Architekturanalyse des X-Algorithmus: Systemgrenzen, Pipeline-Orchestrierung und Verifizierung viraler Reichweiten-Claims

Vorbemerkung

Gegenstand dieser Untersuchung ist die umfassende architektonische und quellcodebasierte Analyse des offengelegten GitHub-Repositories xai-org/x-algorithm. Die Recherche und Code-Überprüfung wurde unter der Maßgabe strikter methodischer Strenge durchgeführt, um virale Behauptungen bezüglich Reichweitenverlusten, Reichweiten-Metriken und algorithmischen Strafen auf der Plattform X einem datenbasierten Faktencheck zu unterziehen. Das Ziel ist eine quellennah belegte Prüfung strittiger Behauptungen aus einem viralen Creator-Thread, der als Ausgangspunkt der Claims dient, hierbei jedoch konsequent als reiner Behauptungskatalog und nicht als Tatsachenquelle behandelt wird.

Das analysierte Repository repräsentiert den Stand des Main-Branches vom Frühjahr 2026. Es handelt sich um die in Rust geschriebene, auf dem Grok-1-Transformer basierende Architektur, die das Kernsystem, namentlich den "Home Mixer", für den personalisierten "For You"-Feed abbildet. Ausdrücklich zu betonen sind die Open-Source-Grenzen dieser Veröffentlichung. Die bereitgestellte Codebasis gewährt profunde Einblicke in die strukturelle Pipeline, einschließlich der Filter-Abfolgen, Scorer-Logiken und Hydration-Phasen, ist jedoch nicht vollständig identisch mit der isoliert lauffähigen Produktionsumgebung. Zentral für das Verständnis der Systemgrenzen ist das bewusste Auslassen von Modellgewichten in Parametrisierungsdateien wie params.rs, von internen Utility-Funktionen sowie von spezifischen Implementierungsdetails diverser Service-Clients.

Diese Untersuchung differenziert streng zwischen belegbarer Code-Architektur, dokumentierter Intention, begründbarer Inferenz und spekulativer Interpretation. Externe Drittquellen, private Blogs, Social-Media-Kommentare oder Medienberichte fließen nicht in die Beweisführung ein. Das primäre und einzige Beweismaterial bildet der maschinell auslesbare Quellcode, die Repository-Struktur und die beiliegenden Dokumentationsdateien aus der offiziellen Publikation von xAI.

I. Repository-Status und Grenzen des offenen Codes

Bevor die spezifischen Mechaniken des Reichweiten-Algorithmus verifiziert werden können, muss das Fundament der Beweisführung exakt definiert werden. Die zentrale Frage lautet, inwieweit das Repository finale Aussagen über das Produktionsverhalten zulässt und welche

systemischen blinden Flecken in der Evaluation berücksichtigt werden müssen.

Das Repository enthält den strukturellen Produktionscode des Empfehlungssystems, das In-Network-Inhalte und Out-of-Network-Inhalte orchestriert und kombiniert. Die Datei phoenix_candidate_pipeline.rs orchestriert diese Phasen eindeutig und demonstriert die Integration von Clients wie dem PhoenixRetrievalClient und der ThunderSource.[1] Das Top-Level-README definiert explizit einen Paradigmenwechsel innerhalb der Plattformarchitektur: Die vollumfängliche Abkehr von handgeschriebenen Heuristiken hin zu einem Grok-basierten Transformer-Modell, welches komplexe Embedding-Repräsentationen nutzt, um Interaktionswahrscheinlichkeiten vorherzusagen.[2] Die Candidate Pipeline nutzt eine klar strukturierte Abfolge von Sourcing, Hydration, Filtering, Scoring und Selektion.[1]

Trotz der Transparenz der Pipeline-Orchestrierung weist das System bewusste Auslassungen auf, die aus dem Open-Source-Release exkludiert wurden. Dies limitiert die finale Beweisbarkeit quantitativer Metriken erheblich. Die Datei params.rs, welche alle zentralen Konstanten, Schwellenwerte und Gewichtungen wie beispielsweise p::REPLY_WEIGHT oder MAX_POST_AGE enthält, ist nicht öffentlich einsehbar.[4] Verzeichnisse für Service Clients fehlen in ihrer Implementierungstiefe, was den Blick auf externe Microservices wie den GizmoduckClient oder den SocialGraphClient verwehrt. Darüber hinaus bleiben die Modellgewichte, die Trainingsdaten und die exakte Architektur der Embeddings des Grok-Transformers eine Blackbox. Der Transformer liefert Wahrscheinlichkeiten, deren tieferes statistisches Zustandekommen nicht im Rust-Code des Mixers abgebildet ist.[3] Schließlich sind Utility-Module, die essenzielle Funktionen wie normalize_score oder get_related_post_ids beinhalten, ebenfalls ausgeschlossen.[5]

Die strukturelle Logik der Pipeline ist direkt belegbar und stimmt offensichtlich mit der Makroarchitektur der Produktion überein. Es lässt sich zweifelsfrei determinieren, welche Filter in welcher Reihenfolge greifen und welche Variablen in den linearen Scorern existieren. Die quantitative Ausprägung dieser Variablen entzieht sich jedoch der Analyse. Aus dem Code lässt sich ablesen, welche Metriken das System erfasst, es lässt sich jedoch nicht berechnen, wie stark diese ins Gewicht fallen. Jegliche Aussagen, die absolute Proportionalitäten zwischen verschiedenen Interaktionstypen behaupten, sind daher ohne die Einsicht in die Datei params.rs methodisch unzulässig und spekulativ.

