Synth-RAG: A Hybrid Retrieval-Augmented Generation System for MIDI Synthesizer Manuals using ColPali Vision-Language Models

Adhithya Bhaskar
ISE 547 Project
adhithya@usc.edu

December 7, 2025

Abstract

Retrieval-Augmented Generation (RAG) systems have become essential for querying large document collections, yet traditional approaches struggle with visually rich documents containing tables, diagrams, and complex layouts. This project presents Synth-RAG (https://adbx.github.io/synth-rag/), a hybrid RAG system designed for querying PDF manuals of MIDI synthesizers. Our approach leverages ColPali, a vision-language model that processes PDF pages directly as images, preserving visual information that text-only methods discard. We implement a novel two-stage retrieval architecture: the first stage uses HNSW-indexed mean-pooled multivectors alongside dense (FastEmbed) and sparse (BM25) embeddings for fast candidate retrieval, while the second stage performs precise reranking using original ColPali multivectors with MaxSim scoring. The system is augmented with a LangGraph-powered agentic workflow that combines manual retrieval with web search fallback for comprehensive question answering. We evaluate performance using the RAGBench dataset with RAGAS and TruLens metrics. Evaluation on the RAGBench emanual dataset demonstrates strong performance with RAGAS faithfulness scores of 0.80–0.83 and TruLens groundedness of 0.79–0.85 across runs of 10 to 1,000 queries, with a hallucination detection AUROC of 0.70.

1. Introduction

Retrieval-Augmented Generation (RAG) has emerged as a powerful paradigm for building question-answering systems over large document collections. By combining retrieval mechanisms with large language models (LLMs), RAG systems can provide grounded, accurate responses with citations to source documents. However, traditional RAG approaches face significant limitations when dealing with visually rich documents such as technical manuals, which contain tables, diagrams, flowcharts, and complex page layouts that carry semantic meaning beyond their textual content.

MIDI synthesizer manuals exemplify this challenge. These technical documents contain intricate signal flow diagrams, parameter tables, button layouts, and visual representations of synthesis architectures that are essential for understanding the equipment. Traditional OCR-based approaches lose critical visual information:

- **Visual layout and structure**: The spatial arrangement of elements conveys relationships
- **Table structures**: Parameter mappings and specifications are often tabular

- **Diagrams and figures**: Signal flows and user interfaces are visual
- **Font emphasis**: Bold, italic, and sizing indicate importance

This project addresses these limitations by developing Synth-RAG, a hybrid retrieval-augmented generation system with the following objectives:

1. **Vision-Language Processing**: Utilize ColPali, a state-of-the-art vision-language model, to process PDF pages directly as images, capturing both textual and visual semantics.
2. **Hybrid Search Architecture**: Implement a multi-vector search strategy combining dense embeddings (FastEmbed), sparse embeddings (BM25), and ColPali multivector representations for robust, query-agnostic retrieval.
3. **Two-Stage Retrieval**: Develop an efficient retrieval pipeline using mean-pooled multivectors for fast first-stage retrieval with HNSW indexing, followed by precise reranking with original ColPali embeddings.
4. **Agentic Workflow**: Create a LangGraph-powered agent that intelligently combines manual retrieval with web search for comprehensive answers.
5. **Systematic Evaluation**: Benchmark the system using the RAGBench dataset with established metrics including RAGAS faithfulness/context relevancy and TruLens groundedness/context relevance.

The remainder of this report is organized as follows: Section 2 describes the datasets used, Section 3 details the experimental setup and system architecture, Section 4 presents results, Section 5 provides discussion and analysis, Section 6 outlines future directions, and Section 7 concludes.

2. Data

This section describes the datasets used for development and evaluation of the Synth-RAG system.

2.1. Primary Dataset: MIDI Synthesizer Manuals

The primary dataset consists of PDF manuals for Elektron MIDI synthesizers, specifically:

- **Elektron Digitone II**: FM synthesis workstation manual
- **Elektron Digitakt**: Drum machine and sampler manual
- Additional synthesizer documentation

The dataset is organized into two subsets:

Subset	Number of PDFs	Purpose
Test	3	Development and quick iteration
Full	8	Complete evaluation

Table 1: MIDI synthesizer manual dataset organization

These manuals exhibit characteristics that make them challenging for traditional text-based RAG:

- **Complex visual layouts**: Multi-column pages with sidebars and callouts
- **Technical diagrams**: Signal flow charts, block diagrams, and interface layouts

- **Parameter tables**: MIDI CC mappings, synthesis parameters, and specifications
- **Annotated screenshots**: UI elements with numbered references

2.2. Benchmarking Dataset: RAGBench

For systematic evaluation, we utilize the RAGBench dataset from Hugging Face (`rungalileo/ragbench`), specifically the `emanual` sub-dataset which contains electronic manual question-answer pairs.

