Process Copilot Mini

AI-powered industrial process assistant with document Q&A (RAG) and alarm analysis capabilities.

Built for learning GenAI/RAG concepts in industrial automation context. This is a complete, runnable prototype that demonstrates production-ready patterns while remaining educational and extensible.

Features

Document Q&A (RAG with Citations)

- **Input**: Questions about technical PDFs/manuals/SOPs
- **Output**: Grounded answers with bullet citations [title, page, score]
- **Behavior**: Returns "I don't know" for low-confidence retrieval, never hallucinates

Alarm/Process Explainer

- **Input**: Process tag and time window over alarm CSV data
- **Output**: Data summary (trends, thresholds) + procedural guidance with citations
- **Behavior**: Combines quantitative analysis with document-based procedures

Quick Start

Prerequisites

- Python 3.11+
- 4GB+ RAM (for sentence transformers)
- Git

Installation

```
# Clone and navigate
git clone <repository>
cd process-copilot-mini

# Create virtual environment
python -m venv .venv
source .venv/bin/activate  # Windows: .venv\Scripts\activate

# Install dependencies
pip install -r requirements.txt
```

```
# Copy environment template
cp .env.example .env
```

Add Sample Documents

```
# Create PDF directory
mkdir -p data/pdfs

# Add your PDF manuals to data/pdfs/
# For demo: create text files and save as PDFs, or use provided sample content
```

Build Knowledge Base

```
# Process PDFs and build vector index
python -m src.ingest   # Extract text from PDFs
python -m src.index    # Build FAISS vector index
```

Run the API

```
# Start FastAPI server
uvicorn src.app:app --host 0.0.0.0 --port 8000

# Test endpoints
curl http://localhost:8000/health
curl -X POST http://localhost:8000/ask -H "Content-Type: application/json" -d '{"query":
```

Run the UI (Optional)

```
# In another terminal
streamlit run ui/app_ui.py
```

Docker Deployment

```
# Build image
docker build -t process-copilot-mini .

# Run container
docker run -p 8000:8000 process-copilot-mini
```

API Endpoints

`GET /health`

Health check endpoint

```
{
  "status": "ok",
  "timestamp": "2024-08-20T18:00:00",
  "version": "1.0.0"
}
```

`POST /ask`

Document Q&A with RAG

```
curl -X POST http://localhost:8000/ask \
  -H "Content-Type: application/json" \
  -d '{"query": "What are the safety procedures for high temperature alarms?"}'
```

Response:

```
{
  "answer": "For high temperature alarms, immediately check...",
  "citations": [
    {"title": "Safety_Manual", "page": 15, "score": 0.892},
    {"title": "Operating_Procedures", "page": 23, "score": 0.756}
  ]
}
```

`GET /explain_alarm`

Alarm analysis with guidance

```
curl "http://localhost:8000/explain_alarm?tag=Temp_101&start=2024-08-20T15:30:00&end=2024
```

Response:

```
{
  "summary_from_data": "Process tag Temp_101 analysis:\n• Data points: 60 over 1.0 hours.
  "answer": "Based on the rising temperature trend, follow these procedures...",
  "citations": [...]
}
```

Project Structure

```
process-copilot-mini/
├── README.md              # This file
├── requirements.txt       # Python dependencies
├── Dockerfile             # Container definition
```

```
├── .env.example          # Environment template
├── data/
│   ├── pdfs/            # Technical PDFs (add your documents here)
│   └── samples/
│       └── alarms.csv   # Sample alarm data
├── models/
│   └── vector_index/    # FAISS index files (generated)
├── src/
│   ├── config.py        # Configuration management
│   ├── utils.py         # Helper functions and logging
│   ├── ingest.py        # PDF processing and chunking
│   ├── index.py         # Vector indexing with FAISS
│   ├── rag.py           # RAG system with confidence gating
│   ├── alarms.py        # Alarm data analysis
│   └── app.py           # FastAPI application
├── ui/
│   └── app_ui.py        # Streamlit interface
└── tests/
    ├── test_ingest.py   # Ingestion tests
    ├── test_rag.py      # RAG pipeline tests
    └── test_alarms.py   # Alarm analysis tests
```

Configuration

Key environment variables in `.env`:

```
# Paths
DATA_DIR=./data
INDEX_DIR=./models/vector_index

# Model settings
EMBEDDING_MODEL=all-MiniLM-L6-v2
RETRIEVAL_K=5
SCORE_THRESHOLD=0.35

# API settings
API_HOST=0.0.0.0
API_PORT=8000
LOG_LEVEL=INFO
```

Adding Your Own Documents

1. **Add PDFs**: Place technical manuals in `data/pdfs/`
2. **Rebuild Index**: Run `python -m src.index`
3. **Test**: Ask questions about your documents via API or UI

Supported formats: PDF files with extractable text (not scanned images)

🔍 Example Queries

Document Q&A:

- "What is the normal operating temperature range?"
- "How do you calibrate the pressure transmitter?"
- "What maintenance is required for the heat exchanger?"
- "What are the emergency shutdown procedures?"

