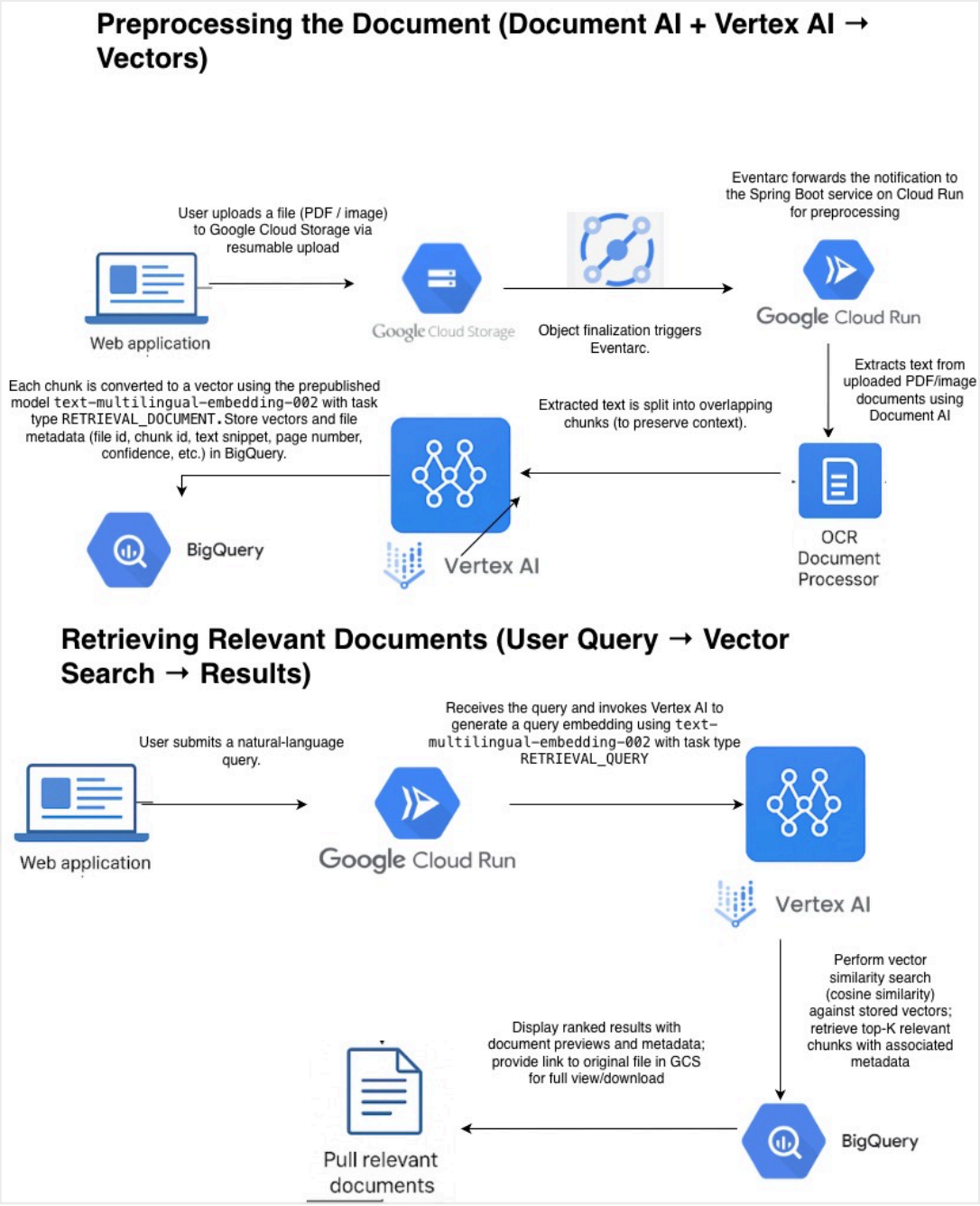
AI-Powered Personal Document Assistance System

Overview

The **AI-Powered Personal Document Assistance System** allows users to securely upload private documents (PDFs and images) and later retrieve relevant content using **natural language queries**.
Built entirely on **Google Cloud Platform (GCP)**, the system employs a combination of **Document AI**, **Vertex AI**, **Cloud Run**, and **BigQuery** to perform intelligent text extraction, embedding, and retrieval through **vector similarity search**.

This architecture ensures **data privacy, scalability, and multilingual adaptability** by leveraging Google's advanced AI models and server-less infrastructure.

System Architecture



Core Flow:

Upload & Storage

- Users upload documents through a web interface.
- Files are transferred to **Google Cloud Storage (GCS)** via **Resumable Upload** for reliability and large-file support.
- Upon successful upload, GCS triggers an **Eventarc event** (OBJECT_FINALIZE)

to initiate downstream processing.

Document Preprocessing (Cloud Run + Document AI)

- **Eventarc** forwards the event to a **Spring Boot microservice** deployed on **Cloud Run**.
- The microservice retrieves the uploaded document from GCS and invokes **Document AI (OCR Processor)** to extract the full text content.

