READ MODE · ANSWER FIRST

How X For You Timeline Ranking Works

An interactive, visual explanation of How X For You Timeline Ranking Works, from setup through the complete system.

CHEAT SHEET · 7 ESSENTIAL IDEAS

The whole story in 7 lines

How X For You Timeline Ranking Works becomes easier to reason about when every stage is connected as one system.

1. Show query hydrators as concurrent branches that enrich one request object before any source fetches begin.
2. Show the source stage as a fan-in, not a quota split. Both branches merge into one pool before ranking.
3. Keep the teaching point clear: hydrators run in parallel and annotate candidates without changing candidate order or count.
4. This stage should read like a narrowing conveyor belt. Each filter takes the survivors from the previous one.
5. The critical fact is candidate isolation: each candidate can attend to the viewer and history, but not to other candidates in the same...
6. Make this feel like a score-processing pipeline, not just a sort. Each pass transforms the ordering logic a little more.
7. Teach that selection is not the same thing as serving. Visibility and conversation dedupe still trim the shortlist, while a side effect...

STAGE 1 OF 9

Setup

Every time you open X and scroll the For You tab, a massive pipeline runs behind the scenes to pick about 50 posts out of millions. This explainer will walk you through that pipeline, one stage at a time.

First, some vocabulary. Home Mixer is the name of the service that orchestrates the entire feed. Think of it as the kitchen that assembles your meal from many ingredient sources.

A candidate is any post that could potentially appear in your feed. The pipeline starts with hundreds of candidates and whittles them down to a handful.

Hydration means attaching extra information to a thin data object. A candidate starts as just an ID. Hydration fills in the author name, post text, media, and everything else needed to rank or display it.

Phoenix is the machine learning model that predicts how likely you are to interact with each candidate. It outputs probabilities for actions like "favorite", "reply", and "repost".

Here is the journey ahead. We will move through seven stages: hydrating your viewer context, sourcing candidates, enriching them, filtering bad fits, scoring with Phoenix, blending scores, and finally serving the timeline to your device. Let us begin with context hydration.

STAGE 2 OF 9

Context Hydration

When you tap the For You tab, a feed request arrives at Home Mixer carrying your viewer ID, locale, and a few client flags. That is the entire starting point. It is far too little information to personalize a feed.

A user action hydrator kicks off immediately. It fetches your recent likes, replies, reposts, and clicks, then compresses them into a sequence the ranking model will use as a taste signal.

