Reverse-Engineering Google AI Mode: What Discovery Engine Reveals About Google's AI Search Architecture

Metehan Yesilyurt

AI Search & Information Retrieval Researcher · metehan.ai

Published: November 26, 2025 • Canonical: https://metehan.ai/blog/reverse-engineering-google-ai-mode/

Let me be clear upfront: I'm not claiming Discovery Engine **is** exposing the full architecture of Google AI Mode & AIO. They're different products serving different purposes.

But here's what I am saying: When it comes to AI-powered search with a proper algorithm and tech stack, Google is clearly ahead. And they're not hiding their technology. They're selling it.

Google's Discovery Engine is a product with a proper UI & API endpoints.

Click to see the Discovery Engine documentation here.[1]

Click to see the Discovery Engine's full aspects here.[2] [HexDocs link is here, I know it sounds familiar :)]

Before starting, let me show you a sneak peek at what's inside.

.... "description": "Optional. A unique identifier for tracking visitors. For example, this could be implemented with an HTTP cookie, which should be able to uniquely identify a visitor on a single device. This unique identifier should not change if the visitor logs in or out of the website. This field should NOT have a fixed value such as `unknown_visitor`. This should be the same identifier as UserEvent.user_pseudo_id and CompleteQueryRequest.user_pseudo_id The field must be a UTF-8 encoded string with a length limit of 128 characters. Otherwise, an `INVALID_ARGUMENT` error is returned.", "type": "string" }, "contentSearchSpec": { "description": "A specification for configuring the behavior of content search.", "$ref": "GoogleCloudDiscoveryengineV1SearchRequestContentSearchSpec" }, "rankingExpression": { "description": "Optional. The ranking expression controls the customized ranking on retrieval documents. This overrides ServingConfig.ranking_expression. The syntax and supported features depend on the `ranking_expression_backend` value. If `ranking_expression_backend` is not provided, it defaults to `RANK_BY_EMBEDDING`. If ranking_expression_backend is not provided or set to `RANK_BY_EMBEDDING`, it should be a single function or multiple functions that are joined by \"+\". *ranking_expression = function, { \" + \", function }; Supported functions:* double *relevance_score* double *dotProduct(embedding_field_path) Function variables:* `relevance_score`: pre-defined keywords, used for measure relevance between query and document. *`embedding_field_path`: the document embedding field used with query embedding vector.* `dotProduct`: embedding function between `embedding_field_path` and query embedding vector. Example ranking expression: If document has an embedding field doc_embedding, the ranking expression could be `0.5 * relevance_score + 0.3 * dotProduct(doc_embedding)`. If ranking_expression_backend is set to `RANK_BY_FORMULA`, the following expression types (and combinations of those chained using + or *operators) are supported:* `double` *`signal`* `log(signal)` *`exp(signal)`* `rr(signal, double \u003e 0)` -- reciprocal rank transformation with second argument being a denominator constant. *`is_nan(signal)` -- returns 0 if signal is NaN,*