Split	Number of Examples
Train	1,054
Validation	132
Test	132
Total	**1,318**

Table 2: RAGBench emanual dataset statistics

Each RAGBench example includes:

- A question about the manual content
- Ground truth documents (relevant passages)
- Ground truth response (expected answer)
- Metadata for evaluation (adherence, relevance, utilization, completeness scores)

The RAGBench dataset provides standardized evaluation with pre-computed ground truth labels, enabling comparison with published baselines.

2.3. Data Preprocessing

The ingestion pipeline performs the following preprocessing steps:

1. **PDF Rendering**: Each PDF page is rendered to RGB images at 2x scale using `pypdfium2`, producing high-quality inputs for the vision-language model.
2. **Text Extraction**: Plain text is extracted per page using `pymupdf` for:
   - Dense and sparse text embeddings
   - Payload metadata in search results
   - Human-readable context snippets
3. **Semantic Chunking**: Extracted text is chunked using `semantic-text-splitter` with 512-token chunks and 50-token overlap for text-based embeddings.

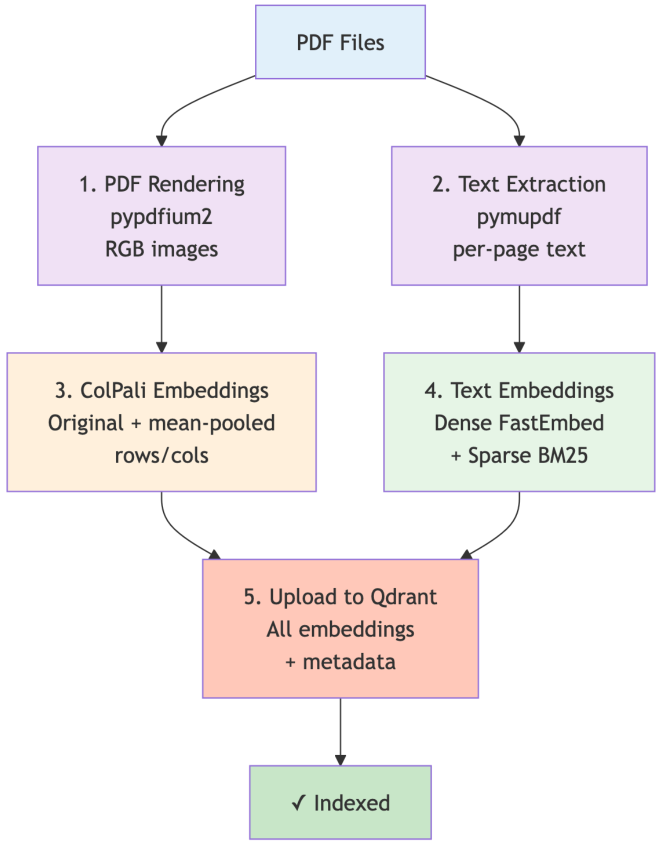

Figure 1: Ingestion pipeline for PDF manual processing. The pipeline processes each PDF through parallel paths: visual processing via ColPali embeddings and text processing via dense and sparse embeddings, before uploading all representations to Qdrant.

3. Experimental Setup

This section details the technical architecture, models, and evaluation methodology.

3.1. Technology Stack

The system is built on a combination of cloud services and open-source libraries. For vector storage and retrieval, we use Qdrant Cloud, a managed vector database service that supports multiple named vectors per point, sparse vectors, and HNSW indexing with configurable parameters. Response generation is powered by the OpenAI API using GPT-4o-mini, chosen for its balance of quality and cost-effectiveness. The agentic workflow includes web search capability via the Brave Search API, which provides fallback information retrieval when manual content is insufficient.

The core embedding pipeline relies on several key libraries: ColPali (`vidore/colpali-v1.3`) for vision-language embeddings, FastEmbed with the `sentence-transformers/all-MiniLM-L6-v2` model for dense text embeddings, and Qdrant's BM25 implementation for sparse keyword embeddings. PDF processing uses `pypdfium2` for high-quality page rendering and `pymupdf` for text extraction. The agentic orchestration is implemented with LangGraph, providing a stateful graph-based workflow for tool selection and response generation. A Gradio-based web interface provides an interactive chat experience for end users.

