DataHub Day 2 Interview Preparation

Project Deep Dive: Game Clickbait Detection System

Alex Yang
Principal Software Engineer, Roblox

Interview Date: November 20, 2024
Prepared: November 19, 2024

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 ∼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 Practice Checklist

Action Item

2-Day Prep Plan:

Day 1 (3-4 hours)

- 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 architecture diagram from memory (whiteboard/paper)
  - Practice answering "Why heuristics?" question out loud
  - Write down 3 questions to ask Abe about DataHub

Day 2 (3-4 hours)

- Morning (2 hours):
  - Review Mock Q&A section
  - Practice STAR narrative out loud (record yourself, aim for 10-12 min)
  - Identify weak spots (what questions make you hesitate?)
- Afternoon (1-2 hours):
  - Mock interview with friend/colleague
  - Practice whiteboard diagram + verbal walkthrough
  - 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

- Skim Executive Summary (page 2)
- Review architecture diagram (can you draw it from memory?)
- Practice: "This project taught me..." (Connection to DataHub)
- Arrive 10 min early, bring water, have paper/pen ready

7 Key Talking Points (Memorize These)

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 $\rightarrow$ SQS $\rightarrow$ Detection Service $\rightarrow$ Frost $\rightarrow$ 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
7. **Connection to DataHub**
   - PAL $\approx$ DataHub metadata registry
   - Cross-team adoption challenges (same at DataHub)
   - Real-time propagation (schema changes $\rightarrow$ lineage updates)
   - Policy framework (governance at DataHub)

8 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.

9 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!

10 Part 2: Search Architecture Deep Dive

10.1 Overview

Key Insight

Your Search Expertise at Roblox:
You are uniquely positioned to discuss search architecture because you've worked across the entire search stack:

- **Retrieval Layer:** RSS (billion-scale vector database), semantic search
- **Ranking Layer:** Clickbait detection signal (quality scoring)
- **Query Understanding:** Spelling correction service, autocomplete
- **Search Quality:** Game clickbait detection, evaluation frameworks
- **Platform Infrastructure:** RAG Platform (retrieval + generation)

Key Message: You're not just a user of search systems - you're a *builder of search infrastructure* that others depend on.

10.2 Roblox Search Architecture: High-Level Overview

10.2.1 The Roblox Search Problem Space

What Makes Roblox Search Unique:

1. **Scale:**
   - Millions of user-generated games (experiences)
   - Billions of search queries annually
   - Real-time content creation (games published every second)

2. **User-Generated Content (UGC) Challenges:**
   - Noisy metadata (creators gaming SEO with keyword stuffing)
   - Quality variance (AAA games vs low-effort clones)
   - IP violations (clickbait, copycats)
   - Sparse signals for new content (cold start problem)
3. **Diverse User Base:**
   - Age range: 6-60+ years old
   - Query types: navigational ("Brookhaven"), exploratory ("fun games"), developmental ("obby")
   - Intent: play now, discover new, find friends' games

10.2.2 End-to-End Search Flow

Draw this architecture during the interview:

```
User Query: "Brookhaven"
    |
[1] Query Understanding
    |-- Spelling Correction (your service)
    |-- Query Normalization (lowercase, trim)
    |-- Intent Detection (navigational vs exploratory)
    \-- Query Expansion (synonyms, related terms)
    |
[2] Retrieval (Multi-Stage)
    |-- [2a] Keyword Retrieval (Elasticsearch)
    |   |-- Title/Description match (BM25 scoring)
    |   \-- Returns: Top 1000 candidates
    |-- [2b] Semantic Retrieval (RSS - your system)
    |   |-- Query -> CLIP embedding
    |   |-- Vector search against game thumbnails
    |   \-- Returns: Top 500 candidates
    \-- [2c] Hybrid Fusion
        \-- Merge keyword + semantic results -> Top 200 candidates
    |
[3] Ranking
    |-- [3a] Feature Generation (Frost)
    |   |-- Engagement: CTR, play time, favorites
    |   |-- Quality: clickbait score (your system), creator reputation
    |   |-- Freshness: publish date, update recency
    |   |-- Personalization: user history, age appropriateness
    |-- [3b] Learning-to-Rank (LTR) Model
    |   |-- Gradient-boosted trees (XGBoost/LightGBM)
    |   |-- Trained on user feedback (clicks, play sessions)
    |   \-- Outputs: relevance score per candidate
    \-- [3c] Re-Ranking & Filtering
```

```
        |-- Diversity: avoid duplicate creators in top 10
        |-- Safety: filter age-inappropriate content
        \-- Policy: demote clickbait (your system)
    |
[4] Serving
    |-- Cache: Redis for popular queries
    |-- Latency: P99 < 500ms
    \-- A/B Testing: gradual rollout of ranking changes
    |
Search Results Displayed to User
```

