DataHub Day 2 Interview Preparation

Part 1: Project Deep Dive (Game Clickbait Detection)
Part 2: Search Architecture Discussion

Alex Yang
Principal Software Engineer, Roblox

Interview Date: November 20, 2025
Prepared: November 19, 2025

Contents

1 Interview Overview

1.1 Format

- **Interviewer:** Abe (DataHub)
- **Duration:** 45-60 minutes
- **Part 1:** Project Deep Dive (25-30 min)
- **Part 2:** Search Architecture Discussion (20-25 min)

1.2 Selected Project

Game Clickbait Detection System (Q1 2024, Roblox)
Why this project:

- Complete end-to-end ownership (problem definition → architecture → deployment)
- High-visibility impact (CEO escalation → townhall recognition)
- Cross-team leadership (5 teams: DSA, Human Eval, Brand Safety, ML Platform, Game Discovery)
- Systems thinking (IP protection, search quality, policy framework)
- Measurable success (99%+ precision, real-time detection)
- Recent and fresh (2024)

2 30-Second Executive Summary

Critical Point

Practice delivering this in 30 seconds:
"In Q1 2024, our CEO escalated an urgent search quality issue: copycat games with identical names and images polluted results when searching for popular games like 'Brookhaven'. A team had spent 2 months on a CLIP-based image similarity approach, but when I evaluated it, precision was near-random.
I identified the core issue within days, threw away the previous work, and redesigned from first principles. The result: a real-time, multi-modal IP protection system with 99%+ precision that processes game updates in 10 seconds, protects hundreds of assets across billions of games, and was recognized at company townhall."

3 Project Narrative (STAR Framework)

3.1 Situation (2-3 minutes)

3.1.1 The Problem

CEO Observation (Holiday 2023):

- Searching "Brookhaven" (top Roblox game) returned pages of copycat games
- Copycats used identical/similar names and thumbnail images
- Search results looked "ugly" and hurt platform trust
- Users clicking copycats experienced low-quality, unrelated gameplay

3.1.2 Previous Team's Approach (2 months of work)

Method: CLIP-based image similarity detection

1. Pipeline to get top 100 popular games
2. Extract game thumbnail images
3. Generate CLIP embeddings for all game thumbnails (daily/weekly)
4. Find games with thumbnails semantically close to top 100
5. Demote these games in ranking

My Evaluation (within days):

- Ran human evaluation on their results with different thresholds (0.9, 0.95)
- **Result: Precision near-random (∼50%)**
- Many false positives (legitimate games with similar art styles)
- Many false negatives (modified images evaded detection)
- **Critical flaw: Completely ignored title duplication signal**

Key Insight

Why the previous approach failed:

- Treated as pure image similarity problem
- CLIP clusters by visual style, not deceptive intent
- False positives: Anime-style games, similar genres (tycoon, obby)
- False negatives: Color shifts, crops, overlays evade detection
- Ignored strongest signal: title text copying

3.1.3 Problem Reframing (First Principles)

Key question I asked: "Why is this even a problem?"
If authentic Brookhaven ranks #1, users can identify it. So what are we *really* solving?
Two distinct problems identified:

1. **IP Protection (Primary)**

- Platform responsibility to protect creator intellectual property
- Games stealing name + image = trademark/copyright violation
- Roblox's reputation as fair UGC platform at stake

2. **Search Quality (Secondary)**

- Copycats are typically low-quality (quick cash grabs)
- Crowd out higher-quality, diverse alternatives
- Users want: original game (#1) + quality similar games, NOT 50 low-effort clones

Design Principle: Multi-signal detection for IP violations, not just image matching

3.2 Task (1-2 minutes)

3.2.1 My Responsibilities

- **Technical architecture:** Design end-to-end detection system
- **Cross-team coordination:** Align 5 teams on approach and deliverables
- **Policy framework:** Define what constitutes IP violation
- **Implementation:** Build real-time detection service and offline backfill
- **Evaluation:** Establish metrics and validation pipeline
- **Deployment:** Launch to production with A/B testing

3.2.2 Constraints

- **Time:** CEO urgency - need results in Q1 2024
- **Scale:** Hundreds of millions of existing games to process
- **Latency:** Real-time updates (¡1 minute from game publish to ranking impact)
- **Precision:** Must avoid false positives (creator backlash risk)
- **Coverage:** Top 30 games covers 20-30% of search queries (Pareto principle)

3.2.3 Success Criteria

- 95%+ precision on clickbait detection
- Real-time detection (¡1 min latency)
- Backfill all existing games
- Zero creator backlash (clear, fair policies)
- CEO satisfaction (visible problem resolution)

3.3 Action (5-7 minutes)

3.3.1 Core Innovation: Protected Assets Library (PAL)

Concept: Centralized registry of IP assets to protect
Schema:

```
Field: type        (icon, title, logo, etc.)
Field: creatorId
Field: universeId
Field: rootPlaceId
Field: titleMatchPolicy (for title assets: exact, substring, fuzzy)
```

Violation Logic: Any signal triggered → demotion in ranking
Three Asset Types:

- **Signal A:** Text (game names)
- **Signal B:** Images (full thumbnails)
- **Signal C:** Logos (brand marks within images)

3.3.2 Signal A: Text Protection (Rule-Based Heuristics)

Why rule-based, not semantic similarity?

- "Microhard" doesn't violate "Microsoft" IP (semantic but not legal violation)
- Need legal precision, not fuzzy matching
- Semantic similarity failed in evaluation (no practical utility)
- Heuristics achieved 99%+ precision on golden dataset

Three Match Policies (Flexible Framework):

1. **exact:** Strict 1:1 match
   - Use case: "Evade" (common English word, don't over-protect)
   - Normalization: lowercase, remove spaces, strip emojis/brackets
2. **substring:** Contains match
   - Use case: "Welcome to Bloxburg" prevents "Welcome to Bloxburg Christmas!"
   - Protects against title stuffing: "BrookhavenRP Adopt Me! Pet Simulator"
3. **fuzzy:** Edit distance tolerance
   - Use case: "Brookhaven" with fuzzy=2 catches "bro0khaven", "Br00khaven"
   - Prevents simple evasion tactics

Built-in Exceptions (Nuanced Policy):

- **Fan-made allowed:** "Brookhaven RP [Fan Version]" NOT clickbait

- **Same creator/group:** Can create own spinoffs (ownership check)
- **Typo tolerance:** "Fan Versoin" still triggers (prevents evasion)

Cross-functional Work:

- Collaborated with Community Program Manager on policy drafting
- Community announcement rollout (several weeks)
- Clear creator communication through official channels
- **Result:** Zero creator backlash

Impact: Solved ∼30% of clickbait cases

3.3.3 Signal B: Image Protection (CLIP + PAL)

Key Difference from Previous Approach:

- **Previous:** Compare ALL games to top 100 (noisy, high false positive rate)
- **Mine:** Compare only against curated PAL library (precise, targeted)

Implementation:

- Human Eval team curates protected assets using Label Studio
- CLIP embeddings generated for full thumbnails
- Embeddings stored in RSS (Roblox Similarity Search - vector database)
- Fine-tuned similarity thresholds per PAL entry
- Monthly pipeline reviews top 30 games + ad-hoc updates

PAL Evolution:

- Started: ∼30-100 entries (top games)
- Current: Hundreds of entries
- Planned: External brands (Disney, Coca-Cola, etc.)