3.2. Embedding Models

The system employs three complementary embedding approaches:

3.2.1. ColPali Vision-Language Model

ColPali (`vidore/colpali-v1.3`) is a vision-language model that generates multivector embeddings from document images. For each page, ColPali produces approximately 1,030 vectors of 128 dimensions (32×32 image patches plus special tokens).

Variant	Dimensions	Purpose
Original	1030×128	Precise reranking (no HNSW)
Row-pooled	32×128	Fast vertical structure matching
Col-pooled	32×128	Fast horizontal structure matching

Table 3: ColPali embedding variants

Mean-pooling the original multivectors by rows and columns creates compact representations that can be efficiently indexed with HNSW while preserving structural information.

3.2.2. Dense Embeddings (FastEmbed)

Dense text embeddings are generated using `sentence-transformers/all-MiniLM-L6-v2` via FastEmbed, producing 384-dimensional vectors optimized for semantic similarity.

3.2.3. Sparse Embeddings (BM25)

Sparse keyword-based embeddings use `Qdrant/bm25` with IDF weighting for exact term matching, complementing the semantic dense embeddings.

3.3. Vector Database Schema

All embeddings are stored in Qdrant with the following named vector configuration:

```
{
  "colpali_original": [1030 × 128],  // No HNSW (reranking only)
  "colpali_rows": [32 × 128],        // HNSW indexed
  "colpali_cols": [32 × 128],        // HNSW indexed
  "dense": [384],                    // HNSW indexed
  "sparse": {indices, values}        // Inverted index
}
```

Each point stores payload metadata including manual name, page number, extracted text, and image path.

3.4. Two-Stage Retrieval Architecture

The retrieval pipeline implements a two-stage approach that balances efficiency with accuracy. The first stage performs fast prefetch using HNSW-indexed vectors, retrieving candidate documents via dense embeddings for semantic similarity, sparse embeddings for keyword matching, and mean-pooled ColPali representations for both vertical (row-pooled) and horizontal (column-pooled) structure matching. Each prefetch retrieves the top 50 candidates, which are then merged for the reranking stage.

The second stage performs precise reranking using original ColPali multivectors with MaxSim scoring:

$$\text{MaxSim}(Q, D) = \sum_{i=1}^{|Q|} \max_{j=1}^{|D|} \text{sim}\left(q_i, d_j\right)$$

where Q is the query embedding and D is the document embedding. This scoring function finds the maximum similarity for each query vector across all document vectors, providing fine-grained matching that captures nuanced visual and textual relationships.

3.5. Agentic RAG with LangGraph

The system includes a LangGraph-powered agent with two primary tools: a Manual Retriever Tool that performs hybrid search with ColPali reranking to find relevant manual pages, and a Web Search Tool using the Brave Search API that serves as a fallback when manual retrieval is insufficient. The agent follows strict behavioral rules: it always queries manuals first without exception, cites sources with manual name and page number, structures responses with clear sections, and uses web search only as supplementary information.

3.6. Language Model

Response generation uses OpenAI's GPT-4o-mini model with temperature set to 0 for deterministic outputs. This model was selected for its strong instruction-following capabilities and cost-effectiveness for high-volume benchmarking. The model receives retrieved contexts as part of a structured prompt and generates grounded answers with citations to specific manual pages. The choice of a smaller model also enables faster response times, which is particularly important for the interactive chat interface.

3.7. Evaluation Metrics

Performance is evaluated using two complementary frameworks:

3.7.1. RAGAS Metrics

- **Faithfulness**: Measures how grounded the response is in retrieved contexts
- **Context Relevancy**: Measures how relevant retrieved contexts are to the question

3.7.2. TruLens Metrics

- **Groundedness**: Similar to faithfulness; checks if response is supported by context
- **Context Relevance**: Evaluates if contexts contain information to answer the question

3.7.3. Aggregate Metrics

- **Hallucination AUROC**: Area under ROC curve for hallucination detection
- **Relevance RMSE**: Root mean squared error for relevance predictions
- **Performance**: Query time, generation time, total latency

4. Results

This section presents experimental results from evaluating the Synth-RAG system on the RAG-Bench emanual dataset.

4.1. RAGBench Benchmark Results

We evaluated the system on the RAGBench emanual dataset across three runs of increasing size: 10 queries (test split), 100 queries (test split), and 1,000 queries (train split). All runs used `gpt-4o-mini` with top-k=5 retrieval and prefetch limit of 50.