II. Follower Count und Social Graph

Eine zentrale und weithin diskutierte Streitfrage ist die Relevanz der Followerzahl für die individuelle Reichweite eines Accounts. Die virale Behauptung suggeriert, dass Follower-Metriken lediglich als Display-Daten fungieren und von sämtlichen Scorern innerhalb der algorithmischen Pipeline ignoriert werden.

claim_	claim_	repo_	symb	code_ evide	status	confid	interp retati	public _claim

id	text	path	ol	nce		ence	on	_safe?
FC_01	Follower count does nothing for reach anymore.	home-mixer/candidate_pipeline/	GizmoduckCandidateHydrator	author_followers_count wird aktiv aus Nutzerprofilen hydriert.[7]	teilweise belegt	Hoch	Variable existiert und wird für Pipeline geladen, fehlt aber im expliziten Code des linearen WeightedScorer.	Nein
FC_02	It is not fed into any scorer.	home-mixer/scorers/weighted_scorer.rs	compute_weighted_score	author_followers_count fehlt in der Formel des WeightedScorer.[5]	teilweise belegt	Mittel	Fehlt im linearen Scorer, könnte aber im Transformer oder bei der Retrieval-Phase als		Nein

							Embedding-Feature genutzt werden.	
FC_03	Social graph / Following list is irrelevant.	home-mixer/candidate_pipeline/	ThunderSource, AuthorSocialgraph Filter	In-Network-Posts werden via Thunder geladen; Filter prüfen den Graphen.[1]	widerlegt	Hoch	Follow-Beziehung ist primäre Source für Thunder; elementar für In-Network-Ausspielung.	Nein

Die Überprüfung der architektonischen Mechanik offenbart ein differenziertes Bild bezüglich der Nutzung von Netzwerk-Metriken. Der Code belegt eindeutig, dass die Followerzahl eines Autors in Form der Variable author_followers_count vom GizmoduckCandidateHydrator aus den Profildaten extrahiert und in den Systemkontext geladen wird.[7] Diese Hydratisierung geschieht vor der eigentlichen Filtering- und Scoring-Phase. Betrachtet man jedoch isoliert die Datei weighted_scorer.rs, in der die lineare Kombination der Signale zur Berechnung des finalen Scores stattfindet, so taucht die Variable author_followers_count dort nicht als direkter gewichteter Faktor auf.[5] Gleichzeitig existieren starke netzwerkbezogene Sourcing-Komponenten: Die ThunderSource ist explizit dafür zuständig, Content von Accounts abzurufen, denen der Nutzer aktiv folgt.[1] Zudem nutzt der AuthorSocialgraphFilter explizit die sozialen Verbindungen, um Mutes oder Blocks konsequent anzuwenden und den Feed zu bereinigen.[8]

Dass die numerische Followerzahl nicht direkt im WeightedScorer mit einem simplen Gewicht multipliziert wird, bedeutet architektonisch in keinem Fall, dass sie für die Reichweite

bedeutungslos ist. Hier liegt eine fundamentale Fehlinterpretation moderner Recommender-System-Architekturen vor. Die Hydratisierung eines Features wie author_followers_count indiziert stark, dass komplexe nachgelagerte Scorer, insbesondere der auf Machine Learning basierende PhoenixScorer, auf diese Datenpunkte zugreifen.[1] Ein Transformer-Modell kann die Followerzahl als Teil eines Nutzer-Embeddings nutzen, um die Authentizität, die historische Autorität oder das virale Potenzial eines Posts einzuordnen. Darüber hinaus garantiert die Follow-Beziehung an sich – unabhängig von der absoluten Zahl – eine In-Network-Evaluierung durch die ThunderSource.[2] Ein Post wird bei den eigenen Followern systematisch in die engere Candidate-Pipeline eingespeist, während er für eine Out-of-Network-Reichweite erst durch den PhoenixRetrievalClient über Vektorähnlichkeiten identifiziert werden muss.[1] Der WeightedScorer enthält zudem Signale wie den follow_author_score [4], was belegt, dass die prädiktive Wahrscheinlichkeit, dass ein Post zu einem neuen Follower führt, ein massives positives Reichweitensignal darstellt.

Die Behauptung, Followerzahlen hätten absolut keine Auswirkung mehr und seien reines kosmetisches Display-Material, ist aus systemarchitektonischer Sicht nicht haltbar. Zwar fehlt ein platter linearer Multiplikator im offengelegten Scorer-Code, doch die bewusste Hydratisierung der Zahl sowie die essenzielle strukturelle Bedeutung des Social Graphs im Candidate-Sourcing widerlegen die These einer pauschalen Irrelevanz. Öffentliche Aussagen in dieser normativen Absolutheit ignorieren die Komplexität von Embedding-Räumen und sind faktisch unzutreffend.

III. Phoenix Prediction Heads und Multi-Action Prediction

Eine intensiv diskutierte Behauptung betrifft die Funktionsweise des Phoenix-Modells, spezifisch die genaue Anzahl der sogenannten "Prediction Heads" und das damit verbundene Konzept einer "Prediction Trap", bei der Reichweite algorithmisch abgeriegelt wird, bevor eine signifikante Nutzerbasis den Inhalt überhaupt zu sehen bekommt.

claim_id	claim_text	repo_path	symbol	code_evidence	status	confidence	interpretation	public_claim_safe?
PH_01	Phoenix uses 19 prediction heads.	scorers/weighted_scorer.rs	PhoenixScores Felder	Es existieren exakt 15 positive	direkt belegt	Hoch	Die Summe der spezifisch codierten	Ja

				Signale und 4 negative Signale in der Score-Kombination.[4]			Score-Typen ergibt exakt 19.	
PH_02	Predicts reach before anyone sees the post.	README.md, phoenix/	Grok Transformer	Das System prognostiziert Interaktions-Wahrscheinlichkeiten für Aktionen wie P(like) auf Basis latenter Modelle.[3]	starke Inferenz	Hoch	Standardverhalten latenter Retrieval- und Scoring-Pipelines vor der Impression.	Ja
PH_03	Prediction Trap limits reach autonomou	README.md	Candidate Isolation	Scoring erfolgt kontextfrei; schwache	starke Inferenz	Mittel	Das System priorisiert kontextfreie	Ja

	sly.			Prognosen führen zu direktem Ausschluss durch den Selector.[1]			Vorhersagen; schlechte Embeddings bedeuten null Impressionen.	