1 otherwise. `fill_nan(signal1, signal2 | double)` -- if signal1 is NaN, returns signal2 | double, else returns signal1. Here are a few examples of ranking formulas that use the supported ranking expression types: - `0.2 * semantic_similarity_score + 0.8 * log(keyword_similarity_score)` -- mostly rank by the logarithm of `keyword_similarity_score` with slight `semantic_smilarity_score` adjustment. - `0.2 * exp(fill_nan(semantic_similarity_score, 0)) + 0.3 * is_nan(keyword_similarity_score)` -- rank by the exponent of `semantic_similarity_score` filling the value with 0 if it's NaN, also add constant 0.3 adjustment to the final score if `semantic_similarity_score` is NaN. - `0.2 * rr(semantic_similarity_score, 16) + 0.8 * rr(keyword_similarity_score, 16)` -- mostly rank by the reciprocal rank of `keyword_similarity_score` with slight adjustment of reciprocal rank of `semantic_smilarity_score`. The following signals are supported: *`semantic_similarity_score`: semantic similarity adjustment that is calculated using the embeddings generated by a proprietary Google model. This score determines how semantically similar a search query is to a document.* `keyword_similarity_score`: keyword match adjustment uses the Best Match 25 (BM25) ranking function. This score is calculated using a probabilistic model to estimate the probability that a document is relevant to a given query. *`relevance_score`: semantic relevance adjustment that uses a proprietary Google model to determine the meaning and intent behind a user's query in context with the content in the documents.* `pctr_rank`: predicted conversion rate adjustment as a rank use predicted Click-through rate (pCTR) to gauge the relevance and attractiveness of a search result from a user's perspective. A higher pCTR suggests that the result is more likely to satisfy the user's query and intent, making it a valuable signal for ranking. *`freshness_rank`: freshness adjustment as a rank* `document_age`: The time in hours elapsed since the document was last updated, a floating-point number (e.g., 0.25 means 15 minutes). *`topicality_rank`: topicality adjustment as a rank. Uses proprietary Google model to determine the keyword-based overlap between the query and the document.* `base_rank`: the default rank of the result", "type": "string" }, "rankingExpressionBackend": { "description": "Optional. The backend to use for the ranking expression evaluation.", "type": "string", "enumDescriptions": ["Default option for unspecified/unknown values.", "Deprecated: Use `RANK_BY_EMBEDDING` instead. Ranking by custom embedding model, the default way to evaluate the ranking expression. Legacy enum option, `RANK_BY_EMBEDDING` should be used instead.", "Deprecated: Use `RANK_BY_FORMULA` instead. Ranking by custom formula. Legacy enum option, `RANK_BY_FORMULA` should be used instead.", "Ranking by custom embedding model, the default way to evaluate the ranking expression.", "Ranking by custom formula."], "enumDeprecated": [false, true, true, false, false], "enum": ["RANKING_EXPRESSION_BACKEND_UNSPECIFIED", "BYOE", "CLEARBOX", "RANK_BY_EMBEDDING", "RANK_BY_FORMULA"] }, "safeSearch": { "description": "Whether to turn on safe search. This is only supported for website search.", "type": "boolean" }, "userLabels": { "description": "The user labels applied to a resource must meet the following requirements: *Each resource can have multiple labels, up to a maximum of 64.* Each label must be a key-value pair. *Keys have a minimum length of 1 character and a maximum length of 63 characters and cannot be empty. Values can be empty and have a maximum length of 63 characters.* Keys and values can contain only lowercase letters, numeric characters, underscores, and dashes. All characters must use UTF-8 encoding, and international characters are allowed. *The key portion of a label must be unique. However, you can use the same key with multiple resources.* Keys must start with a lowercase letter or international character. See Google Cloud Document[3] for more details.", "type": "object", "additionalProperties": { "type": "string" } }, "naturalLanguageQueryUnderstandingSpec": { "description": "Optional. Config for natural language query understanding capabilities, such as extracting structured field filters from the query. Refer to this documentation[4] for more information. If `naturalLanguageQueryUnderstandingSpec` is not specified, no additional natural language query understanding will be done.", "$ref":

"GoogleCloudDiscoveryengineV1SearchRequestNaturalLanguageQueryUnderstandingSpec" }, "searchAsYouTypeSpec": { "description": "Search as you type configuration. Only supported for the IndustryVertical.MEDIA vertical.", "$ref": "GoogleCloudDiscoveryengineV1SearchRequestSearchAsYouTypeSpec" }, "displaySpec": { "description": "Optional. Config for display feature, like match highlighting on search results.", "$ref": "GoogleCloudDiscoveryengineV1SearchRequestDisplaySpec" }, "crowdingSpecs": { "description": "Optional. Crowding specifications for improving result diversity. If multiple CrowdingSpecs are specified, crowding will be evaluated on each unique combination of the `field` values, and max_count will be the maximum value of `max_count` across all CrowdingSpecs. For example, if the first CrowdingSpec has `field` = \"color\" and `max_count` = 3, and the second CrowdingSpec has `field` = \"size\" and `max_count` = 2, then after 3 documents that share the same color AND size have been returned, subsequent ones should be removed or demoted.", "type": "array", "items": { "$ref": "GoogleCloudDiscoveryengineV1SearchRequestCrowdingSpec" } }, "session": { "description": "The session resource name. Optional. Session allows users to do multi-turn /search API calls or coordination between /search API calls and /answer API calls. Example #1 (multi-turn /search API calls): Call /search API with the session ID generated in the first call. Here, the previous search query gets considered in query standing. I.e., if the first query is \"How did Alphabet do in 2022?\" and the current query is \"How about 2023?\", the current query will be interpreted as \"How did Alphabet do in 2023?\". Example #2 (coordination between /search API calls and /answer API calls): Call /answer API with the session ID generated in the first call. Here, the answer generation happens in the context of the search results from the first search call. Multi-turn Search feature is currently at private GA stage. Please use v1alpha or v1beta version instead before we launch this feature to public GA. Or ask for allowlisting through Google Support team.",

