The Intelligent Support Hub is an AI-driven ticket triage platform that augments support operations through:

- Semantic similarity search (context > keywords)

- Automated solution recommendation with confidence scoring

- Category and priority inference

- Infrastructure built entirely on serverless GCP primitives (BigQuery + Vertex AI)

**High-Level Data & Processing Flow**

User / Incoming Ticket  ──►  Combine title + description

                                                              │

                                                             ▼

       Generate embedding (text-embedding-004 via BigQuery remote model)

                                                              │

                                                             ▼

     VECTOR\_SEARCH against precomputed ticket embeddings

                                                             │

      ┌─────────────────────┴─────────────────────┐

Similar Ticket Retrieval                 Recommendation / Categorization

            │                                           │

           ▼                                           ▼

Return structured JSON                 Confidence scores + exemplar tickets

**GCP Resources & Configuration**

|  |  |  |
| --- | --- | --- |
| **Resource Type** | **Name / Pattern** | **Purpose** |
| Project | `big-query-project-476503` | Hosting all artifacts |
| Dataset | `support\_tickets` | Logical storage for tables & model |
| Table | `tickets` | Raw + enriched ticket records |
| Table | `tickets\_with\_embeddings` | Raw ticket columns + embedding vector |
| Remote Model | `support\_tickets.textembedding\_model` | Wrapper to Vertex AI embedding endpoint |
| Connection | `<project>.US.vertex-ai` | Secure remote inference channel |

**Data Model Details**

Primary logical entity: Support Ticket.

Key derived column: `combined\_text` = `title + ' ' + description` used for embedding generation ensuring both context layers are represented.

Embedding Table Schema Enhancement:

- All original ticket columns preserved for downstream analytics.

- Embedding column: `embedding` (ARRAY<FLOAT64>, length 768) produced via `ML.GENERATE\_EMBEDDING`.

**Embedding Generation Strategy**

We precompute embeddings for all historical tickets instead of generating them ad hoc at query time.

Advantages:

- Reduced latency (search only needs to embed the query text)

- Predictable cost profile

- Allows for batch index creation in future

SQL pattern (implemented programmatically):

```sql

CREATE OR REPLACE TABLE `support\_tickets.tickets\_with\_embeddings` AS

WITH embeddings AS (

    SELECT content, ml\_generate\_embedding\_result AS embedding

    FROM ML.GENERATE\_EMBEDDING(

        MODEL `support\_tickets.textembedding\_model`,

        (SELECT ticket\_id, combined\_text AS content FROM `support\_tickets.tickets`)

    )

)

SELECT t.\*, e.embedding

FROM `support\_tickets.tickets` t

JOIN embeddings e ON t.combined\_text = e.content;

```

**Semantic Search Mechanics**

We leverage BigQuery's `VECTOR\_SEARCH` function.

Conceptual form:

```sql

SELECT base.ticket\_id, distance

FROM VECTOR\_SEARCH(

    TABLE `support\_tickets.tickets\_with\_embeddings`,

    'embedding',

    (SELECT ml\_generate\_embedding\_result AS embedding FROM ML.GENERATE\_EMBEDDING(...)),

    distance\_type => 'COSINE',

    top\_k => 10

);

```

Distance → Similarity: `similarity = (1 - distance) \* 100` (heuristic scaling for interpretability).

**Recommendation Algorithm**

1. Retrieve top-N similar tickets (larger N for statistical stability).

2. Filter: `status = 'resolved' AND resolution IS NOT NULL AND satisfaction\_score >= threshold`.

3. Aggregate by resolution text:

   - frequency

   - avg satisfaction

   - avg resolution time

   - avg similarity

4. Score formula:

```

confidence = round(

    (frequency \* 0.3 + avg\_satisfaction \* 15 + avg\_similarity \* 0.5)

    / (greatest(avg\_resolution\_time, 1) \* 0.1)

, 2)

```

This balances popularity, quality, relevance, and operational efficiency.

**Categorization & Priority Inference**

Approach: Nearest-neighbor voting.

- Gather top-K similar tickets.

- Compute vote counts per `category` and per `priority`.

- Confidence = `(votes / K) \* 100`.

- Also returns average similarity metric for transparency.

**End-to-End Setup Steps**

**### Local Environment**

Powershell:

git clone <repo-url>

cd "Intelligent Support Hub"

python -m venv .venv

.\.venv\Scripts\activate

pip install -r requirements.txt

**### GCP Initialization (one-time)**

1. Enable APIs: BigQuery, Vertex AI.

2. Create dataset: `support\_tickets` (can be done via console or CLI).

3. Grant IAM roles to service account.

4. Place credentials JSON and set `GOOGLE\_APPLICATION\_CREDENTIALS` (or use gcloud auth).

**### Application Bootstrap**

python main.py --setup  # Creates tables, loads sample, builds embeddings, ensures model

**### Functional Usage**

python main.py --search "password reset not working"

python main.py --recommend "cannot login to account"

python main.py --categorize "payment failed at checkout"

**Testing Strategy**

- Smoke: Validate initialization, dataset presence, row counts, simple search.

- Functional: Assert recommendation returns non-null for known queries.

- Edge Cases:

  - Empty text → should return no similar tickets gracefully.

  - Very short text → still embeds; similarity may degrade.

  - High similarity threshold → may produce zero recommendations.