RAG Integration With Gemini Live API

RAG (Retrieval-Augmented Generation) allows the voice agent to answer user questions based strictly on your business documents instead of relying on the model's general knowledge. When a user asks a question, the system performs three key steps:

1. **Retrieve** relevant information from a knowledge base
2. **Augment** the model with that information
3. **Generate** a natural response that uses the retrieved content

This ensures the agent provides accurate, business-specific answers without generating incorrect responses.

Full Processing Pipeline

When a user asks a question such as *"What are your business hours?"*, the system performs the following operations:

2.1 Speech → Text

Gemini converts the user's audio into text.

2.2 Intent Understanding

Gemini identifies that the question requires company information and triggers a function call, e.g.:

lookup_company_info(query="What are your business hours?")

2.3 Retrieval (RAG Search)

The RAG system:

1. Converts the question into a numerical embedding

2. Compares this vector to all stored question embeddings

3. Retrieves the top-k most similar Q&A pairs using a FAISS index

2.4 Context Injection

The retrieved Q&A pairs are passed back to Gemini in a structured format.

2.5 Response Generation

Gemini generates a natural response using only the provided context.

2.6 Text → Speech

Gemini streams the audio response back to the user.

System Initialization

At startup, the RAG module performs:

1. **Load Q&A pairs** from the FAQ file
2. **Generate embeddings** for each question
3. **Build a FAISS index** for fast vector similarity search

Function Tool Architecture

The function tool enables Gemini to interact with the RAG system.

Example (simplified):

```
@function_tool
async def lookup_company_info(query: str) -> str:
    logger.info(f"Tool called with query: {query}")
    try:
```

```
            return rag.get_context(query, top_k=2)
        except Exception:
            return "I'm having trouble accessing the FAQ database right now."
```

What Gemini Sees

```
{
  "name": "lookup_company_info",
  "description": "Search knowledge base for company info.",
  "parameters": {
    "type": "object",
    "properties": { "query": { "type": "string" } },
    "required": ["query"]
  }
}
```

It Matters because:

- Docstring → tool description for Gemini
- Tells Gemini when to call the tool
- Schema ensures correct invocation

Agent Instructions

```
Agent(
    instructions=(
        "Use ONLY the knowledge base.\n"
```

```
        "1. ALWAYS call lookup_company_info for company questions\n"
        "2. Answer ONLY using the tool result\n"
        "3. If no info: 'I don't have that information.'\n"
        "4. No account-specific answers\n"
        "5. Redirect off-topic questions."
    ),
    tools=[lookup_company_info]
)
```

They Work because:

Rule 1 → Forces RAG usage

Rule 2 → No hallucinations

Rules 3–5 → Handles errors, privacy, and off-topic queries

The Request-Response Flow

Step-by-Step Integration

User asks: "What are your business hours?"

A. Gemini Receives Audio Stream

Audio input → Speech-to-text

Output: "What are your business hours?"

B. Gemini Analyzes Query

Reasoning:

- User is asking about company information

- My instructions say "ALWAYS use lookup_company_info"

- I need to call the function tool

C. Gemini Generates Function Call

json

```
{
  "function": "lookup_company_info",
  "arguments": {
    "query": "What are your business hours?"
  }
}
```

D. Function Tool Executes

python

```
# Python function receives a call
query = "What are your business hours?"


# Calls RAG system
result = rag.get_context(query, top_k=2)
```

E. RAG System Processes Query

RAG retrieval: `rag_system.py`

python

```
def get_context(self, query: str, top_k: int = 2) -> str:
    # 1. Embed the query
    query_embedding = self.model.encode([query])
    # Output: [0.145, -0.231, 0.412, ...] (384 numbers)

    # 2. Search FAISS index
    distances, indices = self.index.search(query_embedding, top_k)
    # Finds 2 closest matching questions

    #3. Format context
    context_parts = [ ]
    for idx, distance in zip(indices[0], distances[0]):
        context_parts.append(f"""Q: {self.questions[idx]}
A: {self.answers[idx]}""")

    return "\n\n".join(context_parts)
```

Returns formatted context:

Q: What are your business hours?