Metric	n=10 (test)	n=100 (test)	n=1000 (train)
RAGAS Faithfulness	0.795 ± 0.228	0.798 ± 0.256	0.832 ± 0.245
RAGAS Answer Relevancy	0.735 ± 0.369	0.711 ± 0.412	0.603 ± 0.455
TruLens Groundedness	0.852 ± 0.118	0.812 ± 0.136	0.792 ± 0.137
TruLens Context Relevance	0.633 ± 0.277	0.653 ± 0.353	0.662 ± 0.350
Hallucination AUROC	–	0.702	–
Relevance RMSE	0.671	0.630	–

Table 4: RAGBench emanual evaluation metrics across three benchmark runs

Key observations from the benchmark results:

- **Faithfulness improves with scale**: RAGAS faithfulness increases from 0.795 to 0.832 as sample size grows, suggesting consistent grounding behavior.
- **Answer relevancy varies by split**: The train split shows lower answer relevancy (0.603) compared to test (0.735), likely reflecting different question difficulty distributions.
- **High groundedness**: TruLens groundedness remains consistently high (0.79–0.85) across all runs, indicating responses are well-supported by retrieved contexts.
- **Context relevance stable**: TruLens context relevance is consistent around 0.63–0.66, showing reliable retrieval quality.

4.2. Query Latency

Latency measurements were collected across all benchmark runs. The pipeline timing breaks down into query processing (embedding generation + hybrid search with reranking) and LLM response generation.

Stage	n=10	n=100	n=1000
Query + Retrieval	0.11s	0.22s	0.06s
LLM Generation	3.24s	3.26s	3.18s
Total	**3.35s**	**3.49s**	**3.25s**

Table 5: Query latency breakdown across benchmark runs (mean time in seconds)

The results show that LLM response generation dominates the total latency at approximately 3.2 seconds per query, while retrieval is comparatively fast at 0.06–0.22 seconds. The variation in query time across runs reflects caching effects and system load rather than algorithmic differences.

4.3. Qualitative Example

The following example demonstrates the agentic RAG system's ability to synthesize information from multiple manuals and supplement it with web search results.

User Query:

How can I setup encoders of the Faderfox EC-4 MIDI controller such that encoders 2, 3 & 4 control the Reverb of the Digitone II on MIDI channels 6, 7, & 8 respectively? Then, I also want to use encoder 1 to control encoders 2, 3 & 4 simultaneously on the Faderfox.

Agent Response:

Information from Manuals

To set up the Faderfox EC-4 MIDI controller to control the reverb of the Digitone II using encoders 2, 3, and 4 on MIDI channels 6, 7, and 8 respectively, and to use encoder 1 to control encoders 2, 3, and 4 simultaneously, follow these steps:

1. Faderfox EC-4 Setup:

- The Faderfox EC-4 is a universal MIDI controller with 16 push-encoders and a large OLED display. It allows for various MIDI commands, including control change in both relative and absolute modes (Faderfox EC4 Manual V03, Page 3).
- To assign MIDI channels to specific encoders, you need to access the setup mode by pressing the gray SETUP key. This will display all available setups, and you can select the desired setup by pressing one of the 16 associated encoders (Faderfox EC4 quick start guide, Page 0).

2. Digitone II MIDI Control:

- The Digitone II allows for MIDI control of its parameters, including reverb, through MIDI CC messages. You will need to refer to the MIDI implementation chart in the Digitone II manual to find the specific CC numbers for reverb control (Digitone II Appendix & MIDI CC, Page 19).

3. Linking Encoders:

- The Faderfox EC-4 supports a link function that allows you to link multiple encoders. This can be used to control multiple parameters simultaneously. You can link encoders 2, 3, and 4 to encoder 1 so that adjusting encoder 1 will simultaneously adjust the other three (Faderfox EC4 Manual V03, Page 3).

Additional Web Search Results:

- The Faderfox EC-4 features a "link function" that allows you to link up to 16 encoders or pushbuttons for multiple commands, which is useful for controlling multiple parameters simultaneously (Faderfox EC4, Perfect Circuit).
- The EC-4′s MIDI learn functionality can be used to quickly assign incoming MIDI signals to the encoders, making it easier to set up the desired control mappings (Faderfox EC4, Perfect Circuit).

By following these steps and utilizing the link function, you can effectively set up your Faderfox EC-4 to control the reverb on the Digitone II as desired.

