Phoenix: Candidate-Isolated Transformer Ranking in X's Recommendation System

A Technical Analysis of the 2026 Open-Source Release

Daniel An
entopiqai@gmail.com

January 2026

Abstract

On January 19, 2026, X (formerly Twitter) open-sourced Phoenix, a Grok-based transformer recommendation system that replaced their previous feature-engineered Heavy Ranker. This paper provides a technical analysis of the release, with particular focus on a novel architectural choice: **candidate isolation**, where post candidates cannot attend to each other during transformer inference. We analyze the implications of this design for score consistency, caching efficiency, and ranking fairness. We compare the 2023 and 2026 architectures, enumerate the 19 engagement signals now predicted, and discuss what remains undisclosed—specifically the weight parameters that combine these signals into final ranking scores. Our analysis suggests that while X has provided unprecedented transparency into recommendation system architecture, the opacity of learned weights and training data limits reproducibility and independent auditing of algorithmic amplification patterns.

1 Introduction

Social media recommendation algorithms determine what billions of users see daily, yet their inner workings remain largely opaque. X's decision to open-source their recommendation system—first in March 2023 with the Heavy Ranker [1], and now in January 2026 with Phoenix—represents a significant step toward algorithmic transparency.

Note: All code references in this paper have been verified against the public repository at `github.com/xai-org/x-algorithm` *(commit* `aaa167b`*, January 19, 2026). Repository statistics at time of analysis: 6.1k stars, 1k forks, Apache 2.0 license.*

The 2026 release marks a fundamental architectural shift: from a feature-engineered system with explicit, published weights to a transformer-based system where relevance is learned end-to-end. This transition reflects broader trends in recommendation systems toward neural approaches, but X's implementation includes several distinctive design choices worthy of analysis.

This paper makes the following contributions:

- A comparative analysis of X's 2023 Heavy Ranker and 2026 Phoenix architectures
- Technical examination of the **candidate isolation** attention pattern, a novel approach ensuring batch-independent scoring
- Enumeration and analysis of the 19 engagement signals predicted by Phoenix
- Discussion of transparency limitations: what remains undisclosed and why it matters

2 Background: The 2023 Heavy Ranker

2.1 Architecture Overview

The 2023 release [1] exposed a traditional machine learning pipeline with hundreds of hand-engineered features. The system predicted multiple engagement types and combined them using explicit weights:

Table 1: Published 2023 Heavy Ranker Weights

Engagement Type	**Weight**
Reply engaged by author	75.0
Reply	13.5
Profile click (with engagement)	12.0
Good click (reply/like in thread)	11.0
Good click v2 (2+ min dwell)	10.0
Retweet	1.0
Favorite (like)	0.5
Video playback 50%	0.005
Negative feedback v2	-74.0
Report	-369.0

The scoring formula was straightforward:

$$\text{Score} = \sum_i w_i \cdot P(\text{engagement}_i) \tag{1}$$

This transparency enabled researchers and creators to understand amplification dynamics. The 75x weight on "reply engaged by author" versus 0.5x on likes revealed X's explicit preference for conversation-generating content.

2.2 Feature Engineering Complexity

The Heavy Ranker relied on extensive feature engineering:

- Social graph features (follower relationships, community embeddings)
- Content features (media type, post age, hashtags, URLs)
- User features (account age, verification status, historical engagement rates)
- Interaction features (previous user-author engagement history)

This approach required constant maintenance as user behavior evolved and created a complex data pipeline spanning multiple services.

3 Phoenix: Architecture Analysis

3.1 System Overview

Phoenix operates as a two-stage recommendation pipeline:

1. **Retrieval Stage**: A two-tower model narrows millions of candidates to approximately 1,000 using approximate nearest neighbor (ANN) search
2. **Ranking Stage**: A Grok-based transformer scores and orders the retrieved candidates

The system draws candidates from two sources:

- **Thunder**: In-network posts from followed accounts, served from an in-memory store with sub-millisecond latency. Thunder uses DashMap (a concurrent hashmap) for lock-free access, with three separate timeline indices: original posts, secondary posts (replies/retweets), and video posts.
- **Phoenix Retrieval**: Out-of-network posts discovered through embedding similarity search using a two-tower model with L2-normalized embeddings and dot-product similarity.