10.3 Your Contributions to Roblox Search

10.3.1 1. RSS (Roblox Similarity Search) - Vector Database Infrastructure

Your Role: Initiator and lead engineer (2022-2023, V1)

Problem Solved:

- Roblox needed billion-scale vector search for semantic retrieval
- Off-the-shelf solutions (Pinecone, Weaviate) didn't meet requirements:
  - Cost at Roblox scale (∼100M+ vectors)
  - Latency for real-time serving (P99 ¡ 100ms)
  - Data residency (must run in Roblox data centers)

Technical Approach:

1. **Ported Milvus (Open Source) to Roblox Infrastructure**
   - Milvus: distributed vector database (built on FAISS)
   - Adapted for Roblox's Nomad-based microservice environment
   - Integration with Roblox storage layer (S3, persistent volumes)
2. **Deployment Architecture**
   - Distributed across Roblox data centers
   - Sharded by collection (games, avatars, assets)
   - HNSW index for approximate nearest neighbor (ANN) search
   - Replica sets for high availability
3. **Scale Metrics**
   - Billions of vectors indexed
   - P99 latency: ∼50-100ms for k-NN search (k=100)
   - ∼50 production use cases across Roblox (search, recommendations, safety)

First Major Use Case: Avatar Marketplace Semantic Search

- **Problem:** Keyword search failed for visual queries ("red hoodie", "anime hair")

- **Solution:**
  - Generated CLIP embeddings for all avatar items (images)
  - Indexed in RSS (multi-modal: text query → image results)
  - Hybrid retrieval: keyword (Elasticsearch) + semantic (RSS)
- **Impact:** Improved discovery for long-tail items (less popular items surfaced)

Trade-Offs Discussion (Critical for Interview):

Decision	Benefits	Trade-Offs
Milvus vs Build from Scratch	Faster time-to-market, battle-tested at scale	Less control, vendor lock-in (mitigated by open source)
HNSW vs FLAT index	Sub-100ms latency (ANN)	Lower recall than exhaustive search (acceptable for top-k)
In-house vs SaaS	Cost savings, data residency, customization	Operational burden (maintenance, upgrades)
Sharding by collection	Isolation, independent scaling per use case	Cross-collection queries require multiple searches

10.3.2 2. RAG Platform - Retrieval-Augmented Generation

Your Role: Lead architect and engineer (2024, V1 MVP)

Problem Solved:

- Multiple teams building ad-hoc RAG systems (AI Safety, DevRel, ROS)
- Duplicated effort: data ingestion, embedding generation, vector DB integration
- No standardized evaluation or monitoring

Technical Approach:

1. **Ported AnythingLLM (Open Source) to Roblox BEDEV2**
   - AnythingLLM: full-stack RAG platform (document processing, embeddings, LLM integration)
   - Adapted for Nomad-based microservice deployment
   - Integration with Roblox auth, data access controls
2. **Platform Components**
   - **Data Ingestion:** PDF, DOCX, CSV parsers; chunking strategies
   - **Embedding Generation:** CLIP, Sentence-BERT, domain-specific models
   - **Vector Storage:** RSS integration (reused your vector DB)
   - **LLM Orchestration:** Multi-LLM support (GPT-4, Claude, internal models)
   - **Agent Framework:** Tool use, memory, prompt templates
3. **Web Platform (No-Code RAG)**

- UI for non-engineers to build RAG apps
- Upload documents → configure retrieval → deploy chatbot
- Reduced development cycle: 3 months → 1 week

Adoption & Impact:

- **10+ production use cases within 6 months**
- **Use Case 1: Moderation Mate (Safety Team)**
  - RAG over moderation policies and case histories
  - Helps moderators make consistent decisions
  - 5K QPS at peak
- **Use Case 2: Community Insights (DevRel)**
  - RAG over creator feedback, forum posts, support tickets
  - Helps DevRel team understand creator pain points
  - Research tool, not user-facing

RAG-Specific Challenges (Discuss if Asked):

1. **Data Privacy & Access Control**
   - Problem: RAG system has access to sensitive documents
   - Solution: Row-level security, document-level permissions
   - Integration with Roblox's RBAC (role-based access control)
2. **Retrieval Quality (Precision vs Recall)**
   - Problem: Retrieving irrelevant chunks hurts LLM output quality
   - Solution: Hybrid search (keyword + semantic), reranking, chunk overlap
   - Evaluation: human eval on relevance, LLM-as-judge for answer quality
3. **Latency (P99 ¡ 5 seconds for user-facing)**
   - Breakdown: Retrieval (100ms) + Reranking (200ms) + LLM (3-4s)
   - Optimization: parallel retrieval, streaming LLM responses