Text Chunking & Vector Embedding

- Extracted text is segmented into **overlapping chunks** (typically 300–500 tokens with 20–30% overlap) to preserve contextual continuity.
- Each chunk is embedded into a **vector representation** using **Vertex AI's pre-trained model**:
  text-multilingual-embedding-002 with task type RETRIEVAL_DOCUMENT.
- These embeddings, along with metadata (file ID, filename, chunk index, text snippet, etc.), are stored in **BigQuery**.

Query & Retrieval

When a user submits a natural-language query:

- The query text is embedded using the same **Vertex AI model** (task_type = RETRIEVAL_QUERY).
- **BigQuery** executes a **vector similarity search** (cosine distance) to identify the most relevant text chunks.

Retrieved results are ranked, aggregated, and displayed on the web interface with **contextual previews** and links to the original files.

Detailed Component Architecture

Component	Technology	Description
Frontend UI	HTML5, CSS, JavaScript (Thymeleaf)	Web interface for document upload, progress display, and querying.
Storage Layer	**Google Cloud Storage (GCS)**	Stores uploaded files; triggers Eventarc when files are finalized.

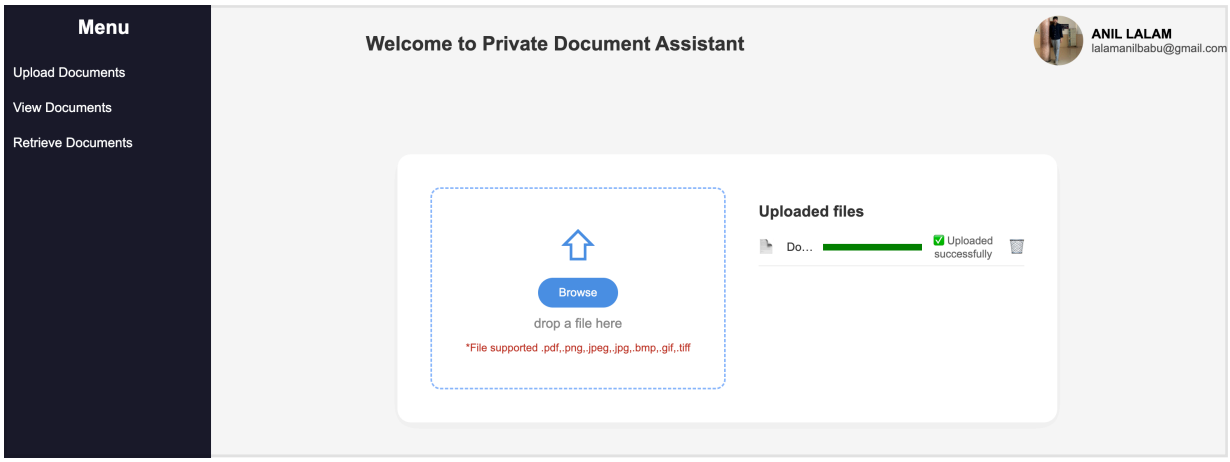
Event Routing	**Eventarc**	Detects finalized uploads in GCS and invokes Cloud Run endpoint.
Compute Layer	**Cloud Run (Spring Boot)**	Core microservice that handles event ingestion, text extraction, chunking, embedding, and querying.
Text Extraction	**Document AI (OCR Processor)**	Extracts text content from PDFs, images, and scanned documents.
Embedding Generation	**Vertex AI Embedding API**	Converts text chunks and queries into 768-dimensional vector embeddings.
Vector Database	**BigQuery (Vector column support)**	Stores embeddings and performs cosine similarity search for retrieval.

