METEHAN.AI RESEARCH REPORT · PREPRINT TECHNICAL REPORT

"Google Discover Architecture: Clusters, Classifiers, OG Tags, NAIADES -

Metehan Yesilyurt

AI Search & Information Retrieval Researcher · metehan.ai

Published: February 24, 2026 • Canonical: https://metehan.ai/blog/google-discover-architecture/

Google Discover serves content to hundreds of millions of users every day, yet its internal mechanics remain largely opaque. Most SEO guidance about Discover comes from Google's own documentation or anecdotal publisher observations. In this post, I want to share a different perspective: what we can learn by examining the observable telemetry, event naming conventions, and client-side state of the Google App.

\-UPDATED-

Important note: Everything described below reflects what the client-side code reveals at a specific point in time. This is Google. They can change any of these systems, ranking signals, pipeline stages, telemetry counters, feature flags, on the server side at any moment, without any client update. What you read here is a snapshot, not a permanent blueprint. Treat it as a lens into how these systems work today, not a guarantee of how they will work tomorrow.

Where something is an inference rather than a direct observation, I say so. I needed to handle a "great amount of data" and removed many parts.

Think of this as reading the nutrition label on a packaged food. You cannot see the factory, but the label tells you quite a lot about what is inside.

Special thanks to Valentin Pletzer[1]

This post is a summary. I created a full dashboard. It's accessible at metehanai.substack.com's latest post. [2]

The Content Pipeline

Discover's content pipeline can be mapped to several observable stages, each leaving distinct telemetry traces:

1. **Content Ingestion:** Google crawls and indexes content. Entity extraction assigns Knowledge Graph MIDs (`/m/xxxxx`) or (`/g/xxxxx`) to recognized topics.
2. **Structured Data & Open Graph Parsing:** The client-side parser extracts page metadata with a defined priority order: **Schema.org JSON-LD is checked first**, followed by Open Graph tags, then Twitter Card tags, then generic HTML meta tags. This is a hardcoded fallback chain, not a preference. If JSON-LD has the field, OG tags for that field are never reached.
3. **Content Classification:** Content is assigned to cluster types and classified for the feed hierarchy.

4. **Content Filtering** — The system operates two separate filter levels: `filter_collection_status` (publisher/domain level) and `filter_entity_status` (single URL level). These are tracked as Discover-specific streamz metrics at `/client_streamz/android_gsa/discover/app_content/`.
5. **User Interest Matching:** Content entity MIDs are matched against the user's interest profile through the NAIADES personalization system (verified subtypes below).
6. **Ranking (Server-Side):** Ranking happens server-side. The client-side code reveals what data is packaged and sent to the server, but the actual ranking models are not observable from the client.
7. **Feed Assembly:** Content is organized and delivered to the device via gRPC streaming, background WorkManager sync, beacon push, or cache.
8. **Feedback Loop:** User interactions (dismissals, follows, saves) feed back into personalization. Dismissed content is permanently tombstoned.

What is notable here is the *ordering*. The collection-level filter runs *before* interest matching and ranking. This means a publisher blocked at the collection level never even reaches the ranking stage, regardless of how relevant their content might be to a user.

Structured Data & Open Graph: What the Code Actually Parses

Publishers often wonder which meta tags Discover actually uses. The decompiled parser class (`dkpg.java` , self-described as `SchemaOrg{parsedMetatags, jsonLdScripts}`) reveals the exact tags and their priority order.

The critical finding: Schema.org JSON-LD structured data is checked **first** for title, author, and publisher — not Open Graph tags. OG tags are the fallback. This is hardcoded in the fallback chain logic.