Analysis:

The agent first retrieved relevant pages from the indexed manuals, then supplemented with web search. The response demonstrates multi-source synthesis:

1. **Faderfox EC-4 Setup:** The system identified that the EC-4 is a universal MIDI controller with 16 push-encoders and retrieved instructions for accessing setup mode via the SETUP key to assign MIDI channels to specific encoders (Faderfox EC4 Manual V03, Page 3; Quick Start Guide, Page 0).
2. **Digitone II MIDI Control:** The agent retrieved information about MIDI CC control for reverb parameters, directing the user to the MIDI implementation chart (Digitone II Appendix & MIDI CC, Page 19).
3. **Linking Encoders:** The system identified the EC-4′s link function that allows linking multiple encoders for simultaneous control, enabling encoder 1 to adjust encoders 2, 3, and 4 together (Faderfox EC4 Manual V03, Page 3).
4. **Web Search Supplement:** The agent's web search fallback provided additional context about the EC-4′s MIDI learn functionality for quick assignment of incoming MIDI signals (Perfect Circuit).

This example illustrates several system capabilities: cross-manual retrieval (combining Faderfox and Digitone II documentation), specific page citations, and intelligent use of web search to supplement manual content with practical usage information.

5. Discussion

The query from the example is a complex question that requires knowledge and understanding of both devices in question, MIDI protocol, how they are connected and how to configure them. The agent demonstrates the ability to retrieve relevant information from the manuals with references to page numbers and supplement it with web search results to provide a comprehensive answer.

5.1. Limitations

Several limitations should be acknowledged:

1. **Benchmarking Scope**: The RAGBench evaluation currently uses text-only embeddings rather than the full ColPali pipeline due to dataset format constraints.
2. **Single Domain**: Evaluation focuses on the `emanual` sub-dataset; generalization to other domains requires additional testing.
3. **Agentic Evaluation**: The benchmarking system evaluates hybrid search only; the agentic RAG workflow requires separate evaluation methodology.
4. **Computational Requirements**: ColPali requires significant GPU memory (2GB model), limiting deployment options.

6. Potential Future Directions

Based on the current implementation and identified limitations, several concrete extensions are possible. The RAGBench evaluation pipeline currently uses text-only embeddings for compatibility, so a natural next step would be full ColPali integration with benchmarking. This would involve rendering RAGBench documents as images for ColPali processing, enabling direct comparison of ColPali retrieval against text-only baselines and measuring the impact of visual understanding on specific query types. Additionally, developing evaluation methodology for the LangGraph agent would provide insights into tool selection accuracy, multi-step reasoning quality, and how agent responses compare to single-retrieval approaches.

The system could benefit from extended dataset evaluation across RAGBench's 12 sub-datasets spanning different domains, including `covidqa` for medical information, `finqa` for financial document understanding, and `techqa` for technical support documentation. This would test generalization capabilities beyond the `emanual` domain. Domain adaptation presents another opportunity: fine-tuning ColPali on synthesizer manual layouts could improve recognition of signal flow diagrams and enhance MIDI-specific terminology understanding through domain-specific training data.

Finally, retrieval optimization remains an open area for improvement. Experimenting with different mean-pooling strategies beyond row and column pooling, implementing query expansion techniques to improve recall, and exploring cross-encoder reranking as an alternative or complement to MaxSim scoring could all potentially enhance retrieval quality.

7. Conclusion

This project presented Synth-RAG, a hybrid retrieval-augmented generation system for querying MIDI synthesizer manuals. The system addresses the challenge of processing visually rich technical documents by combining:

- **ColPali vision-language embeddings** that process PDF pages as images, preserving visual layout and structure
- **Hybrid search** combining dense, sparse, and multivector representations for robust retrieval
- **Two-stage retrieval** with efficient HNSW-indexed prefetch and precise MaxSim reranking
- **Agentic workflow** with intelligent tool selection and web search fallback

Evaluation on the RAGBench emanual dataset demonstrated strong performance: RAGAS faithfulness scores ranged from 0.80 to 0.83, TruLens groundedness remained consistently high at 0.79–0.85, and the system achieved a hallucination detection AUROC of 0.70. The retrieval pipeline proved efficient, with query processing completing in under 0.25 seconds while LLM generation dominated overall latency at approximately 3.2 seconds per query.

The key takeaway is that vision-language models offer a promising approach for RAG systems dealing with documents where visual information carries semantic meaning. By processing documents as images rather than extracted text, systems can better understand tables, diagrams, and layouts that are essential for technical documentation.

---

8. Supporting Links

8.1. Demo Video

https://jumpshare.com/share/ZuUYyPeYZe3fpdPzjMwQ

8.2. GitHub Repository

https://github.com/adbX/synth-rag

8.3. Project Website

https://adbx.github.io/synth-rag/