Data Flow

Document Ingestion

1. **User → Uploads file → GCS via web app (Resumable Upload)**

File sent to **GCS** via **Resumable Upload**



Java code handles session URL creation for resumable upload

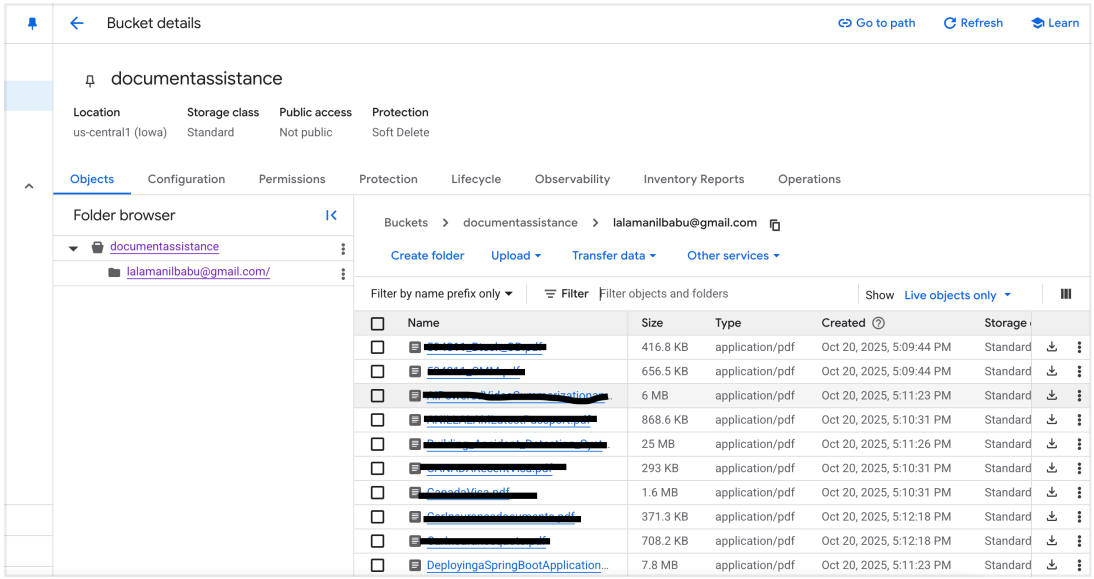
```java
@Override
public Map<String, String> initiateResumableUploadSessionUri(String objectName, String contentType) {
    // TODO Auto-generated method stub

    if (!NotNullEmptyUtility.notNullEmptyCheck(objectName)) {
        LOGGER.info("objectName cannot be null or empty");
        throw new PrivateDocumentException("objectName cannot be null or empty", 400);
    }

    if (!NotNullEmptyUtility.notNullEmptyCheck(contentType)) {
        LOGGER.info("Object contentType cannot be null or empty");
        throw new PrivateDocumentException("object ContentName cannot be null or empty", 400);
    }

    if (!ApplicationConstants.SUPPORTED_CONTENT_TYPE.contains(contentType)) {
        LOGGER.info(contentType + " is not supported. Supported contentTpes are:"
                + ApplicationConstants.SUPPORTED_CONTENT_TYPE);
        throw new PrivateDocumentException(contentType + " is not supported. Supported contentTpes are:"
                + ApplicationConstants.SUPPORTED_CONTENT_TYPE, 400);

    }

    String accessToken = AccessTokenUtility.getAccessToken();
    if (null == accessToken) {
        LOGGER.info("accessToken is null or empty.");
        throw new PrivateDocumentException("accessToken is null or empty", 500);
    }
    Map<String, String> metadata = new HashMap<String, String>();
    metadata.put("name", objectName);
    metadata.put("contentType", contentType);
    try {
        String metadataJson = ObjectMapperSingleton.INSTANCE.getObjectMapper().writeValueAsString(metadata);
        LOGGER.info("metadataJson:" + metadataJson);
        // Http Client
        HttpClient httpClient = HttpClient.newBuilder().version(Version.HTTP_2).build();

        LOGGER.info(ApplicationConstants.INITIATE_RESUMABLE_UPLOAD_URL);
        // Build HttpRequest
        HttpRequest httpRequest = HttpRequest.newBuilder()
                .uri(URI.create(ApplicationConstants.INITIATE_RESUMABLE_UPLOAD_URL)).timeout(Duration.ofSeconds(20))
                .header("Authorization", "Bearer " + accessToken)
                .header("Content-Type", "application/json; charset=UTF-8")
                .POST(BodyPublishers.ofString(metadataJson)).build();

        // Send Http Request
        HttpResponse<Void> httpResponse = httpClient.send(httpRequest, BodyHandlers.discarding());

        if (200 != httpResponse.statusCode() && 201 != httpResponse.statusCode()) {

            String msg = String.format("Failed to initiate resumable upload. HTTP %d", httpResponse.statusCode());
            LOGGER.info(msg);
            throw new PrivateDocumentException(msg, 500);
        }

        String uploadSessionUrl = httpResponse.headers().firstValue("Location")
                .orElseThrow(() -> new PrivateDocumentException("No Location header returned from GCS", 500));

        LOGGER.info("uploaded Session url:" + uploadSessionUrl);

        return Map.of("uploadUrl", uploadSessionUrl);

    } catch (JsonProcessingException e) {
        // TODO: handle exception
        String msg = "Exception while converting metadata object to json:" + e.getMessage();
        LOGGER.info(msg);
        throw new PrivateDocumentException(msg, 500);
    } catch (InterruptedException e) {
        // TODO: handle exception
        e.printStackTrace();
        Thread.currentThread().interrupt();
        throw new PrivateDocumentException("Interrupted while Initiating Resumable upload to get session uri", 500);

    } catch (IOException e) {
        // TODO: handle exception
        e.printStackTrace();
        throw new PrivateDocumentException("I/O exception while Initiating Resumable upload to session uri", 500);
    }

}
```