Verified Fallback Chains (from `java files`)

Title:

1. Schema.org structured data (`TEXT_TYPE_TITLE`)
2. `og:title` (via `property` attribute)
3. `twitter:title` (via `name` attribute)
4. `title` (generic `name` attribute)

Author:

1. Schema.org structured data (`TEXT_TYPE_AUTHOR`)
2. `author` (via `name` attribute)

Publisher:

1. Schema.org structured data (`TEXT_TYPE_PUBLISHER`)
2. `og:site_name` (via `property` attribute)

Image:

1. `og:image` (via `property` attribute)

2. `twitter:image` (via `name` attribute)
3. `og:image:secure_url` (via `property` attribute)
4. `twitter:image:src` (via `name` attribute)
5. `image` (generic `name` attribute)

Language:

1. Primary language detection (execution-based)
2. `og:locale` (via `property` attribute)
3. `inLanguage` (Schema.org JSON-LD)
4. Hardcoded fallback to `"en"` (conditional on server config flag)

Paywall Classification (from `dkri.java:173–176`, `dkqd.java`)

The system checks paywall status in this exact order:

1. First: `isAccessibleForFree` (Schema.org JSON-LD boolean) — defaults to `true` if absent
2. Then: `article:content_tier` recognized values are exactly three strings:

`"free"` the expected default `"metered"` counted as paywalled * `"locked"` counted as paywalled

If multiple `article:content_tier` values are found on the same page, the code logs a warning: `"More than one content tier found"` (event code 38468). Use only one value.

Blocking Meta Tags (from `dkri.java:304–416`)

Two meta tags halt the pipeline entirely with an exception (`dkma`):

- **`nopagereadaloud`** triggers `DISALLOWED_FOR_READOUT`
- **`notranslate`** triggers `DISALLOWED_FOR_TRANSLATION`

When either is detected as a meta tag, the system throws an error and stops processing that page. If your CMS or translation plugin injects `notranslate` as a meta tag, your content may not enter this parsing pipeline.

JSON/OG Rewrite in Action

DonanimHaber OG Source Code

DonanimHaber SERP Title

Now let's see in Discover below.

DonanimHaber OG Discover

And this is the og:image link: <https://www.donanimhaber.com/images/images/haber/202436/src_340x1912xtaalas-yapay-zek-ciplerinde-devrim-yaratabilir.jpg>

This JPG link placed in only og:image & twitter:image tags, not in the schema.org (Is it proven? No, the other image is 1400x788px wide[3] in schema tag, it's Google, you can decide it.)

Content Filtering: The Two-Level Architecture

Discover's content filtering operates multiple levels, each tracked by its own Discover-specific streamz metric, identified ones are below:

- **Collection level** (`/client_streamz/android_gsa/discover/app_content/filter_collection_status`) — blocks content from a publisher/domain. Parameterized by `reason`.
- **Entity level** (`/client_streamz/android_gsa/discover/app_content/filter_entity_status`) — blocks a single URL. Parameterized by `reason`.

When a user selects "Don't show content from \[Publisher]" in the card menu, this triggers the collection-level filter. A single article that generates enough negative feedback can suppress an entire publication. That reaction applies to all content from that domain, not just the triggering article. Supressing algorithm can extend the publisher-level filtering.

Important caveat: These are client-side telemetry counters. They confirm the filter mechanisms exist and are tracked, but the exact server-side thresholds, recovery mechanisms, and how "reason" values map to user actions are not observable from the client. Google can change these configurations at any time without a client update.

Tombstoning (from `bska.java`)

Dismissed content is permanently recorded with a `tombstoned` boolean field on the content state object. The content state also tracks `stallState`, `lastKnownState` (INSERTED/REMOVED/UNKNOWN), and `purged` — creating a complete lifecycle record. Tombstoned content does not resurface.

