Genkit 102

RAG & Function Calling - Building Agents

Agenda

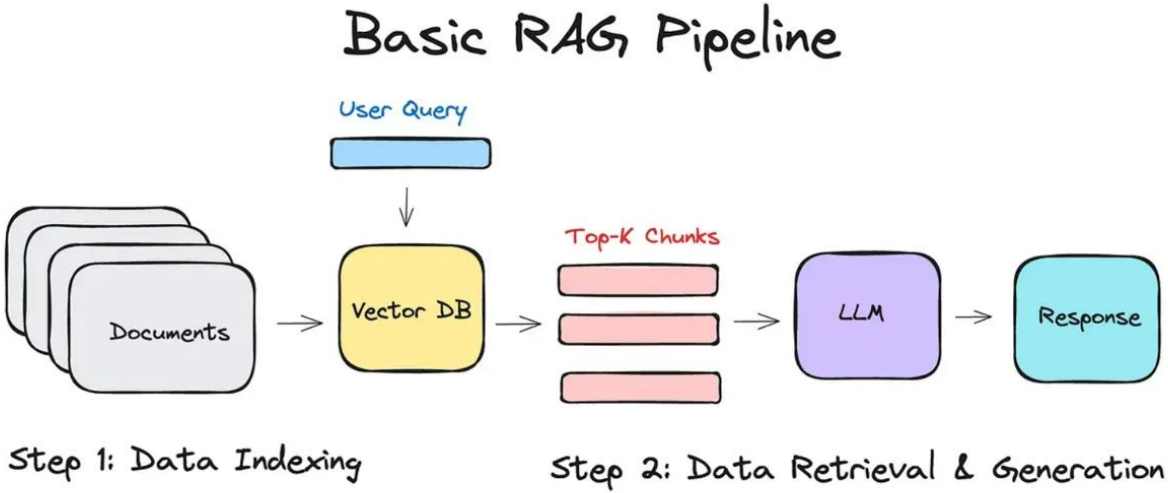
- RAG
  - What is RAG?
  - Using RAG with Genkit
- Function/Tool Calling
  - What is Tool Calling
  - Defining Tools
  - Using Tools
- Genkit Developer UI - Testing Genkit Application

What is RAG?

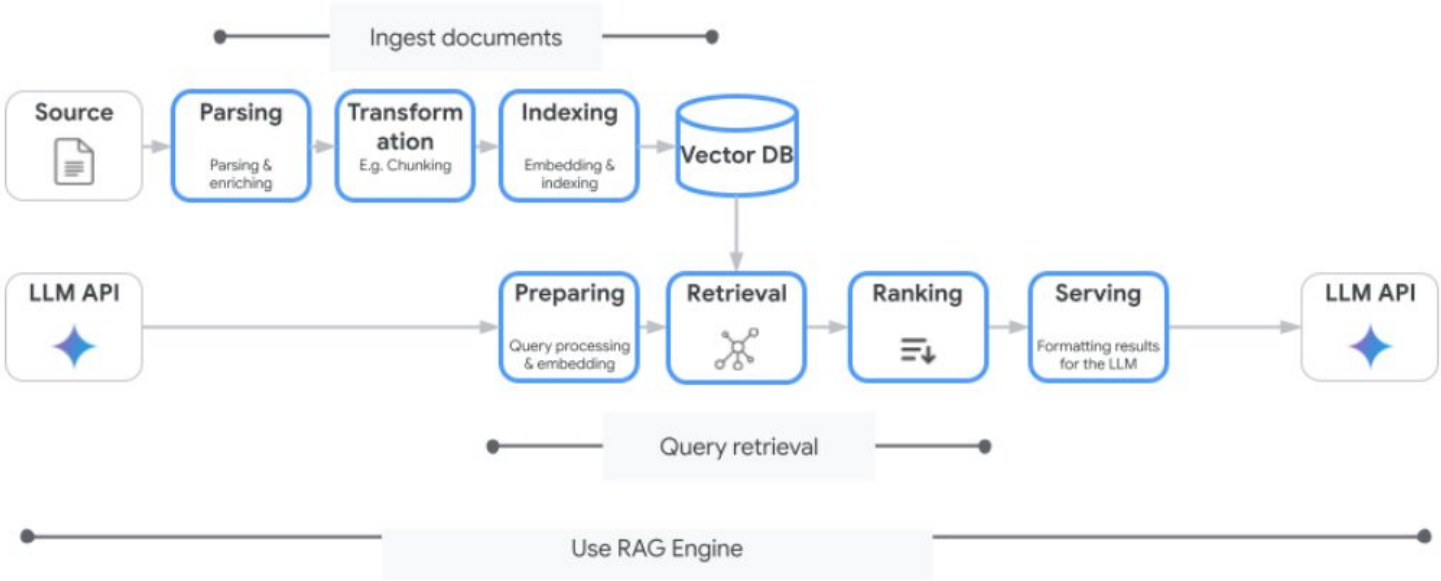
What is RAG?