Java script Code Upload file to GCS in Chunks:

```
async function uploadFileWithResumableGCS(email, file, li) {
    const progress = li.querySelector("progress");
    const status = li.querySelector(".upload-status");
    try {
        const encodedObjectName = encodeURIComponent(email + "/" + file.name);
        const encodedType = encodeURIComponent(file.type);
        const resumableUrl = `/initiateResumableUpload?objectName=${encodedObjectName}&contentType=${encodedType}`;

        console.log(resumableUrl);
        const response = await fetch(resumableUrl);
        console.log('status:' + response.status)
        if (!response.ok) {
            status.textContent = "❌ Not uploaded (session URI failed)";
            return;
        }
        const data = await response.json();
        console.log('response json:' + data);
        const uploadUrl = data.uploadUrl;
        console.log('uploadUrl:' + uploadUrl);
        // upload the chunks

        if (!uploadUrl) {
            status.textContent = "❌ Not uploaded (session URI failed)";
            return;
        }

        status.textContent = "Uploading...";

        const chunkSize = 5 * 1024 * 1024; //5MB
        let offset = 0;
        while (offset < file.size) {
            const chunk = file.slice(offset, offset + chunkSize);
            const chunkStart = offset;

            const chunkEnd = Math.min(offset + chunkSize - 1, file.size - 1);
            const totalSize = file.size;
```

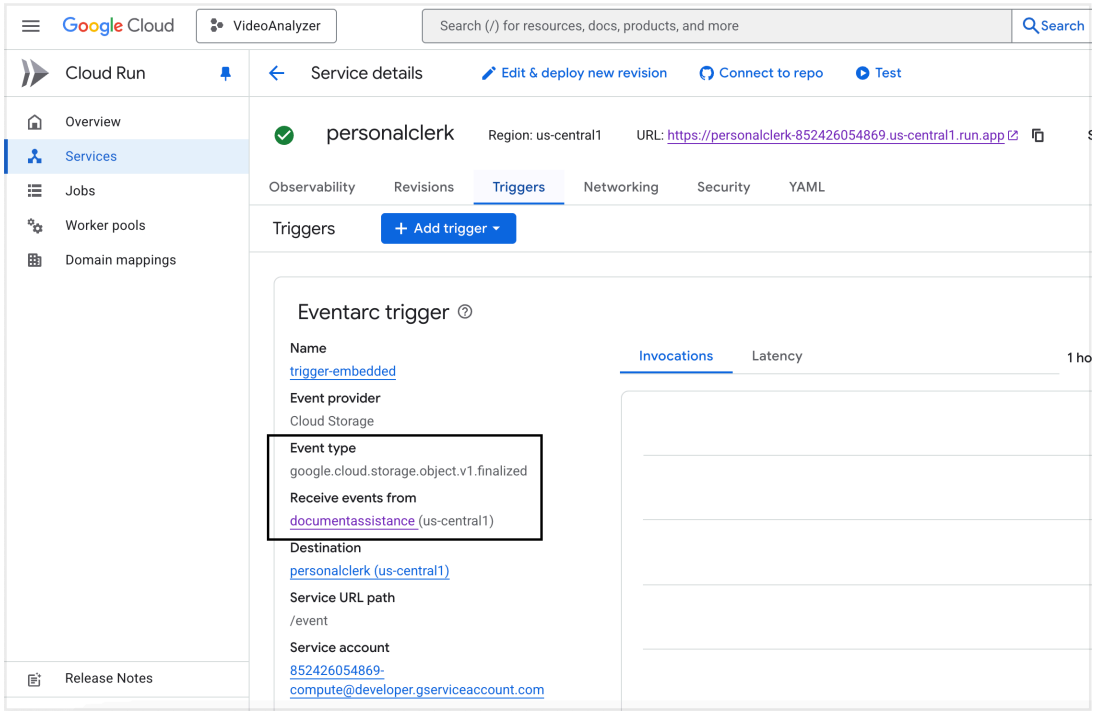
```
let success = false;
let attempts = 0;
const maxRetries = 3;

while (!success && attempts < maxRetries) {


    try {
        const res = await fetch("/uploadChunks", {
            method: "POST",
            headers: {
                "Content-Type": 'application/octet-stream',
                "SessionUrl": uploadUrl,
                "Content-Length": chunk.size.toString(),
                "Content-Range": `bytes ${chunkStart}-${chunkEnd}/${totalSize}`,
            },
            body: chunk
        });
        if (res.status === 200) {
            const finalMetadata = await res.text();
            console.log('upload Completed! final metadata received:', finalMetadata);
            progress.value = 100;
            offset = file.size;
            success = true;
        } else {
            if (res.status === 308) {
                offset = offset + chunk.size;
                progress.value = Math.round((offset / totalSize) * 100);
                success = true;
            } else {

                const errorBody = await res.text();
                console.error('GCS Error Details:', errorBody);
                throw new Error(`Chunk upload failed: HTTP ${res.status}`);
            }
        }
```

Documents are stored in GCS Bucket



2. **GCS → triggers Eventarc → Cloud Run service**

Detects finalized uploads and routes the event to **Cloud Run**.