10.3.3 3. Autocomplete Services (Avatar, Studio, Brand)

Your Role: Sole engineer (Avatar: 2020, Studio: 2021, Brand: 2025 as mentor)
Problem Solved:

- Users need instant query suggestions as they type
- Reduce typos, guide users to popular content
- Different domains: Avatar Marketplace, Studio (creator tools), Brand partnerships

Technical Approach:

1. **Trie-Based Data Structure**
   - Prefix tree for fast prefix matching (O(k) for query length k)
   - Precomputed suggestions for top 10K prefixes
   - In-memory (Redis) for P99 ¡ 10ms latency
2. **Suggestion Ranking**
   - Signals: query frequency, result CTR, recency
   - Personalization: user history, age group
   - A/B testing framework for ranking changes
3. **Offline Pipeline (Daily)**
   - Aggregate query logs (Hive)
   - Filter: remove PII, profanity, low-frequency queries
   - Build trie, compute scores, publish to Redis

Scale Metrics:

- Avatar Autocomplete: All Roblox users (∼70M DAU)
- Studio Autocomplete: Roblox creators (∼2M+ monthly)
- P99 latency: 10ms (critical for user experience)
- Update frequency: Daily (captures trending queries)

10.3.4 4. Spelling Correction Service

Your Role: Sole engineer (2022)

Problem Solved:

- User queries contain typos (”Brookhven” → ”Brookhaven”)
- Children are primary users (higher typo rate)
- Poor spelling = zero results = bad user experience

Technical Approach:

1. **Dictionary-Based Correction**
   - Dictionary: all valid game titles, popular queries
   - Algorithm: Edit distance (Levenshtein), phonetic similarity (Soundex)
   - Candidate generation: words within edit distance 2
2. **Context-Aware Ranking**
   - Given ”Brookhven”, candidates: [”Brookhaven”, ”Brookhaven RP”, ”Brook Haven”]
   - Rank by: query frequency, result availability, user context
3. **Production Deployment**
   - Microservice (bedev2), REST API
   - Integration: Search (query rewriting), Autocomplete (suggestion generation)
   - 3 production adoptions: Avatar Marketplace, Studio, Community

10.3.5 5. Game Clickbait Detection (Covered in Part 1)

Search Quality Signal:

- Clickbait score (0-1) stored in Frost feature table
- Used by ranking model as quality signal (demote low-quality copycats)
- Real-time updates (10-second latency from game publish to ranking)

10.4 Key Search Topics: Deep Dive

10.4.1 Indexing Pipeline

Roblox Game Indexing Flow:

```
Game Created/Updated Event (SNS)
    |
[1] Document Builder Service
    |-- Fetch metadata: title, description, creator, genre, etc.
    |-- Generate features: engagement stats, quality scores
    |-- Enrich: clickbait score (your system), safety flags
    \-- Output: JSON document
    |
[2] Elasticsearch Indexing
    |-- Index mapping: title (text), description (text), genre (keyword)
    |-- Sharding: by universe_id (game ID)
    |-- Replica sets: 3x for availability
    \-- Refresh interval: 1 second (near real-time)
    |
[3] Vector Indexing (RSS)
    |-- Generate CLIP embedding for game thumbnail
    |-- Index in RSS (separate collection: "game_thumbnails")
    \-- Async (doesn't block Elasticsearch indexing)
    |
[4] Feature Store (Frost)
    |-- Compute ranking features (engagement, quality, freshness)
    |-- Publish to Frost tables
    \-- Used by ranking service at query time
```

Indexing Challenges:

1. **Freshness vs Consistency**
   - Problem: Game metadata changes frequently (title updates, thumbnail swaps)
   - Solution: Near real-time indexing (1-second refresh) vs eventual consistency
   - Trade-off: Slight staleness acceptable for better performance
2. **Schema Evolution**
   - Problem: Adding new fields (e.g., clickbait_score) requires reindexing

- Solution: Backward-compatible mappings, gradual rollout
- Zero-downtime reindexing: alias swap pattern

3. **Scale (Millions of Games)**
   - Problem: Full reindex takes hours/days
   - Solution: Incremental indexing (only changed documents)
   - Backfill: parallel workers, checkpointing for recovery

10.4.2 Query Understanding

Roblox Query Understanding Pipeline:

1. **Normalization**
   - Lowercase, trim whitespace
   - Remove special characters (emojis, punctuation)
   - Unicode normalization (é → e)
2. **Spelling Correction (Your Service)**
   - Detect misspellings: edit distance, phonetic similarity
   - Suggest corrections: "Brookhven" → "Brookhaven"
   - User experience: "Did you mean..." vs auto-correct
3. **Intent Detection**
   - **Navigational:** User wants specific game ("Adopt Me")
   - **Exploratory:** User wants category ("scary games", "obby")
   - **Transactional:** User wants to create/edit ("new game template")
4. **Query Expansion**
   - Synonyms: "scary" → ["horror", "spooky", "creepy"]
   - Related terms: "obby" → ["obstacle course", "parkour"]
   - Learned from user behavior (query reformulations)

Challenge: Roblox-Specific Terminology

- "Obby" = obstacle course (Roblox slang, not in standard dictionaries)
- "Brookhaven RP" = roleplay (RP is domain-specific abbreviation)
- Solution: Custom dictionary, query log mining, creator tagging

10.4.3 Ranking & Relevance

Ranking Signal Categories:

1. **Textual Relevance (Traditional IR)**
   - BM25 score (keyword match in title, description)
   - TF-IDF (term frequency, inverse document frequency)
   - Exact match boost (query = title → higher score)
2. **Engagement Signals**
   - CTR (click-through rate): clicks / impressions
   - Play time: average session duration
   - Favorites: user saves to favorites list
   - Retention: users return to game repeatedly
3. **Quality Signals**
   - Clickbait score: 0-1 (your system, lower = higher quality)
   - Creator reputation: established vs new creator
   - Safety flags: age-appropriate, policy violations
   - User ratings: thumbs up/down
4. **Freshness Signals**
   - Publish date: new games get temporary boost
   - Update recency: recently updated games prioritized
   - Decay function: boost diminishes over time
5. **Personalization Signals**
   - User history: games played, genres preferred
   - Social: friends' games, group memberships
   - Age appropriateness: filter by user's age group
   - Language: localization, regional preferences

Learning-to-Rank (LTR) Model:

- **Model Type:** Gradient-boosted decision trees (XGBoost/LightGBM)
- **Training Data:** User interactions (clicks, play sessions, favorites)
- **Features:** ∼50-100 features across categories above
- **Objective:** Maximize engagement (play time, retention) while maintaining quality
- **Evaluation:** Offline (NDCG, MRR) + Online (A/B tests on CTR, play time)

Trade-Off: Engagement vs Quality

- **Problem:** Clickbait games have high CTR (by design) but low quality
- **Naive approach:** Rank purely by CTR → clickbait dominates
- **Solution:** Multi-objective optimization
  - Objective 1: Maximize engagement (CTR, play time)
  - Objective 2: Maximize quality (low clickbait score, high ratings)
  - Weighted combination: $\alpha \cdot engagement + (1 - \alpha) \cdot quality$
- **Your contribution:** Clickbait score as quality signal (part of Objective 2)

10.4.4 Semantic Search (CLIP + RSS)

Why Semantic Search?

- Keyword search fails for visual/conceptual queries:
  - "red hoodie" (visual attribute, not in text metadata)
  - "anime style" (artistic style, hard to describe in keywords)
  - "games like Brookhaven" (conceptual similarity, not keyword match)

Semantic Search Pipeline:

```
User Query: "anime style games"
    |
[1] Query Embedding
    |-- CLIP text encoder: query -> 512-dim vector
    |-- Normalize: L2 norm = 1
    \-- Output: query embedding
    |
[2] Vector Search (RSS)
    |-- k-NN search: find top-k nearest game thumbnail embeddings
    |-- Distance metric: cosine similarity (dot product for normalized vectors)
    |-- Index: HNSW (approximate nearest neighbor)
    \-- Returns: Top 500 candidates with similarity scores
    |
[3] Hybrid Fusion
    |-- Keyword results (Elasticsearch): Top 1000
    |-- Semantic results (RSS): Top 500
    |-- Fusion method: Reciprocal Rank Fusion (RRF)
    \-- Output: Top 200 candidates for ranking
```

Hybrid Search Trade-Offs:
CLIP Model Details:

- **Base model:** OpenAI CLIP (ViT-B/32 or ViT-L/14)
- **Fine-tuning:** Considered but not done in V1 (off-the-shelf worked well)
- **Future:** Fine-tune on Roblox-specific data (game thumbnails + user clicks)

Approach	Strengths	Weaknesses
Keyword Only	Precise for exact matches, fast	Misses conceptual similarity
Semantic Only	Finds conceptual matches	Misses exact matches, slower
Hybrid (Keyword + Semantic)	Best of both worlds	Increased complexity, tuning needed

