
X For You Feed Algorithm

Official repository README and August 2026 release documentation

Contents

0.1	Table of Contents	1
0.2	Latest Updates	2
0.2.1	August 14th, 2026	2
0.2.2	August 13th, 2026	2
0.3	Overview	2
0.4	System Architecture	3
0.4.1	Request Path	3
0.4.2	Labeling Path	3
0.5	Components	5
0.5.1	Home Mixer and Candidate Pipeline	5
0.5.2	Candidate Sources	5
0.5.3	Retrieval Index	5
0.5.4	Ranking	5
0.5.5	Content Understanding	5
0.5.6	Visibility Filtering	6
0.6	How It Works	6
0.6.1	Scoring and Ranking	6
0.6.2	Filtering	7
0.7	Experiments and Configuration	7
0.8	What's not in this repo?	8
0.8.1	Deployment-related code	8
0.9	Under the Hood Label Transparency Tool	8
0.10	Key Design Decisions	8
0.10.1	1. Multi-Action Prediction	8
0.10.2	2. Candidate Isolation in Ranking	8
0.10.3	3. Hash-Based Embeddings	9
0.10.4	4. Ranking and Visibility Are Separate	9
0.10.5	5. Composable Pipeline Architecture	9
0.11	License	9

This repository contains the core code that determines which posts a viewer sees in the For You feed on X. It combines in-network content (from accounts the viewer follows) with out-of-network content (discovered through ML-based retrieval and other mechanisms), filters content based on a variety of inputs, and ranks posts using a transformer model.

0.1 Table of Contents

- [Latest Updates](#)
 - [August 14th, 2026](#)
 - [August 13th, 2026](#)
- [Overview](#)
- [System Architecture](#)
 - [Request Path](#)
 - [Labeling Path](#)
- [Components](#)
- [How It Works](#)
 - [Scoring and Ranking](#)
 - [Filtering](#)
- [Experiments and Configuration](#)
- [What's not in this repo?](#)
- [Under the Hood Label Transparency Tool](#)
- [Key Design Decisions](#)
- [License](#)

0.2 Latest Updates

0.2.1 August 14th, 2026

Notable updates:

- How weights work. There's a common misconception about how weights related to actions (e.g. Like, Share, Block, Report, etc) work in ranking. The weights scale the predicted probabilities of such actions (or predicted continuous values, e.g. dwell time) — they do *not* scale the raw engagement counts, so e.g. it'd be incorrect to see that a report has 468 times higher weight than a like and conclude that e.g. "1 report cancels out 468 likes". The weights are a multiple on your own predicted probability of Liking, Reporting, etc, which is substantially driven by your own behavior. We've [added comments to the code](#) so that LLMs or people reading it are more likely to understand it correctly.
- Brazil 2026 Elections. As [announced by X](#), in accordance with Brazilian electoral law, For You now runs `Brazil2026ElectionFilter`, which removes posts from accounts reported to Brazil's Electoral Court for the 2026 election, unless the viewer explicitly follows the account. A benefit of open-source is that you can see that changes like this exist, and exactly how they work — take a [look at the code](#).

0.2.2 August 13th, 2026

This release:

- Adds key configuration parameters (including weights used to blend predicted action values into a score for a post)
- Adds code for systems that impact whether a post is filtered from the For You feed
- Replaces the Phoenix demonstration model with the code used to train the models the feed uses, as well as synthetic data generation code so one can run a proof-of-concept training run of Phoenix.

Among new systems included are:

1. Visibility filtering: [visibility-filtering/](#) determines whether to show a post, drop it, or show it behind an interstitial.
2. The systems that produce labels that drive visibility filtering's responses: rules that apply labels ([botmaker/](#), [botmaker-rules/](#), [scarecrow/](#)), models that score accounts on various dimensions ([agatha/](#), [bdsm/](#), [user-cred-v2/](#)), models that examine images and video ([media-model-proxy/](#), [clip/](#)), and enforcement ([abuse-enforcement-service/](#)).
3. Phoenix model code: [phoenix/](#) now contains code that trains and runs the model, plus synthetic data generation.
4. SimClusters: [simclusters/](#), an additional source of posts from accounts the viewer does not follow that is called in retrieval alongside Thunder and Phoenix retrieval.