3. **Cloud Run → Calls Document AI → Extracts text**

Code to extract raw text from documents using Document OCR (Document AI)

```java
public static String processDocument(byte[] filedata, String mimeType) {
    String rawText = null;
    if (null != documentProcessorServiceClient) {
        if (null != filedata) {
            String name = String.format("projects/%s/locations/%s/processors/%s", ApplicationConstants.PROJECT_ID,
                    ApplicationConstants.DOCUMENT_AI_PROCESSORS_LOCATION, ApplicationConstants.PROCESSOR_ID);
            // converting the image data to a Buffer and base64 encode it.
            ByteString content = ByteString.copyFrom(filedata);
            RawDocument document = RawDocument.newBuilder().setContent(content).setMimeType(mimeType).build();
            // Configure the process request
            ProcessRequest request = ProcessRequest.newBuilder().setName(name).setRawDocument(document).build();
            ProcessResponse processResponse = documentProcessorServiceClient.processDocument(request);
            Document documentResponse = processResponse.getDocument();
            LOGGER.info("Document processing is completed..");
            rawText = documentResponse.getText();
        } else {

            LOGGER.info("filedata bytes is null. Please provide valid byte array.");
        }
    } else {
        LOGGER.info("documentProcessorServiceClient is null. Please check application logs");
    }

    return rawText;

}
```

4. **Text → Chunked + Overlapped**

Extracted text is divided into overlapping chunks to preserve semantic continuity.

```java
package com.document.personal.assistance.utility;

import java.util.ArrayList;

public class TextChunkerUtility {

    public static List<String> chunkText(String text, int maxChars, int overlap) {
        List<String> chunkList = new ArrayList<String>();
        int start = 0;

        int minChunckSize = maxChars / 2;

        while (start < text.length()) {
            if ((text.length() - start) <= maxChars) {
                chunkList.add(text.substring(start));
                break;
            }
            int end = Math.min(start + maxChars, text.length());
            int lastboundary = Math.max(Math.max(text.lastIndexOf(".", end - 1), text.lastIndexOf("!", end - 1)),
                    text.lastIndexOf("?", end - 1));
            if (lastboundary > start + minChunckSize) {
                end = lastboundary + 1;
            }
            String chunk = text.substring(start, end).trim();
            if (!chunk.isEmpty()) {
                chunkList.add(chunk);
            }
            start = Math.max(end - overlap, start + 1);
        }
        return chunkList;
    }

}
```

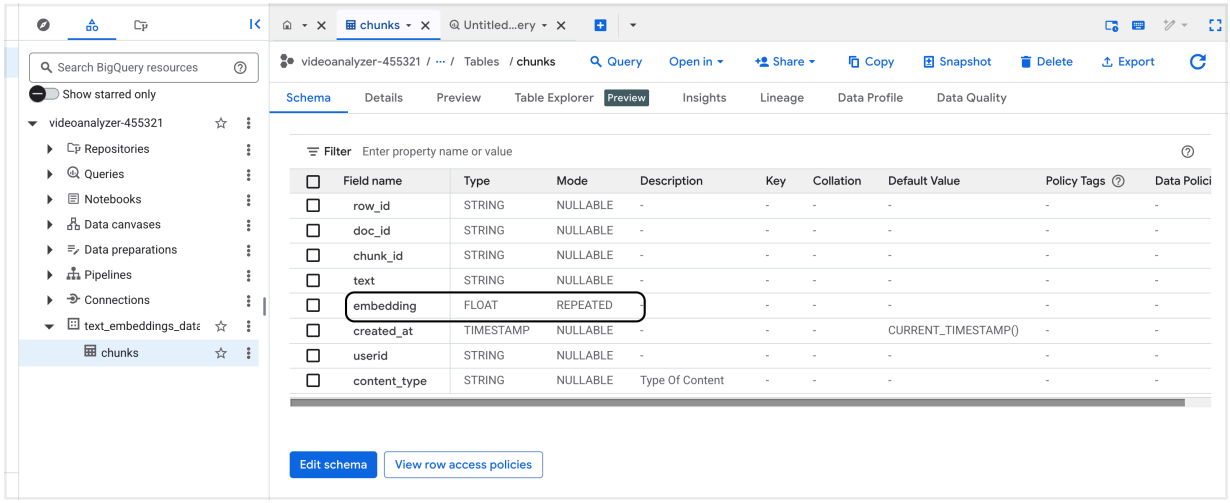
5. **Each chunk → Embedded via Vertex AI → Embeddings returned**