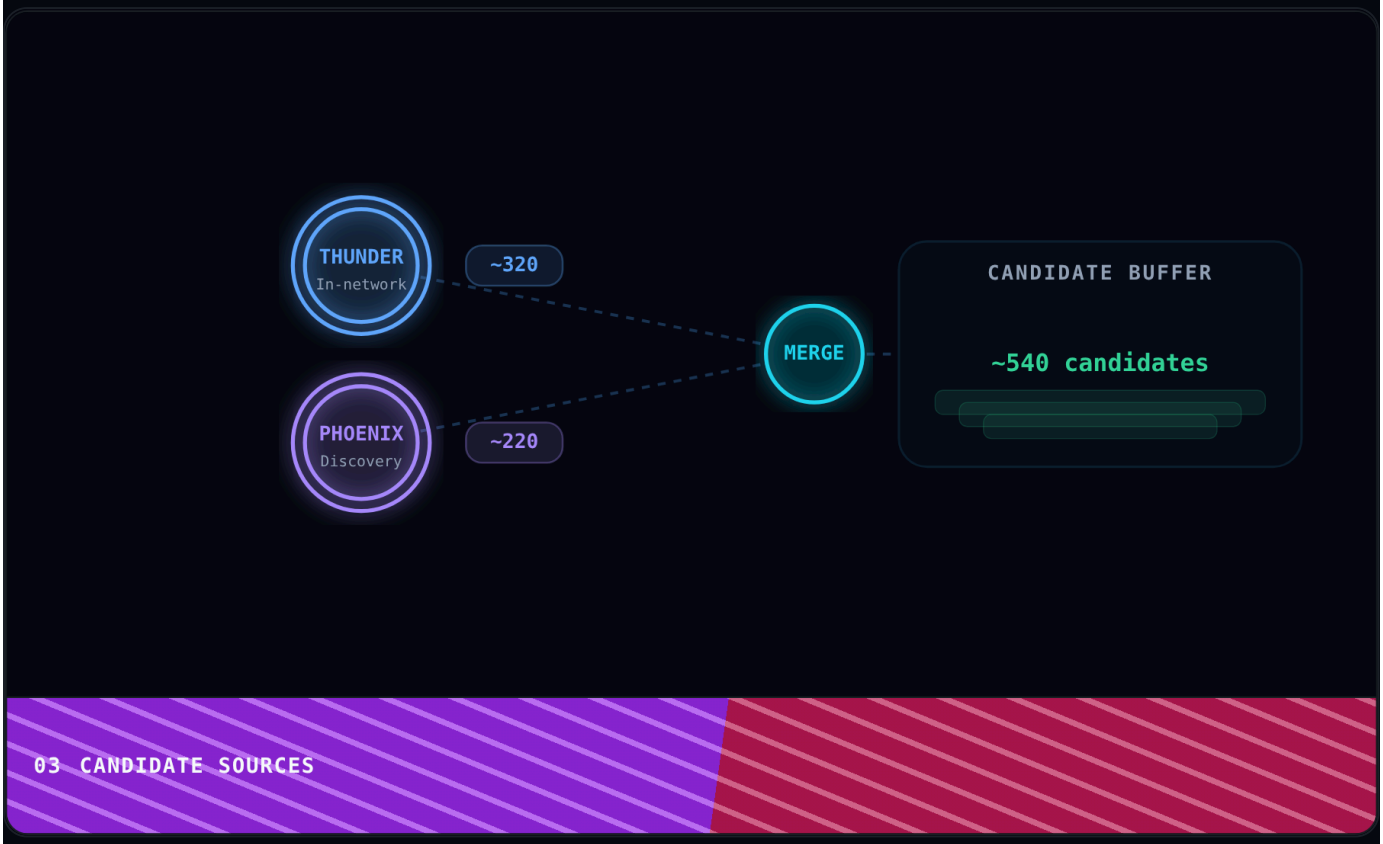
At the same time, a user features hydrator loads your follow list, muted keywords, blocked accounts, and related social preferences. These two branches run in parallel so neither waits for the other.

Try switching the Request control in the sidebar to In-network. Notice how the hydrated query flags discovery as off. An in-network request tells later stages to skip Phoenix retrieval entirely.

Both branches merge into a single hydrated query object. This is the shared input that every downstream stage will use. Without it, the system would not know who you follow or what you recently engaged with.

Context hydration is complete. The key takeaway: ranking starts before any posts are fetched, by first understanding the viewer. Next, we will see where the actual candidate posts come from.

Candidate Sources



03 CANDIDATE SOURCES

Now that the system knows who you are and what you have been doing recently, it needs posts to show you. Home Mixer asks two separate candidate sources to contribute.

Thunder handles in-network content. It keeps an in-memory store of recent posts from accounts you follow and returns a batch based on your follow graph size. More follows means more Thunder candidates.

Phoenix retrieval handles discovery. Using your behavior history from the previous stage, it searches an embedding index for posts from accounts you do not follow but would likely enjoy.

Switch the Mode control in the sidebar to In-network. Notice the Phoenix branch goes dark. An in-network request tells the system to skip discovery entirely, so only followed accounts contribute posts.

Both branches feed the same merge point. This is a critical design choice. The feed is one mixed pool from the start, not two separate timelines stitched together. Ranking will treat all candidates equally.

The merged buffer now holds hundreds of lightweight candidate objects. They have IDs and source tags, but not much else. Next, we will enrich them with the details needed for filtering and ranking.

STAGE 4 OF 9

Candidate Enrichment

We just merged candidates from Thunder and Phoenix into one pool. But those candidates are still mostly bare IDs. Before the system can filter or rank anything, it needs to know what each post actually contains.

Core data hydration runs first. It attaches the post text, reply relationships, creation timestamps, and engagement counts. Think of it as filling in the body of each card.

Profile hydration adds author screen names, follower counts, verification status, and media metadata. Switch the Media control to Video to see duration fields appear on video candidates.

An important detail: hydrators annotate candidates without removing or reordering them. The pool size stays the same. The candidates just become richer objects that downstream stages can inspect meaningfully.