A: We are open Monday to Friday from 9 AM to 6 PM, and Saturday from 10 AM to 4 PM. We are closed on Sundays and public holidays.

Q: How can I contact customer support?

A: You can reach our customer support team via email at support@company.com, call us at 1-800-SUPPORT, or use the live chat on our website available 24/7.

F. Function Returns to Gemini

python

return result *# String containing Q&A context*

Gemini now has the retrieved context from your knowledge base.

G. Gemini's Prompt State

Gemini now sees:

System Instructions:

"You are a helpful customer support assistant. Answer questions using ONLY the knowledge base..."

User Query:

"What are your business hours?"

Function Result (lookup_company_info):

Q: What are your business hours?

A: We are open Monday to Friday from 9 AM to 6 PM, and Saturday from 10 AM to 4 PM. We are closed on Sundays and public holidays.

Q: How can I contact customer support?

A: You can reach our customer support team via email at support@company.com...

H. Gemini Generates Response

Internal processing:

1. Extract relevant info from the function result
2. Follow instructions: "Answer ONLY with information from tool results"
3. Generate a natural language response
4. Maintain conversational tone

Generated text:

"We are open Monday to Friday from 9 AM to 6 PM, and Saturday from 10 AM to 4 PM. We are closed on Sundays and public holidays."

I. Text-to-Speech

python

```
model = google.beta.realtime.RealtimeModel(
    model="gemini-2.0-flash-exp",
    voice="Aoede",  # Natural voice
)
```

Gemini converts text → natural speech → streams to the user

Semantic Search Integration

How RAG Understands Different Phrasings

User query variations:

- "What are your business hours?"
- "What time do you open?"
- "When are you available?"
- "Are you open on weekends?"

All map to the same knowledge base entry:

Question: What are your business hours?

Answer: We are open Monday to Friday from 9 AM to 6 PM...

The Magic: Embeddings

python

 # Query embedding

query = "What time do you open?"

query_embedding = model.encode([query])

 # Output: [0.148, -0.228, 0.408, ...] (384 dimensions)

 # Knowledge base question embedding

kb_question = "What are your business hours?"

kb_embedding = [0.145, -0.231, 0.412, ...] *# Pre-computed*

 # Calculate similarity (L2 distance)

distance = sqrt(sum((q - kb)^2))

 # Result: 0.42 (low distance = high similarity)

Distance interpretation:

0.0 - 0.3 = Excellent match (near identical meaning)

0.3 - 0.8 = Good match (semantic similarity)

0.8 - 1.5 = Fair match (related topic)

1.5+ = Poor match (different topic)

Top-K Retrieval Strategy

Why Retrieve 2 Results Instead of 1

Code:

python

result = rag.get_context(query, top_k=2)

Example scenario:

User: "How do I contact support?"

Top-1 only:

Result: "Contact us at support@company.com"

Response: "You can reach us at support@company.com."

✓ Correct but minimal

Top-2:

Result 1: "Contact us at support@company.com, call 1-800-SUPPORT..."

Result 2: "Our business hours are Monday to Friday 9 AM to 6 PM..."

Response: "You can reach us at support@company.com or call 1-800-SUPPORT. We're available Monday to Friday from 9 AM to 6 PM."

Handling Edge Cases

When RAG Finds No Good Match

Scenario: User asks, "What's the weather?"

python

 # RAG searches

results = self.search("What's the weather?", top_k=1)

 # Best match: ("What are your business hours?", distance=2.85)

 # Distance too high - no relevant match

if distance > 1.5:

    return "No relevant information found in knowledge base."

Gemini receives:

Function result: "No relevant information found in knowledge base."

Gemini's response (following instructions):

"I don't have information about the weather. I'm specialised in helping with questions about our company, products, and services. Is there anything about our business I can help you with?"