Each chunk is embedded into a numerical vector using the Vertex AI model (text-multilingual-embedding-002)

```java
    public static List<float[]> getEmbeddings(List<String> chunkList, String taskType) {
        List<float[]> embeddings = new ArrayList<float[]>();
        if (null != chunkList && !chunkList.isEmpty()) {
            if (null != predictionServiceClient) {
                // Important: For embeddings, you don't deploy your own endpoint.
                // You call the published model directly.
                String endPointName = String.format("projects/%s/locations/%s/publishers/google/models/%s",
                        ApplicationConstants.PROJECT_ID, ApplicationConstants.LOCATION_ID,
                        ApplicationConstants.TEXT_EMBEDDING_MODEL);
                System.out.println("endPointName is:" + endPointName);

                int exactChuckSize = chunkList.size() / ApplicationConstants.NUMBER_OF_CHUNKS_PER_REQUEST;
                int leftOverChunk = chunkList.size() % ApplicationConstants.NUMBER_OF_CHUNKS_PER_REQUEST;
                int track = 0;
                if (exactChuckSize > 0) {
                    for (int i = 0; i < exactChuckSize; i++) {
                        // predictRequest
                        PredictRequest.Builder request = PredictRequest.newBuilder().setEndpoint(endPointName);
                        List<String> subsetChunkList = chunkList.subList(track,
                                track + ApplicationConstants.NUMBER_OF_CHUNKS_PER_REQUEST);
                        predictionRequestCall(subsetChunkList, request, embeddings, track, taskType);
                        track = track + ApplicationConstants.NUMBER_OF_CHUNKS_PER_REQUEST;
                        System.out.println("***** at track:" + track);
                    }
                }

                if (leftOverChunk > 0) {
                    // predictRequest
                    PredictRequest.Builder request = PredictRequest.newBuilder().setEndpoint(endPointName);
                    List<String> subsetChunkList = chunkList.subList(track, chunkList.size());
                    predictionRequestCall(subsetChunkList, request, embeddings, track, taskType);
                    System.out.println("**** at leftOver track:" + (chunkList.size() - track));
                }

            } else {
                System.out.println("predictionServiceClient is null. Please check your application logs");
                throw new PrivateDocumentException(
                        "predictionServiceClient is null. Please check your application logs", 500);
            }
        } else {
            System.out.println("chunlklist is null or empty.");
            throw new PrivateDocumentException("chunlklist is null or empty for userprompt.", 500);
        }

        return embeddings;
    }
```

6. **Cloud Run → Stores embeddings + metadata → BigQuery**

Create the dataset and table in Google Big Query:



Embeddings and metadata are stored in **BigQuery** for efficient vector search.

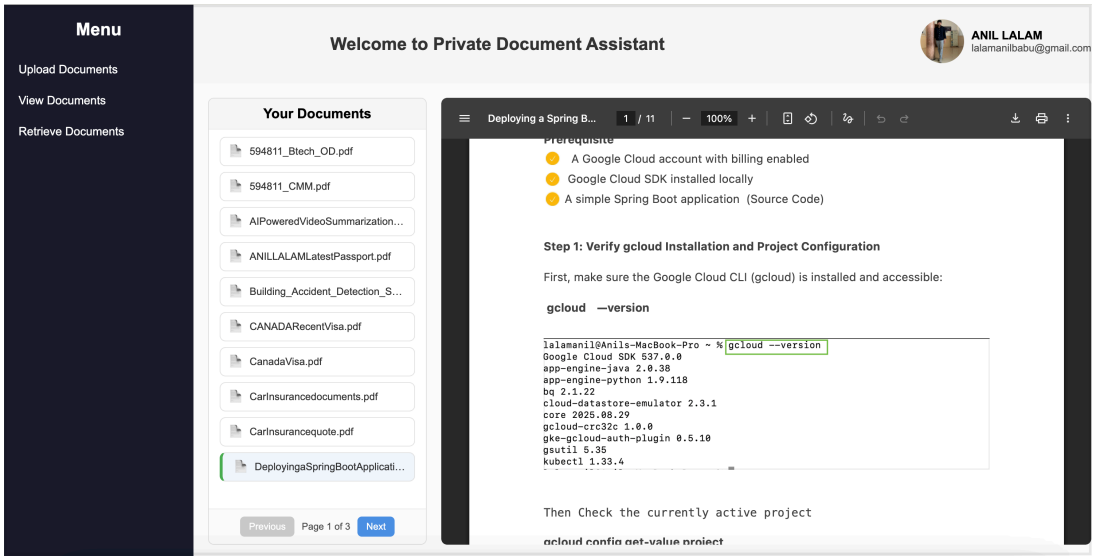
```
            List<Map<String, Object>> rowList = new ArrayList<Map<String, Object>>();
            for (int i = 0; i < vectorList.size(); i++) {
                Map<String, Object> row = new HashMap<String, Object>();
                row.put("row_id", objectName + "_chunck" + chunkCount);
                row.put("doc_id", objectName);
                row.put("chunk_id", "chunck" + chunkCount);
                row.put("text", chunkList.get(i));
                row.put("userid", userid);
                row.put("content_type", contentType);
                float[] floatvectors = vectorList.get(i);
                List<Double> emblist = new ArrayList<Double>();
                for (float f : floatvectors) {
                    emblist.add((double) f);
                }
                row.put("embedding", emblist);
                rowList.add(row);
                chunkCount++;
            }
            LOGGER.info("Total vector rows trying to insert into Bigquery is:" + rowList.size());

            if (!rowList.isEmpty()) {
                BigQueryUtility.insertRowsBatch(rowList, ApplicationConstants.TABLE_NAME);
            }
```

```
public static void insertRowsBatch(List<Map<String, Object>> rowData, String tableName) {
    if (null != bigQuery) {
        TableId tableId = TableId.of(ApplicationConstants.DATA_SET, tableName);
        InsertAllRequest.Builder builder = InsertAllRequest.newBuilder(tableId);
        for (Map<String, Object> row : rowData) {
            builder.addRow(row);
        }
        try {
            InsertAllResponse response = bigQuery.insertAll(builder.build());
            if (response.hasErrors()) {
                response.getInsertErrors().forEach((index, error) -> {
                    System.err.println("Row at index " + index + " failed with errors:" + error);
                });

            } else {
                LOGGER.info("Inserted " + rowData.size() + " rows to " + tableId);
            }

        } catch (BigQueryException e) {
            // TODO: handle exception
            e.printStackTrace();
        }

    } else {
        LOGGER.info("bigQuery object is null. Please check application logs");
    }

}
```