- Retrieval-Augmented Generation (RAG) is an AI technique that enhances language models by retrieving factual, up-to-date information from external sources to generate more accurate and contextually relevant answers.
- RAG makes AI more reliable by grounding it in real-time, verifiable data, which reduces factual errors and builds user trust.

What is RAG?


Basic RAG Pipeline
User Query
Documents
Vector DB
Top-K Chunks
LLM
Response
Step 1: Data Indexing
Step 2: Data Retrieval & Generation

What is RAG?


Ingest documents
Source
Parsing
Parsing & enriching
Transform ation
E.g. Chunking
Indexing
Embedding & indexing
Vector DB
LLM API
Preparing
Query processing & embedding
Retrieval
Ranking
Serving
Formatting results for the LLM
LLM API
Query retrieval
Use RAG Engine

Using RAG with Genkit

Step 1: Prepare your Data - Indexing

- The index is keeps track of your documents in such a way that you can quickly retrieve relevant documents given a specific query.
- Commonly achieved via vector databases

Step 1: Prepare your Data - Vector DBs

- Documents are indexed using multidimensional vectors called embeddings.
- A text embedding (opaquely) represents the concepts expressed by a passage of text; these are generated using special-purpose ML models.
- By creating text embeddings, a vector database is able to cluster conceptually related text and retrieve documents related to a novel string of text (the query) without knowing the meaning

Step 1: Steps to index your Data

- Split up large documents into smaller documents – “**chunking**”.
  - LLMs have a limited context window.
  - Genkit doesn’t provide built-in chunking libraries
    - Use Third Party Libraries
- Generate embeddings for each chunk
  - Some DB Support this
  - You can use an embedding model e.g.
- Save both chunk and embedding to the DB

Step 1: Steps to index your Data

```
const ai = genkit({
  plugins: [
    // googleAI provides the gemini-embedding-001 embedder
    googleAI(),

    // the local vector store requires an embedder to translate from text to vector
    devLocalVectorstore([
      {
        indexName: 'menuQA',
        embedder: googleAI.embedder('gemini-embedding-001'),
      },
    ]),
  ],
});
```

codesnap.dev

Step 1: Steps to index your Data

```
async function extractTextFromPdf(filePath: string) {
  const pdfFile = path.resolve(filePath);
  // Implementation to read and parse the PDF
}

export const indexMenu = ai.defineFlow(
  {
    name: 'indexMenu',
    inputSchema: z.string(),
    outputSchema: z.object({ success: z.boolean() }),
  },
  async (filePath) => {
     // Implementation Details here
  }
);
```

codesnap.dev

Step 1: Steps to index your Data

```
async (filePath) ⇒ {
  // 1. Extract text from the PDF
  const pdfTxt = await ai.run('extract-text', () ⇒ extractTextFromPdf(filePath));

  // 2. Chunk the extracted text
  const chunks = await ai.run('chunk-it', async () ⇒ chunk(pdfTxt, { /* chunking options */ }));

  // 3. Convert chunks into Genkit Documents
  const documents = chunks.map((text) ⇒ {
    return Document.fromText(text);
  });

  // 4. Index the documents
  await ai.index({
    indexer: menuPdfIndexer,
    documents,
  });

  return { success: true };
}
```

codesnap.dev

Step 2: Retrieving

- A retriever is a concept that encapsulates logic related to any kind of document retrieval.
  - E.g. retrieval from vector stores,
  - can be any function that returns data.

Step 2: Retrieving - Example

```
import { devLocalRetrieverRef } from '@genkit-ai/dev-local-vectorstore';

// Reference to the local retriever
export const menuRetriever = devLocalRetrieverRef('menuQA');
```

codesnap.dev

Step 2: Retrieving - Example

```
import { defineFirestoreRetriever } from '@genkit-ai/firebase';
import { firestore } from './firebase-config'; // Your Firebase config
import { vertexAI } from '@genkit-ai/vertexai';

export const placesRetriever = defineFirestoreRetriever(ai, {
  name: 'placesRetriever',
  firestore,
  collection: 'places',
  contentField: 'knownFor',
  vectorField: 'embedding',
  embedder: vertexAI.embedder('text-embedding-005'),
});
```