Google Sells Its Search Technology

Google Cloud Discovery Engine (also called Vertex AI Search) is Google's enterprise search product. It's designed for companies that need AI-powered search for their internal data, documentation, product catalogs, or customer-facing applications.

This isn't some obscure API buried in documentation. It's available right now in Google Cloud Console. The first setup is time-consuming. You need to configure data stores, processing pipelines, and ranking controls but it's all there, exposed in the UI.

And Google keeps updating it. New models. More signals. Better configuration options. They're actively developing this product because enterprises pay for it.

Why This Matters for AI Search Optimization

Here's my main point: **We're always looking for hard technical data about how AI search works.**

We reverse-engineer ChatGPT's citation system through browser DevTools. I analyze Perplexity's ranking factors through systematic testing. We build hypotheses from fragments of information.

But with Google? The technical data lives in the Discovery Engine. The signal names. The ranking components. The chunking parameters. The model options. It's documented, configurable, and visible.

I'm not saying AI Mode uses identical code. But Google built the Discovery Engine with Google's engineering team, similar infrastructure, and a similar philosophy that powers their consumer products. When Discovery Engine exposes seven distinct ranking signals with names like "Gecko" and "Jetstream," that tells us something about how Google thinks about AI search ranking.

We can learn from it.

Important Note: The Cloud Console mentions Gecko, but there is also an embedding model named Gecko and it's deprecated now for API users.[5]I will continue mentioning Gecko in this post. Since the Discovery Engine names it "Gecko score."

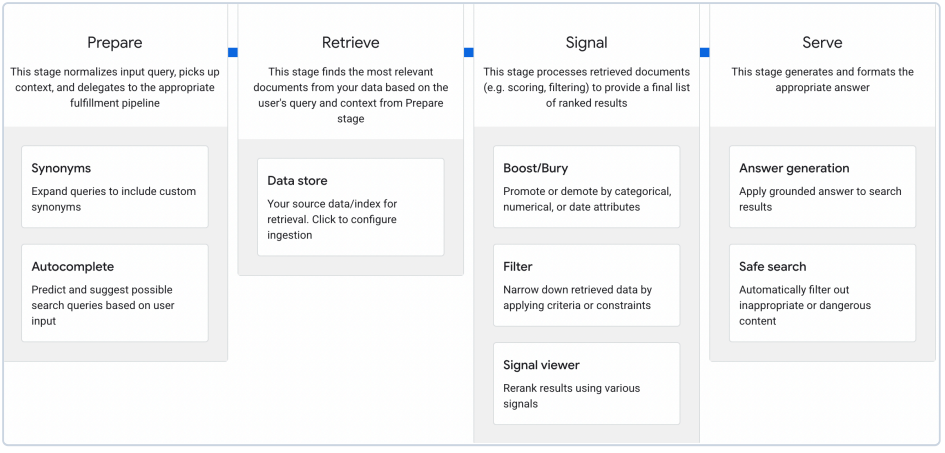
I've been using the Discovery Engine tool for about a year. It keeps getting updated, and even though I use it only for my own website or my clients' websites, I can say it is constantly evolving and improving. Many of the features I discuss in this content were added recently. I also had the opportunity to present a project I conducted through Discovery Engine, along with the insights I uncovered on the main stage at BrightonSEO in April 2025. In the conferences I'll attend in 2026, I plan to dive into much more technical topics. This is only the visible side of the iceberg.

