tradeoffs:** Be ready to discuss latency vs. accuracy, precision vs. recall, complexity vs. maintainability

13.2 The Night Before

- Review this guide's summaries (sections 2, 4, 5)
- Re-read Faire's mission and engineering blog posts
- Prepare 5 questions to ask
- Get 8 hours of sleep

13.3 You've Got This!

You've prepared thoroughly. Remember:

- **Communication & perfection:** Interviewers want to see your thought process
- **Ask questions:** Shows curiosity and collaboration
- **Stay calm:** If stuck, take a breath, think aloud, ask for hints
- **Be yourself:** Faire values authenticity and passion

Good luck! You're going to do great!