This update is also paired with a new [Under the Hood](#) transparency tool that allows people to see aggregate statistics about the labels on their account and posts that can limit visibility.

0.3 Overview

The For You feed is assembled per request. Posts come from two places:

1. In-Network — [thunder/](#) keeps recent posts from the accounts a viewer follows in memory
2. Out-of-Network — [phoenix/](#) retrieval and [simclusters/](#) find posts from accounts the viewer does not follow

Both are ranked together by the same model. Phoenix reads the viewer's recent engagement history and predicts, for each post, how likely the viewer is to take each action on it. Those predictions are combined into one score using weights held in the code — see [Scoring and Ranking](#).

Two pipelines do the work. The Post Pipeline finds, ranks and filters posts. The Blending Pipeline wraps it and adds what the model does not rank: ads, Who to Follow recommendations, prompts.

Ranking sets the order. Whether a post can be shown at all is decided separately, by [visibility-filtering/](#), from the viewer's own actions such as blocks and mutes and from labels that other systems here attach to posts and accounts.

0.4 System Architecture

0.4.1 Request Path

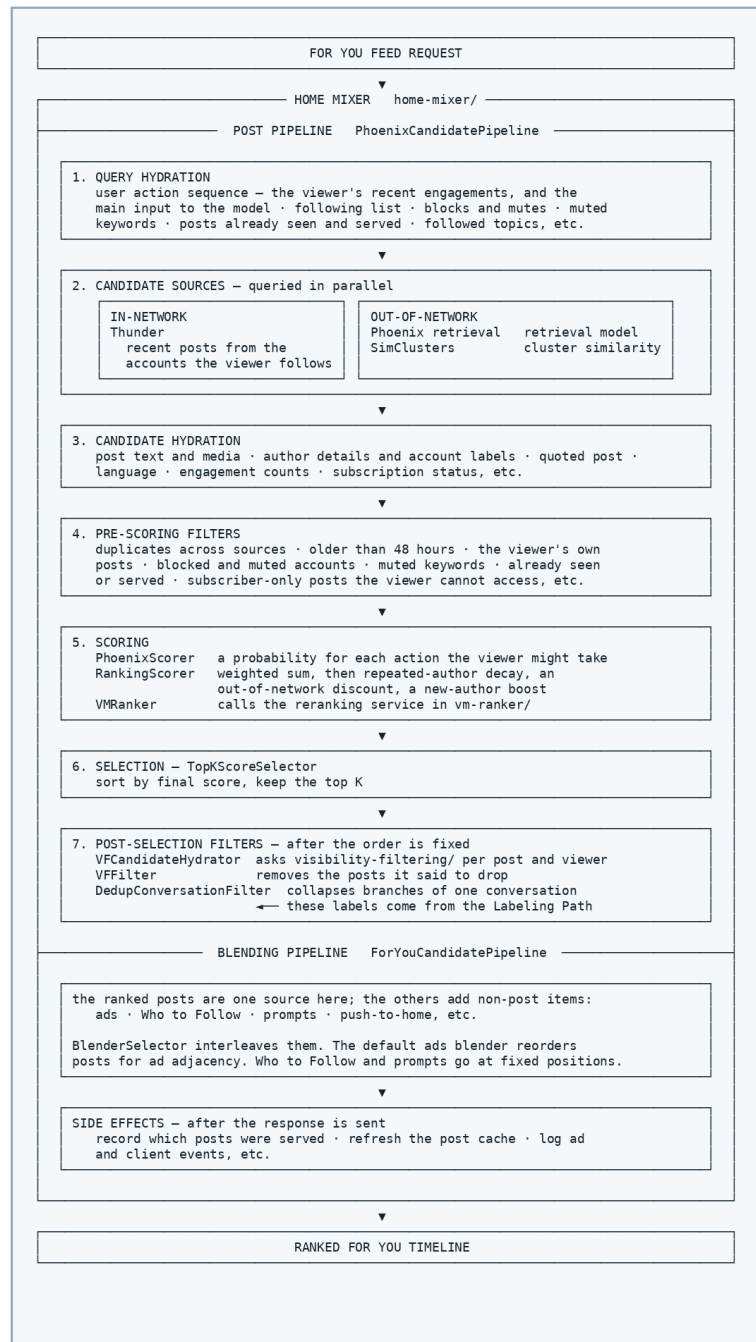


Figure 1: System architecture diagram

Stages can be switched on and off individually, with defaults in [home-mixer/params/param.rs](#) — see [Experiments and Configuration](#) for how those defaults relate to what runs in production.