Before reaching the Phoenix ranker, candidates pass through 13 filters including muted keyword filtering, conversation deduplication, author social graph filtering, and previously-seen post removal.

3.2 The Transformer Architecture

X states that the transformer implementation is "ported from the Grok-1 open source release by xAI, adapted for recommendation system use cases." The `TransformerConfig` in `grok.py` reveals key architectural choices:

- **Grouped Query Attention**: Separate `num_q_heads` and `num_kv_heads` parameters indicate GQA for memory efficiency
- **RoPE**: Rotary positional embeddings for position encoding
- **RMSNorm**: Root mean square normalization (vs. LayerNorm)
- **Widening factor**: FFN width multiplier (default 4.0)

Key characteristics include:

- **Input**: User embedding, engagement history embeddings, candidate post embeddings
- **Output**: Probability distributions over 19 engagement types for each candidate
- **Special attention masking**: Candidates cannot attend to each other

The elimination of hand-engineered features is explicit in X's documentation: "We have eliminated every single hand-engineered feature and most heuristics from the system. The Grok-based transformer does all the heavy lifting."

3.3 Candidate Isolation: A Novel Design Choice

The most technically interesting aspect of Phoenix is its attention masking strategy. During inference, the attention mask is structured such that:

- All tokens can attend to user context (user embedding + engagement history)
- Candidate tokens **cannot** attend to other candidate tokens
- Each candidate can only see the user context, not its ranking competitors

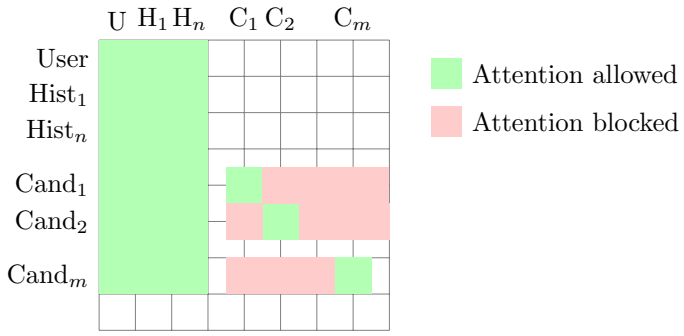

Figure 1: Candidate Isolation Attention Mask. Candidates can attend to user context but not to each other.

3.3.1 Implications of Candidate Isolation

This design has several important consequences:

The attention mask is implemented in `phoenix/grok.py` via the `make_recsys_attn_mask` function:

Listing 1: Attention mask creation (from grok.py)

```
def make_recsys_attn_mask(
    seq_len: int,
    candidate_start_offset: int,
    dtype: jnp.dtype = jnp.float32,
) -> jax.Array:
    """Create attention mask for recommendation system inference.

    Creates a mask where:
    - Positions 0 to candidate_start_offset-1 (user+history): causal
      attention
    - Positions candidate_start_offset onwards (candidates): can attend to
      user+history and themselves (self-attention), but NOT to other
         candidates

    This ensures each candidate is scored independently based on
    user+history context.
    """
```

The implications of this design are significant:

1. Score Consistency: A candidate's score is independent of which other candidates appear in the same batch. The score for post p given user u is deterministic:

$$\text{Score}(p|u) = f_\theta(\text{embed}(p), \text{context}(u)) \tag{2}$$

This contrasts with architectures where candidates can attend to each other, where:

$$\text{Score}(p|u, \mathcal{C}) = f_\theta(\text{embed}(p), \text{context}(u), \{\text{embed}(c) : c \in \mathcal{C}\}) \tag{3}$$

2. Caching Efficiency: Batch-independent scores enable aggressive caching. Once scored, a post's relevance to a user can be cached without invalidation when other candidates change.

3. Ranking Fairness: No candidate is advantaged or disadvantaged by its batch neighbors. This prevents subtle biases where high-quality posts might suppress or boost nearby content.

4. Computational Trade-off: The model cannot learn cross-candidate patterns (e.g., "show diverse topics" or "don't show three videos in a row"). Such diversity must be enforced post-hoc through separate scorers. X implements this via the Author Diversity Scorer (`author_diversity_scorer.rs`), which uses a decay-based multiplier: scores for repeated authors are attenuated by $(1.0 - \text{floor}) \times \text{decay\_factor}^{\text{position}} + \text{floor}$, where position tracks how many times the author has appeared.