Query & Retrieval

1. **User → Enters natural-language query → Web UI**

The user submits a natural-language question via the web UI.
A “View Documents” option displays previously uploaded files.



2. **Web UI → Sends query → Cloud Run service**

Java script code to send user query to Cloud Run Service to get relevenat Documents based on Context.

```
async function sendQuery() {
    const query = queryInput.value.trim();
    if (!query) {
        return;
    }
    appendMessage("user", query);
    queryInput.value = "";
    appendMessage("assistant", "Searching your documents...");
    const body = {
        userId: emailid,
        userPromptForDocSearch: query,
        relevanceCutoff: 0.37 // we can adjust this or make it configurable
    };
    try {
        const response = await fetch("/userdocsbyprompt", {
            method: "POST",
            headers: { "Content-Type": "application/json" },
            body: JSON.stringify(body)
        });
        const docs = await response.json();
        if (!response.ok) {
            throw new Error(`HTTP ${response.status}`);
        }
        if (!Array.isArray(docs.documents) || docs.documents.length === 0) {
            chatMessages.lastChild.textContent = "No relevant documents found.";
            docList.innerHTML = "";
            preview.style.display = "none";
            preview.innerHTML = ""; //clear old preview
            return;
        }
        chatMessages.lastChild.textContent = `Found ${docs.documents.length} matching documents.`;
        documents = docs.documents;
        renderDocuments(currentpage);
    } catch (err) {
        console.error("Error retrieving docs:", err);
        chatMessages.lastChild.textContent = "❌ Error retrieving documents.";
    }

}
```

3. **Cloud Run → Embeds query → Vertex AI (task type: RETRIEVAL_QUERY)**

The Cloud Run service converts the user query into a vector using Vertex AI with task_type = RETRIEVAL_QUERY

```java
@Override
public List<VectorSearchResultsModel> searchDocumentsForUserPrompt(String userid, String promptString,
        float relevanceCutoff) {
    // TODO Auto-generated method stub
    if (!NotNullEmptyUtility.notNullEmptyCheck(userid)) {
        throw new PrivateDocumentException("userid cannot be null or empty. Please provide valid userid", 400);
    }
    if (!NotNullEmptyUtility.notNullEmptyCheck(promptString)) {
        throw new PrivateDocumentException("promptString cannot be null or empty. Please provide valid userid",
                400);
    }
    List<float[]> vectorlist = TextEmbeddingWithPredictionServiceUtility.getEmbeddings(Arrays.asList(promptString),
            ApplicationConstants.TASK_TYPE_RETRIEVAL_QUERY);
    if (null != vectorlist && !vectorlist.isEmpty()) {
        float[] vector = vectorlist.get(0);
        List<Double> doubleVector = new ArrayList<Double>(vector.length);
        for (float f : vector) {
            doubleVector.add((double) f);
        }
        LOGGER.info("vector for user prompt is: " + doubleVector);
        List<VectorSearchResultsModel> list = BigQueryUtility.vectorSearch(userid,
                ApplicationConstants.TOP_CHUNK_COUNT, doubleVector);
        return GeneralUtility.filterVectorSearchBasedOnRelevenceCutoff(list, relevanceCutoff);
    } else {
        LOGGER.info("vectorList for user prompt is null or empty.");
        throw new PrivateDocumentException("vectorList for user prompt is null or empty.", 500);
    }

}
```