0.4.2 Labeling Path

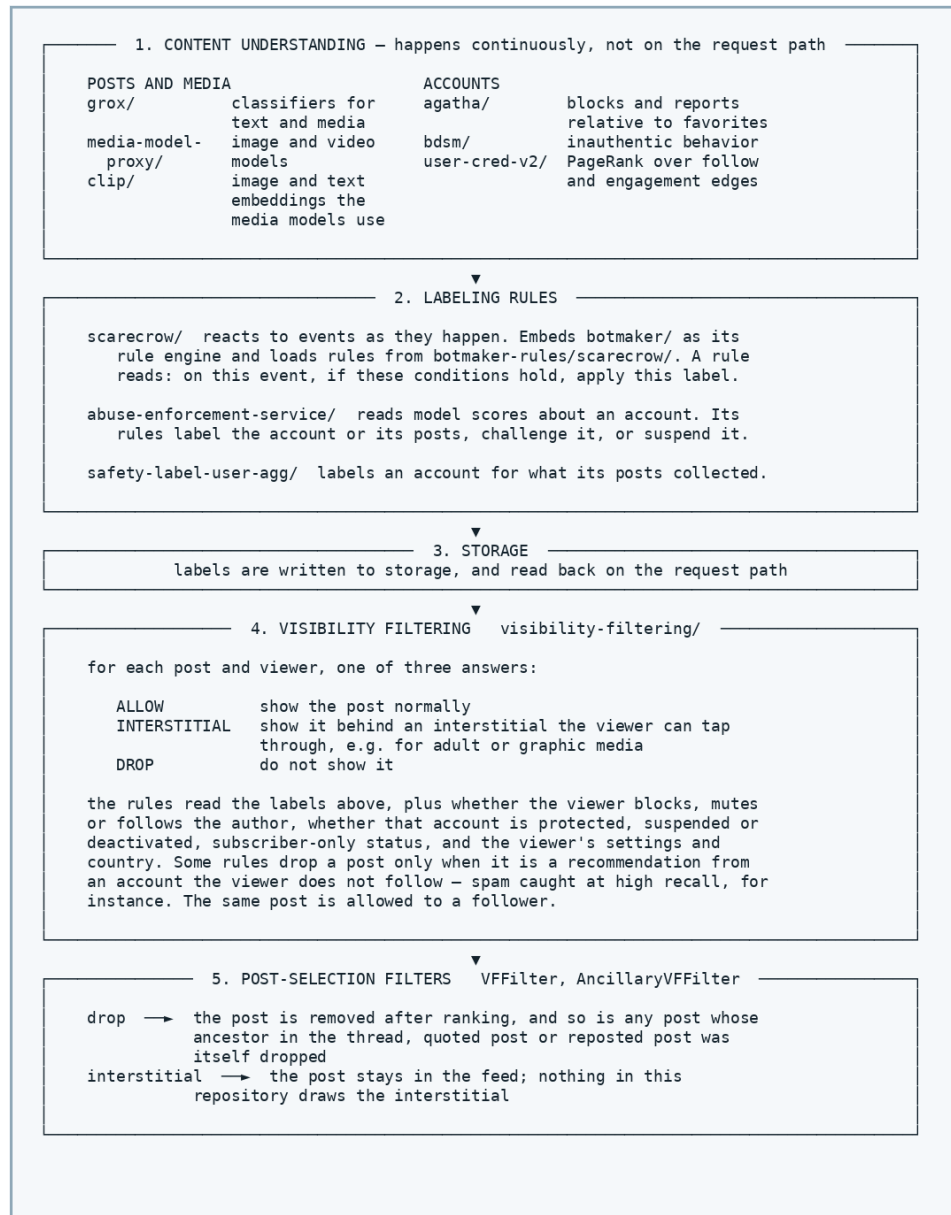


Figure 2: System architecture diagram

0.5 Components

0.5.1 Home Mixer and Candidate Pipeline

Component	What it does
home-mixer/	Builds the For You feed: the pipeline stages, the scoring weights, and calls other systems on the request path.
candidate-pipeline/	The framework <code>home-mixer</code> is built on. Defines the stage types — source, hydrator, filter, scorer, selector, side effect — and runs them, in parallel where it can.

0.5.2 Candidate Sources

Component	What it does
thunder/	Holds recent posts in memory as they are published, and returns those from the accounts a viewer follows.
phoenix/ retrieval simclusters/	Embeds the viewer and each post as vectors, and returns the posts nearest the viewer. Clusters accounts and posts by who engages with what, then uses the clusters to find candidates.

0.5.3 Retrieval Index

Component	What it does
phoenix-rankall/	Maintains the index of posts Phoenix retrieval queries, updating it as events arrive.
phoenix-rankall-strato/	The event layer that determines which index a post belongs in, consulting visibility filtering first.

0.5.4 Ranking

Component	What it does
phoenix/ ranking vm-ranker/	Predicts how likely the viewer is to take each action on each post. Training and serving code, in JAX with a Rust serving layer. The service <code>VMRanker</code> calls once posts are scored. It reorders them with a determinantal point process over their embeddings, giving up a little score for less similarity between neighbours.

0.5.5 Content Understanding

These produce the scores and labels that Visibility Filtering reads.

Component	What it does
grox/	Runs as posts are published. Classifiers for categories such as spam, adult content and violent media, plus numeric representations of a post's text and images.