3.4 Engagement Signals: 19 Predicted Outcomes

Phoenix predicts probability distributions over 19 engagement types, a significant expansion from the 2023 system:

Table 2: Phoenix Engagement Predictions (verified from `runners.py`)

Index	Positive Signals	Index	Negative Signals
0	p_favorite_score	14	p_not_interested_score
1	p_reply_score	15	p_block_author_score
2	p_repost_score	16	p_mute_author_score
3	p_photo_expand_score	17	p_report_score
4	p_click_score		
5	p_profile_click_score		
6	p_vqv_score		
7	p_share_score		
8	p_share_via_dm_score		
9	p_share_via_copy_link_score		
10	p_dwell_score		
11	p_quote_score		
12	p_quoted_click_score		
13	p_follow_author_score		
18	p_dwell_time [continuous]		

Notable additions since 2023:

- **Share disaggregation**: DM shares and copy-link shares are now separate signals, enabling the model to distinguish private sharing behavior
- **VQV score**: Video quality view metric (likely completion rate or quality-weighted views)
- **Quoted click**: Engagement with quote tweets specifically
- **Continuous dwell time**: Not just binary dwell, but predicted time spent

4 What Remains Undisclosed

4.1 The Weight Black Box

The 2023 release published explicit weights (Table 1). The 2026 release does not. The `home-mixer/scorers/weight` file exists in the repository, and the ranking output in `phoenix/runners.py` shows the structure:

Listing 2: Ranking output structure (from runners.py)

```
return RankingOutput(
    scores=probs,
    ranked_indices=ranked_indices,
    p_favorite_score=probs[:, :, 0],
    p_reply_score=probs[:, :, 1],
    p_repost_score=probs[:, :, 2],
    # ... indices 3-17 for other engagement types
    p_dwell_time=probs[:, :, 18],
)
```

However, the actual weight values that combine these 19 probability scores into a final ranking score are not disclosed. X's documentation states: "Positive actions (like, repost, share) have positive weights. Negative actions (block, mute, report) have negative weights"—but the specific values are absent.

4.2 Transformer Weights and Training Data

Beyond the scoring weights, the transformer model weights themselves are not released. We receive the architecture but not the trained parameters. This means:

- The learned representations are unknown
- What the model considers "relevant" cannot be directly inspected
- Training data composition and any debiasing procedures are opaque

4.3 Implications for Algorithmic Auditing

The architecture-without-weights release pattern limits independent auditing:

- Researchers cannot reproduce ranking decisions
- Bias analysis requires black-box probing rather than weight inspection
- Content creators cannot optimize with certainty

This represents a middle ground between full opacity and full transparency—structurally transparent but parametrically opaque.

5 Related Work: Production Recommendation Systems

Phoenix exists within a broader ecosystem of production recommendation systems. We compare its architecture to other publicly documented systems to contextualize its design choices.

5.1 YouTube (Covington et al., 2016)

YouTube's seminal recommendation paper [4] established the two-stage paradigm (candidate generation + ranking) that Phoenix follows. Key architectural choices:

- **Two-stage pipeline**: Candidate generation reduces millions of videos to hundreds; ranking network scores these candidates

- **Watch time optimization**: The ranking model predicts expected watch time via weighted logistic regression, not click probability
- **Feature architecture**: User watch history and search queries are averaged into fixed-width embeddings; candidate videos are scored independently
- **Approximate nearest neighbor**: Serving uses ANN lookup (similar to Phoenix's two-tower retrieval)

Critically, YouTube's 2016 system scores candidates *independently*—each video is scored based on user context without attending to other candidates in the batch. Phoenix's candidate isolation achieves a similar property but within a transformer architecture, where cross-candidate attention would be the default.