Precision: 95%+ on image-based clickbait

3.3.4 Signal C: Logo Protection (YOLOv8 + CLIP)

Two-Stage Detection Pipeline:

1. **Stage 1: YOLOv8 Logo Detection (Fine-tuned by me)**
   - Detect logo regions within game thumbnails
   - Training: Human Eval drew bounding boxes in Label Studio
   - Fine-tuned YOLOv8 on Roblox-specific logo corpus
   - Accuracy: 95%+

2. **Stage 2: CLIP Logo Comparison**
    - Extract detected logo regions
    - Generate CLIP embeddings for logos
    - Compare against PAL logo library (vector search)

Cross-Team Integration:

- Leveraged Brand Safety team's logo detection endpoint
- Established versioning protocol for future model updates:
    - Old endpoint maintained
    - New versioned endpoints for A/B testing
    - Enables safe model migrations

Combined Precision: 99.5%+ on logo-based clickbait

3.3.5 Real-Time Detection Architecture

Design Shift:

- **Previous:** Weekly batch processing (stale, reactive)
- **Mine:** Event-driven, real-time (10-second latency)

System Architecture:

```
SNS Topics (Game Lifecycle Events):
  - PRODUCTION_PlaceEntityChangeEvents
  - PRODUCTION_UniverseDisplayInformationChangeEvents
  - PRODUCTION_UniverseEntityChangeEvents
          ↓
    SQS Queue (~10 QPS)
          ↓
  Queue Processor Service
          ↓
Clickbait Detection Service (bedev2 microservice)
  |-- Text rule matching (heuristics)
  |-- CLIP embedding generation -> RSS query
  \-- YOLO logo detection -> Brand Safety endpoint
          ↓
Frost Feature Store
  Table: sdp_production_nonpii.clickbait_games_features_v0
  Columns: universe_id, is_clickbait, metadata (JSON), updated_time_unix
          ↓
Experience Document Builder (SNS topic)
          ↓
Elasticsearch Index (for ranking)
          ↓
Search Retrieval & Ranking Pipeline
```

Key Design Decisions:

- **Event-driven:** Triggered by game lifecycle, not polling (efficient)
- **SQS buffering:** Handles traffic spikes, decouples services (resilient)
- **Microservice:** Modular CLIP/YOLO inference (scalable)
- **Feature serving:** Signals propagate to ranking in ∼10 seconds (fast)
- **Non-blocking:** Doesn't delay game creation/updates (user-friendly)

API Features:

- REST endpoints for on-demand detection
- Manual override API for edge cases (parodies, legitimate references)
- JSON metadata field for detailed signal breakdown

3.3.6 Offline Backfill Pipeline

Challenge: Process hundreds of millions of existing games
Solution: Modular MLP Pipelines

1. **Separate Pipeline per Feature** (title, thumbnail, logo)
   - Load games corpus from Hive
   - Load PAL
   - Run similarity detection (text heuristics or vector search)
   - Output to intermediate Hive tables
2. **Powerhouse Aggregation Pipeline**
   - Load input tables: `sdp_icon_similarity`, `sdp_title_similarity`
   - Aggregate scores into boolean `is_clickbait` column
   - Publish batch to Frost feature store

Benefits of Modular Design:

- Scale compute independently per feature
- Iterate on one feature without reprocessing all
- Recover from failures without full reruns
- Clear separation of concerns (detection vs aggregation)

3.3.7 Cross-Team Collaboration

Five Teams Coordinated:

1. **DSA (Jinlong Ji)**
   - Owned initial backfill pipeline architecture
   - Top-K game collection logic
   - Hive table management
2. **Human Eval (Patricia Morales)**
   - Created PAL using Label Studio
   - Drew logo bounding boxes for YOLOv8 training
   - Ongoing evaluation pipeline (human-in-the-loop)
3. **Brand Safety Team**
   - Provided logo detection endpoint
   - Versioning protocol for future model updates
4. **ML Platform Team**
   - Inference infrastructure (CLIP/YOLO serving)
   - Model deployment support
5. **Game Discovery Team**
   - Search ranking integration
   - Final ownership post-launch
   - Ongoing policy iteration

Key Coordination Challenge:
Handling abandonment of 2 months of work:

- Presented evaluation results showing near-random precision
- Data spoke for itself - approach wasn't salvageable
- Principal ML Engineer moved to other priorities (no friction)
- Key: Show problem clearly + propose concrete alternative (not just criticize)

3.4 Result (1-2 minutes)

3.4.1 Metrics

Precision/Recall:

- **Previous approach:** ∼50% precision (near-random)
- **Text rules:** 99%+ precision, solved 30% of cases
- **CLIP (full image):** 95%+ precision
- **YOLO + CLIP (logos):** 99.5%+ precision

Scale:

- **PAL:** Hundreds of protected assets
- **Coverage:** 20-30% of top search queries
- **Backfill:** Hundreds of millions of existing games processed
- **Real-time:** ∼10 QPS game updates, ∼10-second latency end-to-end

Business Impact:

- **CEO satisfaction:** Problem visibly resolved for top queries
- **Townhall recognition:** Company-wide acknowledgment
- **Policy framework:** Established IP protection standards for platform
- **Expandable:** Foundation for other fraud detection (bait-and-switch teleport games)
- **Zero creator backlash:** Clear communication, fair policies

3.4.2 Timeline

- **Week 1:** Identified previous approach failure, proposed redesign
- **Milestone 1 (March 1):** Title text protection prod launch + A/B
- **Milestone 2 (March 15):** Thumbnail protection prod launch
- **Milestone 3 (March 22):** Logo protection prod launch
- **Q1 2024 End:** Evaluation pipeline, optimization, handoff to Game Discovery
- **Ongoing:** Maintained and evolved system for several quarters

3.4.3 Handoff

- Successfully transferred ownership to Game Discovery team
- Modular architecture enabled them to iterate on policies without rewriting core services
- Continued evolution: bait-and-switch detection, expanded PAL, etc.

3.5 Technical Challenges & Lessons (2-3 minutes)

3.5.1 Challenge 1: Balancing Precision vs Coverage

Problem: Can't manually curate PAL for millions of games
Solution:

- Focus on top 30 games (Pareto principle: 20-30% query coverage with minimal effort)
- Human-in-the-loop for high-profile additions
- Scalable pipeline for monthly reviews

Lesson: Perfect is the enemy of good - solve 80% of the problem with 20% of the effort

3.5.2 Challenge 2: Real-Time Latency Requirements

Problem: Game creation can't be blocked; search results must reflect changes quickly
Solution:

- Async event processing (SQS decoupling)
- Non-blocking: Detection happens post-creation
- Acceptable staleness: 10-second propagation vs instant blocking

Lesson: Understand business constraints - near real-time (seconds) $\gg$ batch (days)

3.5.3 Challenge 3: Fine-Tuning YOLOv8 for Logos

Problem: Off-the-shelf YOLO detects generic objects, not game logos
Solution:

- Created custom dataset with Human Eval (bounding boxes in Label Studio)
- Fine-tuned YOLOv8 on Roblox-specific logo corpus
- Achieved 95%+ accuracy on domain-specific task

Lesson: Domain-specific ML requires domain-specific data - generic models aren't enough

