

# 答えは、問いに似た 文書にはなかった。

— 55万件超から **"本当の知恵"** を引き出す RAG の作り方



問い



本当の知恵

## SPEAKER

GMOインターネットグループ株式会社  
グループ研究開発本部  
AI研究開発室 エグゼクティブリード  
片野 道雄

2026.06.29

# 01 このセッションについて



## TOPIC

社内 55万件超のドキュメントから、根拠付きで回答する社内ナレッジ検索システム

GMOブランドセキュリティ社との取り組みで試作した プロトタイプ

## 技術スタック（全体像）



### CLIENT

**Claude Code**

MCP クライアント



### SERVER

**Cloud Run**

MCP Server / SSE



### SEARCH

**Spanner**

ベクトル × 全文の  
ハイブリッド検索



### DISTILLATION

**Vertex AI  
/ Claude Sonnet**

Gemini 2.5 Flash Batch



### INDEX / STATE

**Firestore**

原文



### ORCHESTRATION

**Cloud Workflows**

12 ステップ

## 02 前提：普通の RAG のしくみ



RAG = LLM に "検索結果" を渡して答えさせる

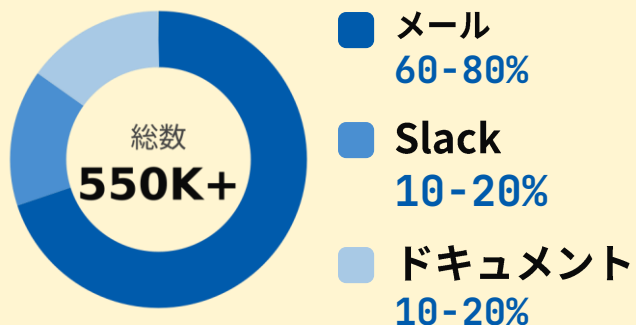


- 1 全文書を LLM に渡せない → 関連分だけ 渡す
- 2 「似ている」の正体 = 埋め込みベクトルのコサイン類似度
- 3 弱点 = 「問いに意味が近いチャンク」しか取れない

## 03 課題：「似た文書」では答えに届かない

社内 55万件超で素朴な RAG を試すと…

### DATA COMPOSITION



1 一番"似た"チャンク ≠ 答え (類似度 ≠ 有用性)