A source annotation hydrator also marks whether each card originally came from Thunder or Phoenix. That label will matter later when the score cascade applies out-of-network dampening.

The result is a stack of fully enriched cards. Each one now carries everything a filter or scorer needs to make a decision. Next, we will start removing the ones that should not appear in the feed.

STAGE 5 OF 9

Filter Pass

The enriched pool still contains duplicates, stale posts, and content you have already seen or explicitly blocked. Remember how hydration from the last stage gave each card full metadata? Filters now use that metadata to clean the pool.

Duplicate and retweet cleanup goes first. If the same post arrived from both Thunder and Phoenix, only one copy survives. Retweet chains are collapsed to the original.

Freshness and self-post filters come next. Posts older than a certain threshold are removed, along with anything you authored yourself. Nobody wants to see their own tweet ranked back at them.

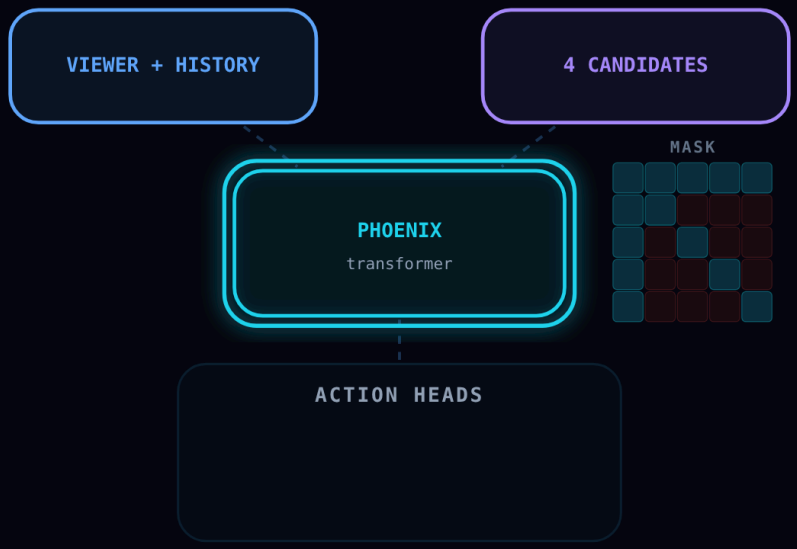
History filters apply what the client remembers. Seen IDs, recently served posts, and muted keyword matches are all removed. Try increasing the Repeats control in the sidebar. More repeat pressure means the history filter bites harder.

The final filter checks blocked and muted authors. These are personal social graph rules, not content quality judgments. A post from a blocked author is dropped regardless of its score.

What remains is a smaller, cleaner pool worth sending into the expensive ML scoring stage. The filter chain saved Phoenix from wasting compute on posts that would never be shown. Next, Phoenix will score the survivors.

STAGE 6 OF 9

Phoenix Ranker



06 PHOENIX RANKER

The filter chain from the last stage cleaned the pool. Now Phoenix receives the survivors along with the viewer context we built in Stage 1. This is where machine learning enters the picture.

The viewer embedding and action history are encoded first. They act as shared context, like a teacher who knows the student before grading each essay. Every candidate will be evaluated against this same viewer state.

Here is the most important detail in this stage: the attention mask. Each candidate can read the viewer tokens and history, but candidates cannot see each other. This isolation means a post's score depends only on you and the post itself.