3.5.4 Challenge 4: Policy Rollout Without Creator Backlash

Problem: New restrictions could anger creators (platform risk)
Solution:

- Clear, fair rules (spinoff patterns allowed)
- Transparent communication (official community announcements)
- Gradual enforcement (education before penalties)
- Explainable heuristics ("Your title matches X with fuzzy=1" vs "Model score 0.87")

Lesson: Technical solutions need social/policy layer - engineering alone isn't enough

3.5.5 Challenge 5: Defining "Clickbait" vs "Poor Experience"

Problem: Initial discussions blurred clickbait (pre-game deception) with poor in-game quality

My Framing:

- **Clickbait** = misleading *pre-game signals* (title, thumbnail) regardless of in-game content
- **Poor experience** = separate problem requiring in-game scene analysis (future scope with RFM models)

Why This Mattered:

- If evaluated based on in-game content, pre-game detection algorithms would falsely appear to fail
- Example: Game copies "Brookhaven" title/image but has transformative gameplay → still clickbait for IP protection

Solution:

- Created golden dataset focused *only* on pre-game signals
- Separated in-game content analysis to "poor experience detection" (out of scope for V1)
- Enabled clear algorithm evaluation and explainable enforcement

Lesson: Precise problem definition is foundational - ambiguity leads to misaligned evaluation

3.5.6 What I'd Do Differently

1. **Earlier evaluation framework**
   - Built golden dataset evaluation late
   - Should've been day-one priority to validate approach faster
2. **More automated PAL maintenance**
   - Top 30 games required manual monthly review
   - Could've built automated pipeline detecting trending games for Human Eval queue
3. **Better internal documentation**
   - Leadership didn't fully understand project complexity (part of reason for seeking new opportunities)
   - Should've created exec-level summary decks: problem → solution → impact

What Worked Well:

- Modular architecture paid off - Game Discovery could iterate on policies without rewriting core services
- Cross-team coordination was smooth due to clear work streams and ownership
- First principles thinking caught fundamental flaw early (saved months of wasted effort)

4 Connection to DataHub

Key Insight

How This Project Maps to DataHub's Challenges:
"This project taught me that **metadata quality** - whether it's game titles, thumbnails, or IP assets - is foundational to platform trust. At DataHub, metadata quality is the *core product*.
I see direct parallels:

- **Protected Assets Library (PAL) ≈ DataHub's metadata registry**
  - Both are centralized sources of truth
  - Both require curation, governance, and maintenance
  - Both enable downstream systems (search ranking vs data discovery)
- **Cross-team adoption** was key to my success at Roblox
  - Got 5 teams (DSA, Human Eval, Brand Safety, ML Platform, Game Discovery) to trust and contribute to PAL
  - DataHub faces same challenge: getting engineering, data science, and business teams to trust and contribute metadata
- **Real-time propagation** (game updates → search ranking in 10 seconds)
  - DataHub needs similar: schema changes → lineage updates → downstream awareness
  - Event-driven architecture patterns directly applicable
- **Policy framework** (exact/substring/fuzzy matching policies)
  - Flexible policies ¿ one-size-fits-all
  - DataHub needs governance policies: who can edit what, approval workflows, data classification rules

I'm excited about DataHub because it's solving the *same fundamental problem* - metadata quality and governance - at the data infrastructure level rather than search/discovery."

5 Mock Q&A Scenarios

5.1 Technical Deep Dive Questions

5.1.1 Q: Why didn't you use a machine learning model for title detection?

Answer:
"I evaluated both approaches. For title matching, heuristics significantly outperformed semantic similarity models for three reasons:

1. **Explainability:** We needed to communicate to creators *exactly why* they were flagged for

policy compliance. 'Your title matches protected title X with fuzzy distance 1' is clear; 'model confidence 0.87' is not. This was critical for avoiding creator backlash.

2. **Semantic similarity misalignment:** ML models would flag 'Zombie Night' as similar to 'Halloween Night' based on semantic meaning, but that's not IP violation. Clickbait is about exact/near-exact copying, not conceptual similarity.

3. **Pattern recognition for evasion:** Common evasion tactics like 'title stuffing' - cramming multiple popular titles like 'BrookhavenRP Adopt Me! Pet Simulator' - are trivial to detect with substring rules but hard for models to learn without massive training data and constant retraining.

4. **Empirical results:** When we evaluated semantic similarity on our golden dataset, it had 'no practical utility' - near-random precision. Heuristics achieved 99%+ precision.

We *did* use ML where appropriate: CLIP for image similarity and YOLOv8 for logo detection, where embeddings and object detection excel. The key was choosing the right tool for each signal."

5.1.2 Q: How did you handle the massive backfill of hundreds of millions of games?

Answer:

"I designed separate MLP pipelines per feature - title, thumbnail, logo - so we could run them independently and iterate on one without reprocessing all. The architecture:

1. **Per-Feature Pipelines:**
   - Each loads games corpus from Hive
   - Loads PAL for that feature type
   - Runs similarity detection (text heuristics or vector search)
   - Outputs to intermediate Hive tables: `sdp_title_similarity`, `sdp_icon_similarity`
2. **Powerhouse Aggregation Pipeline:**
   - Aggregates scores from intermediate tables
   - Publishes final `is_clickbait` boolean to Frost in batch

This modular design had three benefits:

- **Independent scaling:** Could scale compute per feature based on workload (CLIP inference vs simple string matching)
- **Fault tolerance:** If one pipeline failed, didn't need full rerun - just reprocess that feature
- **Iterability:** Could improve title matching heuristics without re-running CLIP embeddings on billions of images

We coordinated with ML Platform team to ensure adequate CLIP inference capacity for the backfill burst."

5.1.3 Q: What metrics did you use to evaluate success?

Answer:

"We had multi-level metrics across model performance, business impact, and operations:
Model Performance (Precision/Recall):

- Golden dataset evaluation on 2000+ labeled examples
- Per-signal accuracy: Title (99%), CLIP (95%), YOLO+CLIP (99.5%)
- Confusion matrix analysis to identify systematic errors

Business Impact:

- CEO satisfaction: Problem visibly resolved for top queries like 'Brookhaven'
- Query coverage: Top 30 protected games covered 20-30% of search queries (Pareto principle validated)
- PAL growth: Started with ∼30 entries, grew to hundreds (system scaling successfully)
- Creator feedback: Zero backlash (policy clarity worked)

Operational Metrics:

- Real-time latency: P50/P99 for end-to-end detection
- Throughput: Sustained ∼10 QPS game updates without queue buildup
- Backfill completion: Time to process all existing games
- False positive rate: Manual override API usage as proxy

The combination gave us confidence in both technical correctness and business value."

5.1.4 Q: How did you ensure high precision to avoid false positives?

Answer:

"False positives were our biggest risk - flagging legitimate games would hurt creators and damage platform trust. I used a multi-layered approach:

1. **Conservative Thresholds:**
   - Started with stringent similarity thresholds (high confidence bar)
   - Fine-tuned per PAL entry based on that asset's visual distinctiveness
   - Used confusion matrix to optimize precision vs recall trade-off
2. **Human-in-the-Loop:**
   - Label Studio pipeline for ongoing evaluation
   - High-profile games get human review before PAL addition
   - Feedback loop: flagged games reviewed weekly, errors fed back to threshold tuning
3. **Nuanced Policies:**

- Built-in exceptions: fan-made content, same creator ownership
- Flexible match policies: 'Evade' uses exact (common word), 'Brookhaven' uses fuzzy=2
- Manual override API for edge cases (parodies, legitimate references)

4. **Ranking Demotion, Not Filtering:**
   - Demoted in ranking rather than hard-filtered
   - Users still have choice; we just deprioritize suspected clickbait
   - Reduced impact of false positives

The result: 99%+ precision with zero creator backlash, which validated our approach."

