Pydantic RAG

Agentic RAG chatbot built with Pydantic AI, Weaviate vector database, and local Ollama inference. Features hybrid search, conversation memory, and multiple RAG modes.

Features

- **Hybrid Search** - Combines BM25 keyword search with semantic vector search (configurable alpha)
- **3 RAG Modes** - Auto (agent decides), Force (always search), Disabled (plain chat)
- **Multimodal Support** - Image + text RAG with CLIP embeddings and VLM analysis at query time
- **Conversation Memory** - Multi-turn conversations with full context via `message_history`
- **Token Tracking** - Real-time token usage display with context limit warnings
- **Local Inference** - GPU-accelerated LLM and embedding generation via Ollama

Tech Stack

Component	Technology
Agent Framework	Pydantic AI
Vector Database	Weaviate (with text2vec-ollama)
LLM Inference	Ollama (llama3.2)
Embeddings	nomic-embed-text (768 dimensions)
Web UI	Gradio
Orchestration	Docker Compose

Prerequisites

- Docker with Compose v2
- NVIDIA GPU with CUDA drivers (for GPU inference)
- NVIDIA Container Toolkit (`nvidia-docker`)

Quick Start

1. **Clone and start services**:

```
git clone <repo-url>
cd pydantic-rag
docker compose up -d
```

2. **Wait for model downloads** (first run only):

```
# Watch Ollama logs until models are ready
docker logs -f ollama
```

3. **Access the UI** at http://localhost:7860

Document Ingestion

Place documents in `data/documents/` and run the ingestion script:

```
# Create documents directory
mkdir -p data/documents

# Add your documents (supports .txt, .md, .pdf, .py, .js, .ts, .c, .cpp,
.cu, .h)
cp /path/to/your/docs/* data/documents/

# Install dependencies and run ingestion
pip install weaviate-client pypdf
python scripts/ingest.py --name "my project"
```

Ingestion options:

```
python scripts/ingest.py --help
python scripts/ingest.py --reset                                  #
Reset collection only (no ingestion)
python scripts/ingest.py --name "my project" --reset              #
Delete and recreate collection, then ingest
python scripts/ingest.py --name "eu ai regulations" --extensions .pdf  #
Only PDFs with label
python scripts/ingest.py --name "docs" --documents-dir ./my-docs  #
Custom source folder
python scripts/ingest.py --name "code" --extensions .py,.md       # Only
Python and Markdown files
python scripts/ingest.py --name "config" --extensions .yml,Dockerfile  #
Extensions and exact filenames
python scripts/ingest.py --name "images" --multimodal             #
Multimodal mode with CLIP (includes images)
python scripts/ingest.py --name "mm-docs" --multimodal --reset    #
Reset and reingest in multimodal mode
python scripts/ingest.py --documents-dir
/home/alex/projects/prototypes/92-claude-agent-sdk/ --extensions
.py,.png,.txt,.md,.sh,.yml,Dockerfile --name "claude agent sdk" --
multimodal
```

PROF

> **Note**: The `--name` option is required when ingesting documents. Use `--reset` alone to recreate the schema without ingesting.

Documents are chunked (800 tokens, 200 overlap) and embedded automatically by Weaviate's text2vec-ollama module.

Usage

RAG Modes

Mode	Behavior
Auto	Agent decides when to search documents based on the question
Force	Always searches documents before answering
Disabled	Plain chat without document retrieval

Chat Interface

1. Select a RAG mode
2. Type your question and press Enter
3. View token usage in the top-right display
4. Click "Reset Chat" to clear conversation history

Status Checks

Use the "Check Ollama" and "Check Weaviate" buttons to verify connections and see available models/collections.

Architecture

```
┌─────────────────┐     ┌─────────────────┐     ┌─────────────────┐
│   Gradio UI     │────▶│  Pydantic AI    │────▶│     Ollama      │
│   (port 7860)   │     │     Agent       │     │  (port 11434)   │
└─────────────────┘     └─────────────────┘     └─────────────────┘
                                 │
                                 │
                                 ▼
                        ┌─────────────────┐
                        │    Weaviate     │
                        │  (port 8080)    │
                        └─────────────────┘
```

PROF

Configuration

Environment variables (set in `docker-compose.yml`):

Variable	Default	Description
`OLLAMA_BASE_URL`	`http://ollama:11434`	Ollama API endpoint
`WEAVIATE_URL`	`http://weaviate:8080`	Weaviate endpoint

Variable	Default	Description
`CHAT_MODEL`	`llama3.2`	LLM for chat
`EMBED_MODEL`	`nomic-embed-text`	Model for embeddings
`CHAT_MODEL_MULTIMODAL`	`mistral-small3.1`	Vision-language model for multimodal
`MULTIMODAL_MODE`	`false`	Enable multimodal mode

Configuration Modes

The system supports two operating modes:

Text-Only Mode (Default)

Uses text embeddings for document retrieval.

Component	Value
Embedding	`nomic-embed-text` via text2vec-ollama
Chat Model	`llama3.2`
Collection	`Document`
Best for	Pure text documents

Multimodal Mode

Uses CLIP embeddings for cross-modal (text + image) retrieval with VLM analysis at query time.

Component	Value
Embedding	`CLIP ViT-B-32` via multi2vec-clip
Chat Model	`mistral-small3.1` (vision + tool calling)
Collection	`MultimodalDocument`
Best for	Mixed text and images

PROF

Architecture:

```
Ingestion:
  Image → CLIP embedding → Store embedding + raw blob in Weaviate
  Text  → CLIP embedding → Store embedding + content in Weaviate

Query time:
  Query → CLIP embedding → Retrieve top-K results
                               ↓
              For image results: extract raw blobs
                               ↓
```

```
                Pass query + images to mistral-small3.1
                                ↓
                VLM reasons over actual images (not pre-generated
captions)
```

To enable multimodal mode:

1. Ensure `multi2vec-clip` container is running (included in docker-compose.yml)
2. Set `MULTIMODAL_MODE=true` in docker-compose.yml for the app service
3. Ingest documents with the `--multimodal` flag:

   ```
   python scripts/ingest.py --name "my docs" --multimodal
   ```

4. Restart the app:

   ```
   docker compose up -d --build app
   ```

Notes:

- `mistral-small3.1` is ~13GB quantized, requires ~24GB VRAM for good performance
- Images are stored as raw blobs during ingestion (no caption generation - faster ingestion)
- At query time, retrieved images are passed directly to the VLM for analysis
- Hybrid search: text uses BM25 + vector, images use vector-only search (CLIP)
- Up to 3 images are passed to the VLM per query to avoid context overflow

Troubleshooting

Models not loading: Check Ollama logs with `docker logs ollama`. First startup downloads ~2GB of models (more for multimodal mode with mistral-small3.1).

PROF

Weaviate connection errors: Ensure Weaviate is healthy with `docker compose ps`. The app will show connection status.

Out of GPU memory: llama3.2 (3B) requires ~4GB VRAM. For multimodal mode, mistral-small3.1 requires ~24GB VRAM.

Images not being analyzed: Ensure you're using "Force" RAG mode, as "Auto" mode may not always trigger search. Images require explicit BLOB property retrieval from Weaviate.