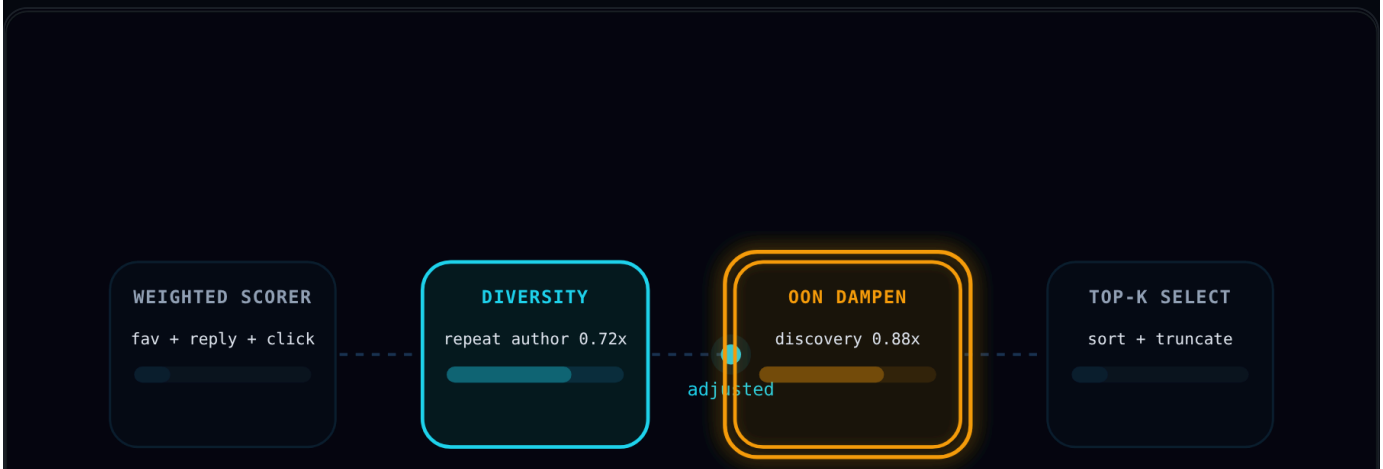
Try adjusting the Cards control to change how many candidates are scored. Notice the mask grid grows, but each candidate still has access only to the viewer row. Candidate to candidate cells stay blocked.

The action heads at the output emit probabilities for favorite, reply, repost, click, dwell, and negative signals. Switch the History control to Noisy and notice how the probability bars compress. A scattered history gives the model less to work with.

Phoenix has done its job. Each candidate now carries a set of action probabilities instead of just metadata. But the timeline still needs a single sortable number. That transformation happens next in the score cascade.

STAGE 7 OF 9

Score Cascade



07 SCORE CASCADE

Remember the action probabilities from Phoenix? Each candidate now has scores for favorite, reply, repost, click, and dwell. But the timeline needs a single number per candidate so it can sort them into an order.

The weighted scorer blends those probabilities using tuned coefficients. Favorites and replies carry more weight than clicks. The result is one relevance score per candidate.

Raw relevance is not enough. If one popular author posted five times today, all five could land at the top. The author diversity pass attenuates repeated authors so your feed stays varied.