Time flies :)

The Four-Stage Pipeline: Prepare → Retrieve → Signal → Serve

After deep-diving into Discovery Engine's configuration panels, I've mapped the four-stage pipeline Google uses, including the **seven specific ranking signals** exposed in the Signal Viewer.

These are actual configuration options, signal names, and architectural decisions that Google shows to enterprise customers.



Figure

Stage 1: Prepare, Query Understanding and Normalization

What happens here: The system normalizes the input query, picks up context, and routes it to the appropriate fulfillment pipeline.

Synonyms Configuration

Discovery Engine allows adding synonym control lists **with time ranges**. This is significant. It means Google can apply different synonym mappings for different time periods. Think elections, it's adjustable.

Why does this matter? Query understanding isn't static. The meaning of terms evolves, and Google can dynamically adjust how queries are expanded based on temporal context.

Autocomplete Configuration

The autocomplete settings reveal Google's approach to query prediction:

Setting	Default/Options	Implication
Maximum suggestions	Configurable	Controls candidate query diversity
Minimum trigger length	Character-based	Determines when prediction kicks in
Matching order	"Suggestion start with the term" OR

"Suggestion can start from anywhere in the term" | Prefix matching vs. substring matching |

Query suggestions model	Multiple sources (see below)	Different data sources for suggestions
Deny list	Configurable	Explicit blocking of unwanted suggestions

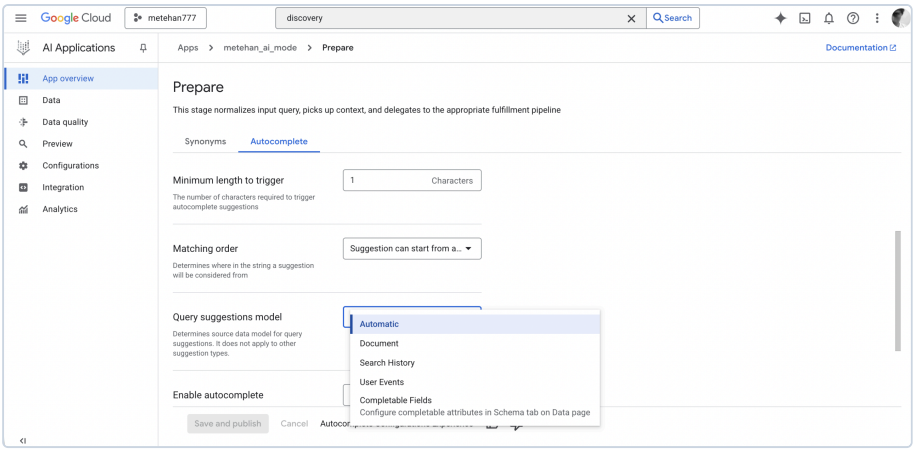
Query Suggestions Model Options:

- **Automatic:** System decides the best source
- **Document:** Suggestions derived from indexed document content
- **Search History:**Suggestions based on past searches
- **User Events:**Suggestions driven by user interaction data
- **Completable Fields:** Suggestions from schema-defined completable attributes

The "matching order" setting is particularly interesting. Google offers two options:

- **Prefix matching:** Suggestions must start with the typed term
- **Substring matching:** Suggestions can match anywhere in the term

This toggle suggests Google's query understanding can operate in both modes. For AI Mode, substring matching would allow more flexible query interpretation, connecting partial inputs to broader concept matches.



Figure

Schema Configuration: How Google Indexes Structured Data

This section will look familiar if you followed Mark Williams-Cook's coverage of the Google ranking leak.[6] Discovery Engine exposes a schema(boolean, array) & prediction configuration panel with field-level controls that mirror what we saw in those leaked documents.