media-model-proxy/ clip/	Serves the image and video models: adult content, violence and gore, hateful symbols, subject matter, and matching against known media. Trains the image and text embedding model whose media embeddings the classifiers above take as input.
agatha/	Offline batch jobs that label an account from how others respond to its posts: blocks, reports and spam reports relative to favorites, plus spam-suspension and adult-content labels.

Component	What it does
 bdsm/	Reads the sequence of actions an account takes over time to identify signs of inauthentic or abusive behavior.
 user-cred-v2/	Runs PageRank over the follow graph and engagement edges, and turns the resulting mass into a per-account score.
 adult-content/ pnsfwmedia/	Trains and calibrates a classifier for adult media. An adult-media classifier that combines CLIP media embeddings with account-level scores, including the calibrated score from agatha.

0.5.6 Visibility Filtering

Component	What it does
 visibility-filtering/ scarecrow/ botmaker/ botmaker-rules/	Determines whether a post is shown to a viewer. Rules in rules/registry.rs . Applies label rules to events as they happen. Embeds botmaker as its rule engine. That rule engine: the language rules are written in, its compiler, and its runtime. The rules scarecrow loads. To reduce the risk of gaming to circumvent these systems, some rules aren't currently in this repository.
 abuse-enforcement-service/ safety-label-user-aggr/ visibility-filtering-client/ under-the-hood/	Acts on model scores about an account rather than on events: labels it or its posts, challenges it, or suspends it. Labels an account for what its posts collected. The client callers use to reach visibility filtering, and the post safety-label types it answers with. Builds the per-account Under the Hood report: daily jobs collect the labels applied to an account and its posts, which the serving layer aggregates over a period.