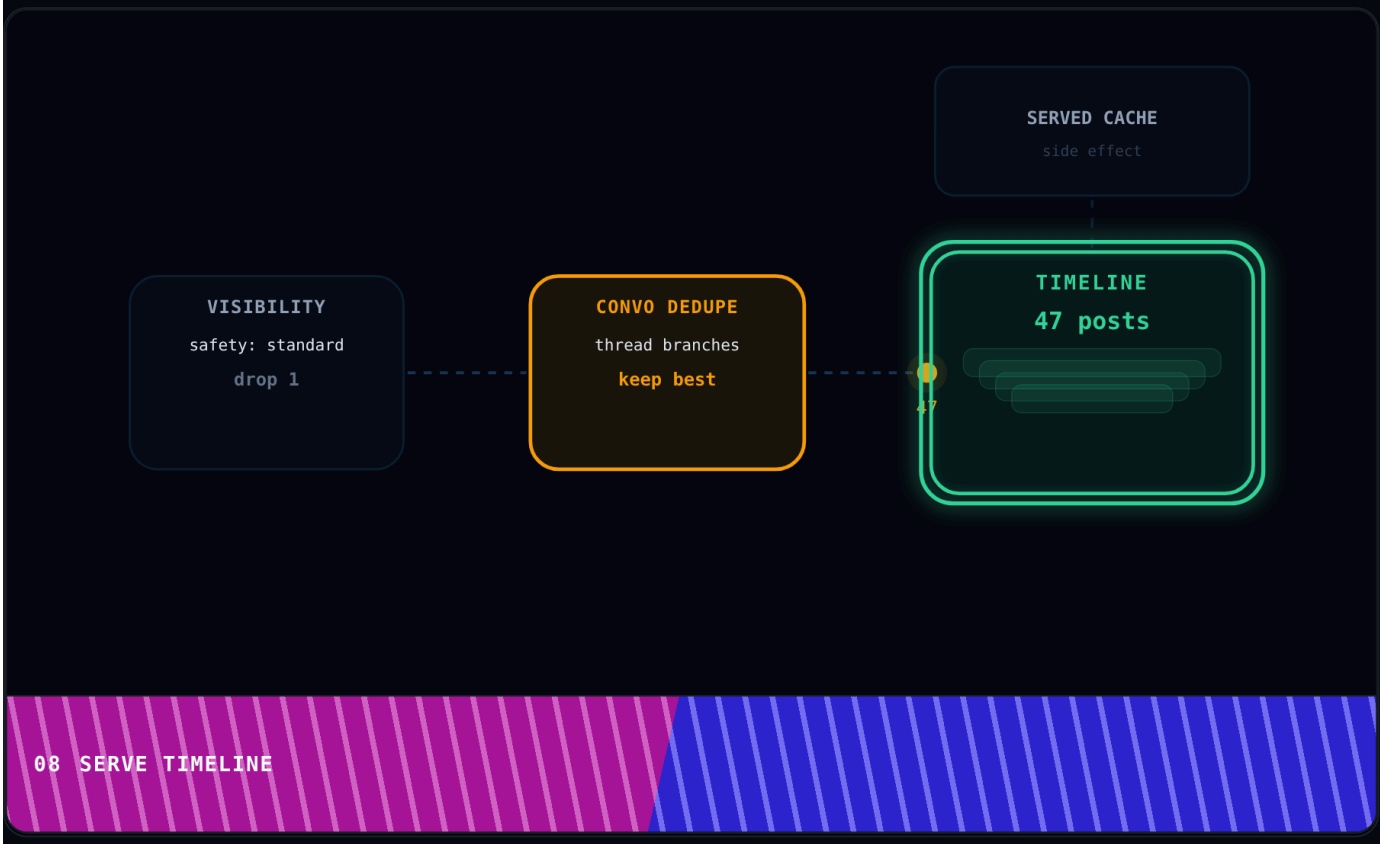
An out-of-network dampener then lightly penalizes discovery content. Switch the Bias control to Follow-first and notice how the dampening multiplier drops. This knob controls how much the feed favors followed accounts over discoveries.

The Top-K selector sorts by the final adjusted score and keeps only the top slice. The rest are discarded. What survives is a shortlist that already looks like a feed order.

The score cascade turned multi-dimensional predictions into a single ranked list. But selection is not the same as serving. There is one more safety and dedupe pass before the client sees anything. Let us look at that final stage.

STAGE 8 OF 9

Serve Timeline



08 SERVE TIMELINE

The score cascade gave us a ranked shortlist. You might think the job is done, but it is not. Selection and serving are two different things. The shortlist still needs a safety check before it reaches your device.

A visibility filter checks each post for deletion, spam flags, or policy violations that appeared after indexing. Deleted or flagged posts are removed silently. Switch the Safety control to Strict to see more borderline posts get caught.

Conversation dedupe comes next. If a reply and its parent both made the shortlist, this pass keeps only the more relevant one so the feed does not show redundant thread branches.