10.4.5 Personalization

Cold Start Problem (New Users):

- **Problem:** No user history → can't personalize
- **Solution:**
  - Default to popularity (most played games globally)
  - Age-based filtering (show age-appropriate content)
  - Gradual learning: collect clicks/plays, update profile

Cold Start Problem (New Games):

- **Problem:** No engagement data → can't rank well
- **Solution:**
  - Freshness boost: new games get temporary ranking boost
  - Creator reputation: leverage creator's past game quality
  - Content-based features: genre, description, thumbnail quality

Privacy Considerations:

- User history stored with consent (privacy policy)
- Anonymized for model training (no PII)
- User control: "Clear history" option

10.4.6 Evaluation & Metrics

Offline Metrics (Pre-Deployment):

1. **NDCG (Normalized Discounted Cumulative Gain)**
   - Measures ranking quality (higher-ranked relevant results = better)
   - NDCG@10: focus on top 10 results (most important for users)
2. **MRR (Mean Reciprocal Rank)**
   - Position of first relevant result (1/rank)
   - Good for navigational queries ("Adopt Me" → should be #1)

3. **Precision@k / Recall@k**
    - Precision@10: % of top 10 results that are relevant
    - Recall@10: % of relevant results found in top 10

Online Metrics (A/B Testing):

1. **CTR (Click-Through Rate)**
    - Clicks / Impressions (how often users click results)
    - Broken down by position (position 1 vs position 10)
2. **Play Time**
    - Average session duration after search
    - Measures result quality (did user find what they wanted?)
3. **Zero Result Rate**
    - % of queries with no results
    - Target: ¡5% (spelling correction helps)
4. **Query Reformulation Rate**
    - % of users who search again after initial query
    - High rate = initial results unsatisfactory

Human Evaluation:

- Periodic manual review of top queries (top 1000)
- Raters assess: relevance, quality, diversity
- Identify edge cases, systematic errors
- Feed back into model training, feature engineering

10.5 Search Architecture Trade-Offs

10.5.1 Latency vs Quality

Component	Latency Impact	Quality Impact
Retrieval Depth	More candidates = slower	More candidates = better recall
Ranking Model Complexity	Deeper trees = slower	More features = better accuracy
Semantic Search	Vector search adds 50-100ms	Improves conceptual matches
Personalization	User profile lookup adds 20ms	Improves relevance

Roblox's Choice:

- Target P99 latency: 500ms (acceptable for search)
- Budget breakdown: Retrieval (100ms) + Ranking (200ms) + Overhead (200ms)
- Trade-off: Accept slightly slower for better quality (user retention ¿ speed)

10.5.2 Freshness vs Consistency

- **Real-time indexing:** Game available in search within seconds
- **Eventual consistency:** Ranking features (engagement) update hourly/daily
- **Acceptable staleness:** Users don't expect instant engagement stats

10.5.3 Recall vs Precision

- **High recall (cast wide net):** Retrieve 1000 candidates, risk irrelevant results
- **High precision (narrow focus):** Retrieve 100 candidates, risk missing relevant results
- **Roblox's approach:** High recall in retrieval (1000), precision in ranking (filter to top 20)

10.5.4 Personalization vs Diversity

- **Problem:** Over-personalization creates filter bubbles
  - User only sees games similar to past plays
  - Misses discovery of new genres, creators
- **Solution:** Inject diversity
  - 70% personalized results + 30% diverse/popular results
  - Diversity constraints: no more than 2 games from same creator in top 10
  - Exploration: occasionally show random high-quality games

10.6 Potential Interview Questions & Answers

10.6.1 Q: Walk me through what happens when a user types a search query.

Answer (End-to-End Flow):

"Let me walk through a concrete example: user searches 'scary games'.

Stage 1: Query Understanding (50ms)

- Normalize: lowercase, trim spaces
- Check spelling: 'scarry' → 'scary' (my spelling service)
- Detect intent: exploratory (not navigational)
- Expand: add synonyms ['horror', 'spooky', 'creepy']

Stage 2: Retrieval (100ms)

- Keyword retrieval (Elasticsearch):
  - Match 'scary' in title, description, tags
  - BM25 scoring: games with 'scary' in title rank higher
  - Returns: Top 1000 candidates
- Semantic retrieval (RSS - my system):

  - Generate CLIP embedding for 'scary games' text
  - Vector search: find games with similar thumbnail embeddings
  - Returns: Top 500 candidates
- Hybrid fusion: Merge results using Reciprocal Rank Fusion
  - Combines keyword + semantic scores
  - Output: Top 200 candidates for ranking