In the Data Store → Website → Schema section, each field can be configured with these attributes:

Attribute	Function
Field Name	The identifier for this data field (e.g., `Product Name`, `Customer ID`, `Author`)
Type	Data type: text, numbers, dates, or boolean (yes/no)
Array	Whether the field holds multiple values (e.g., `["Waterproof", "Durable", "Lightweight"]`) or a single value
Searchable	Improves recall for this field in search queries. Only text fields can be searchable.
Indexable	Enables filtering, ordering, and faceting by this field. Object fields cannot be indexable.
Retrievable	Returns this field in search results

Why this matters:

Google explicitly separates **Searchable**, **Indexable**, and **Retrievable** as distinct properties. This means:

- A field can be **searchable but not retrievable:** it influences ranking but isn't shown to users
- A field can be **indexable but not searchable:** it can be filtered/sorted but doesn't contribute to text matching
- A field can be **retrievable but not indexable:** it's returned in results but can't be used for filtering

This three-way distinction reveals how Google thinks about structured data processing. Your schema markup isn't just "indexed or not", different properties of your structured data serve different functions in the search pipeline.

Optimization Implications

This stage happens *before* your content is even considered. The query gets transformed, expanded, and contextualized through time-aware synonym mappings.

Actionable insight: Your content needs to cover the full semantic field around your target topics across time. Historical terminology and current terminology both matter; Google's time-ranged synonyms mean they're actively managing semantic drift.

Stage 2: Retrieve, Document Selection and Processing

What happens here: The system finds the most relevant documents from the data store based on the processed query and applies document-level processing.

Data Store Configuration

Discovery Engine accepts multiple data sources: Search Console verified websites, downloaded HTMLs, document datasets, and **sitemaps**.

Document Processing: The Chunking Pipeline

This is where it gets technical. Discovery Engine's processing configuration gives some hints on how Google chunks documents for retrieval:

Parser Options:

- Default document parser
- Layout parser[7] with **table annotation** and **image annotation**
- **Gemini enhancement** (Preview) Using LLM to enhance document understanding *(And just wow, for layout parsing, can AI help with it???)*

Chunking Configuration:

- Advanced chunking can be enabled
- Chunk size limit: **500 tokens** (maximum, not fixed)
- Option to **include ancestor headings in chunks**

The 500-token chunk size limit tells us Google Discovery Engine caps retrieval segments at roughly 375 words maximum. Chunks can be smaller, but won't exceed this limit. This suggests content should be structured so that key information is self-contained within ~500 token sections.

The "include ancestor headings in chunks" option is significant when enabled; heading hierarchy travels with each chunk. If a chunk comes from content under H1 > H2 > H3, those headings are preserved as context. This means your heading structure isn't just for readers; it can provide topical context to each retrieved segment. *(WOW!)*

The **Gemini enhancement** option is a preview feature that uses LLM processing during indexing. This suggests Google is already using AI to improve document understanding at the retrieval stage, not just the serving stage.

Optimization Implications

Actionable insights:

1. **Structure content in sections under ~500 tokens.** Each major point should be self-contained within roughly 500 tokens (about 375 words) since that's the maximum retrieval unit size.

2. **Use clear heading hierarchies.** When ancestor headings are included, chunks carry their H1/H2/H3 context. A well-structured heading hierarchy means your chunks are topically labeled.
3. **Optimize tables and images.** Table and image annotation means structured data in tables and descriptive image content are being parsed and indexed.
4. **Submit comprehensive sitemaps.** Google accepts sitemap data for the data store, confirming that sitemap signals influence retrieval.

Stage 3: Signal, The Seven Ranking Signals Revealed

What happens here: Retrieved documents are processed through multiple signal layers to produce a final ranked list. This is where the magic happens and Discovery Engine's **Signal Viewer** exposes the exact signals.