The NAIADES Personalization System (from `fiqc.java`)

The code reveals a personalization system called NAIADES with multiple content subtypes, all confirmed as enum values:

Subtype	Enum Value	What It Suggests
SUBTYPE_PERSONAL_UPDATE_MID_BASED_NAIADES	793	Entity/Knowledge Graph-based personalization
SUBTYPE_PERSONAL_UPDATE_QUERY_BASED_NAIADES	792	Search query-based personalization
SUBTYPE_PERSONAL_UPDATE_QUERY_BASED_NAIADES_PERSISTENT_LOGGING	805	Same with persistent logging
SUBTYPE_PERSONAL_UPDATE_RECALL_BOOST	797	Increases retrieval priority from the candidate pool
SUBTYPE_PERSONAL_UPDATE_WPAS	800	Web Publisher Articles Signal
SUBTYPE_PERSONAL_UPDATE_WPAS_PERSISTENT_LOGGING	811	Same with persistent logging
SUBTYPE_PERSONAL_UPDATE_AIM_THREAD_NAIADES	856	AIM (AI Mode) thread-based

`WPAS` (Web Publisher Articles Signal) likely corresponds to Google News Publisher Center registration, meaning content from registered publishers gets a distinct classification in the personalization pipeline. **`RECALL_BOOST`** can suggest increased retrieval priority from the candidate pool, boosting content during retrieval, before ranking. (We can't see server-side configuration, attention please)

Caveat: These are enum names in a content subtype classification system. They confirm the categories exist, but how much weight each subtype carries in ranking is a server-side decision we cannot observe.

Suppression and Counterfactual Experiments

Discover runs counterfactual experiments. The code confirms:

- `SHOW_SKIPPED_DUE_TO_COUNTERFACTUAL` (from `fevu.java`, enum value 16) — content withheld for A/B testing
- `VISIBILITY_REPRESSED_COUNTERFACTUAL` (from `eyxv.java`) — a Visual Element logging state used across the Google App (not Discover-specific) that marks elements deliberately suppressed for experiment measurement
- `background_refresh_rug_pull_count` (from `bupa.java`, under `/client_streamz/android_gsa/discover/`) a Discover-specific counter tracking cases where content was pushed to the feed and then removed during a background refresh. 100% verified

Figure

The "rug pull" counter is particularly notable. It tracks cases where content was delivered to the feed and then retroactively removed. This means Discover can withdraw content that was already in the feed, not just filter it before display.

The Beacon Push System

Most Discover content arrives through pull-based feed requests, but there is also a push channel. The Beacon system allows Google's servers to proactively push content to a user's device.

From the decompiled code (`bqmt.java`), Beacon currently handles **exactly two content types**:

- **Sports scores** (`SportsScoreAmbientDataDocument`) ordinal 0
- **Investment/finance recaps** (`InvestmentRecapAmbientDataDocument`) ordinal 1
- Anything else triggers `"Unsupported BeaconContent type: %s"` and is rejected

Beacon has its own metrics (from `bupa.java`):

```TEXT
/client_streamz/android_gsa/discover/beacon/incoming_sports_notifications_count
/client_streamz/android_gsa/discover/beacon/donated_sports_documents_count
/client_streamz/android_gsa/discover/beacon/dropped_sports_notifications_count
/client_streamz/android_gsa/discover/beacon/appsearch_cleared_count
```

Sports content has **10+ dedicated notification types** (from `fkld.java`): `SPORTS_AWARENESS_NOTIF`, `SPORTS_GAME_CRICKET_MILESTONE_NOTIF`, `SPORTS_BREAKING_NEWS_NOTIF`, `SPORTS_LIVE_ACTIVITY_NOTIF`, `SPORTS_PREGAME_ANALYSIS_AIM_NOTIF`, `SPORTS_LEAGUE_INSIGHTS_AIM_NOTIF`, `SPORTS_STANDINGS_NOTIF`, and more. General breaking news has just one: `BREAKING_NEWS_NOTIF`. The structural investment in sports notification infrastructure is significantly larger. It may share very similar infrastructure with Google News // This part seems dynamic, so it can change anytime.