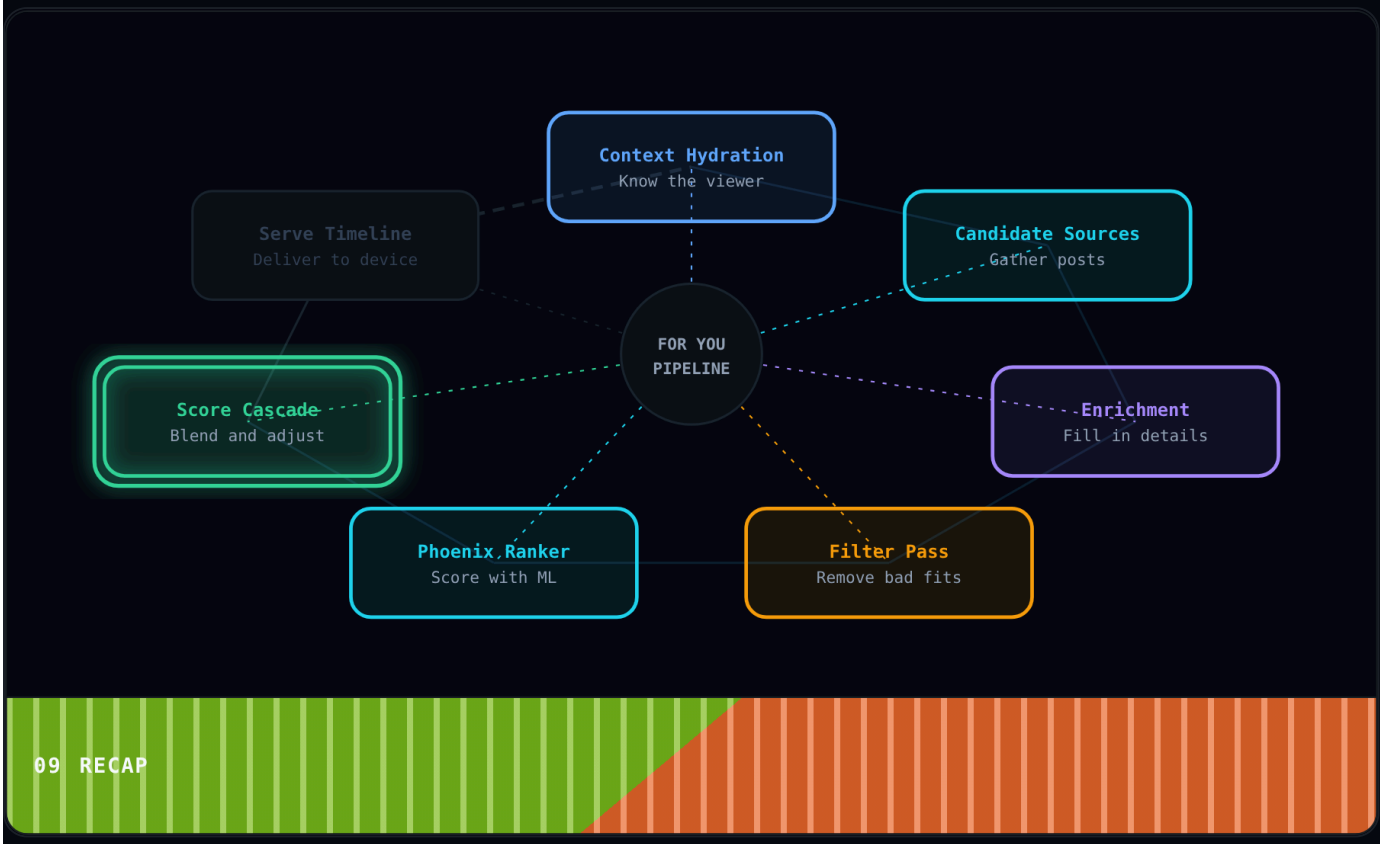
The survivors are arranged into the final response payload. This is the actual data structure the client will use to render your timeline, not just a list of scores.

A side effect fires before the response is sent. It writes what was served into a cache so the next request knows what you already saw. Remember the history filter from Stage 4? This cache is what feeds it.

The response leaves Home Mixer and your device renders the posts you see. That is the full pipeline. Now let us step back and see the whole picture together.

STAGE 9 OF 9

Recap



09 RECAP

We started with context hydration. A thin request was enriched with your recent behavior and social preferences. Without that viewer context, nothing downstream could personalize the feed.

Then we sourced candidates from two worlds. Thunder brought posts from accounts you follow. Phoenix retrieval found discoveries from outside your graph. Both merged into one pool.

Enrichment hydrators filled in the details. Post text, author metadata, media signals. The candidates went from bare IDs to fully described cards without any reordering or removal.

The filter chain removed duplicates, stale posts, seen content, and blocked authors. Each filter narrowed the pool one step at a time, saving expensive ML scoring for the survivors.

Phoenix scored each survivor by predicting action probabilities. The attention mask isolated candidates from each other so every score depended only on the viewer and the post itself.

The score cascade blended those predictions into one sortable number, then applied diversity and dampening rules to keep the feed varied and anchored in the follow graph.

A final visibility and dedupe pass cleaned the shortlist before it reached your device. The served-history cache then fed back into the next request, closing the loop.

That is the full X For You pipeline. Millions of potential posts are narrowed to about fifty through a chain of hydration, retrieval, filtering, ML scoring, policy adjustments, and safety checks. Every stage depends on the one before it.