Stage 3: Ranking (200ms)

- Feature generation (Frost):
  - Engagement: CTR, play time, favorites
  - Quality: clickbait score (my system), creator reputation
  - Freshness: publish date, update recency
  - Personalization: user's age group, past scary game plays
- LTR model inference:
  - XGBoost model with ∼100 features
  - Outputs relevance score per candidate
- Re-ranking & filtering:
  - Diversity: max 2 games from same creator in top 10
  - Safety: filter age-inappropriate content for this user
  - Policy: demote clickbait (score ¿ 0.8 threshold)

Stage 4: Serving (50ms)

- Check cache (Redis): popular queries cached for 5 minutes
- Format results: thumbnails, titles, metadata
- Log: query, user, results for offline analysis
- Return: Top 20 results to client

Total Latency: P99 ∼ 400-500ms (within budget)"

10.6.2 Q: How do you handle very popular queries that could overwhelm the system?

Answer:

"Popular queries like 'Adopt Me' or 'Brookhaven' can spike to thousands of QPS. We use multi-layer caching:

Layer 1: CDN Edge Cache

- Results cached at edge nodes (close to users geographically)
- TTL: 5 minutes (balance freshness vs load)

- Invalidation: on ranking model updates or game policy changes
- Benefit: Offloads 80-90% of popular query traffic

Layer 2: Application Cache (Redis)

- Full result set cached in Redis (including ranking scores)
- TTL: 1 minute (more aggressive, needs fresher data)
- Personalization: cache key includes user age group (coarse-grained)
- Benefit: Handles cache misses from Layer 1

Layer 3: Partial Result Cache

- Cache retrieval candidates (top 200) separately from ranking
- Ranking still computed per-request (for personalization)
- Benefit: Saves expensive retrieval step, allows real-time personalization

Graceful Degradation:

- If ranking service is overloaded (P99 ¿ 1s):
  - Fall back to simpler ranking (BM25 only, skip LTR model)
  - Still functional, just less optimal
- If Elasticsearch is overloaded:
  - Serve stale cache (extend TTL to 10 minutes)
  - Better than showing errors to users

Rate Limiting:

- Per-user rate limit: 10 queries/second (prevent abuse)
- Per-IP rate limit: 100 queries/second (prevent DDoS)

This multi-layer approach ensures we can handle traffic spikes without degrading user experience."

10.6.3 Q: What's your strategy for ranking results for a brand new user?

Answer:

"New users have no history, so we use a combination of defaults and rapid learning:
Initial Strategy (Session 1):

- **Age-based filtering:**
  - User age (from account signup) determines content appropriateness
  - Age 6-12: family-friendly games only
  - Age 13+: broader content, still filtered by safety policies

- **Popularity as proxy:**
  - Default to globally popular games (high play time, favorites)
  - Assumption: popular games are high-quality, good introduction
- **Diversity:**
  - Show diverse genres (roleplay, obby, tycoon, simulator)
  - Goal: learn user preferences quickly through exploration

Rapid Learning (Sessions 2-10):

- **Implicit feedback:**
  - Track clicks: which games did user click?
  - Track play time: which games did user play for ¿ 5 minutes?
  - Track favorites: which games did user save?
- **Genre preference detection:**
  - If user plays 3+ roleplay games → boost roleplay genre
  - If user plays 0 scary games → reduce scary genre
- **Creator affinity:**
  - If user plays multiple games from Creator X → boost Creator X's other games

Long-Term Personalization (Sessions 10+):

- Full personalization model (user embedding, collaborative filtering)
- Social signals: friends' games, group memberships
- Temporal patterns: play more tycoon games on weekends

Balancing Exploration vs Exploitation:

- **Exploitation:** Show games similar to past plays (80%)
- **Exploration:** Inject random high-quality games (20%)
- Prevents filter bubble, allows discovering new interests

The key is **gradual personalization**: start with safe defaults, learn quickly from early interactions, then fully personalize."

10.6.4 Q: How do you detect and fix search quality issues?

Answer:

"Search quality degrades over time (user behavior changes, new content, model drift). We use multiple detection mechanisms:

Detection Layer 1: Automated Monitoring

- **Query-level metrics:**
  - Zero result rate (alert if ¿ 10%)
  - CTR drop (alert if drops ¿ 5% week-over-week)
  - Query reformulation rate (alert if ¿ 30%)
- **System-level metrics:**
  - Latency (alert if P99 ¿ 1s)
  - Error rate (alert if ¿ 1%)