Die Evaluation der Moduldatei home-mixer/scorers/weighted_scorer.rs erbringt einen der detailliertesten strukturellen Nachweise im gesamten Repository. Der Code referenziert ein Datenobjekt, welches als s: &PhoenixScores deklariert ist, und prozessiert exakt 19 unterschiedliche Wahrscheinlichkeits-Scores in einer linearen Kombination.[4] Zu den positiv gewichteten Signalen, von denen es exakt fünfzehn gibt, gehören Interaktionsmetriken wie favorite_score, reply_score, retweet_score, quote_score und share_score. Ebenso werden Sichtbarkeits- und Verweildauermetriken quantifiziert, darunter click_score, photo_expand_score, profile_click_score, quoted_click_score, dwell_score, dwell_time, follow_author_score und der spezifisch für Bewegtbildinhalte konzipierte vqv_score (Video Quality Value).[4] Sharing-Aktivitäten werden granular durch den share_via_dm_score und den share_via_copy_link_score abgebildet. Die vier negativen Signale umfassen den not_interested_score, block_author_score, mute_author_score und den report_score.[4] Gemäß der Repositoriumsdokumentation führt das Grok-basierte Modell eine elaborierte Multi-Action-Prediction durch. Es berechnet die statistischen Wahrscheinlichkeiten für all diese Aktionen simultan für jeden eingehenden Kandidaten, noch bevor dieser die finale Selektionsphase durchläuft.[3]

Das Konzept einer "Prediction Trap" stellt eine polemische, dramaturgische Umschreibung eines etablierten Standardverhaltens in modernen hyper-skalierten Recommender-Systemen dar. Da das Phoenix-Ranking-Modell unter dem architektonischen Paradigma der "Candidate Isolation" arbeitet, bewertet der Transformer jeden Content-Kandidaten vollkommen isoliert auf Basis des individuellen Nutzer-Kontexts und der intrinsischen Post-Embeddings.[3] Eine Abhängigkeit von anderen Posts im aktuellen Feed-Batch ist ausgeschlossen. Wenn das Two-Tower-Retrieval und der Scorer für einen bestimmten Post minimale Interaktionswahrscheinlichkeiten prognostizieren – beispielsweise aufgrund einer fehlenden historischen Resonanzstruktur –, erreicht dieser Kandidat keinen ausreichenden Summenwert im WeightedScorer. Folglich fällt der Post durch das Raster des TopKScoreSelector und erhält physisch keine Impressionen im Frontend des Nutzers.[1] Der Content wird somit nicht erst einem breiten Publikum exponiert und bei schlechter Performance abgestraft, sondern von

vornherein probabilistisch als irrelevant aussortiert.

Die numerische Behauptung über die Existenz von 19 Prediction Heads ist durch den vorliegenden Quellcode als absolut und direkt belegt zu klassifizieren. Die Deskription, dass das Machine-Learning-Modell Interaktionen prognostiziert, bevor Sichtbarkeit im Feed gewährt wird, beschreibt zutreffend die fundamentale Natur einer Two-Tower-Architektur mit nachgelagertem Transformer-Ranking. Der Terminus "Trap" mag normativ gefärbt sein, erfasst aber präzise das Kaltstart-Problem für neue oder systemisch unvorteilhaft bewertete Inhalte: Ein Mangel an prädiktiver Attraktivität in den latenten Repräsentationen führt zu einem direkten Ausschluss von der Sichtbarkeit, ohne dass organische Explorationstruppen als Evaluierungsinstanz dienen.

IV. Weighted Scorer und die unsichtbaren Parameter

Die Kontroverse darüber, welche spezifischen Aktionen wie Likes, Replies oder Reposts algorithmisch wie stark gewichtet werden, bildet den Kern zahlreicher Strategien von Content-Erstellern. Die Behauptungen operieren oft mit fixen Multiplikatoren, die es zu überprüfen gilt.

claim_id	claim_text	repo_path	symbol	code_evidence	status	confidence	interpretation	public_claim_safe?
WS_01	Replies score higher than likes.	weighted_scorer.rs	p::REPLY_WEIGHT, p::FAVORITE_WEIGHT	Beide Gewichte existieren und werden appliziert, die numerischen Werte liegen aber in params.rs verbor	offen	Niedrig	Ohne die Rohwerte ist keine proportionale oder vergleichende Aussage verifizierbar.	Nein

				gen.[4]				
WS_02	Negative feedback destroys score.	weighted_scorer.rs	p::BLOCK_AUTHOR_WEIGHT etc.	Negative Signale werden im Code explizit aggregiert und vom Score subtrahiert oder negativ summiert.[3]	direkt belegt	Hoch	Negative User-Interaktionen wirken als gewichtete Reduktionsfaktoren im finalen Ranking.	Ja
WS_03	Visual posts get dwell/photo boost.	weighted_scorer.rs	photo_expand_score, vqv_score	Dedizierte Scoring-Felder und zugehörige Weights wie p::PHOTO_EXPAND_WEIGHT sind implementiert.[4]	direkt belegt	Hoch	Die Pipeline erfasst visuelle Interaktionsformen und honoriert diese durch eigene Prediction	Ja

							Heads .	