The Seven Ranking Signals

Discovery Engine's Signal Viewer shows a table with these columns for every search result:

Signal	Description	What It Measures
Base Ranking	Initial relevance score from the core ranking algorithm	Pre-adjustment relevance
Embedding Adjustment	Semantic similarity between query and document embeddings	**Gecko score:** Google's embedding model
Semantic Relevance	Cross-attention model score	**Jetstream:** handles context and negation better than embeddings
Keyword Matching	Frequency and relevance of query keywords	**BM25** or a similar algorithm
Predicted Conversion	Likelihood of user engagement	**Three-tier system:** Popularity → PCTR → Personalized PCTR
Freshness	Recency score	Time-sensitive query adjustment
Boost/Bury	Manual business rule adjustment	Explicit promotion/demotion

Figure

This is the actual ranking stack. Let me break down what each signal means for optimization:

Signal 1: Base Ranking

The initial relevance score before any adjustments. This is likely Google's traditional ranking algorithm output, the foundation that everything else modifies.

Signal 2: Embedding Adjustment (Gecko Score)

Google's **Gecko** embedding model measures semantic similarity between the query and document embeddings. This is the vector similarity score.

Optimization implication: Pure keyword matching isn't enough. Your content needs to be semantically aligned with query intent at the embedding level.

Signal 3: Semantic Relevance (Jetstream)

This is huge. Google uses a **cross-attention model called Jetstream** that "better understands context and negation compared to embeddings."

Good news: Jetstream is open-source. https://github.com/AI-Hypercomputer/JetStream[8]

Interesting news: It's designed for TPUs.

Cross-attention models process query and document together, allowing for nuanced understanding that pure embedding similarity misses. The explicit mention of **negation handling** suggests Jetstream can understand "not X" and "without Y" patterns that embeddings struggle with. *(MUVERA???)*

Optimization implication: Write content that explicitly addresses what something **is** and what it **isn't**. Negation-aware ranking means clarifying distinctions matters.

Signal 4: Keyword Matching (BM25)

Classic keyword frequency and relevance scoring, likely BM25 or a variant. This confirms that traditional keyword optimization isn't dead; it's one of seven signals in the stack.

Optimization implication: Keywords still matter. They're not the only signal, but they're explicitly in the ranking stack.

Signal 5: Predicted Conversion (PCTR/PCVR)

It's very interesting because Google uses the word "prediction" with a scoring in a public service.

This is the engagement signal. **PCTR** (Predicted Click-Through Rate) and **PCVR** (Predicted Conversion Rate) are calculated from historical user interaction data.

Discovery Engine's **Data Quality** section reveals this isn't a single signal It's a **three-tier system** with quality thresholds:

TIER 1: Popularity Derived from user interactions across documents. The more users interact with a document, the stronger the boost. Key details:

- Aggregated across all datastores with events in the Google Cloud project
- If one datastore has sufficient events, the signal is enabled project-wide
- BUT: Documents in datastores without events don't get boosted, even if the project-level threshold is met

TIER 2: PCTR Model (Predicted Click-Through Rate) Predicts the probability of a user clicking on a document given the context. Google explicitly states this is "an important factor considered in ranking."

- App-specific (tied to a particular search application)
- Only counts events from datastores linked to the current app
- Has minimum and optimal thresholds for data quality

TIER 3: Personalized PCTR Model Incorporates user-specific signals: user metadata, individual search history, and behavioral patterns.

- Only activates after **100,000+ queries** have been served
- User-specific, not just document-specific
- Represents the highest tier of engagement signal sophistication

Each tier shows **Optimal**, **Suboptimal**, and **Blocking** status, meaning there are hard thresholds where the signal either works well, works partially, or is disabled entirely due to insufficient data.

Optimization implication: Engagement isn't just one signal. It's a layered system. Basic popularity comes first, then predicted CTR modeling, then personalized predictions. Sites with more user interaction data unlock higher tiers of ranking signals.

Signal 6: Freshness

Recency scoring is adjusted based on query type. Discovery Engine notes this is "especially important for time-sensitive queries."

Optimization implication: Freshness is query-dependent. For time-sensitive topics, recent content gets boosted. For evergreen topics, freshness may matter less.

Signal 7: Boost/Bury

Manual adjustment from -1 to +1 based on business rules. The configuration interface allows:

- Selecting a data store
- Applying filters using AND/OR/NOT operators with grouping via parentheses
- Exact phrase matching with quotes
- Slider adjustment from Bury (-1) to Boost (+1)

Optimization implication: Google can (and does) apply manual ranking adjustments based on categorical rules. This is likely how E-E-A-T categories and source authority are implemented, as boost/bury rules on source attributes.