5.1.5 Q: Walk me through your collaboration with the Brand Safety team on logo detection.

Answer:

"The Brand Safety team had an existing logo detection model that hadn't been updated since deployment. I needed to integrate it while planning for future model evolution. Here's how I approached it:

Discovery Phase:

- Met with Brand Safety team to understand their model (YOLOv8-based)
- Identified constraint: model was static, no update cadence
- Learned their model detected generic logos, not game-specific ones

Integration Strategy:

- Used their endpoint for initial logo region detection (bounding boxes)
- Fine-tuned my own YOLOv8 on Roblox-specific logo corpus (using Label Studio dataset)
- Two-stage pipeline: their model for generic detection → my model for game logo classification

Future-Proofing:

- Established versioning protocol: if they update model, maintain old endpoint
- Create new versioned endpoint (`/v2/detect-logos`)
- Enables A/B testing before migration
- Asked if they could incorporate PAL into their training (potential future improvement)

Result:

- Smooth integration, 99.5%+ precision on logo-based clickbait
- Reusable pattern for future cross-team ML integrations
- Brand Safety team appreciated the versioning protocol (adopted it for other consumers)

This taught me that cross-team ML integration requires planning for model evolution, not just initial integration."

5.2 Systems Design Questions

5.2.1 Q: Why event-driven architecture for real-time detection? What are the trade-offs?

Answer:

"I chose event-driven architecture for three reasons:

Benefits:

1. **Efficiency:** Only process games that changed (vs polling all games periodically)
   - Game updates are sparse events (∼10 QPS vs millions of games)
   - Polling would waste 99.9% of compute checking unchanged games
2. **Low Latency:** React immediately to changes (10-second end-to-end)
   - Users see updated ranking within seconds of game publish
   - vs batch: hours/days of staleness
3. **Decoupling:** SQS buffer isolates detection service from upstream events
   - Handles traffic spikes without dropping events
   - Detection service can scale independently
   - Failed processing goes to dead letter queue for recovery

Trade-Offs:

1. **Complexity:** More moving parts (SNS, SQS, multiple services)
   - Mitigation: Used mature AWS services with high availability
   - Clear ownership boundaries between teams
2. **Eventual Consistency:** 10-second delay before ranking reflects changes
   - Acceptable: game creation isn't blocked, creators don't see immediate impact anyway
   - Much better than batch (hours/days)
3. **Ordering:** SQS doesn't guarantee FIFO (standard queue)
   - Not a problem: we only care about final state, not order of updates
   - Each update is idempotent (upsert to Frost by universe_id)

For this use case, the benefits far outweighed the trade-offs. Event-driven was the right choice."

5.2.2 Q: How would you handle a sudden 10x spike in game update traffic?

Answer:

"The architecture has natural buffers, but I'd implement additional safeguards:

Existing Resilience:

- SQS queue absorbs spikes (messages wait until processed)
- Detection service can scale horizontally (more bedev2 instances)

- Frost writes are async (doesn't block upstream)

Additional Mitigations for 10x Spike:

1. **Rate Limiting at SQS:**
   - Set max queue depth threshold (e.g., 100K messages)
   - If exceeded, temporarily drop low-priority events (minor game updates)
   - Prioritize new game creation events over minor metadata changes
2. **Auto-Scaling Detection Service:**
   - CloudWatch alarm on SQS ApproximateNumberOfMessages
   - Trigger auto-scaling policy: add bedev2 instances when queue depth $>$ 10K
   - Scale down when queue drains to save cost
3. **Batching at Frost:**
   - Instead of per-message Frost write, batch 100 updates
   - Reduces Frost write load by 100x
   - Trade-off: slightly higher latency (batch flush interval)
4. **Circuit Breaker for Downstream:**
   - If CLIP/YOLO inference services are overloaded, fail fast
   - Requeue to SQS for retry with exponential backoff
   - Prevents cascading failures
5. **Monitoring & Alerting:**
   - P99 latency alerts (if $>$ 60 seconds, something's wrong)
   - Queue depth alerts (if $>$ 50K, need manual intervention)
   - Dead letter queue alerts (if messages failing, investigate)

The key principle: **graceful degradation**. Accept slightly higher latency under extreme load, but never drop data or crash services."

5.3 Project Management Questions

5.3.1 Q: How did you convince stakeholders to abandon 2 months of work?

Answer:

"This was delicate - I had to show the problem clearly without making the previous team look bad. Here's how I approached it:

1. **Data-First Approach (Week 1):**
   - Immediately ran human evaluation on their CLIP results
   - Tested multiple thresholds (0.9, 0.95) on Brookhaven case
   - Result: precision $\sim$50% (near-random)

- Documented: false positives (legitimate anime-style games), false negatives (modified images)

2. **Root Cause Analysis:**
   - **Not:** "Your approach is wrong" (accusatory)
   - **Instead:** "We're solving the wrong problem" (reframe)
   - Showed CLIP clusters by visual style, not deceptive intent
   - Identified missing signal: title duplication (strongest clickbait indicator)
3. **Concrete Alternative (Not Just Criticism):**
   - Presented multi-signal architecture: text + image + logo
   - Showed how each signal addresses specific failure modes
   - Estimated timeline: M1 (title) in 3 weeks, full system in Q1
4. **Meeting Dynamics:**
   - Invited Principal ML Engineer to the discussion (respect)
   - Framed as "we discovered this together through evaluation"
   - He acknowledged CLIP-only approach had limitations
   - Moved to other priorities (no friction)
5. **Leadership Alignment:**
   - CEO urgency gave me leverage (problem needs solving *now*)
   - PM supported pivot after seeing evaluation data
   - Committed to clear milestones (M1, M2, M3) for accountability

Key Lesson: When proposing to scrap work, lead with **data**, offer **concrete alternatives**, and **avoid blame**. Make it about the problem, not the people."

5.3.2 Q: What was the biggest challenge in coordinating 5 teams?

Answer:

"The biggest challenge was **aligning on scope and deliverables** - each team had different priorities and timelines. Here's what I did:

Challenge 1: Diverging Priorities

- DSA team wanted comprehensive backfill (all games, all time)
- Human Eval had limited bandwidth (can't label millions of games)
- ML Platform focused on inference efficiency (latency optimization)
- Game Discovery wanted fast deployment (CEO pressure)

Solution: Clear Work Streams

- Created detailed milestones doc with team ownership (M1: title by DSA, M2: thumbnail by me + Human Eval, M3: logo by Brand Safety)

- Weekly sync meetings (30 min, focused agenda)
- Slack channel for async updates (`#clickbait-detection-project`)
- **Decision-making authority:** I had final call on architecture; PM had final call on launch timing

Challenge 2: Human Eval Bottleneck

- Label Studio tasks required expert judgment (what is clickbait?)
- Human Eval team had competing priorities (safety moderation)

Solution: Prioritization + Iteration

- Phase 1: Top 30 games only (minimal viable PAL)
- Created detailed labeling guidelines to reduce ambiguity
- Automated easy cases (exact title match) to reduce human load
- Phase 2: Expand PAL based on query volume analysis

Challenge 3: Brand Safety Model Dependency

- Their logo model was static (no update schedule)
- Risk: if model degrades, we're stuck

Solution: Versioning Protocol + Fine-Tuning

- Agreed on versioned endpoints for future updates
- Fine-tuned my own YOLOv8 as backup (not fully dependent)
- Offered to share PAL data for their model retraining (mutual benefit)

Key Lesson: Cross-team projects need **clear ownership**, **explicit dependencies**, and **fallback plans**. Over-communicate, document decisions, and reduce critical path dependencies where possible."