0.6 How It Works

0.6.1 Scoring and Ranking

Phoenix predicts a probability for each action:

Engagement	favorite · reply · repost · quote · share · share via DM · share via copy link
Clicks	post · profile · link · photo expand · video open · quoted post
Attention	video quality view · dwell · dwell time · click dwell time · active seconds
Author	follow author
Negative	not interested · mute author · block author · report · not dwelled

RankingScorer combines them:

$$\text{Final Score} = \sum (\text{weight}_i \times P(\text{action}_i))$$

Positive actions carry positive weights, negative actions negative ones. The weights are in [home-mixer/params/param.rs](#); the arithmetic is in [home-mixer/scorers/ranking_scorer.rs](#).

There is a common misconception to be aware of about the weights: they scale the predicted probabilities (or predicted continuous values, e.g. dwell time) — they do *not* scale the raw engagement counts, so e.g. it'd be incorrect to see that a report has 468 times higher weight than a like and conclude that e.g. “1 report cancels out 468 likes”. The weights are a multiple on your own predicted probability of Liking, Reporting, etc, which is substantially driven by your own behavior.

Three adjustments follow:

- Author Diversity: each post after an author's first is multiplied by a decaying factor, down to a floor.
- Out-of-Network Discount: posts from accounts the viewer does not follow are multiplied by a factor below 1, as are replies and reposts from accounts the viewer does follow.

- New-Author Boost: posts from authors whose impressions are below a threshold are lifted toward a target position.

VMRanker then calls [vm-ranker/](#), a separate service that reorders the result.

0.6.2 Filtering

Pre-Scoring Filters ([home-mixer/filters/](#)), in order:

Filter	Removes
DropDuplicatesFilter	The same post returned by more than one source
CoreDataHydrationFilter	Posts whose text and metadata failed to load
AgeFilter	Posts older than 48 hours
SelfTweetFilter	The viewer's own posts
00NRetweetReplyFilter	Reposts and replies from accounts the viewer does not follow, and replies whose parent is missing
00NNsfwSimclustersFilter	SimClusters posts whose author is flagged for adult content, when the viewer does not follow them
RetweetDeduplicationFilter	Repeated reposts of the same post
IneligibleSubscriptionFilter	Subscriber-only posts the viewer cannot access
PreviouslySeenPostsFilter	Posts the viewer has already been shown
PreviouslySeenPostsBackupFilter	The same, from a second record of impressions
PreviouslyServedPostsFilter	Posts already served earlier in the session
MutedKeywordFilter	Posts matching the viewer's muted keywords
AuthorSocialgraphFilter	Posts from accounts the viewer blocks or mutes
VideoFilter	Video posts, when the request excludes video
TopicIdsFilter	Posts outside the requested topics, and posts in excluded topics
NewUserMinEngagementFilter	For new accounts, out-of-network posts below an engagement threshold
InventoryHoldoutFilter	A configured percentage of posts, chosen deterministically per post and viewer

Already-seen posts are handled twice over: ThunderSource is passed the list and leaves them out, the other sources are not, so their repeats are caught by the filters above.

Post-Selection Filters:

Filter	Removes
VFFilter	Posts visibility-filtering/ answered drop for
AncillaryVFFilter	Posts whose parent, quoted or reposted post was itself dropped
DedupConversationFilter	Additional branches of the same conversation

Two things to know about how the rules run:

- The first rule that answers drop ends the evaluation.
- A further set of rules applies only when the post is a recommendation from an account the viewer does not follow, and those rules can only drop — spam caught at high recall, for instance. The same post is allowed to a follower. Both sets are listed in evaluation order in [visibility-filtering/rules/registry.rs](#).