Alarm Analysis:

- Tag: `Temp_101`, Time: `2024-08-20 15:30:00` to `2024-08-20 16:30:00`
- Tag: `Pressure_202`, Time: `2024-08-20 15:00:00` to `2024-08-20 16:00:00`

🔍 How It Works

RAG Pipeline

1. **Ingestion**: PDFs → text extraction → chunking with metadata
2. **Indexing**: Text chunks → sentence embeddings → FAISS vector index
3. **Retrieval**: Query → embedding → similarity search → ranked chunks
4. **Generation**: Retrieved context + query → grounded answer + citations
5. **Confidence**: Low similarity scores → "I don't know" responses

Alarm Analysis

1. **Data Loading**: CSV with timestamp, tag, value, alarm_state columns
2. **Filtering**: Extract data for specific tag and time window
3. **Analysis**: Compute trends, statistics, alarm transitions
4. **Guidance**: Query documents for relevant procedures using RAG
5. **Combination**: Merge data insights with procedural recommendations

🔍 Technical Choices Explained

Why FAISS?

- Fast similarity search for production scale
- Supports exact and approximate search
- Easy persistence and loading
- Industry standard for vector databases

Why Sentence Transformers?

- Pre-trained models for semantic understanding
- Efficient inference without GPU requirements
- Good balance of quality and speed
- Easy to swap models for different domains

Why Template-based Generation?

- Educational clarity (easy to understand)
- No external API dependencies
- Deterministic responses for testing
- Easy to replace with real LLM later

Why Confidence Gating?

- Prevents hallucination when documents don't contain answers
- Maintains user trust through honest "I don't know" responses
- Critical for industrial safety applications

Extension Ideas

Immediate Improvements:

- Replace template generation with OpenAI API or local LLM
- Add more sophisticated chunking (sentence boundaries, headers)
- Implement query expansion and re-ranking
- Add authentication and rate limiting

Advanced Features:

- Multi-modal RAG (images, diagrams from PDFs)
- Real-time data streaming for alarms
- Workflow automation based on alarm patterns
- Integration with historian databases (OSIsoft PI, etc.)

Production Readiness:

- Distributed vector search with multiple indices
- Caching layer for frequent queries
- Monitoring and observability
- A/B testing for different retrieval strategies

Testing

```
# Run individual modules
python -m src.ingest    # Test PDF processing
python -m src.index     # Test vector indexing
python -m src.rag       # Test RAG pipeline
python -m src.alarms    # Test alarm analysis

# Run unit tests
python -m pytest tests/

# Test API endpoints
curl http://localhost:8000/health
```

Troubleshooting

"No index available": Run `python -m src.index` to build vector index

"No PDF files found": Add PDF files to `data/pdfs/` directory

Memory errors: Reduce `CHUNK_SIZE` or use smaller embedding model

Import errors: Activate virtual environment and reinstall requirements

API connection failed: Ensure FastAPI server is running on correct port

Learning Path (Reverse Engineering Guide)

Recommended reading order for understanding:

1. `config.py` → Environment management, configuration patterns
2. `utils.py` → Logging, text processing, helper functions
3. `ingest.py` → PDF parsing, chunking strategies, metadata handling
4. `index.py` → Embeddings, vector search, FAISS operations
5. `rag.py` → Retrieval-generation pipeline, confidence gating
6. `alarms.py` → Time-series analysis, industrial data patterns
7. `app.py` → API design, request handling, error management

Tracing a request end-to-end:

1. User asks question in UI (`app_ui.py`)
2. HTTP request to FastAPI (`app.py` `/ask` endpoint)
3. RAG system processes query (`rag.py`)
4. Vector search finds relevant chunks (`index.py`)
5. Template generates answer with citations (`rag.py`)
6. Response returned through API to UI

🎯 Interview Talking Points

Technical Depth:

- "I implemented semantic chunking to preserve context across document boundaries"
- "Used cosine similarity thresholding to prevent hallucination in low-confidence scenarios"
- "Designed modular architecture for easy LLM swapping and testing"
- "Combined quantitative process analysis with qualitative document retrieval"

Industrial Relevance:

- "Built confidence gating because wrong answers in process control can be dangerous"
- "Integrated alarm trend analysis with procedural guidance for complete decision support"
- "Used industrial data patterns (tag naming, alarm states) familiar to process engineers"
- "Designed for offline operation to meet industrial network security requirements"

Production Considerations:

- "Implemented proper error handling and structured logging for production deployment"
- "Used Docker for consistent deployment across environments"
- "Designed RESTful APIs for easy integration with existing industrial systems"
- "Added health checks and monitoring endpoints for operational visibility"

Contributing

This is an educational project. Feel free to:

- Add more sophisticated PDF processing
- Implement different embedding models
- Add more industrial data analysis features
- Improve the UI/UX
- Add comprehensive tests

License

Built with: FastAPI, sentence-transformers, FAISS, Streamlit, pandas
Purpose: Learning GenAI/RAG for industrial automation applications