5.3.3 Q: You mentioned leadership didn't fully understand the project. What happened there?

Answer (Honest but Diplomatic):

"This was a learning experience for me about the importance of **executive communication**, not just technical execution.

What Happened:

- Project had high visibility (CEO escalation, townhall recognition)
- But my direct leadership didn't fully grasp the technical complexity or business impact
- This contributed to my decision to explore new opportunities

Where I Could've Done Better:

1. **Executive-Level Artifacts:**
   - I wrote detailed technical docs (architecture, APIs, pipelines)
   - Should've also created 1-pager for leadership: Problem → Solution → Impact
   - Metrics dashboard showing business value (query coverage, precision, CEO satisfaction)
2. **Proactive Updates:**
   - I communicated in team meetings and Slack
   - Should've done monthly 1:1 highlights with skip-level manager
   - "Here's what we shipped, here's the impact, here's what's next"
3. **Connecting to Org Goals:**
   - I framed as "fixing clickbait problem"
   - Could've framed as "establishing IP protection framework for platform" (strategic)
   - Or "improving search quality metrics" (measurable org goal)

What I Learned:

- Technical excellence alone doesn't guarantee recognition
- Senior/Principal engineers must translate technical work into business value *for leadership*
- Documentation, metrics, and storytelling are as important as code

Looking Forward:

- At DataHub, I'd apply these lessons from day one
- Create exec-friendly artifacts alongside technical design docs
- Proactively communicate impact, not just features shipped

This experience taught me that **impact without visibility is wasted effort** - something I'm now very conscious of."

6 Part 2: Search Architecture Deep Dive

6.1 Roblox Search Architecture Overview

Key Insight

Architecture Diagrams Available:
You have two search architecture references to review:

- `Snd-Platform_architecture-Roblox-Simple.svg` - Roblox Search & Discovery Platform
- `search-architecture-for-coupang.excalidraw` - Coupang E-commerce Search

Review the Roblox diagram visually before the interview - it shows your actual production system. The Coupang architecture has been incorporated into this document for comparative context.

6.1.1 Detailed Architecture (From Production Diagram)

Core Systems (Top Layer):

1. **Search Service**
   - **Service Manager:** Query Understanding & Transformation, API Manager
   - **Search Service:** Core search orchestration, Query Manager
   - Routes queries to indexing and ranking systems
2. **SnD Content Service**
   - **Content Service:** Manages content aggregation
   - **Blending & Re-Ranking:** Combines results from multiple sources
   - **Content API Manager:** External API layer
   - Your clickbait signals feed into this blending layer!

Vertical Search Services (Left Side):
Specialized search capabilities that feed into the main pipeline:

- **T&S Filtering:** Trust & Safety filtering (includes your clickbait detection!)
- **Query Similarity, Sequence Understanding, Embeddings:** Vector-based search
- **Nature Entity Recognition Service:** NER for query parsing
- **Autocomplete Spell Correction:** Query suggestions and typo fixing
- **Category Discovery Service:** Genre/category classification

Indexing System (Center):

- **Elastic Search Cluster:** Multiple shards for horizontal scaling

- **ES Indexer Service:** Writes documents to ES
- **Indexing Manager:** Orchestrates indexing pipeline
- **Doc Processor:** Transforms raw data into searchable documents
- **Update Listener:** Real-time updates from event streams
- Your clickbait detection connects here via event-driven updates!

Ranking System (Center-Right):

- **Ranking Framework/System:** Orchestrates multi-signal ranking
- **Model Serving Endpoint:** Serves ML models for ranking
- **Model Training:** Trains ranking models offline
- **Ranking Data Pipeline:** ETL for ranking features
- **ML Platform:** Infrastructure for model deployment
- Your clickbait signals are ranking features in this system!

Recommendation System (Right Side):
Similar architecture to Ranking System:

- **Recommendation Framework/System:** Personalized recommendations
- **Model Serving Endpoint:** Real-time inference
- **Model Training:** Collaborative filtering, embeddings
- **Ranking Data Pipeline:** User behavior data
- **ML Platform:** Shared infrastructure

Data Layer (Bottom):

- **Messaging Queue:** Event-driven communication (likely Kafka/SQS)
- **SQL Store:** Vertical data, metadata, game catalog
- **Signal Platform:** Centralized signal aggregation (like Frost!)
- **AWS Data Store:**
  - Data buckets (raw data, logs)
  - Signal buckets (computed features - your clickbait signals here!)
  - Model buckets (trained models)

Supporting Systems (Bottom):

- **Monitoring, Alerting, Visualization:** Operational health
- **Metrics & Evaluation:** Search quality metrics (NDCG, MRR, etc.)
- **Human Eval:** Label Studio for golden dataset evaluation
- **Diagnostic Tools:** Debugging and performance analysis

6.1.2 Data Flow Through the System

Query Path (User Search):

```
User Query
    ↓
Search Service (Query Understanding & Transformation)
    ↓
Vertical Search Services (parallel):
  - Autocomplete/Spell Correction
  - Query Similarity/Embeddings
  - Category Discovery
  - T&S Filtering (Clickbait check!)
    ↓
Elasticsearch Cluster (retrieval)
    ↓
Ranking System
  - Model Serving (ML models)
  - Ranking Data Pipeline (features from Signal Platform)
  - Multi-signal scoring
    ↓
SnD Content Service (Blending & Re-Ranking)
    ↓
Results returned to user
```

Indexing Path (Game Updates):

```
Game Update Event
    ↓
Messaging Queue
    ↓
Update Listener (Indexing System)
    ↓
Doc Processor (transform to ES document)
    ↓
Indexing Manager
    ↓
ES Indexer Service → Elasticsearch Cluster
    ↓
Signal Platform (update features)
    ↓
Available for search queries
```

Your clickbait detection sits in this flow:

```
Game Update → Detection Service → Frost (Signal Platform)
    → ES Indexer → Ranking System
```

6.1.3 Key Architectural Patterns

1. Modular Service Architecture:

- Separate concerns: Search, Indexing, Ranking, Recommendation, Content
- Each can scale independently
- Clear API boundaries between services

2. Vertical Services Pattern:

- Specialized services for specific capabilities (T&S, NER, Autocomplete)
- Compose into main search pipeline
- Your clickbait detection is a T&S vertical service!