0.7 Experiments and Configuration

As we work to improve the algorithm we regularly run experiments on a small percentage of timeline traffic. Our aim is for experiments running at a notable share of traffic — e.g. 10% or more — to be visible in this repository.

To enable experimentation, many tunable values are read from a configuration system rather than written into the code. To help people understand the production defaults, we run cron scripts that set the defaults in this repository's code to be the primary production values, for example in [home-mixer/params/param.rs](#).

[docs/BIDIRECTIONAL_BOOST_CHANGE.md](#) follows a widely-discussed timeline change, exemplifying what you would see as a param value changes over time.

0.8 What's not in this repo?

We believe transparency is important for trust, and our aim is for the public to be able to understand how posts are distributed on X, so they can audit, critique or even help improve the system.

One challenge with making code that impacts post distribution public is that people could use it to try to game the system. To reduce the risk of this, there are a limited set of files not currently published in the repository, e.g.:

- Grox prompts. E.g. the j2 files with the specific LLM prompts used in Grox.
- Some botmaker rules

However, we still want the public to have insight into these systems. To accomplish that, we're piloting a new [transparency tool](#) that will show people the visibility-impacting labels that have been applied to their account and posts. This approach has multiple benefits:

- one can see the outcomes of these systems (and whether they affect their own account)
- one can see whether labels have been manually applied outside of automated systems
- one can match any labels present on their account to the code to understand if or how the visibility of their posts is affected, and critique it if desired

We believe the combination of code + transparent outputs is a powerful one for public transparency, and welcome feedback.

0.8.1 Deployment-related code

The focus of the repository is transparency into the code that affects post visibility in the For You timeline. All of the code here is inspectable, and some of the code is even designed to be runnable end-to-end — e.g. training and running the Phoenix scoring model. Where code is meant to be built and run, the relevant manifests are in the repo, e.g. [phoenix/](#) ships a Cargo workspace, a `pyproject.toml`, a [quickstart](#) and synthetic data generation, so a small model can be trained and served end-to-end. Elsewhere, code may not necessarily include build- or deployment-related files or generally self-explanatory infrastructure imports (e.g. `xai_service_runner` or `xai_kafka`). If there's anything not here that you believe would help your understanding of the algorithm, please let us know.

0.9 Under the Hood Label Transparency Tool

We're piloting a new transparency tool that lets people see aggregate statistics about the visibility-impacting labels on their account and posts. Paired with the code in this repository, we believe this gives people valuable insight into the visibility of their posts.

The tool is [available here](#) — we'll be shaping it based on your feedback and expanding availability over time. The jobs and serving code that build the report are in [under-the-hood/](#).

0.10 Key Design Decisions

0.10.1 1. Multi-Action Prediction

Rather than predicting a single “relevance” score, the model predicts probabilities for many actions. Combining them into one number is a separate, explicit step.

0.10.2 2. Candidate Isolation in Ranking

During transformer inference, candidates cannot attend to each other—only to the viewer context. This ensures the score for a post doesn't depend on which other posts are in the batch, making scores consistent and cacheable.

0.10.3 3. Hash-Based Embeddings

Both retrieval and ranking use multiple hash functions for embedding lookup, so there is no vocabulary to maintain and a new post is representable immediately.

0.10.4 4. Ranking and Visibility Are Separate

Ranking decides the order. Visibility filtering decides whether a post can be shown at all. Different services, different inputs, different rules.

0.10.5 5. Composable Pipeline Architecture

The candidate-pipeline crate provides a flexible framework for building recommendation pipelines with:

- Separation of pipeline execution and monitoring from business logic
 - Parallel execution of independent stages and graceful error handling
 - Easy addition of new sources, hydrations, filters, and scorers
-

0.11 License

Licensed under the Apache License 2.0. See [LICENSE](#).