5.2 TikTok/ByteDance Monolith (Liu et al., 2022)

ByteDance's Monolith system [5], documented in their 2022 RecSys workshop paper, emphasizes **real-time online training**—a capability absent from Phoenix's architecture:

- **Collisionless embedding tables**: Cuckoo hashmaps ensure unique ID representations, avoiding the embedding collision problem common in large-scale systems
- **Online training**: Model parameters update with minute-level latency from user feedback, capturing trending content and shifting interests
- **Sparse feature optimization**: Expirable embeddings and frequency filtering reduce memory footprint for long-tail entities
- **DeepFM architecture**: Uses factorization machines for sparse feature interaction, not transformers

Unlike Phoenix, Monolith prioritizes *temporal responsiveness* over architectural elegance. Phoenix's transformer approach requires batch training and periodic deployment; Monolith continuously adapts. This represents a fundamental design tradeoff between model sophistication and real-time adaptation.

5.3 Instagram (Meta, 2025)

Meta's engineering blog [6] reveals that Instagram operates over 1,000 ML models across different surfaces (Feed, Stories, Reels, Explore), each with multi-stage funnels:

- **Multi-surface architecture**: Unlike Phoenix's unified ranker, Instagram uses separate models for each content type with surface-specific optimization
- **Three-stage funnel**: Sourcing (retrieval) $\rightarrow$ Early-Stage Ranking (ESR) $\rightarrow$ Late-Stage Ranking (LSR), operating on progressively fewer candidates
- **Model stability metric**: Calibration and normalized entropy (NE) define model health, enabling automated quality monitoring
- **Engagement signals**: Watch time, likes per reach, and sends per reach (DM shares) are primary ranking factors for Reels

Instagram's scale (1,000+ models) contrasts with Phoenix's apparent single-ranker design. The tradeoff: Phoenix achieves architectural simplicity at the cost of surface-specific optimization.

5.4 Architectural Comparison

Table 3: Production Recommendation System Comparison

Property	Phoenix	YouTube	Monolith	Instagram
Core architecture	Transformer	DNN	DeepFM	Multi-model
Candidate attention	Isolated	Independent	N/A	Unknown
Training paradigm	Batch	Batch	Online	Batch
Retrieval method	Two-tower	ANN	ANN	Multi-stage
Public documentation	Full arch.	Paper	Paper	Blog
Weights disclosed	No	No	No	No

5.5 The Novelty of Candidate Isolation

Among documented production systems, Phoenix's explicit candidate isolation is unique. YouTube and traditional systems achieve score independence through pointwise scoring (each item scored separately). Transformer-based systems typically allow candidates to attend to each other, enabling relative comparisons but creating batch-dependent scores.

Phoenix's `make_recsys_attn_mask()` function represents a deliberate architectural choice to preserve score independence within a transformer framework. To our knowledge, this specific pattern—allowing candidates to attend to user context but not to each other—has not been previously documented in production recommendation literature.

6 Comparative Analysis: 2023 vs 2026

Table 4: X Platform Architectural Evolution

Aspect	2023 Heavy Ranker	2026 Phoenix
Core model	Gradient boosted trees	Grok transformer
Feature engineering	Extensive (100s of features)	Eliminated
Engagement signals	10	19
Weights published	Yes	No
Candidate interaction	N/A (pointwise)	Isolated (cannot attend)
Retrieval	SimClusters + GraphJet	Two-tower + ANN
Caching potential	Limited	High (score independence)
Interpretability	High (explicit weights)	Low (learned)

7 Implications for Creators and Researchers

7.1 The Death of Weight Optimization

In 2023, creators could reason: "Replies that get author responses are worth 150x a like (75.0 / 0.5), so I should post content that sparks conversation and then engage with replies."

In 2026, such calculations are impossible. The transformer learns user-specific relevance from engagement history. What works for reaching User A may not work for User B.

7.2 Research Directions

The Phoenix release enables several research directions:

1. **Attention pattern analysis**: Studying what the model attends to in engagement history
2. **Candidate isolation effects**: Comparing isolated vs. cross-attending transformer RecSys
3. **Black-box auditing**: Probing techniques to infer effective weights from observed rankings
4. **Reproducibility studies**: Reimplementing the architecture with different training data

8 Future Work

This analysis opens several avenues for follow-on research that could significantly strengthen our understanding of Phoenix and similar systems:

8.1 Empirical Weight Inference via Black-Box Probing

While Phoenix does not disclose scoring weights, it may be possible to *infer* effective weights through systematic black-box experimentation:

- **Controlled content experiments**: Create test accounts and post controlled content varying single engagement dimensions (e.g., posts designed to elicit replies vs. likes vs. reposts)
- **Impression tracking**: Measure reach and ranking position across systematic variations
- **Statistical inference**: Use regression analysis on observed rankings to estimate effective signal weights

This approach faces challenges: X's terms of service may prohibit systematic probing, the algorithm is personalized per-user making generalization difficult, and sample sizes needed for statistical significance may require months of data collection. However, such empirical studies would provide the first quantitative characterization of Phoenix's effective behavior.