codesnap.dev

Step 3: Using Retriever

```
export const menuQAFlow = ai.defineFlow(
  {
    name: 'menuQAFlow',
    inputSchema: z.object({ query: z.string() }),
    outputSchema: z.object({ answer: z.string() }),
  },
  async ({ query }) => {
    // 1. Retrieve relevant documents
    const docs = await ai.retrieve({
      retriever: menuRetriever,
      query: query,
      options: { k: 3 }, // Retrieve top 3 documents
    });

    // 2. Generate a response using the retrieved context

  }
);
```

codesnap.dev

Step 4: Invoking the Model

```
export const menuQAFlow = ai.defineFlow(
  {
    // ...
  },
  async ({ query }) => {
    // ... Retrieve documents

    // 2. Generate a response using the retrieved context
    const { text } = await ai.generate({
      prompt: `Prompt`,
      docs: docs
    });

    return { answer: text };
  }
);
```

codesnap.dev

Function/Tool Calling with Genkit

Function/Tool Calling with Genkit

- A structured way to give LLMs the ability to make requests back to the application that called it.
- Give models ability to get things done
- You define the tools you want to make available to the model, and the model will make tool requests to your app as necessary to fulfill the prompts you give it.
- Dependent on model support

Tool Calling Example

```
const getWeather = ai.defineTool(
  {
    name: 'getWeather',
    description: 'Gets the current weather in a given location',
    inputSchema: z.object({
      location: z.string().describe('The location to get the current weather for'),
    }),
    outputSchema: z.string(),
  },
  async (input) => {
    // Here, we would typically make an API call or database query. For this
    // example, we just return a fixed value.
    return `The current weather in ${input.location} is 63°F and sunny.`;
  },
);
```

codesnap.dev

Tool Calling - Example

```
const response = await ai.generate({
  prompt: 'What is the weather in Baltimore?',
  tools: [getWeather],
});
```

codesnap.dev

```
const weatherPrompt = ai.definePrompt(
  {
    name: 'weatherPrompt',
    tools: [getWeather],
  },
  'What is the weather in {{location}}?',
);

const response = await weatherPrompt({ location: 'Baltimore' });
```

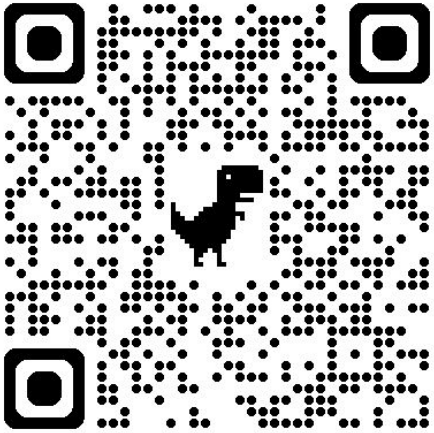
codesnap.dev

Function/Tool Calling with Genkit

- Prompt the LLM and also include list of tools the LLM can use
- The LLM either generates:
  - a complete response or
  - a tool call request in a specific format.
- If the caller receives a complete response:
  - the request is fulfilled and the interaction ends;
- If the caller receives a tool call:
  - it performs whatever logic is appropriate
  - sends a new request to the LLM containing the original prompt or some variation of it as well as the result of the tool call.
- The LLM handles the new prompt as in Step 2

Genkit Developer UI

Real-World Example - Github Griller

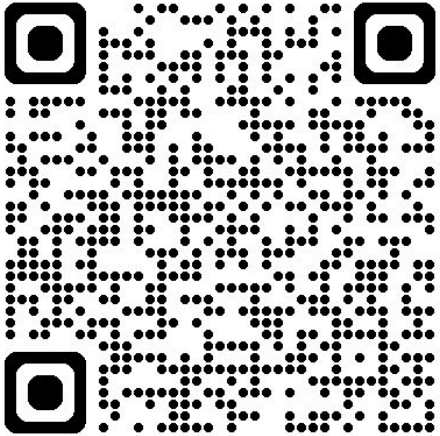


https://github.com/mainawycliffe/ripper-the-github-griller

Google Codelabs

1. Automatically Deploy Generative AI Node.js Genkit Web Application from Version Control to Cloud Run | Google Codelabs
2. Automatically Deploy Generative AI Go with Genkit Web Application from Version Control to Cloud Run | Google Codelabs
3. Build gen AI features powered by your data with Genkit | Firebase Codelabs

Last Weeks Genkit Codelabs



Genkit Codelabs - Unstacked Labs

Q&A

- The End -