Detection Layer 2: Human Evaluation

- **Monthly audits:**
  - Sample 100 top queries (cover 50%+ of search volume)
  - Human raters assess: relevance (1-5 scale), quality, diversity
  - Compare to baseline (previous month, competitor platforms)
- **Issue categorization:**
  - Retrieval issues: relevant games not retrieved
  - Ranking issues: relevant games retrieved but ranked poorly
  - Quality issues: low-quality games ranked highly

Detection Layer 3: User Feedback

- "Report search result" button (crowdsourced quality)
- Support tickets mentioning search problems
- Social media monitoring (Twitter, Discord complaints)

Root Cause Analysis (Example: Clickbait Problem):

1. **Symptom:** CEO reports poor results for "Brookhaven"
2. **Hypothesis generation:**
   - Retrieval issue? (relevant games not found)
   - Ranking issue? (copycat games ranked too high)
   - Data quality issue? (games have misleading metadata)
3. **Investigation:**
   - Manual inspection: top 20 results for "Brookhaven"

- Finding: 15/20 are copycats with identical names/images
- Root cause: **Ranking issue** (engagement signals favor clickbait)

4. **Fix:** Clickbait detection system (my project)
   - Add quality signal to ranking (clickbait score)
   - Demote games with high clickbait scores
5. **Validation:**
   - A/B test: treatment group sees demoted clickbait
   - Metrics: CTR, play time, user satisfaction surveys
   - Result: CTR slightly down (clickbait has high CTR by design), but play time up (users find quality games)

Continuous Improvement Loop:

- Detect → Diagnose → Fix → Validate → Repeat
- Quarterly model retraining (new data, new features)
- Annual architecture review (e.g., switch to transformer-based ranking)

The key is **multi-signal detection** (automated + human + user feedback) and **rigorous validation** (A/B tests, not just intuition)."

10.6.5 Q: How would you add a new ranking signal to the system?

Answer:
"I'll walk through this using the clickbait score as a concrete example, since I actually did this:
Step 1: Signal Definition & Feasibility (Week 1)

- **Define signal:** Clickbait score (0-1, higher = more likely clickbait)
- **Hypothesis:** Demoting clickbait will improve search quality (play time, user satisfaction)
- **Feasibility check:**
  - Can we compute it? (Yes: text rules + CLIP + YOLO)
  - At what latency? (Real-time: 10 seconds for new games)
  - Coverage? (Top 30 games initially, expandable)

Step 2: Offline Pipeline Development (Weeks 2-4)

- Build detection service (covered in Part 1)
- Backfill existing games (hundreds of millions)
- Store in Frost feature table: `clickbait_games_features_v0`
- Validate: human eval on sample (95%+ precision)

Step 3: Feature Integration (Week 5)

- **Frost schema update:**
  - Add `clickbait_score` column to ranking features table
  - Ensure backward compatibility (default 0.0 for missing values)
- **Ranking service integration:**
  - Modify feature generation code to fetch `clickbait_score`
  - Add to LTR model input (new feature column)

Step 4: Model Training (Week 6)

- **Offline training:**
  - Retrain XGBoost model with new feature (100 → 101 features)
  - Use historical data (query logs, user interactions)
  - Objective: maximize play time, minimize clickbait score
- **Offline evaluation:**
  - Test set: held-out queries from last month
  - Metrics: NDCG@10, MRR, clickbait in top 10 (should decrease)
  - Result: NDCG improves +2%, clickbait in top 10 drops -40%

Step 5: Online A/B Testing (Weeks 7-9)

- **Experiment setup:**
  - Control: Current ranking (no clickbait signal)
  - Treatment: New ranking (with clickbait signal)
  - Traffic split: 95% control, 5% treatment (safe rollout)
- **Metrics tracked:**
  - Primary: Play time (should increase)
  - Secondary: CTR (may decrease, clickbait has high CTR)
  - Guardrail: Zero result rate, latency (should not regress)
- **Analysis:**
  - Play time: +5% (statistically significant)
  - CTR: -2% (acceptable trade-off)
  - User surveys: satisfaction up +3%
- **Decision:** Ship to 100%

Step 6: Gradual Rollout (Week 10)

- Ramp: 5% → 25% → 50% → 100% over 2 weeks
- Monitor for issues (latency spikes, error rate)

- Rollback plan: instant revert to control if critical issue

Step 7: Post-Launch Monitoring (Ongoing)

- Weekly metrics review (play time, clickbait in top 10)
- Monthly model retraining (incorporate new data)
- Iterate on clickbait detection (expand PAL, improve precision)

Key Lessons:

- **Validate early:** Offline eval before A/B (avoids wasted user exposure)
- **Gradual rollout:** Catch issues before full launch
- **Multiple metrics:** Don't optimize for single metric (CTR alone would favor clickbait)
- **Cross-team coordination:** Needed DSA for backfill, Game Discovery for ranking integration