Die Analyse der Moduldatei home-mixer/scorers/weighted_scorer.rs illustriert den Mechanismus der finalen linearen Evaluation. Innerhalb der Funktion compute_weighted_score wird eine gewichtete mathematische Summe gebildet. Der Quellcode demonstriert dies durch Aufrufe wie Self::apply(s.favorite_score, p::FAVORITE_WEIGHT) + Self::apply(s.reply_score, p::REPLY_WEIGHT).[5] Es ist evident, dass die prädiktiven Wahrscheinlichkeiten, die aus den Prediction Heads des Transformer-Modells stammen, mit statischen Faktoren skaliert werden. Diese Faktoren werden aus einem externen Parametermodul p, referenziert als crate::params, importiert.[5] Negative Signale wie das Blockieren oder Stummschalten eines Autors werden parallel verrechnet und reduzieren den Gesamtwert durch die Anwendung von spezifischen negativen Konstanten wie p::BLOCK_AUTHOR_WEIGHT.[4] Weiterhin implementiert der Code eine konditionale Logik für Videodateien: Der Video Quality Value, vqv_score, wird nur dann mit einem Gewicht versehen, wenn die Dauer des Medieninhalts einen vordefinierten Minimum-Wert, p::MIN_VIDEO_DURATION_MS, überschreitet.[5] Nach der umfassenden Summierung der Faktoren wird die Utility-Funktion normalize_score aufgerufen, deren exakter normalisierender Algorithmus aufgrund der Exklusion des util-Moduls eine Blackbox bleibt.[5]

Die vorliegende Architektur demonstriert zweifellos die Multidimensionalität des Rankings, verschleiert jedoch systematisch die quantitative Kalibrierung der einzelnen Dimensionen. Die Isolation numerischer Konstanten in eine exkludierte params.rs-Datei entspricht Best Practices im Software Engineering für Machine-Learning-Pipelines, um Logik-Kompilierung von Hyperparameter-Tuning zu trennen. Der Code beweist unumstößlich, dass Mutes, Blocks und Reports zu einer direkten algorithmischen Abwertung des Content-Scores führen. Jede Behauptung, die konkrete proportionale Relationen aufstellt, ist jedoch wissenschaftlich unlauter und basiert auf freier Erfindung. Es ist unmöglich aus dem Code abzuleiten, ob ein Reply einen zehnfach höheren Wert als ein Like aufweist, oder ob Link-Klicks mit einem prozentualen Malus versehen sind. Das System summiert gewichtete Vektoren, aber die Vektorlängen sind unsichtbar.

Dementsprechend sind öffentliche Behauptungen über die reine Struktur der algorithmischen Bewertung – also die Kategorisierung in positive und negative Resonanzsignale – absolut verifizierbar und können sicher kommuniziert werden. Jegliche Aussagen über konkrete Gewichtungsverhältnisse, Punkte-Systeme oder mathematische Hierarchien der Interaktionsformen sind hingegen maßlos und entbehren jeder empirischen Grundlage im Rahmen dieser Open-Source-Veröffentlichung. Wer behauptet, exakte Gewichtswerte aus dem xAI-Repository extrahiert zu haben, stützt sich auf imaginäre Daten.

V. Author Diversity: Die Mathematik der Sichtbarkeitsdrosselung

Ein emotional hoch aufgeladenes Thema für intensive Plattform-Nutzer ist die Frequenz der

Publikation. Die These "Posting too much hurts" postuliert eine systematische Bestrafung für hohe Veröffentlichungsdichten. Gleichzeitig wird eine "Dormancy Penalty" für zu geringe Aktivität befürchtet.

claim_id	claim_text	repo_path	symbol	code_evidence	status	confidence	interpretation	public_claim_safe?
AD_01	Posting too much in a session hurts score.	scorers/author_diversity_scorer.rs	AuthorDiversityScorer	Exponential Decay Multiplier: (1 - floor) * decay^position + floor mindert den Score.[6]	direkt belegt	Hoch	Eine Pipeline-Komponente bestraft sukzessiv multiple Posts desselben Autors innerhalb einer evaluierten Candidate-Liste.	Ja
AD_02	Posting too little hurts (Dormancy penalty).	Repo-weit	-	Es findet sich kein Code befund in Scorern oder	nicht belegt	Hoch	Inaktivität reduziert natürliche Impressionen,	Nein

				Filtern für eine "Dormancy"-Strafe bezüglich Account-Inaktivität.			triggert aber keinen negativen Multiplikator in der offengelegten Logik.	
AD_03	Safe posting band is undisclosed.	params.rs	decay, floor	Konstanten für die Abklingkurve stammen aus nichtöffentlichen Parametern.[6]	direkt belegt	Hoch	Die mathematische Struktur der Drosselung ist klar, die exakte Schwere der Abklingkurve bleibt verborgen.	Ja

Die Analyse der Pipeline-Topologie verortet den AuthorDiversityScorer in der phoenix_candidate_pipeline.rs unmittelbar nach den primären Scorer-Instanzen.[1] Gemäß der implementierten Logik appliziert dieser Scorer eine deterministische Dämpfungsfunktion auf wiederkehrende Autoren innerhalb derselben Liste von Content-Kandidaten, die für eine Request-Session evaluiert wird. Die algebraische Funktion lautet: multiplier = (1 - floor) * decay^position + floor.[6] Die Variable position referenziert hierbei den Index des Auftretens desselben Autors in der aktuell berechneten Liste. Der erste evaluierte Post eines Autors in diesem Batch erhält den Indexwert position = 0, was mathematisch zu einem Multiplikator von 1 führt und somit den ursprünglichen Score unangetastet lässt. Der zweite Post desselben

Autors erhält position = 1 und wird mit dem decay-Faktor gedämpft. Der Parameter floor agiert als Asymptote, die verhindert, dass der Score im Falle von extremen Frequenzen auf den absoluten Nullwert kollabiert; eine absolute Auslöschung findet nicht statt.

