



YouTube RAG Chat System







Ask questions about any YouTube video using AI-powered retrieval!



Open in Streamlit



Features

-  **YouTube Integration:** Extract transcripts from any YouTube video
-  **AI-Powered Q&A:** Ask natural language questions about video content
-  **Smart Retrieval:** Uses vector embeddings for accurate context retrieval
-  **Interactive Chat:** Streamlit-based chat interface
-  **Example Questions:** Pre-loaded suggestions to get you started
-  **Debug Mode:** View retrieved context for transparency



Live Demo

Try it now: [XYZ](#)




How It Works

1. **Extract:** Pulls transcript data from YouTube videos using the YouTube Transcript API
2. **Chunk:** Splits transcripts into manageable pieces for better retrieval
3. **Embed:** Creates vector embeddings using OpenAI's embedding models
4. **Retrieve:** Finds relevant transcript segments based on your questions
5. **Generate:** Uses GPT to answer questions based on retrieved context



Usage

Getting Started

1. Visit the [live app](#)
2. Enter a YouTube URL or video ID in the sidebar
3. Click " Process Video"
4. Start asking questions about the video!

Supported Input Formats

Full URL: `https://youtube.com/watch?v=Gfr50f6ZBvo`
Short URL: `https://youtu.be/Gfr50f6ZBvo`
Video ID: `Gfr50f6ZBvo`

Example Questions

- "Can you summarize the video?"

- "What are the main topics discussed?"
- "Who are the key people mentioned?"
- "What are the key takeaways?"

Architecture

Core Components

```
📁 Project Structure
├── streamlit_app.py      # Main Streamlit application
├── transcript_processor.py # YouTube transcript extraction
├── vector_store_manager.py # Vector database operations
├── rag_chain.py          # RAG chain implementation
├── config.py             # Configuration settings
├── requirements.txt      # Dependencies
└── README.md            # This file
```

Technology Stack

- **Frontend:** Streamlit
- **LLM:** OpenAI GPT-4o-mini
- **Embeddings:** OpenAI text-embedding-3-small
- **Vector Store:** FAISS
- **Framework:** LangChain
- **Transcript API:** YouTube Transcript API

Local Development

Prerequisites

- Python 3.8+
- OpenAI API key

Installation

1. Clone the repository

```
git clone <repository-url>
cd youtube-rag-chat
```

2. Install dependencies

```
pip install -r requirements.txt
```

3. Set up environment variables

```
# Create .env file  
echo "OPENAI_API_KEY=your_openai_api_key_here" > .env
```

4. Run the application

```
streamlit run streamlit_app.py
```

5. Open your browser

```
http://localhost:8501
```

Configuration

Edit `config.py` to customize:

```
# Model configurations  
EMBEDDING_MODEL = "text-embedding-3-small"  
LLM_MODEL = "gpt-4o-mini"  
LLM_TEMPERATURE = 0.2  
  
# Text splitting  
CHUNK_SIZE = 1000  
CHUNK_OVERLAP = 200  
  
# Retrieval settings  
RETRIEVAL_K = 4  
SEARCH_TYPE = "similarity"
```

Requirements

```
youtube-transcript-api  
langchain-community  
langchain-openai  
faiss-cpu  
tiktoken  
python-dotenv  
streamlit
```

Use Cases

- **Educational Content:** Quickly find specific information in lectures or tutorials
- **Research:** Extract key insights from research presentations
- **Business:** Analyze webinars, conference talks, or product demos
- **Entertainment:** Get summaries of podcasts or interview content
- **Learning:** Create study materials from educational videos

⚠ Limitations

- **Transcript Availability:** Only works with videos that have captions/transcripts
- **Language Support:** Currently optimized for English content
- **Video Length:** Very long videos may require chunking for optimal performance
- **Rate Limits:** Subject to OpenAI API rate limits

🔒 Privacy & Security

- **No Data Storage:** Transcripts are processed in memory only
- **API Security:** OpenAI API keys are securely managed
- **No User Tracking:** No personal information is collected or stored

🤝 Contributing

We welcome contributions! Here's how to get started:

1. Fork the repository
2. Create a feature branch (`git checkout -b feature/amazing-feature`)
3. Commit your changes (`git commit -m 'Add amazing feature'`)
4. Push to the branch (`git push origin feature/amazing-feature`)
5. Open a Pull Request

Development Guidelines

- Follow PEP 8 style guidelines
- Add docstrings to new functions
- Include error handling
- Test with various video types

📊 Performance

- **Processing Time:** ~10-30 seconds for typical videos
- **Memory Usage:** Optimized for efficiency with chunking
- **Accuracy:** High-quality responses using GPT-4o-mini
- **Scalability:** Stateless design for easy deployment

🐛 Troubleshooting

Common Issues

"Failed to extract transcript"

- Ensure the video has captions/subtitles available
- Try with a different video
- Check if the video is public and accessible




"OpenAI API Error"

- Verify your API key is correct
- Check your OpenAI account credits
- Ensure you have access to the required models

"Video processing stuck"

- Very long videos may take more time
- Try with a shorter video first
- Check your internet connection

Getting Help

-  Create an issue on GitHub
-  Check existing issues for solutions
-  Review the documentation



Roadmap

- ☐ **Multi-language Support:** Support for non-English videos
- ☐ **Video Series:** Process multiple videos as a knowledge base
- ☐ **Advanced Search:** Semantic search across video collections
- ☐ **Export Features:** Save conversations and insights
- ☐ **API Access:** Programmatic access to the RAG system
- ☐ **Custom Models:** Support for other LLM providers



License

This project is licensed under the MIT License - see the [LICENSE](#) file for details.



Acknowledgments

- **OpenAI** for the powerful language models
- **LangChain** for the excellent RAG framework
- **Streamlit** for the intuitive web interface
- **YouTube Transcript API** for transcript access
- **FAISS** for efficient vector similarity search

Built with ❤️ for the AI community

[Live Demo](#) • [Report Bug](#) • [Request Feature](#)