"

10.6.6 Q: What's your biggest challenge with search at Roblox's scale?

Answer:

"The biggest challenge is the **tension between freshness and quality signals**.
The Problem:

- **New games** (created every second):
  - Need to surface quickly (creators expect instant visibility)
  - But have no engagement data (no CTR, play time, favorites)
  - Can't rely on quality signals (clickbait detection requires metadata)
- **Quality signals** (engagement, ratings):
  - Take time to accumulate (days/weeks)
  - New games start at a disadvantage (low ranking)
  - Creates barrier for new creators (can't break through)

Specific Example (Illustrative):

- Creator X publishes high-quality game at 10:00 AM
- Game is indexed in Elasticsearch within seconds (real-time)
- But ranking model sees:
  - Engagement: 0 clicks, 0 play time (no data yet)
  - Quality: 0 favorites, no clickbait score yet (processing delay)
  - Creator reputation: unknown (new creator)
- Result: Game ranks poorly despite being high-quality

- Creator frustrated: "My game doesn't show up in search!"

Our Mitigations (Partial Solutions):

1. **Freshness Boost:**
   - New games (published ¡ 24 hours) get temporary ranking boost
   - Decays exponentially over time (day 1: +10%, day 7: +1%)
   - Gives new games a chance to accumulate engagement
2. **Content-Based Signals (No Engagement Required):**
   - Thumbnail quality (CLIP embedding → aesthetic score)
   - Description completeness (has detailed description?)
   - Genre tagging (properly categorized?)
   - These signals available immediately
3. **Creator Reputation Transfer:**
   - If Creator X has 3 high-quality games previously, new game inherits reputation
   - Assumption: good creators make good games
   - Helps established creators, but not new ones
4. **Rapid Engagement Acceleration:**
   - Show new games in "New & Trending" section (separate from main search)
   - Users browsing this section provide early engagement signals
   - Signals feed back into main search ranking within hours

Why This Remains Hard:

- **Cold start is unsolvable:** Can't predict quality without data
- **Gaming the system:** Freshness boost incentivizes spam (publish many low-quality games)
- **Balance:** Too much freshness boost → low-quality games surface; too little → new creators can't break through

Future Directions I'd Explore:

- **Pre-launch quality prediction:**
  - Analyze game before publish (in Studio editor)
  - Predict quality from: code complexity, asset quality, playtesting data
  - Give creators feedback: "Add more detail to your description to improve searchability"
- **Multi-armed bandit:**
  - Treat new game ranking as exploration-exploitation problem
  - Dynamically adjust freshness boost based on early engagement
  - If game gets high CTR in first hour → increase boost; if low CTR → decrease boost

This challenge touches multiple areas: ranking algorithms, creator incentives, platform economics - exactly the kind of **cross-functional systems problem** I enjoy solving."

10.7 Connection to DataHub (Search Context)

Key Insight

How Your Search Experience Applies to DataHub:
"DataHub's core product is **metadata search and discovery** - finding the right dataset, understanding lineage, discovering related data. This maps directly to my search experience:

- **Roblox Game Search ↔ DataHub Dataset Search**
  - Games have metadata (title, description, creator) ↔ Datasets have metadata (schema, owner, tags)
  - Users search for games (keywords, concepts) ↔ Data scientists search for datasets (column names, business context)
  - Quality signals (engagement, clickbait) ↔ Quality signals (data quality scores, usage frequency)
- **RSS (Vector Database) ↔ Semantic Dataset Search**
  - CLIP embeddings for game thumbnails ↔ Embeddings for schema/column descriptions
  - Semantic search: "anime style games" ↔ "customer transaction data"
  - Both solve: keywords alone don't capture conceptual similarity
- **RAG Platform ↔ DataHub Documentation/Lineage**
  - RAG over Roblox docs (policies, guides) ↔ RAG over dataset documentation (data dictionaries, ETL logic)
  - Retrieval: find relevant context ↔ Lineage: find upstream/downstream dependencies
  - Generation: answer questions ↔ Explain: "How is this field computed?"
- **Quality Signals (Clickbait Detection) ↔ Data Quality Signals**
  - Game quality: clickbait score, engagement, ratings ↔ Dataset quality: completeness, freshness, usage
  - Both require: real-time signal generation, integration with ranking
  - Both face: trade-offs between quality and discoverability

I'm excited about DataHub because it's the **same core problem** - search, discovery, quality - but in the data infrastructure domain. The techniques I've built (vector search, RAG, quality signals) directly transfer."

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)