3. Centralized Signal Platform:

- Single source of truth for features (Frost-like)
- Feeds both Ranking and Recommendation systems
- Enables feature reuse across ML models

4. ML Platform Abstraction:

- Shared infrastructure for model training and serving
- Ranking and Recommendation use same platform
- Reduces operational complexity

5. Event-Driven Updates:

- Messaging queue decouples producers from consumers
- Update Listener enables real-time indexing
- Your SNS/SQS pattern mirrors this!

Key Insight

Your Clickbait Detection Project in the Broader Architecture:
Now you can clearly articulate where your project fits:

1. **Component:** Trust & Safety (T&S) Filtering vertical service
2. **Input:** Game update events via Messaging Queue
3. **Processing:** Multi-signal detection (text, CLIP, YOLO)
4. **Output:** Features to Signal Platform (Frost)
5. **Consumption:**
   - Ranking System reads clickbait signals from Signal Platform
   - SnD Content Service applies Blending & Re-Ranking
   - Demotion in final search results
6. **Evaluation:** Human Eval system for golden dataset validation

Interview Talking Point:
"My clickbait detection project is a Trust & Safety vertical service in Roblox's Search & Discovery Platform. It consumes game update events, performs multi-signal analysis, publishes quality signals to our centralized Signal Platform (Frost), and those signals feed into the Ranking System for search result demotion. This is a perfect example of how specialized vertical services compose into a larger search architecture - a pattern I see DataHub could leverage for metadata quality signals."

6.2 Key Topics to Master

6.2.1 Semantic Search & Vector Databases

Your Experience:

- Roblox Similarity Search (RSS) - vector database for CLIP embeddings
- Used in clickbait detection project for image similarity
- Hybrid search: keyword + semantic

Key Points:

- **Embedding generation:** CLIP for images, sentence transformers for text
- **Vector database technology:** RSS (internal), alternatives like FAISS, Pinecone, Weaviate
- **Trade-offs:** Latency (vector search slower than keyword), recall/precision balance
- **Hybrid approaches:** Combine BM25 keyword scores with vector similarity

6.2.2 Ranking & Relevance

Multi-Signal Ranking Framework:

- **Textual relevance:** BM25, TF-IDF scores
- **Popularity signals:** Play counts, engagement metrics
- **Quality signals:** Your clickbait detection, safety scores
- **Personalization:** User preferences, historical behavior
- **Diversity:** Avoid ranking 50 similar games

Evaluation Metrics:

- **NDCG (Normalized Discounted Cumulative Gain):** Measures ranking quality
- **MRR (Mean Reciprocal Rank):** First relevant result position
- **Precision@K:** Fraction of top K results that are relevant
- **A/B testing:** Online evaluation with real users

6.2.3 Query Understanding

Challenges & Solutions:

- **Typos:** "Brokehaven" $\rightarrow$ "Brookhaven"
  - Fuzzy matching (edit distance)
  - Query autocomplete/suggestions
  - Did-you-mean functionality
- **Synonyms:** "Roleplay" = "RP"
  - Synonym dictionaries
  - Query expansion
- **Intent detection:** Is user searching for a specific game or browsing a genre?
  - Navigational (specific game) vs Exploratory (genre/category)
  - Affects ranking strategy

6.3 Search Architecture Patterns (Industry Comparison)

6.3.1 Federated Multi-Cluster Architecture (E-commerce Pattern)

Architecture Overview:

Based on large-scale e-commerce search systems (Coupang, Amazon-style), this pattern differs from Roblox's monolithic approach:

Core Components:

- **Search Root Service:** Orchestration layer that routes queries
- **Specialized Clusters:** Separate search clusters by use case
  - Product Search Cluster (main catalog)
  - Category-Specific Cluster (optimized per vertical)
  - Brand Search Cluster (brand-centric queries)
  - Ads/Sponsored Cluster (monetization)
- **ML Ranking Service:** Centralized LTR (Learning to Rank)
- **Query Understanding Layer:** Separate service for rewriting
- **Knowledge Graph:** Entity expansion and relationships

Technology Stack:

- **Search Engine:** Solr (with Lucene) or Elasticsearch
- **Streaming:** Kafka for real-time ingestion and logging
- **ETL:** Apache Airflow + Spark for batch processing
- **ML:** XGBoost, TensorFlow for ranking models
- **Storage:** Feature Store, HBase (Solr backend), S3, Hive
- **Observability:** ELK Stack (logging), Prometheus (metrics)

Query Flow (Federated Pattern):

```
Client → LB → Gateway → Search Root Service
    ↓
Query Understanding (spell check, synonyms, intent)
    ↓
A/B Test Assignment + Corpus Selection
    ↓
Parallel Fanout to Clusters (Prod, Category, Ads, Brand)
    ↓
Per-Cluster Processing:
  - Lucene: parse, score, filter, facet
  - Shard-level parallelism
  - Cache hits
    ↓
```

```
ML Re-Ranking (Feature Store → LTR Service)
    ↓
Result Merge + Deduplication + Post-Processing
    ↓
Response → Gateway → Client
```

Key Patterns:

1. **Two-Stage Ranking**
   - Stage 1: Fast retrieval with simple scoring (BM25, TF-IDF)
   - Stage 2: ML re-ranking on top K candidates (XGBoost/TensorFlow)
   - Trade-off: Recall (stage 1) vs Precision (stage 2)
   - Similar to your clickbait detection: Fast heuristics + ML signals
2. **Feature Store Pattern**
   - Centralized storage for ML features
   - Precomputed features for low-latency serving
   - Examples: click-through rate, conversion rate, user preferences
   - Connection to your work: Frost feature store for clickbait signals
3. **Hybrid Indexing (Batch + Real-Time)**
   - Batch: Full reindex + alias switch (zero-downtime)
   - Real-time: Kafka streaming for fresh content
   - Trade-off: Consistency vs Freshness
   - Your implementation: SNS/SQS events → real-time clickbait detection
4. **Knowledge Graph Integration**
   - Entity relationships (Brand → Products, Category → Items)
   - Random walk algorithms for expansion
   - Enables semantic search beyond keyword matching
   - Potential Roblox use case: Game → Creator → Similar Games
5. **Comprehensive Observability**
   - Query logs, click logs, ranking logs, error logs
   - Metrics: latency, QPS, cache hit rate, conversion funnel
   - A/B test platform integrated into query routing
   - Zero-result queries, top queries, abandonment analysis

Dimension	Roblox (UGC Games)	E-commerce (Products)
Index Size	Hundreds of millions of games	Billions of products (SKUs)
Update Frequency	High (creators constantly update games)	Medium (inventory changes, new products)
Quality Signals	Clickbait detection, safety scores, engagement	Reviews, ratings, return rates, freshness
Personalization	User game history, genre preferences	Purchase history, browsing, demographics
Query Intent	Navigational (find specific game) vs Exploratory (browse genre)	Transactional (buy) vs Research (compare)
Ranking Goal	Engagement (playtime, retention)	Revenue (conversion, AOV, profit margin)
Monetization	Indirect (Robux in-game)	Direct (product sales, ads)
Trust & Safety	Critical (child safety, IP protection)	Important (counterfeit, compliance)

6.3.2 Roblox vs E-commerce: Architectural Comparison

Common Challenges:

- **Scale:** Both need to handle millions of items and high QPS
- **Quality:** Spam/fraud detection (clickbait vs fake products)
- **Relevance:** Balancing keyword match vs semantic similarity
- **Freshness:** New content needs to appear quickly
- **Diversity:** Avoid showing 50 similar results

