RAG System Test Document

Introduction

This is a comprehensive test PDF document designed to evaluate the text extraction and chunking capabilities of the Goose RAG system. The document contains multiple pages with varied content to test the robustness of the PDF processing pipeline.

Technical Overview

The Retrieval-Augmented Generation (RAG) architecture combines the power of large language models with external knowledge retrieval. This approach enables systems to provide accurate, contextual responses based on specific document collections rather than relying solely on pre-trained knowledge.

System Components

1. **PDF Reader MCP Server**: Handles document ingestion and text extraction from various PDF formats, preserving metadata and structure information.
2. **Text Chunking Engine**: Implements intelligent document segmentation with configurable chunk sizes, overlap strategies, and boundary detection to maintain semantic coherence.
3. **Vector Storage**: Utilizes PostgreSQL with pgvector extension to store high-dimensional embeddings generated by local transformer models for efficient similarity search operations.
4. **Hybrid Retrieval**: Combines dense vector search with sparse BM25 ranking to achieve optimal relevance scoring across diverse query types.

Implementation Details

The system employs a FastAPI backend with SQLAlchemy ORM for database operations. Document processing utilizes asyncio for concurrent handling of multiple ingestion tasks. The embedding model runs entirely locally using sentence-transformers, ensuring data privacy and eliminating external API dependencies.

Performance Considerations

Chunking parameters significantly impact both retrieval quality and system performance. Smaller chunks (400-600 tokens) provide precise context matching but may lose broader semantic relationships. Larger chunks (800-1200 tokens) maintain context but can dilute relevance scores. The default configuration of 800 tokens with 100-token overlap represents an optimal balance for general-purpose document retrieval.

Testing Methodology

Comprehensive testing includes unit tests for individual components, integration tests for end-to-end workflows, and performance benchmarks for latency and throughput evaluation. Smoke tests verify basic functionality across different document types and sizes.