Dieses mathematische Konstrukt dient explizit der Feed-Diversifikation, um zu verhindern, dass ein einzelner hoch-performanter Autor die Ausspielungslisten monopolisiert. Es ist von höchster Wichtigkeit, den Kontext dieses Decays korrekt einzuordnen: Die algorithmische Dämpfung operiert streng innerhalb der Grenzen eines Pipeline-Aufrufs. Wenn das Recommender-System hundert Posts lädt, um die Top-K für den momentanen Reload-Request eines Nutzers zu generieren, verdrängen sich multiple Posts desselben Autors in diesem spezifischen Batch gegenseitig.[1] Ein Content-Ersteller, der eine Salve an Posts in einem engen zeitlichen Korridor absetzt, zwingt diese Posts, in den Candidate-Listen seiner potenziellen Leserschaft um Aufmerksamkeit zu konkurrieren. Die stärksten Posts behalten ihre Punktzahl, die nachfolgenden werden durch die Exponentialkurve künstlich unter die Selektionsschwelle gedrückt. Umgekehrt existiert im offengelegten Repository absolut kein empirischer Beweis für eine sogenannte "Dormancy Penalty", also eine explizite Strafe für seltene Publikationen. Wenn ein Account lange Zeit inaktiv ist, führt dies zu einer Veraltung seiner Transformer-Embeddings und zu einem Mangel an initialen Kandidaten, wird aber nicht durch einen negativen linearen Straf-Score sanktioniert.

Die These, dass exzessives Posten innerhalb kurzer Zeiträume der Gesamtreichweite schadet, ist durch die Decay-Formel strukturell einwandfrei verifiziert. Ein Creator, der eine Flut an Inhalten zeitgleich veröffentlicht, fragmentiert seine eigene Durchsetzungsfähigkeit im Top-K-Selektionsprozess drastisch. Die parallel existierende These einer algorithmischen Strafmaßnahme für temporäre Inaktivität projiziert jedoch menschliche Moralvorstellungen von Beständigkeit auf ein stochastisches Modell und muss anhand des Source Codes als unbelegt zurückgewiesen werden.

VI. Retention, Age Filter und der 48h-Mythos

Die persistente Annahme, dass veröffentlichter Content nach Ablauf von exakt 48 Stunden global und unwiderruflich aus dem Plattformsystem gelöscht werde und somit keinerlei virales Spätpotenzial besitze, verursacht massive Unruhe unter professionellen Erstellern.

claim_id	claim_text	repo_path	symbol	code_evidence	status	confidence	interpretation	public_claim_safe?
AF_01	Posts are gone from	filters/age_filter.rs	MAX_POST_AGE	Der AgeFilter entfer	teilweise belegt	Mittel	Es existiert ein deter	Nein

	the system after 48h.			nt Kandidaten älter als MAX_POST_AGE. Der exakte Wert residiert in exkludierten Parametern.[6]			ministischer, harter Cutoff für den Feed, ein globaler 48h-Parameter ist im Code jedoch unsichtbar.	
AF_02	Thunder auto-trims old posts every 2 mins.	thunder/posts/	ThunderSource	Caching-Systeme wie Thunder verwerfen alte Daten. Die genaue 2-Minuten-Frequenz ist unbelegt.	schwache Inferenz	Niedrig	Ein Auto-Trim-Verhalten ist für In-Memory-Systeme typisch, die konkrete Behauptung des Zeitintervalls bleibt ungestützt.	Nein

AF_03	Nothing from 3 days ago is shown to anyone.	Gesamtsystem	AgeFilter	Der AgeFilter ist ein Modul des Home Mixer (For You-Feed), er limitiert nicht globale Profile, Suchen oder Quotes.[1]	widerlegt	Hoch	Die Einschränkung betrifft isoliert die algorithmische Feed-Empfehlung, nicht das komplette Plattform-Ökosystem.	Nein

Die strukturelle Durchsicht der Filter-Hierarchie demonstriert die Applikation eines AgeFilter innerhalb der Candidate Pipeline. Dieser Filter wird aktiviert, nachdem die initiale Sourcing- und Hydration-Phase abgeschlossen ist.[1] Seine spezifische Aufgabe besteht darin, den kryptografischen Timestamp aus der Twitter Snowflake-ID zu decodieren. Jeder identifizierte Kandidat, dessen Alter den statischen Schwellenwert von MAX_POST_AGE Sekunden überschreitet, wird kompromisslos aus der Liste der Evaluierungen entfernt. Das System agiert hier im Fail-Closed-Modus: Sollte das Parsing der Snowflake-ID fehlschlagen, wird der Post präventiv gelöscht.[6] Diese Mechanik belegt zweifelsfrei die Existenz eines harten und unwiderruflichen Recency-Cutoffs für den algorithmischen "For You"-Feed, welcher durch keine noch so hohe Engagement-Metrik kompensiert werden kann. Der konkrete Wert der Konstante MAX_POST_AGE ist jedoch Bestandteil der unzugänglichen Parametrisierung.