6.3.3 Advanced Topics to Discuss

1. Learning to Rank (LTR):

- Pointwise: Predict relevance score per item
- Pairwise: Learn which of two items should rank higher
- Listwise: Optimize entire ranked list (NDCG optimization)
- Connection: Your multi-signal clickbait detection is a form of LTR

2. Cold Start Problems:

- New games: No play counts, no reviews → rely on metadata quality
- New users: No history → global popularity + explore/exploit
- Solution: Hybrid of content-based + collaborative filtering

3. Query Expansion Strategies:

- Synonym expansion: "RP" → "Roleplay"
- Stemming/Lemmatization: "playing" → "play"

- Acronym handling: "FPS" → "First Person Shooter"
- Embedding-based: BERT/sentence transformers for semantic similarity

4. Indexing Pipeline Optimization:

- Incremental updates vs full reindex
- Index warming: Pre-load caches before switching alias
- Partitioning strategies: By region, by popularity tier, by recency
- Your experience: Modular MLP pipelines for clickbait backfill

6.3.4 Search Patterns Applied to DataHub

Key Insight

How Search Architecture Experience Translates to DataHub:
DataHub is fundamentally a **metadata search and discovery platform**. Many search patterns directly apply:

1. **Multi-Signal Ranking for Data Assets:**
   - **Textual relevance:** Table name, column names, descriptions (BM25)
   - **Popularity:** Query frequency, number of downstream dependencies
   - **Quality:** Documentation completeness, schema validity, freshness
   - **Trust:** Ownership clarity, certification status, lineage depth
   - **Personalization:** User's team, frequently accessed tables, role-based

   Similar to your clickbait detection, DataHub needs quality signals to demote low-quality metadata.
2. **Real-Time Metadata Propagation:**
   - Schema change in production → lineage graph update → downstream awareness
   - Event-driven architecture (Kafka) mirrors your SNS/SQS pattern
   - Challenge: Consistency across distributed metadata sources
   - Your experience: Real-time feature propagation (Frost → ES → Ranking)
3. **Feature Store Pattern = Metadata Store:**
   - Your Frost feature store for ML signals
   - DataHub's metadata graph is analogous: centralized source of truth
   - Both need: versioning, lineage, governance, real-time updates
   - Both enable downstream systems: search ranking vs data quality tools
4. **Cross-Team Adoption Challenges:**
   - Your PAL required buy-in from 5 teams (DSA, Human Eval, Brand Safety, ML Platform, Game Discovery)
   - DataHub requires buy-in from engineering, data science, analytics, business teams
   - Common challenges: Metadata quality, contribution incentives, governance policies
   - Your approach: Clear policies, explainable rules, minimize friction
5. **Query Understanding for Data Discovery:**
   - Users search for "customer purchase history" → need to find:
     - Tables: `users`, `orders`, `transactions`
     - Synonyms: "purchase" = "order" ≈ "transaction"
     - Related concepts: Join paths, upstream sources
   - Knowledge Graph: Table → Columns → Lineage → Owners

6.4 Trade-Offs Discussion

Dimension	Trade-Off
Latency vs Quality	Deeper ranking (more signals, ML models) improves quality but increases latency. Must balance within P99 ¡ 200ms constraint.
Freshness vs Consistency	Real-time indexing (new games appear instantly) vs eventually consistent (batch updates, lower load).
Recall vs Precision	Cast wide net (more results, some irrelevant) vs high-confidence only (fewer results, all relevant).
Personalization vs Diversity	Show user preferences vs help users discover new content types.
Keyword vs Semantic	Exact matches (fast, precise) vs conceptual similarity (slower, broader).
Compute Cost vs Quality	Expensive ML models (BERT, neural ranking) vs simpler heuristics (BM25).

6.5 Potential Probing Questions

1. **End-to-End Flow**
   - "Walk me through what happens when a user types 'Brookhaven' in search."
   - *Answer:* Query normalization → Elasticsearch retrieval → Multi-signal ranking → Clickbait filtering → Results return
2. **Scale Challenges**
   - "How do you handle very popular queries that could overwhelm the system?"
   - *Answer:* Caching (Redis), query result pre-computation for top queries, rate limiting per user
3. **Cold Start**
   - "What's your strategy for ranking results for a brand new user?"
   - *Answer:* Fall back to global popularity signals, no personalization initially, use engagement to build profile
4. **Quality Issues**
   - "How do you detect and fix search quality issues?"
   - *Answer:* Human evaluation pipeline (Label Studio - you used this!), A/B testing, monitoring query abandonment rates, user feedback loops
5. **New Ranking Signal**
   - "How would you add a new ranking signal to the system?"
   - *Answer:* Feature engineering → Offline evaluation → A/B test → Gradual rollout → Monitor metrics

6. **Clickbait Integration**
    - "How does your clickbait detection integrate with search ranking?"
    - *Answer:* Frost feature store → Elasticsearch index → Ranking demotion (not hard filter), preserves user choice

Key Insight

Connection Strategy:
When discussing search architecture, **connect it back to your clickbait project**:

- "The clickbait detection system I built feeds directly into search ranking"
- "We use the same Frost feature store pattern for other ranking signals"
- "RSS (vector database) was central to both semantic search and image-based clickbait detection"
- "The real-time event-driven architecture I designed mirrors the search indexing pipeline"

This shows you understand how your project fits into the bigger picture.

7 Practice Checklist

Action Item

2-Day Prep Plan:

Day 1 (3-4 hours) - Focus on Part 1: Project Deep Dive

- Morning (2 hours):
  - Read this document cover-to-cover
  - Highlight 5-7 key talking points to memorize
  - Practice 30-second elevator pitch (time yourself)
- Afternoon (2 hours):
  - Draw clickbait detection architecture diagram from memory
  - Practice answering "Why heuristics?" question out loud
  - Write down 3 questions to ask Abe about DataHub

Day 2 (3-4 hours) - Focus on Part 2: Search Architecture

- Morning (2 hours):
  - Review Mock Q&A section for Part 1
  - Practice STAR narrative out loud (record yourself, aim for 10-12 min)
  - Review Search Architecture Deep Dive section
  - Practice explaining: "Walk me through what happens when a user searches"
- Afternoon (1-2 hours):
  - Draw Roblox search architecture from memory (high-level)
  - Practice connecting clickbait project to broader search system
  - Review trade-offs table - be ready to discuss each
  - Final review: Key Insights and Critical Points boxes

Night Before Interview

- Review 30-second pitch (know it cold)
- Skim Technical Challenges section (3-5 lessons learned)
- Prepare 3 questions for Abe (show genuine interest)
- Get good sleep!