2 答えが 複数文書に分散 → 単一チャンク不完全

3 ヒットしない領域の知見に到達できない

4 メール / Slack は 1件では文脈が薄い

II ベクトルの近傍探索を磨くだけでは、**"本当に使える答え"**にならない。

## THE ANSWER

検索する "対象" を作り変えた

# 生の文書ではなく

# "蒸留したナレッジ"

# の中にあった

**EUREKA!**

わかった！

## 3つの転換



01

多段蒸留

文書をLLMで要点に凝縮  
→ ナレッジカード化



02

そのナレッジを検索

索引対象を生文書から蒸留カードへ転換

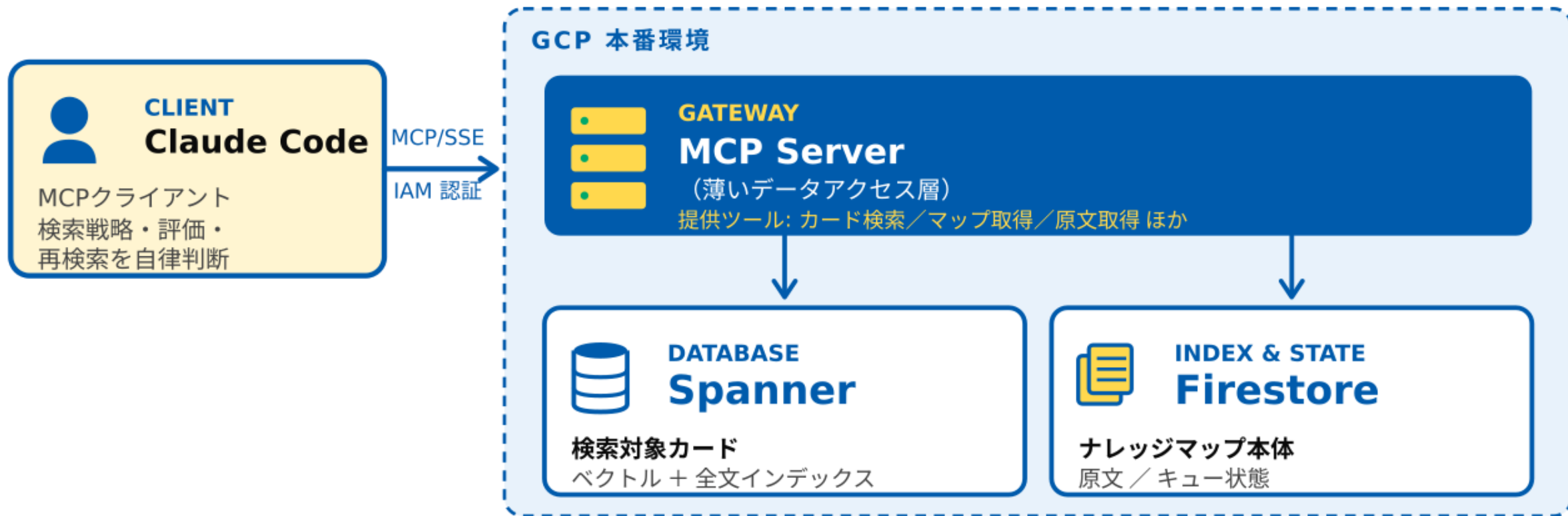


03

ナレッジマップ

全体像の索引をLLMクライアントへ

## 04 システム全体像



### データストアの役割分担（重要）



**Spanner** = 検索が要る部分（蒸留カードのベクトル + 全文インデックス）

**Firestore** = 索引本体・原文・パイプライン状態管理

## 05 工夫①-A データの種類に合わせた蒸留

### ■ 扱うデータは3種類 — メール / Slack / ドキュメント



SOURCE 01

メール

60-80%

⚠ 文脈が薄い

スレッド・往復で意味が成立。  
単体ではノイズになりがち。



SOURCE 02

Slack

10-20%

⚠ 文脈が薄い

会話のまとまり単位で意味が出る。  
単発メッセージはノイズが多い。



SOURCE 03

ドキュメント

10-20%

✓ 単体で文脈成立

そのまま蒸留 OK。



APPROACH

メール・Slack は複数のやり取りを束ねてから蒸留して、意味のあるナレッジに  
ソースの性質に合わせて「蒸留の単位」を変えるのがポイント。

## 06 工夫①-B L1蒸留 — スキーマ強制で構造化

Gemini 2.5 Flash (Batch Prediction) + 構造化出力（スキーマ強制）

### OUTPUT FORMAT



JSON（型保証）

メタデータはスキーマで型を強制



Markdown 本文（柔軟）

本文は自由記述

### 蒸留と同時に付与する構造化情報



01

skip 判定

ノイズ文書を蒸留段階で除外。  
後段に渡す前に間引く。



02

エンティティ抽出

人物 / 会社 / 部署 / PJ /  
製品 / 規則 / 用語 を構造化



03

予兆フラグ

教訓 / 意思決定 / ベストプラ  
クティス / 繰り返しパターン  
/ リスク

★ ①と同じ呼び出しで 同時付与 = 追加コストなし推論 (thinking) は最小限に設定

## 07 工夫①-C L2 蒸留 — クラスタリングで横断知見



**TAKEAWAY** L1 だけでなく、**要約・教訓・索引 (L2) まで事前生成** / 分類体系を自動構築

## 08 工夫② 生の文書でなく "ナレッジ" を検索する

検索インデックスに 載せるものを変えた のが品質の決定打



### BENEFIT 01

#### 1 ヒットの情報密度が高い

複数文書の文脈を 内包。  
1 ヒットで意味のある単位の知見にたどり着く。



### BENEFIT 02

#### 原文はオンデマンド取得

原文は 根拠の裏付け として  
Firestore から必要分のみ取得。

## 09 工夫③ ナレッジマップを LLM クライアントへ



全体像（索引）を2層で持ち、検索の"地図"にする

LAYER 1

Featured Index

保存先

クライアント側キャッシュ

内容

上位カテゴリ／トピック／インサイト／エンティティの索引

LAYER 2

Full Storage

保存先

Firestore

内容

各カードの本体（必要分だけオンデマンド取得）

● セッション開始時に取得 → 常にコンテキスト保持 / サイズはトークン上限でトリミング

### 効果



RECALL 補強

ヒットの外へ自力で到達

関連領域を地図から発見して  
再検索できる



偏りを自己判定

同カテゴリに偏る → 別領域再検索  
クライアント LLM が偏りに気づく



回答統合

検索結果 + 関連カテゴリを統合  
単発検索より厚い回答が作れる

# 10 工夫④ ハイブリッド検索 + RRF 融合

ベクトル（意味）＋フルテキスト（語）を Spanner の SQL 内で 1 つに融合



外部結合 (FULL OUTER JOIN) + RRF 融合

$$\text{RRF} = 1/(\text{k} + \text{rank\_vec}) + 1/(\text{k} + \text{rank\_text})$$

k=60: 標準値

match\_type = both

semantic

keyword

- ▶ 外部結合 + COALESCE で 1 ドキュメント 1 行 に → match\_type を即判定
- ▶ フィルタは 検索と同時に SQL で適用 (NoSQL の絞り込み制約を回避)
- ★ 両方でヒット = 信頼度高 (both) ベクトルは固有名詞に弱い → 全文で補完

# 11 工夫⑤ Agentic Search で "網羅性" を底上げ

従来 RAG が弱い **recall** を、クライアントの自律判断で補う

## ■ マルチレベルクエリ（1 質問を 3 つの粒度で並列検索）



LEVEL 1

自然言語

質問文そのまま

→ 質問の意味に近いカード



LEVEL 2

キーワード

固有名詞・専門用語

→ 全文検索にピンポイント一致



LEVEL 3 ★

擬似回答

"答えに含まれそうな断定文"