Freshness Buckets

The code contains time-based bucketing logic (from `bemp.java:215`):

```JAVA
days < 1  → "0_DAYS"
days < 8  → "1_TO_7_DAYS"
days < 15 → "8_TO_14_DAYS"
days < 31 → "15_TO_30_DAYS"
days < 61 → "31_TO_60_DAYS"
days >= 61 → "TAIL"
```

Important correction: In the decompiled code, this bucketing logic appears in a gesture settings context (`GestureSettingsPreferenceFragment`), not in a Discover-specific class. The bucket names and time ranges are confirmed as exact strings, but their direct connection to Discover's content freshness scoring cannot be verified from the client side alone. The bucketing pattern is consistent with how Google typically handles content age, but I cannot prove these specific buckets are used for Discover feed ranking.

13 Cluster Types

Every card in the Discover feed belongs to a cluster. The following cluster type names are observable:

- `neoncluster` the primary content cluster
- `geotargetingstories` location-based stories
- `deeptrends` and `deeptrendsfable` trending topic narratives
- `freshvideos` recent video content
- `mustntmiss` priority/must-read content
- `newsstoriesheadlines` breaking news
- `homestack` widget cards (weather, sports scores)
- `garamondrelatedarticlegrouping` related article groups
- `trendingugc` user-generated trending content
- `signinlure` sign-in prompts
- `iospromo` cross-platform promotion
- `moonstone` an internal-codename cluster

`mustntmiss` suggests there is a priority queue of content the system considers essential to show. `garamondrelatedarticlegrouping` hints that the system can create related-article groupings — combining separate articles under a shared topic heading.

Real-Time Feed Delivery

Discover does not simply fetch a static list of cards. The code reveals a persistent gRPC connection architecture with distinct service endpoints (verified from `ehdf.java`):

- `google.internal.discover.discofeed.feedrenderer.v1.DiscoverFeedRenderer` standard feed with `QueryInteractiveFeed` and `QueryNextPage`
- `google.internal.discover.discofeed.streamingfeedrenderer.v1.DiscoverStreamingFeedRenderer` streaming variant with `QueryStreamingFeed`
- `google.internal.discover.discofeed.actions.v1.DiscoverActions` `UploadActions` and `BackgroundUploadActions`
- `google.internal.discover.discofeed.reactions.v1.DiscoverReactions` `ListReactions`
- `google.internal.discover.discofeed.recommendations.v1.StoryRecommendations`
- `google.internal.discover.discofeed.homestack.v1.DiscoverHomestackFeedRenderer`

What this means for publishers: your content does not wait for the user to pull-to-refresh. The streaming feed renderer keeps a live connection. The server can inject new cards, reorder existing ones, or remove stale content mid-session. The feed is a living stream, not a snapshot.

What This Means for Publishers

Let me be clear about what this analysis is and is not. It is a set of observations about how the Google App's client-side systems are instrumented. It is not a reverse-engineering of server-side ranking algorithms, which remain on Google's servers and are not directly observable.

That said, some practical observations emerge:

- **Schema.org JSON-LD takes priority over OG tags.** The parser checks structured data first for title, author, and publisher. OG tags are the fallback. If you only implement og:tags without JSON-LD markup, you're relying on the second-choice path.
- **Images are essential.** The image fallback chain is five levels deep — the system tries hard to find an image. Use images at least 1200px wide for hero card eligibility.
- **`og:title` is packaged and sent to Google's servers** as part of the ContentMetadata payload. Whether it's a direct ranking input is plausible but unconfirmed from client-side observation alone. Either way, it is part of the data that informs server-side decisions.
- **Collection-level blocking is tracked as a distinct metric.** The `filter_collection_status` counter confirms this mechanism exists at the publisher/domain level. **However, we can only observe the telemetry counter, not the server-side thresholds or recovery mechanisms. Google can change these at any time.**
- **Publisher Center registration creates a distinct signal.** The `WPAS` (Web Publisher Articles Signal) subtype means registered publishers get different classification treatment in the NAIADES personalization system.
- **`article:content_tier` matters.** The parser explicitly recognizes `free`, `metered`, and `locked`. Use one value only — multiple values trigger a warning.
- **`notranslate` and `nopagereadaloud` meta tags can halt the parsing pipeline.** If your CMS injects these, you need to run experiments and filter Discover traffic.
- **User dismissals are permanent.** Content is tombstoned and does not resurface.
- **Sports publishers have a structural advantage** in the push notification pipeline. 10+ dedicated notification types vs 1 for general breaking news. Client-side confirms, we can't see server-side configuration.