Die weitläufig kommunizierte Zahl von 48 Stunden mag auf empirischen Blackbox-Tests von Nutzern beruhen oder historische Relikte aus älteren Architekturen der ThunderSource reflektieren, welche als flüchtiger In-Memory-Cache extrem kurze Garbage-Collection-Intervalle aufweist. Jedoch ist die Generalisierung, dass Content nach diesem Fenster global aus dem System entfernt werde, massiv irreführend und zeugt von einem tiefgreifenden Missverständnis der Microservice-Infrastruktur. Der home-mixer orchestriert ausschließlich die Bereitstellung des personalisierten Empfehlungsfeeds.[2] Davon unabhängig existieren separate und autarke Retrieval-Pfade für direkte Profilaufrufe,

chronologische Suchanfragen, eingebettete Zitate oder externe Web-Links. Auf diese alternativen Kanäle findet der spezifische AgeFilter des Home Mixers keine Anwendung. Weiterhin basiert die Behauptung, Thunder führe ein deterministisches Auto-Trimming im exakten Abstand von zwei Minuten durch, auf reiner Spekulation, die sich nicht durch vorliegende Code-Auszüge verifizieren lässt.

Es ist als unumstößlicher Fakt zu werten, dass das System für den Empfehlungsfeed eine absolute, zeitlich definierte Halbwertszeit implementiert hat, jenseits derer keine algorithmische Distribution mehr erfolgt. Die Behauptung jedoch, dass Publikationen nach Ablauf von zwei Tagen spurlos aus dem gesamten Datenhaushalt verschwinden und auf einen Wert von null zurückgesetzt werden, negiert die Vielschichtigkeit einer verteilten Service-Landschaft. Solche extremen Pauschalisierungen sind im Kontext einer präzisen Systemanalyse unzulässig.

VII. Repost- und Retweet-Deduplication

Die systematische Unterdrückung von repetitiven Inhalten, insbesondere die vermeintlich harte Bestrafung von Reposts fremder Kunstwerke oder wiederholten Veröffentlichungen durch Content-Kuratoren, bildet ein weiteres hochkontroverses Feld.

claim_id	claim_text	repo_path	symbol	code_evidence	status	confidence	interpretation	public_claim_safe?
DD_01	Every repost of another artist gets buried.	Pipeline	RetweetDeduplicationFilter	Filter wie der DropDuplicatesFilter und RetweetDeduplicationFilter sind aktiv im Einsatz.[1]	teilweise belegt	Hoch	Deduplikation bereinigt Redundanzen in einem Feed-Request, agiert aber nicht als globale	Nein

							algorithmische Abwertung.	
DD_02	Up to 90% impression deduction.	filters/	-	Es existiert kein Beleg für eine mathematische Reduktion oder einen dedizierten Multiplikator im Bereich von 90%.[1]	widerlegt	Hoch	Die implementierten Filter operieren als boolesche Gates (Keep/Drop); die Angabe von 90% ist empirische Fiktion.	Nein
DD_03	Bloom filters reset by session.	filters/previously_seen_posts_filter.rs	PreviouslySeenPostsFilter	Nutzung von Bloom-Filtern für die schnelle Wahrscheinlichkeitsprüfung	starke Inferenz	Hoch	Bloom-Filter auf Client-Ebene verhindern effizient die erneute Präsentation	Ja

				historischer Sichtbarkeit.[1]			bereits konsumierter identischer IDs.	

Die Architektur implementiert in der Pre-Scoring-Phase, unmittelbar nach der Anreicherung durch Hydratoren, eine hochkomplexe und mehrstufige Filter-Phalanx, um Redundanzen im User-Feed rigoros zu minimieren. Flankierend zum basalen DropDuplicatesFilter operiert der spezialisierte RetweetDeduplicationFilter.[1] Um historische Duplikate zu identifizieren, nutzt das System den PreviouslySeenPostsFilter, der eine Kombination aus probabilistischen Bloom-Filtern für riesige Datenmengen und exakten ID-Listen abgleicht.[6] Ein nicht einsehbares Hilfsmodul namens get_related_post_ids() ermittelt im Hintergrund, welche unterschiedlichen Identifikatoren – beispielsweise der Original-Post und dessen vielfache Reposts – systemisch als äquivalente Inhalte klassifiziert werden. Parallel bereinigt der SelfTweetFilter den Feed eines Nutzers prophylaktisch von den eigenen Publikationen, um narzisstische Feedbackschleifen zu kappen.[2]

Der teleologische Zweck dieser massiven Filterarchitektur liegt primär in der Steigerung der User Experience durch strikte Vermeidung von Content-Kollisionen. Die Mechanik lässt sich an einem Beispiel illustrieren: Folgt ein Endnutzer einem populären Künstler, und drei weitere Accounts aus dem eigenen Graphen teilen exakt dasselbe Werk, interveniert der RetweetDeduplicationFilter. Er garantiert, dass der Endnutzer das Artefakt nicht viermal sukzessive beim selben App-Öffnen präsentiert bekommt. Der Filter vollzieht einen harten "Drop", er verwirft den Kandidaten aus der aktuellen Berechnungsliste. Es handelt sich um einen rein binären Filterprozess: Der Kandidat wird zugelassen oder vernichtet. Nirgends in der offengelegten Codebasis manifestiert sich ein linearer Dämpfungsfaktor, der einen Score um arbiträre 90% reduziert. Ein Einbruch der resultierenden Impressions um 90% in der empirischen Praxis mag das direkte Resultat der Tatsache sein, dass der Content bei der großen Mehrheit der potenziellen Empfänger verworfen wird, da diese das Original via Bloom-Filter bereits als "Previously Seen" markiert haben. Dies ist jedoch eine statistische Marktsättigungs-Konsequenz und keine proaktive algorithmische Strafmaßnahme gegen den Ersteller.