4. **Cloud Run → Performs cosine vector similarity in BigQuery**

Cloud Run executes a **cosine similarity** search to identify relevant document chunks

```java
public static List<VectorSearchResultsModel> vectorSearch(String userid, int topK, List<Double> queryVector) {

    List<VectorSearchResultsModel> vectorSearchResultsModels = new ArrayList<VectorSearchResultsModel>();
    if (null != bigQuery) {
        String vectorSearchQuery = getVectorQuery();
        if (null != vectorSearchQuery && !vectorSearchQuery.trim().isEmpty()) {
            // prepare query Configuration with parameters
            QueryJobConfiguration queryJobConfiguration = QueryJobConfiguration.newBuilder(vectorSearchQuery)
                    .addNamedParameter("userId", QueryParameterValue.string(userid))
                    .addNamedParameter("topK", QueryParameterValue.int64(topK))
                    .addNamedParameter("queryVector", QueryParameterValue.array(queryVector.toArray(new Double[0]),
                            StandardSQLTypeName.FLOAT64))
                    .build();
            // Run the Query
            try {
                TableResult tableResult = bigQuery.query(queryJobConfiguration);
                Iterable<FieldValueList> iterable = tableResult.iterateAll();
                for (FieldValueList row : iterable) {
                    VectorSearchResultsModel vectorSearchResultsModel = new VectorSearchResultsModel();
                    vectorSearchResultsModel.setDocumentId(row.get("doc_id").getStringValue());
                    List<Double> distanceList = new ArrayList<Double>();
                    List<FieldValue> distancevalues = row.get("distances").getRepeatedValue();
                    if (null != distancevalues) {
                        distancevalues.forEach(fv -> {
                            distanceList.add(fv.getDoubleValue());
                        });
                    }
                    vectorSearchResultsModel.setDistanceList(distanceList);
                    vectorSearchResultsModel.setNumberOfChunks(row.get("numberOfChucks").getLongValue());
                    vectorSearchResultsModel.setContentType(row.get("contenttype").getStringValue());
                    long micros = row.get("created_at").getTimestampValue();
                    Instant instant = Instant.ofEpochMilli(micros / 1000);
                    LocalDateTime localDateTime = LocalDateTime.ofInstant(instant, ZoneId.of("UTC"));
                    vectorSearchResultsModel.setCreatedAt(localDateTime.toString());
                    vectorSearchResultsModels.add(vectorSearchResultsModel);
                }

            } catch (InterruptedException e) {
                // TODO: handle exception
                LOGGER.info("if the current thread gets interrupted while waiting for the query to complete:"
                        + e.getMessage());
                e.printStackTrace();
                throw new PrivateDocumentException(e.getMessage(), e, 500);

            } catch (JobException e) {
                // TODO: handle exception
```

```
                    // TODO: handle exception
                    LOGGER.info("if the job completes unsuccessfully:" + e.getMessage());
                    e.printStackTrace();
                    throw new PrivateDocumentException(e.getMessage(), e, 500);

                } catch (BigQueryException e) {
                    // TODO: handle exception
                    LOGGER.info("upon failure:" + e.getMessage());
                    e.printStackTrace();
                    throw new PrivateDocumentException(e.getMessage(), e, 500);
                }

            } else {
                LOGGER.info("VectorSearchQuery is either null or empty. Please check application logs");
                throw new PrivateDocumentException(
                        "VectorSearchQuery is either null or empty. Please check application logs", 500);
            }
        } else {
            LOGGER.info("bigQuery object is null. Please check application logs.");
            throw new PrivateDocumentException("bigQuery object is null. Please check application logs.", 500);
        }

        return vectorSearchResultsModels;
    }

}
```

5. **BigQuery → Returns top-K (e.g., top 10) matching chunks**

BigQuery returns the top-K (e.g., top 10) most relevant results.

```
SELECT base.doc_id,
ARRAY_AGG(distance order by distance ASC )   as distances ,
ARRAY_AGG(base.text order by distance ASC)  as text,
count(base.doc_id) as numberOfChucks,
ARRAY_AGG( distinct base.created_at)[offset(0)] as created_at,
ARRAY_AGG(base.content_type)[offset(0)]  as contenttype
from
VECTOR_SEARCH( (select * from `videoanalyzer-455321.text_embeddings_dataset.chunks`
where created_at in (select ARRAY_AGG( created_at
order by created_at DESC LIMIT 1 )[offset(0)]
from `videoanalyzer-455321.text_embeddings_dataset.chunks`
where userid=@userId group by doc_id))
,"embedding",(SELECT @queryVector AS query_embedding) , top_k =>@topK,
distance_type =>'COSINE') group by base.doc_id order by min(distance) ASC
```

6. **Response Delivery** :

Cloud Run sends the ranked results with text snippets and metadata back to the web UI for preview.

Menu

- Upload Documents
- View Documents
- Retrieve Documents

Welcome to Private Document Assistant

ANIL LALAM
lalamanilbabu@gmail.com

Matched Documents

- PrivateDocumentSearchAssist…
- Building_Accident_Detection…
- AIPoweredVideoSummarizatio…
- DeployingaSpringBootApplicat…
- Document Assistance.jpg

Previous | Page 1 of 1 | Next

Please provide the documents related AI projects

Found 5 matching documents.

Ask about your documents.... Send

1 / 28

AI-Powered Accident Detection System Using Java and Google Cloud Vertex AI

1. Introduction

Road accidents claim over a million lives every year worldwide, with countless more left injured. Timely accident detection can drastically reduce response times for emergency services, potentially saving lives and minimizing damage.

In this article, I'll walk you through how I built an AI-powered accident detection system — entirely using **Java for preprocessing, cloud integration, and prediction calls** — combined with **Google Cloud Vertex AI for training and deploying a custom object detection model.**

This work is not just a technical project; it has real-world societal impact, aligning with public safety priorities and contributing toward innovations that support **national interests in road safety and AI development.**

Architecture Diagram