MY Corrections & Caveats

During fact-checking, several items required correction:

Original Claim	Correction
"Exactly 6 OG tags are parsed"	The parser handles 6 OG tags but also parses twitter:image, twitter:title, twitter:image:src, author, title, inLanguage, isAccessibleForFree, image (generic), and Schema.org JSON-LD. Total parsed tags is significantly more than 6
`EVERGREEN_VIBRANT` is a content classification type	It is a **UI color palette name** in `XgadsContext` for theming, alongside CANDY_VIBRANT, GLACIER_VIBRANT, LEMON_VIBRANT etc. Not a content type
`engagement_time_msec` is a Discover-specific engagement signal	It is a **standard Firebase Analytics (GA4) parameter** (`_et` on the wire), used by every app with Firebase Analytics. It measures app-level engagement, not article-level engagement
`freshness_delta_in_seconds` / `staleness_in_hours` are Discover content metrics	In the decompiled code, these appear in **Smartspace weather/air quality metrics**, not in Discover-specific classes
Freshness buckets (1_TO_7_DAYS etc.) are Discover-specific	The bucket strings exist but appear in a **gesture settings** context, not a confirmed Discover class
`VISIBILITY_REPRESSED_COUNTERFACTUAL` is Discover-specific	It is a **Google-wide Visual Element logging state** shared across the entire Google App (Assistant, Lens, Search, etc.)
`isCollectionHiddenFromEmberFeed` is about Discover feed filtering	It is about the **Ember tab** (a separate visual image discovery tab in the Google App), not the Discover feed
`PCTR_MODEL_TRIGGERED` confirms a Discover pCTR model	**Not found** in this SDK version. This doesn't mean it doesn't exist. It may be in server-side config or a different SDK version
OG tags are the primary parsing target	**Schema.org JSON-LD structured data is checked first** for title, author, and publisher. OG tags are the fallback. The OG tags have been in action for a long time.

Methodology Note

All findings in this analysis are derived from decompiling Google App 87,498 classes across 13 DEX files. Of these, 95.5% were obfuscated under package `p000/` with no ProGuard mapping file available.

Deobfuscation was performed through string literal analysis, class hierarchy tracing, gRPC endpoint extraction, and Hilt dependency injection graph reconstruction. Where findings are confirmed via exact string matches in decompiled source, they are labeled as verified. Where findings are inferences based on naming conventions or code proximity, that is noted.

No server-side systems were accessed. Server-side ranking models, experiment allocations, and pipeline stages can all change independently of the client. What we can observe is the instrumentation; the questions the system asks and the answers it records, which reveal the architecture even as the parameters shift underneath.

Google's discovery systems increasingly feed AI Mode too — here are the best AI Mode rank trackers[4] if you want to measure your presence there.

Schema placement matters for discovery systems too — here's where to put schema code on your website[5].

Google Discover architecture is a useful AEO analogy. The same cluster-and-classifier logic shapes which pages earn AI citations in ChatGPT, Perplexity and Gemini. Answer engine optimization borrows from Discover's mental model.

REFERENCES & SOURCES

[1] **Special thanks to Valentin Pletzer —** https://www.linkedin.com/in/valentinpletzer/

[2] **This post is a summary. I created a full dashboard. It's accessible at metehanai.substa... —** https://metehanai.substack.com

[3] **1400x788px wide —** https://www.donanimhaber.com/images/images/haber/202436/1400x788taalas-yapay-zek-ciplerinde-devrim-yaratabilir.jpg

[4] **best AI Mode rank trackers —** https://metehan.ai/articles/best-ai-mode-rank-tracker/

[5] **where to put schema code on your website —** https://metehan.ai/articles/where-to-put-schema-code-in-website-best-practices-for-modern-seo-and-ai-discovery/