8.2 Formal Analysis of Candidate Isolation Properties

The candidate isolation design warrants rigorous theoretical analysis:

- **Score consistency**: Prove that isolated scoring guarantees identical scores for the same candidate across different batch compositions
- **Fairness implications**: Analyze whether isolation prevents certain forms of positional bias (e.g., a high-quality post being suppressed by appearing alongside viral content)
- **Information-theoretic bounds**: Compare the information available to isolated vs. cross-attending transformers—what discriminative power is lost by preventing candidate comparison?
- **Cacheability formalization**: Quantify the cache hit rate improvements enabled by score independence under realistic content arrival distributions

Synthetic experiments comparing isolated vs. cross-attending architectures on controlled datasets could isolate the effects of this design choice on ranking quality, computational efficiency, and fairness metrics.

8.3 Reproducibility Studies

With the architecture fully documented, researchers could reimplement Phoenix with different training data to study:

- How training data composition affects learned representations
- Whether the architecture generalizes to other social media contexts
- The sensitivity of engagement predictions to initialization and hyperparameters

9 Conclusion

X's Phoenix release represents the most detailed public documentation of a production recommendation transformer to date. The candidate isolation design is a genuinely novel contribution with implications for caching, consistency, and fairness in large-scale recommendation systems.

Compared to other documented production systems—YouTube's pointwise scoring, TikTok/ByteDance's online-trained Monolith, and Instagram's multi-model architecture—Phoenix occupies a unique position: a sophisticated transformer architecture that deliberately constrains attention to preserve score independence.

However, the release follows a pattern of structural transparency without parametric disclosure. While we can see *how* the system combines signals, we cannot see *what weights* it assigns or *what patterns* the transformer has learned. This limits the release's utility for algorithmic accountability while still providing valuable architectural insights.

The shift from explicit weights to learned relevance marks a broader industry trend. As recommendation systems become more neural, the question of how to maintain algorithmic transparency becomes increasingly pressing. X's 2026 release, for all its detail, demonstrates both the possibilities and limits of open-sourcing recommendation algorithms.

References

[1] Twitter Engineering. *The Twitter Recommendation Algorithm*. GitHub Repository, March 2023. `https://github.com/twitter/the-algorithm`

[2] X Engineering. *X Algorithm: Phoenix Recommendation System*. GitHub Repository, January 2026. `https://github.com/xai-org/x-algorithm`

[3] xAI. *Grok-1: Open Source Large Language Model*. GitHub Repository, March 2024. `https://github.com/xai-org/grok-1`

[4] Covington, P., Adams, J., & Sargin, E. *Deep Neural Networks for YouTube Recommendations*. Proceedings of the 10th ACM Conference on Recommender Systems (RecSys), 2016.

[5] Liu, Z., Zou, L., Zou, X., Wang, C., Zhang, B., Tang, D., Zhu, B., Zhu, Y., Wu, P., Wang, K., & Cheng, Y. *Monolith: Real Time Recommendation System With Collisionless Embedding Table*. ORSUM Workshop @ ACM RecSys, 2022.

[6] Levis, L., Ma, S., & Nava, E. *Journey to 1000 Models: Scaling Instagram's Recommendation System*. Meta Engineering Blog, May 2025. `https://engineering.fb.com/2025/05/21/production-engineering/journey-to-1000-models-scaling-instagrams-recommendation-system/`

[7] Naumov, M., et al. *Deep Learning Recommendation Model for Personalization and Recommendation Systems*. arXiv:1906.00091, 2019.

[8] Kang, W. & McAuley, J. *Self-Attentive Sequential Recommendation*. IEEE ICDM, 2018.

[9] Chen, Q., Zhao, H., Li, W., Huang, P., & Ou, W. *Behavior Sequence Transformer for E-commerce Recommendation in Alibaba*. DLP-KDD Workshop, 2019.