Doc May Cry

Keywords

Document Summarization
LLM Integration
Text-to-Speech
File Processing
Question Generation
React Frontend
Python API

Primary: Document Summarization, LLM Integration, Text-to-Speech, File Processing, Question Generation, React Frontend, Python API

Secondary: PDF Processing, Markdown Rendering, LaTeX Conversion, Key Point Extraction, Answer Generation, Tailwind CSS, Groq API

Technical: FastAPI, PyMuPDF, python-docx, pypandoc, Web Speech API, react-markdown, Groq SDK

Project Problem Statement

> Current document analysis tools require manual effort to summarize content, extract key points, and generate interactive learning materials. This project investigates a novel platform: **Can we create a web-based system that automatically summarizes uploaded documents (PDF, .txt, .docx, Markdown, LaTeX) using an LLM, provides key point definitions, generates questions with optional answers, and offers real-time text-to-speech for accessibility?**

Primary Inspiration: Advances in large language models (e.g., LLaMA, GPT) for text summarization and question generation, combined with modern web technologies for user-friendly interfaces.

Core Project Questions

1. **Document Summarization**: How effectively can the Groq Cloud LLM summarize diverse document formats while maintaining accuracy and coherence?

2. **Key Point Extraction**: Can the LLM reliably identify and define key concepts with relevant examples from the document?
3. **Question Generation**: How well can the platform generate short and broad questions, and optionally provide accurate answers based on document content?
4. **Text-to-Speech Integration**: Can the system render Markdown/LaTeX as readable text and provide seamless real-time audio playback for summaries and documents?
5. **Scalability and Usability**: How can the platform ensure fast processing, a responsive UI, and accessibility for a wide range of users?

Expected Outcomes

Functional Deliverables

- **Web Platform**: Fully functional React-based frontend with Tailwind CSS styling.
- **Backend API**: Python FastAPI backend for document processing and LLM integration.
- **Supported Formats**: PDF, .txt, .docx, Markdown, LaTeX.
- **Features**:
  - Concise document summaries.
  - Key point definitions with document-specific examples.
  - Short and broad questions with optional answer generation.
  - Text-to-speech for summaries and rendered document content.

Performance Metrics

Summarization Quality:

- **ROUGE Score**: >0.6 for summary coherence and relevance.
- **User Satisfaction**: >85% positive feedback on summary accuracy.

  Processing Speed:
- **API Response Time**: <5 seconds for summarization and feature generation.
- **File Upload**: <10 seconds for files up to 10MB.

  Question Generation:
- **Relevance**: >90% of questions align with document content.
- **Answer Accuracy**: >95% correct answers when generated.

  TTS Performance:
- **Audio Clarity**: >95% intelligibility for rendered text.
- **Rendering Accuracy**: 100% correct conversion of Markdown/LaTeX to readable text.

Technical Contributions

System Architecture: Novel integration of Groq LLM with multi-format document processing.
Frontend Design: Accessible and responsive UI with real-time TTS controls.
API Efficiency: Optimized backend for handling diverse file types and LLM requests.

Algorithmic Innovations

1. **Multi-Format Processing**: Unified pipeline for PDF, .txt, .docx, Markdown, LaTeX.
2. **Dynamic Question Generation**: LLM-driven creation of varied question types.
3. **TTS Optimization**: Seamless rendering of structured formats for audio playback.

Flexible Implementation Steps (Adaptable to 1-Month Timeline)

Phase 1: Foundation & Quick Setup

Priority: Establish a working prototype

Step 1: Environment Setup & Dependencies

- Set up Python backend (FastAPI, PyMuPDF, python-docx, pypandoc).
- Configure React frontend with Tailwind CSS and Vite.
- Obtain Groq Cloud API access and test connectivity.
- **Goal**: Functional development environment in 2-3 days.

Step 2: Baseline Implementation

- Implement file upload and basic text extraction for all formats.
- Create simple API endpoint for Groq LLM summarization.
- Build minimal React UI for file upload and summary display.
- **Goal**: End-to-end prototype with basic summarization.

Phase 2: Core Feature Development

Priority: Implement key features

Step 3: Key Point Extraction

- Develop LLM prompts for identifying key concepts and examples.

- Add API endpoint to return key points with definitions.
- Create React component to display key points.
- **Goal**: Accurate key point extraction with >90% relevance.

Step 4: Question Generation

- Implement LLM-driven question generation (short and broad).
- Add API endpoint for questions and optional answer generation.
- Build React component with "Generate Answers" button.
- **Goal**: Functional question system with toggleable answers.

Phase 3: Text-to-Speech & Optimization

Priority: Enhance accessibility and performance

Step 5: Text-to-Speech Integration

- Implement Web Speech API for real-time TTS.
- Add rendering logic for Markdown/LaTeX to plain text using pypandoc.
- Create React component with play/pause controls for summary and document.
- **Goal**: Seamless TTS for all content types.

Step 6: Performance Optimization

- Optimize API for faster response times (<5 seconds).
- Implement file size/type validation and error handling.
- Enhance UI responsiveness and accessibility (ARIA labels, keyboard navigation).
- **Goal**: Robust and user-friendly system.

Phase 4: Validation & Deployment

Priority: Ensure quality and usability

Step 7: Systematic Testing

- Test summarization, key points, questions, and TTS across all file formats.
- Evaluate performance metrics (ROUGE, response time, accuracy).
- Conduct usability testing with sample users.
- **Goal**: Validated system meeting performance targets.

Step 8: Documentation & Deployment

- Document setup, usage, and API endpoints.
- Deploy frontend (Vercel) and backend (Render).
- Outline future improvements and limitations.
- **Goal**: Deployed platform with clear documentation.

Technical Challenges & Solutions

Challenge 1: Multi-Format Processing

Solution:

- Use PyMuPDF for PDF text extraction.
- Leverage python-docx for .docx files.
- Employ pypandoc for Markdown/LaTeX conversion.
- Implement unified text normalization pipeline.

Challenge 2: LLM Prompt Engineering

Solution:

- Design specific prompts for summarization, key points, and questions.
- Use iterative testing to refine prompt accuracy.
- Implement fallback mechanisms for low-quality LLM outputs.

Challenge 3: TTS for Structured Formats

Solution:

- Convert Markdown/LaTeX to plain text before TTS.
- Use react-markdown for preview rendering in UI.
- Validate audio output for clarity and correctness.

Challenge 4: Scalability & Performance

Solution:

- Implement asynchronous API calls with FastAPI.
- Use caching for frequently accessed summaries.
- Optimize file upload with streaming for large files.
- Employ CDN for frontend asset delivery.

Future Extensions & Development Directions

Immediate Extensions (3-6 months)

- **Additional Formats**: Support for .pptx and image-based PDFs (OCR).
- **Multi-Language Support**: Summaries and TTS in multiple languages.
- **User Accounts**: Save and revisit processed documents.
- **Analytics Dashboard**: Usage statistics and summary insights.

Medium-term Directions (6-12 months)

- **Advanced Questions**: Multiple-choice and fill-in-the-blank formats.
- **Collaboration Features**: Share summaries and questions with teams.
- **Mobile Optimization**: Dedicated iOS/Android apps.
- **Custom LLM Fine-Tuning**: Tailored Groq model for specific domains.

Long-term Vision (1-3 years)

- **AI-Powered Learning Platform**: Adaptive learning paths based on user interactions.
- **Real-Time Collaboration**: Live document analysis with multiple users.
- **Enterprise Integration**: API for integration with LMS and CMS.
- **Automated Content Creation**: Generate presentations or reports from summaries.