Die polemische Konklusion, dass Reposts systematisch begraben würden, verwechselt somit in dramatischer Weise die deterministische Filter-Deduplikation mit einer Ranking-Bestrafung im Machine-Learning-Sinn. Das System bewertet den Kurator nicht ab; es schützt den Konsumenten vor der Sättigung. Die Angabe einer 90-prozentigen Score-Deduction ist ein Mythos, während die Verwendung von probabilistischen Bloom-Filtern und deren flüchtige Session-Resets eine korrekte und präzise Interpretation der asynchronen Caching-Logik

darstellen.

VIII. Creator-Reichweite: Qualitätsmetriken vs. Engagement-Maximierung

Die Synthese der Einzelbefunde zwingt zur Evaluation einer übergeordneten Frage bezüglich der strukturellen Bevorzugung von Inhalten: Ist die häufig kolportierte These, dass "Content Quality secondary" sei, während billiges Engagement dominiere, architektonisch belegbar?

Die Anatomie der Ausspielung beginnt mit dem massiven Einsammeln potenzieller Kandidaten über Thunder für In-Network-Verbindungen und Phoenix für Out-of-Network-Exploration. Wenn ein Creator hochqualitative Texte verfasst, die eine außergewöhnlich lange Lesedauer provozieren, erfasst und honoriert das System diese Qualität unzweifelhaft über die Vektoren dwell_time und dwell_score.[4] Im Falle visueller Meisterwerke messen der photo_expand_score sowie der vqv_score für Videodateien die reaktive visuelle Resonanz der Nutzerschaft. Diese Signale fließen als handfeste positive Vektoren in den WeightedScorer ein.

Provokative Inhalte – sogenannter Ragebait – profitieren strukturell zweifellos von der hohen Wahrscheinlichkeit der Generierung eines reply_score, welcher historisch als starkes Interaktionssignal gilt. Jedoch ist die Architektur mit einer vehementen Gegenkopplung ausgestattet. Das System integriert die negativen Feedbacks not_interested_score, block_author_score, mute_author_score und report_score als harte Reduktionsvektoren.[4] Wenn provokantes Material primär in Mutes oder Blocks resultiert, kollabiert der combined_score im WeightedScorer unweigerlich. Die Behauptung, inhaltliche Qualität sei in diesem System sekundär, ist somit auf algorithmischer Ebene unhaltbar. Das System sucht hochkomplex nach tiefgreifender Verweildauer und Interaktion, die signifikant unterhalb der Repulsionsschwelle bleibt.

Ein Abwärtsstrudel, eine "Downward Spiral", kann sich manifestieren, wenn vergangene Publikationen schwach performen. Das System passt die latenten Embedding-Repräsentationen im Machine-Learning-Modell permanent an. Schwache historische Signale senken die zukünftigen Prediction-Wahrscheinlichkeiten für neue Publikationen. Dieser Mechanismus stellt jedoch keine manuell implementierte Strafmaßnahme für "minderwertige" Accounts dar, sondern ist das inhärente, stochastische Wesen adaptiver Empfehlungssysteme.

Abschlussbewertung

Basierend auf der rigorosen und tiefgreifenden architektonischen Überprüfung des xai-org/x-algorithm-Repositories lässt sich die komplexe Debatte um die Reichweitenmechanik auf fünf essenzielle und überprüfbare Einsichten verdichten.

1. Welche Thread-Claims stimmen?

Die architektonische Struktur der Multi-Action-Prediction mit exakt 19 definierten Heads –

welche positive Vektoren wie Replies, Klicks und Verweildauer sowie negative Vektoren wie Mutes integrieren – ist in der Datei weighted_scorer.rs präzise belegt. Die Wahrscheinlichkeit dieser Interaktionen wird berechnet, noch bevor der Content in die Sichtbarkeit gerankt wird, was das Phänomen der "Prediction Trap" für Accounts mit schwachen Embeddings bestätigt. Die exponentielle Strafe durch den AuthorDiversityScorer, charakterisiert durch die Formel multiplier = (1 - floor) * decay^position + floor, ist ein unwiderlegbarer Codebefund. Das massenhafte Veröffentlichen in kurzen Intervallen fragmentiert die Sichtbarkeit innerhalb einzelner Feed-Anfragen zwingend. Ebenfalls belegt ist die existenzielle Wirksamkeit von visuellem Boost-Verhalten, manifestiert in spezifischen Prediction Heads für Dwell-Time, Photo-Expansion und Video Quality.

2. Welche Thread-Claims sind überzogen?

Die wiederkehrende These, dass die Followerzahl eines Accounts, author_followers_count, vollständig irrelevant geworden sei, verkennt die Systemarchitektur. Die Zahl wird aktiv aus den Profilen hydratisiert und dem Systemkontext zur Verfügung gestellt, wodurch sie potenziell essenzieller Bestandteil der latenten Transformer-Embeddings ist. Zudem determiniert die schiere Existenz einer Follow-Beziehung fundamental die Aufnahme in das In-Network-Sourcing via Thunder. Die Theorie des totalen 48-Stunden-Wipes, basierend auf dem AgeFilter und MAX_POST_AGE, ist extrem überzogen. Dieser deterministische Cutoff beschränkt die Halbwertszeit von Content ausschließlich für die Ausspielung im personalisierten Home Mixer-Feed, löscht die Daten jedoch nicht aus der globalen Speicherinfrastruktur für Profile oder Suchen. Ebenso ist die Annahme, dass Reposts systematisch "begraben" würden, fehlerhaft formuliert. Reposts durchlaufen komplexe Deduplikationsfilter auf Basis von Bloom-Filtern zur Vermeidung von Redundanzen, erfahren jedoch keine direkte Bestrafung ihrer strukturellen Inhaltsqualität.