How These Signals Combine

The Signal Viewer shows all seven signals as separate columns, with Position as the final output. This means Google is using **multi-signal fusion,**likely a learned combination of all seven signals to produce the final ranking.

This is consistent with what I've found in other AI search systems. ChatGPT uses Reciprocal Rank Fusion (RRF)[9]. Perplexity uses 59+ factors in an XGBoost reranker[10]. Google's approach appears to be a seven-signal fusion model.

Stage 4: Serve, Answer Generation and Content Controls

What happens here: The system generates and formats the final answer, applying LLM synthesis and safety filtering.

Search Type Configuration

Discovery Engine offers three search types that **map conceptually to Google's consumer products**:

Discovery Engine Setting	Likely Google Consumer Equivalent
Search - list with results	Traditional Google Search
Search with an answer	AI Overviews
Search with follow-ups	AI Mode

The naming and functionality alignment is too close to be coincidental. "Search with follow-ups" describes exactly what AI Mode does. A conversational search experience with multi-turn context.

LLM Configuration

The serving layer allows model selection and customization:

Model Selection: Gemini 2.5 Flash is the default, but any model can be selected. This means Google can swap underlying models without changing the pipeline.

Answer Customization:

- Custom instructions for tone, style, and length
- Language auto-detection or manual selection
- Related questions toggle
- Image source selection for answer images

The "custom instructions" field is significant. It suggests that AI Mode's answer style is prompt-engineered, not hardcoded. Google can adjust how answers are generated through instruction tuning.

Safety and Quality Gates

Several filters act as final gates before serving:

Filter	Function
Ignore no answer summary	Skip "no summary available" messages
Ignore Adversarial Query	Block LLM responses on detected adversarial queries
Ignore low relevant content	Prevent LLM answers when content relevance is too low

The "low relevant content" filter is important. Even if content makes it through the Signal stage, it can be blocked at Serve if the LLM determines it's not relevant enough to ground an answer.

Optimization Implications

Actionable insights:

1. **Optimize for grounded synthesis.** The LLM is constrained to ground answers in retrieved content. Your content needs clear, extractable statements.

2. **Avoid adversarial patterns.** Content that triggers adversarial detection gets filtered. Write straightforwardly without manipulation patterns.
3. **Maintain high relevance density.** Low relevance content gets filtered. Every section should contribute to query relevance.
4. **Structure for multi-turn.** AI Mode supports follow-ups. Content that answers related questions in the same topic cluster may have advantages.

The Complete Picture: What Discovery Engine Tells Us About AI Search

Here's the flow based on Discovery Engine's architecture and likely a close approximation of how Google approaches AI search more broadly:

Stage 1 - Prepare: User query → Time-aware synonym expansion → Autocomplete prediction model → Transformed query with context

Stage 2 - Retrieve: Transformed query → Data store lookup → Chunked retrieval (up to 500 tokens per chunk, optionally with ancestor headings) → Layout-parsed tables/images → Gemini-enhanced document understanding → Candidate set

Stage 3 - Signal: Candidate set → Seven-signal scoring:

1. Base Ranking (core algorithm)
2. Gecko embedding similarity
3. Jetstream cross-attention relevance
4. BM25 keyword matching
5. Engagement signals (Popularity → PCTR → Personalized PCTR tiers)
6. Freshness scoring
7. Boost/Bury business rules → Multi-signal fusion → Final ranked list

Stage 4 - Serve: Top N results → Gemini 2.5 Flash (or selected model) → Custom instruction application → Adversarial/low-relevance filtering → Grounded answer generation → Related questions → Rendered AI Mode response

What This Means for AI Search Optimization

Discovery Engine is Google's enterprise search product built on the same infrastructure and engineering philosophy as their consumer products. The configuration options reveal how Google thinks about AI-powered ranking.

Is this exactly how AI Mode works? I can't say for certain. But when the same company exposes seven named ranking signals, specific chunk sizes, and three search types that align with their consumer product tiers, that's technical intelligence worth studying.