Interview Day Morning

- **Part 1 prep:** Skim Executive Summary (page 2), review clickbait architecture diagram
- **Part 2 prep:** Review search architecture high-level flow, recall 3 key trade-offs
- Practice: "This project taught me..." (Connection to DataHub)
- Practice: "When a user searches Brookhaven..." (end-to-end flow)

8 Key Talking Points (Memorize These)

Part 1: Project Deep Dive (Clickbait Detection)

1. **First Principles Reframing**
   - Previous team: image similarity problem
   - Me: IP protection + search quality problem
   - Multi-signal detection (text + image + logo), not single model
2. **Flexible Policy Framework**
   - Three match policies: exact, substring, fuzzy
   - Not one-size-fits-all: "Evade" vs "Brookhaven" need different protection
   - Built-in exceptions: fan-made, same creator, typo tolerance
3. **Right Tool for Each Signal**
   - Heuristics for text (explainable, 99% precision)
   - CLIP for images (semantic similarity works here)
   - YOLO for logos (object detection + fine-tuning)
4. **Event-Driven Real-Time Architecture**
   - SNS → SQS → Detection Service → Frost → Search Ranking
   - 10-second latency, non-blocking, scalable
   - vs previous: weekly batch (stale, reactive)
5. **99%+ Precision**
   - Systematic validation on golden dataset (2000+ examples)
   - Conservative thresholds, human-in-the-loop, nuanced policies
   - Zero creator backlash (proof of policy success)
6. **Cross-Team Leadership**
   - Coordinated 5 teams (DSA, Human Eval, Brand Safety, ML Platform, Game Discovery)
   - Clear work streams, weekly syncs, decision-making authority
   - Successful handoff to Game Discovery post-launch

Part 2: Search Architecture Discussion

1. **Search Flow Overview**
   - Query normalization → ES retrieval → Multi-signal ranking → Results
   - Real-time indexing via event-driven pipeline
   - Latency target: P99 ¡ 200ms
2. **Hybrid Search Strategy**

- Keyword (BM25) + Semantic (RSS/CLIP embeddings)
- Trade-off: Precision vs broader discovery
- Used same RSS infrastructure in clickbait detection

3. **Multi-Signal Ranking & LTR**
   - Textual relevance + Popularity + Quality (clickbait!) + Personalization
   - Two-stage ranking: Fast retrieval + ML re-ranking (similar to e-commerce pattern)
   - A/B testing framework for signal tuning
   - Evaluation: NDCG, MRR, precision@K
   - Feature Store pattern (Frost) mirrors industry best practices
4. **Architectural Patterns Knowledge**
   - Can compare Roblox (monolithic) vs E-commerce (federated multi-cluster)
   - Understand trade-offs: UGC games vs product catalog
   - Knowledge Graph potential: Game → Creator → Similar Games
   - Query understanding: Intent detection, typo correction, expansion
5. **Key Trade-Offs**
   - Latency vs Quality (more signals = better results but slower)
   - Freshness vs Consistency (real-time vs batch)
   - Personalization vs Diversity (preferences vs discovery)
   - Scale vs Cost (complex ML models vs simple heuristics)
6. **Connection to DataHub**
   - Search/discovery is central to both Roblox and DataHub
   - Metadata quality challenges: game metadata vs data asset metadata
   - Real-time propagation patterns applicable to lineage updates
   - Feature Store pattern (Frost) analogous to metadata storage
   - My broad search experience (gaming + e-commerce patterns) directly relevant

9 Questions to Ask Abe

1. **Product Strategy:**
   - "What's DataHub's biggest challenge in driving enterprise adoption? Is it technical (scale, latency) or organizational (getting teams to contribute metadata)?"
2. **Technical Architecture:**
   - "How does DataHub handle real-time metadata propagation? If a schema changes in production, how quickly do downstream systems see that in lineage graphs?"
3. **Team Dynamics:**

- "For this role, what's the balance between building new platform features vs enabling customer integrations? How much is greenfield vs existing codebase evolution?"

4. **Search/Discovery Expertise:**

   - "Given my background in search and RAG systems, where do you see opportunities for me to contribute to DataHub's metadata search and discovery experience?"

5. **Personal Growth:**

   - "What does success look like for a Principal Engineer at DataHub in the first 6-12 months? What are the key milestones or projects that would demonstrate high impact?"

Strategy: Pick 2-3 questions that genuinely interest you. Let the conversation flow naturally - you may not need to ask all of them if the discussion covers the topics organically.

10 Final Reminders

Critical Point

Interview Execution Tips:

- **Start High-Level, Then Drill Down**
  - 30-second summary → problem reframing → architecture → technical details
  - Let Abe guide depth (if he asks about YOLO fine-tuning, dive deep; if not, stay high-level)
- **Use Whiteboard/Paper**
  - Draw architecture diagram as you explain
  - Visual aids help both you and interviewer follow along
  - Shows systems thinking
- **Be Honest About Challenges**
  - Don't pretend everything was perfect
  - "What I'd do differently" shows self-awareness and learning
  - Leadership recognition issue: frame as learning about exec communication
- **Show Enthusiasm**
  - This was a project you're proud of - let that show!
  - Technical problem-solving is fun, not drudgery
  - Connect excitement about this work to excitement about DataHub
- **Ask Clarifying Questions**
  - If Abe asks something unclear, ask for clarification
  - Shows careful thinking, not just rushing to answer
  - "Just to make sure I understand, you're asking about X vs Y?"
- **Time Management**
  - Aim for 10-12 min STAR narrative (not 20 min monologue)
  - Leave time for Abe's follow-up questions (the real signal)
  - If running long, ask: "Should I keep going or would you like to dive deeper into something specific?"

Key Insight

You've Got This!
You have:

- 20+ years of engineering experience
- A compelling project with measurable impact (99%+ precision, CEO recognition)
- Deep technical expertise (search, ML, distributed systems)
- Clear narrative arc (problem → insight → solution → impact)
- Strong connection to DataHub's mission (metadata quality, cross-team adoption)

Trust your experience. Be yourself. Show your passion for solving hard problems.
Good luck!

A Quick Reference: Key Numbers

Metric	Value
Previous Approach Precision	∼50% (near-random)
Title Heuristics Precision	99%+
CLIP Image Similarity Precision	95%+
YOLO + CLIP Logo Precision	99.5%+
Real-Time Latency	∼10 seconds end-to-end
SQS Throughput	∼10 QPS game updates
PAL Size (Initial)	∼30-100 entries
PAL Size (Current)	Hundreds of entries
Query Coverage	20-30% of top queries
Protected Games (Top)	30 games (expandable)
Backfill Scale	Hundreds of millions of games
Project Timeline	Q1 2024 (∼3 months)
Teams Coordinated	5 (DSA, Human Eval, Brand Safety, ML Platform, Game Discovery)
Creator Backlash	Zero (policy success)

B Quick Reference: Architecture Components

- **PAL (Protected Assets Library):** Centralized registry (text, image, logo)
- **SNS Topics:** PlaceEntityChangeEvents, UniverseDisplayInformationChangeEvents
- **SQS Queue:** Buffer for game update events (∼10 QPS)
- **Queue Processor:** Consumes SQS, calls Detection Service
- **Detection Service (bedev2):** Multi-signal analysis (text, CLIP, YOLO)
- **RSS (Vector DB):** CLIP embeddings storage and similarity search
- **Frost Feature Store:** Table `clickbait_games_features_v0`
- **Elasticsearch:** Indexed for search ranking integration
- **MLP Pipelines:** Offline backfill (per-feature: title, thumbnail, logo)
- **Powerhouse:** Aggregates scores, publishes to Frost in batch

C Quick Reference: Contact Info

- **Interviewer:** Abe (DataHub)
- **Interview Type:** Day 2 Onsite - Project Deep Dive + Search Architecture
- **Format:** 45-60 minutes
- **Your Role:** Principal Software Engineer, Roblox
- **Project:** Game Clickbait Detection System (Q1 2024)