3. Welche Thread-Claims sind nicht belegbar?

Jede Aussage zu konkreten Gewichtungen bleibt unbeweisbar. Ob Replies einen höheren numerischen Stellenwert als Likes aufweisen, ist durch die Exklusion der Datei params.rs aus dem Open-Source-Release nicht ermittelbar. Behauptungen, die eine exakte "90%-Impression-Deduction" für Reposts postulieren, basieren auf fiktiven Metriken, da die Deduplikationsfilter binäre Entscheidungen treffen und keine prozentualen linearen Abzüge applizieren. Ebenso entbehrt die Angst vor einer algorithmischen "Dormancy Penalty" für temporäre Posting-Inaktivität jeder Grundlage im offengelegten Source Code, da kein Scorer eine solche zeitbasierte Straffunktion aufweist.

4. Welche Aussagen sollte man öffentlich nicht als Fakt behaupten?

In der öffentlichen Kommunikation verbietet sich die Behauptung, man habe das Gewichtungssystem des X-Algorithmus final entschlüsselt. Das Repository gleicht einer gläsernen Maschine: Man identifiziert die Zahnräder, die Abfolge der Filter und die Kategorien der Vermessung, doch die physikalischen Antriebsparameter – die exakten Weights, Schwellenwerte und zeitlichen Konstanten – sind unsichtbar. Jegliche Aussagen, die absolute

Faktoren, Ratios oder Zeitspannen wie exakt 48 Stunden postulieren, suggerieren fälschlicherweise den Zugang zu verborgenen Konfigurationen und sind somit als unseriöse Spekulation zu werten.

5. Welche Experimente wären nötig, um reale Reichweitenwirkung zu testen?

Um die blinden Flecken des Repository-Releases wissenschaftlich fundiert und datenbasiert zu quantifizieren, wären großangelegte, codeunabhängige A/B-Tests in einer Blackbox-Umgebung erforderlich. Für die Weight-Kalibrierung müssten inhaltlich identische Posts durch strikte Kontrollgruppen veröffentlicht werden, wobei Account A primär Likes und Account B primär Replies orchestriert; die Varianz der Out-of-Network-Impressions würde das verborgene Verhältnis von p::REPLY_WEIGHT zu p::FAVORITE_WEIGHT approximieren. Die exakte Form der Decay-Kurve des AuthorDiversityScorer ließe sich durch hochfrequente Posting-Intervalle analysieren, indem die abnehmende Impression-Rate sequenzieller Posts die Parameter decay und floor offenbart. Der genaue Wert von MAX_POST_AGE könnte durch minuziöses Tracking des stündlichen Impression-Zuwachses abgeleitet werden, indem man den Moment des absoluten horizontalen Einbruchs ohne asymptotische Glättung identifiziert. Schließlich würde die Effizienz der Bloom-Filter-Deduplikation durch den Vergleich eines identischen viralen Reposts und eines nativen Reuploads durch accounts mit massiven Follower-Schnittmengen isoliert messbar.

Referenzen

1. x-algorithm/home-mixer/candidate_pipeline ... - GitHub, Zugriff am April 24, 2026, https://github.com/xai-org/x-algorithm/blob/main/home-mixer/candidate_pipeline/phoenix_candidate_pipeline.rs
2. xai-org/x-algorithm: Algorithm powering the For You feed on X - GitHub, Zugriff am April 24, 2026, https://github.com/xai-org/x-algorithm
3. x-algorithm/README.md at main · xai-org/x-algorithm · GitHub, Zugriff am April 24, 2026, https://github.com/xai-org/x-algorithm/blob/main/README.md
4. Hack the X Algorithm - Growth Guide to Twitter/X, Zugriff am April 24, 2026, https://xalgo.replit.app/
5. x-algorithm/home-mixer/scorers/weighted_scorer.rs at main · xai-org ..., Zugriff am April 24, 2026, https://github.com/xai-org/x-algorithm/blob/main/home-mixer/scorers/weighted_scorer.rs
6. How the X Algorithm Actually Decides What You See in “For You” | by Sezer Ufuk Yavuz, Zugriff am April 24, 2026, https://sezerufukyavuz.medium.com/how-the-x-algorithm-actually-decides-what-you-see-in-for-you-b9d4073e2b45
7. Hydrators - X For You Feed Algorithm - Mintlify, Zugriff am April 24, 2026, https://mintlify.com/xai-org/x-algorithm/implementation/hydrators
8. Filters - X For You Feed Algorithm - Mintlify, Zugriff am April 24, 2026, https://mintlify.com/xai-org/x-algorithm/api/home-mixer/filters

9. Design Decisions - X For You Feed Algorithm - Mintlify, Zugriff am April 24, 2026, https://mintlify.com/xai-org/x-algorithm/concepts/design-decisions
10. Zugriff am Januar 1, 1970, https://raw.githubusercontent.com/xai-org/x-algorithm/main/home-mixer/filters/retweet_deduplication_filter.rs