Key Takeaways

1. Seven explicit signals, not a black box

The Signal Viewer exposes seven distinct ranking signals. This isn't a single neural network making opaque decisions. It's a multi-signal fusion system with interpretable components.

2. Gecko + Jetstream = Semantic layer

Google uses both embedding similarity (Gecko) and cross-attention (Jetstream) for semantic understanding. Jetstream's negation handling suggests nuanced query understanding beyond simple similarity.

3. PCTR is a three-tier system, not a single signal

Engagement signals operate in tiers: Popularity → PCTR → Personalized PCTR. Each tier has quality thresholds and unlocks with more user interaction data. Personalized ranking only activates after 100,000+ queries. (And of course, this is about the Discovery Engine)

4. 500-token chunk limit with optional heading context

The retrieval unit has a ~500 token maximum, with an option to include ancestor headings. Structure your content accordingly.

5. Three product tiers from one pipeline

Traditional search, AI Overviews, and AI Mode are all served from the same pipeline with different configurations. The architecture is unified.

6. Adversarial and relevance gates

Final safety filters can block content even after it ranks well. Write naturally and maintain high relevance throughout.

Next Steps

Discovery Engine won't give you the exact weights Google uses in AI Mode. But it gives you something almost as valuable: **a window into how Google engineers think about AI search.**

The seven signals are named. The chunk size limit is defined. The pipeline stages are visible. That's more technical intelligence than we get from any other source.

The question is whether you're paying attention.

Have questions about AI Mode optimization? Follow for more reverse-engineering insights from Google's own products and documentation. Connect with me on X[11] and LinkedIn[12].

To track your own visibility inside AI Mode, see the best AI Mode rank trackers compared[13].

For the measurement side of AI Mode, check AI share of voice benchmarking methods[14], AI search optimization platforms for global marketing teams[15], and the best SEO ranking and reporting software compared[16].

Reverse-engineering Google AI Mode is AEO reconnaissance. The architecture it reveals - retrieval, reranking, citation selection - mirrors what ChatGPT, Perplexity and Gemini do. Answer engine optimization is the practice of designing content for that pipeline.

REFERENCES & SOURCES

[1] **Click to see the Discovery Engine documentation here.** — https://docs.cloud.google.com/generative-ai-app-builder/docs/reference/rest

[2] **Click to see the Discovery Engine's full aspects here.** — https://hexdocs.pm/google_api_discovery_engine/api-reference.html

[3] **Google Cloud Document** — https://cloud.google.com/resource-manager/docs/creating-managing-labels#requirements

[4] **this documentation** — https://cloud.google.com/generative-ai-app-builder/docs/natural-language-queries

[5] **Important Note: The Cloud Console mentions Gecko, but there is also an embedding model ...** — https://ai.google.dev/gemini-api/docs/embeddings?hl=en

[6] **Mark Williams-Cook's coverage of the Google ranking leak.** — https://searchengineland.com/google-exploit-scoring-classifications-449333

[7] **Layout parser** — https://metehan.ai/blog/google-document-ai-layout-parser/

[8] **https://github.com/AI-Hypercomputer/JetStream** — https://github.com/AI-Hypercomputer/JetStream

[9] **Reciprocal Rank Fusion (RRF)** — https://metehan.ai/blog/chatgpt-is-using-reciprocal-rank-fusion-rrf/

[10] **Perplexity uses 59+ factors in an XGBoost reranker** — https://metehan.ai/blog/perplexity-ai-seo-59-ranking-patterns/

[11] **on X** — https://x.com/metehan777

[12] **LinkedIn** — https://www.linkedin.com/in/metehanyesilyurt

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

[14] **AI share of voice benchmarking methods** — https://metehan.ai/articles/ai-share-of-voice-benchmarking-methods/

[15] **AI search optimization platforms for global marketing teams** — https://metehan.ai/articles/ai-search-optimization-platform-for-global-marketing-teams/

[16] **the best SEO ranking and reporting software compared** — https://metehan.ai/articles/best-seo-ranking-reporting-software-tools-compared/