→ 発見力 最高

擬似回答は LLM 呼び出しなしでクライアント生成

単一レベルでは届かないカードを **1 問 6~10 件 追加発見**（検証）

## ■ 自律再検索 ループ



① 検索

マルチレベル並列実行

3つの粒度で候補を集める



② 評価

カバレッジ／偏りを判定

ナレッジマップを地図に



③ 不足なら再検索

AI が探し直す

探索型はほぼ全問発動

# 12 工夫⑥ 大規模分散パイプライン

55 万件超を 現実的なコスト・時間 で。Cloud Workflows 12 ステップ

## APPROACH 01



### MapReduce 的アプローチ

抽出: 中間ファイルをバイト範囲で分割 → Worker が並列処理

インサイト発見: 候補を再帰的に map-reduce マージ + チェックポイント

## APPROACH 02



### キュー + 冪等性 (Firestore で状態管理)

アイテム単位の状態管理 / "ゾンビ"検出 (古い処理中を戻す)

再開ポイント指定で途中再開 / 並列度パラメータ / 分散ロック

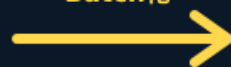
設計の学び — Embedding 生成のボトルネックを潰した  
Embedding 生成も Batch 化 → レート制限を撤廃

BEFORE

最大 3~4日

試算 / レート制限でボトルネック

Batch化



AFTER

約 3~4時間

同じデータ件数で計測

処理時間  
大幅に短縮

# 13 工夫⑦ 低コスト化 — モデルとストアの使い分け

## ■ タスクの難易度でモデルを使い分け、過剰品質を避ける

L1 蒸留（全件）

Gemini 2.5 Flash **Batch**

抽出・要約。Batch で 50%オフ・レート管理不要

L2 カテゴリ／トピック

Gemini 2.5 Flash

分類体系は確定済み → 推論力不要

インサイト発見／蒸留

Claude Sonnet

横断洞察・知識統合は 品質重視

## ■ データストアの使い分け / Spanner の構成



### SPANNER

検索性能・インデックスが要る部分

蒸留カードのベクトル + 全文インデックス



### FIRESTORE

辞書引き・状態管理で十分な部分

L2 索引本体／原文／キュー状態

Spanner Enterprise エディション／100 PU（最小構成）で運用

ベクトル検索は Enterprise 必須（Standard 不可）／100 PU ≒ 月約 2 万円の固定費



月額運用 試算

数万円規模／Spanner の固定費が主

# 14 工夫⑧ 画面を捨てて MCP に振り切った

UI を作らず、AI クライアント + MCP サーバー というモダン構成



## SETUP

セットアップは MCP サーバー登録だけ  
独自 UI ・ 認証画面 ・ 履歴 DB が 不要



## CONTEXT

200K コンテキストをフル活用  
地図 + 検索結果 + ガイドライン + 履歴を 全部載せ



## GUIDELINES

回答ガイドラインも MCP Resources で配信  
更新は デプロイのみで全員反映



## AUTH

Cloud Run IAM に寄せる  
自作プロキシ が IAM トークンを自動付与

## 自作プロキシが要る理由

Claude Code は Google の ID トークンを自前で発行・更新できない。  
ローカルプロキシが 毎リクエストに IAM トークンを付与 → 独自の認証基盤を作らず IAM に寄せられる。



## DESIGN DECISION

サーバーは "薄いデータアクセス層"。賢さはクライアント側に寄せる。

## 15 検証で見たもの



根拠提示

**約10件/問**

全問インライン引用／  
構造化見出し 100%



MCP ツール活用

**1 問で複数併用**

検索 + インサイト + トピック +  
カテゴリ + マップ



Agentic vs 単発

**全評価軸で上回る**

カテゴリ網羅性・多角性  
(LLM-as-judge)



自律再検索

**ほぼ全問発動**

探索型クエリで偏りを  
自動補正



検索レイテンシ

**数秒** / 検索

再検索時は複数ラウンド



運用コスト

**月 数万円**

試算 (Spanner固定費が主)

※ いずれも **プロトタイプ** での検証・試算



USER VALUE

**引用付きで答える = 根拠が追える**

## ■ SUMMARY —では、どうしたか

# 答えは、問いに似た文書にはなかった



01

多段蒸留してナレッジ化

スキーマ強制 + 構造化抽出  
+ クラスタリング



02

そのナレッジを検索

検索対象の転換 = 品質の  
決定打



03

ナレッジマップを LLM へ

recall と自律判断の源

+ ハイブリッド検索 / RRF で precision、Agentic Search で recall

+ MapReduce 的パイプライン で大量処理を安く・確実に

"似たものを返す検索" から、"考えて答えるシステム" へ。

参考 ハイブリッド検索 (RRF) の詳細は技術ブログで公開中

<https://recruit.group.gmo/engineer/jisedai/blog/rag-spanner-hybrid-vector-fulltext-search/>

すべての人にインターネット

**GMO**