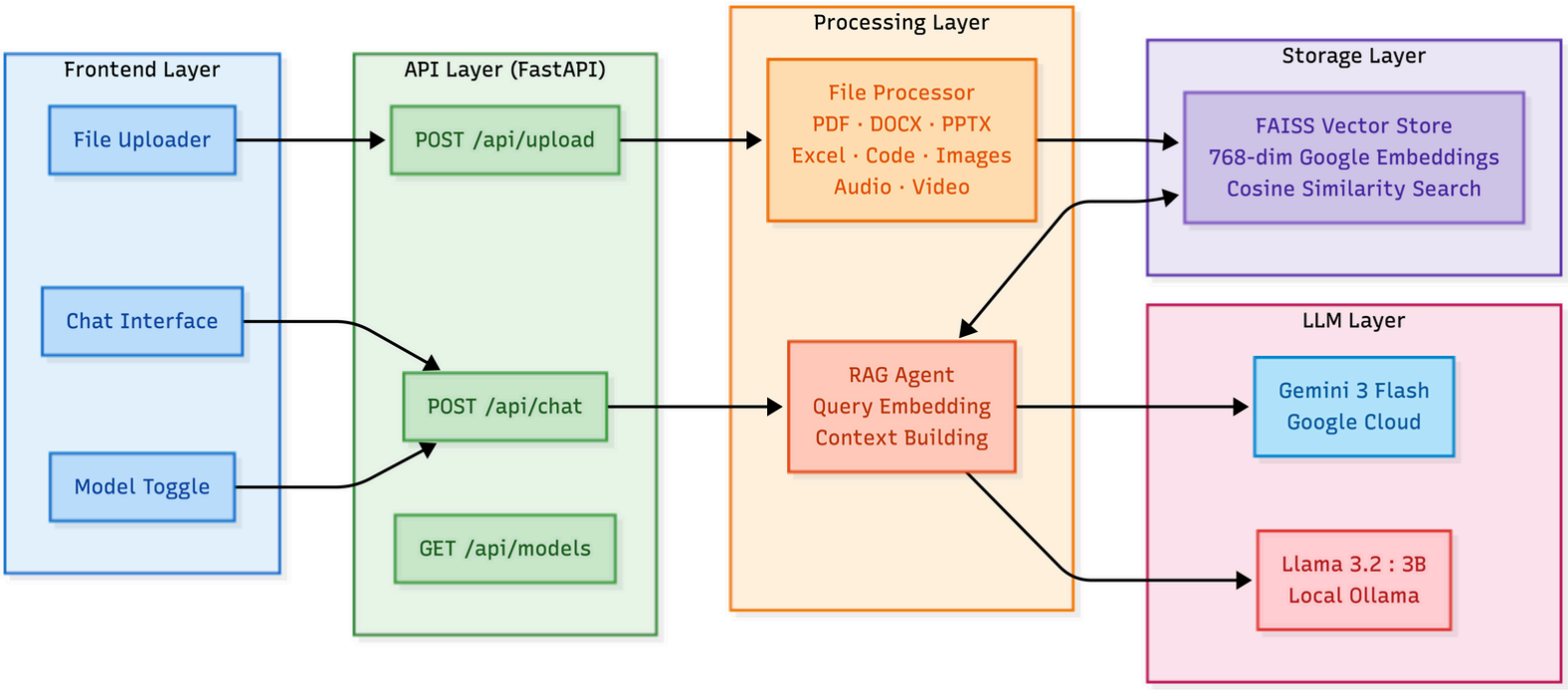
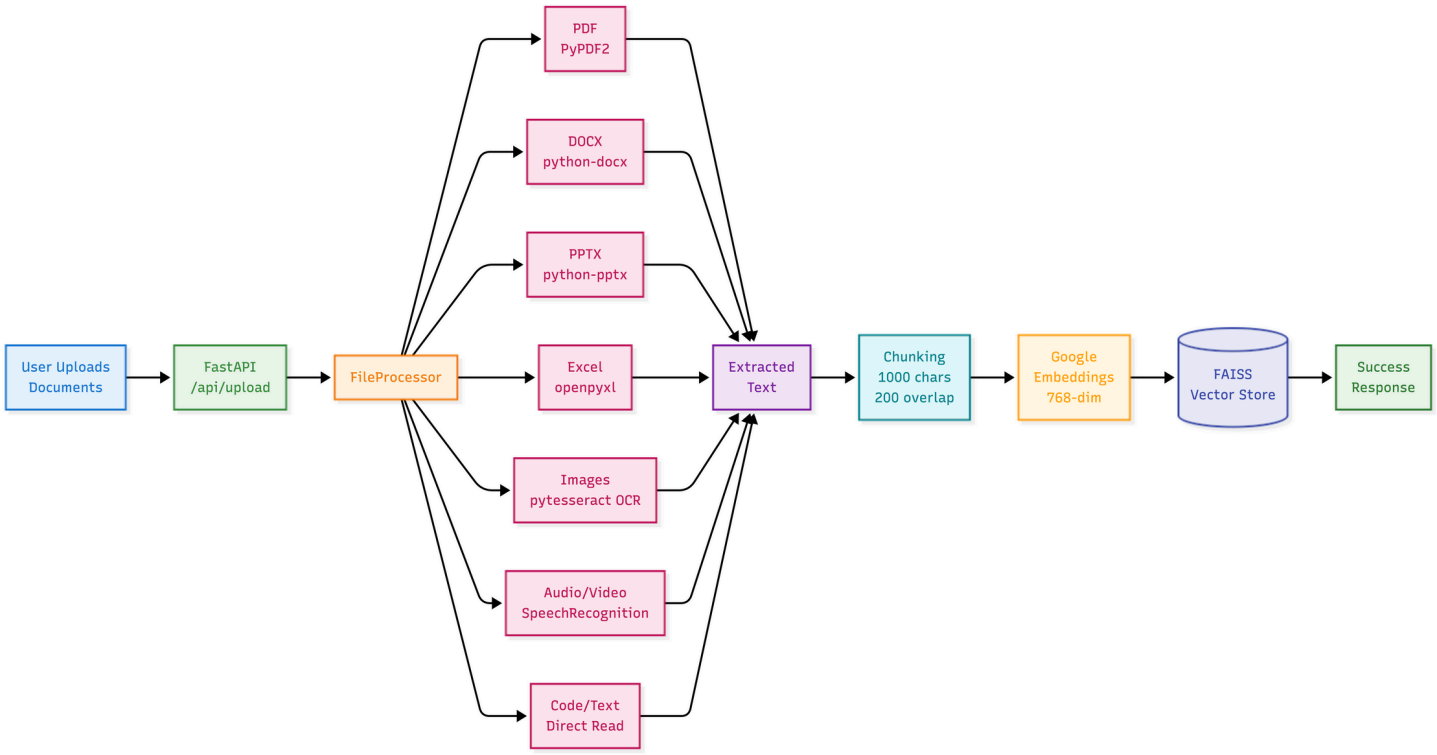
M.A.R.S – Multi-format Agentic RAG System

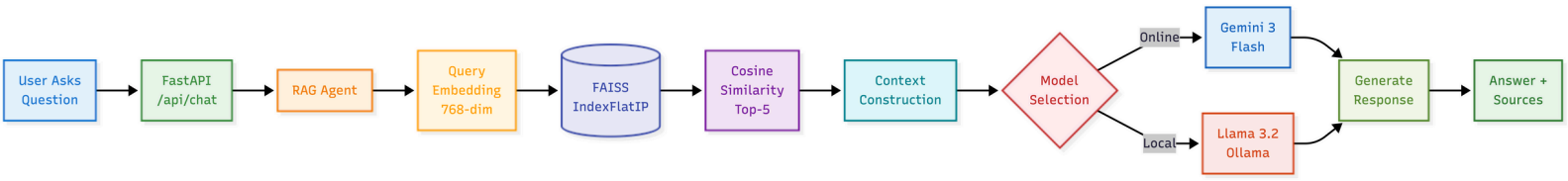
System Architecture



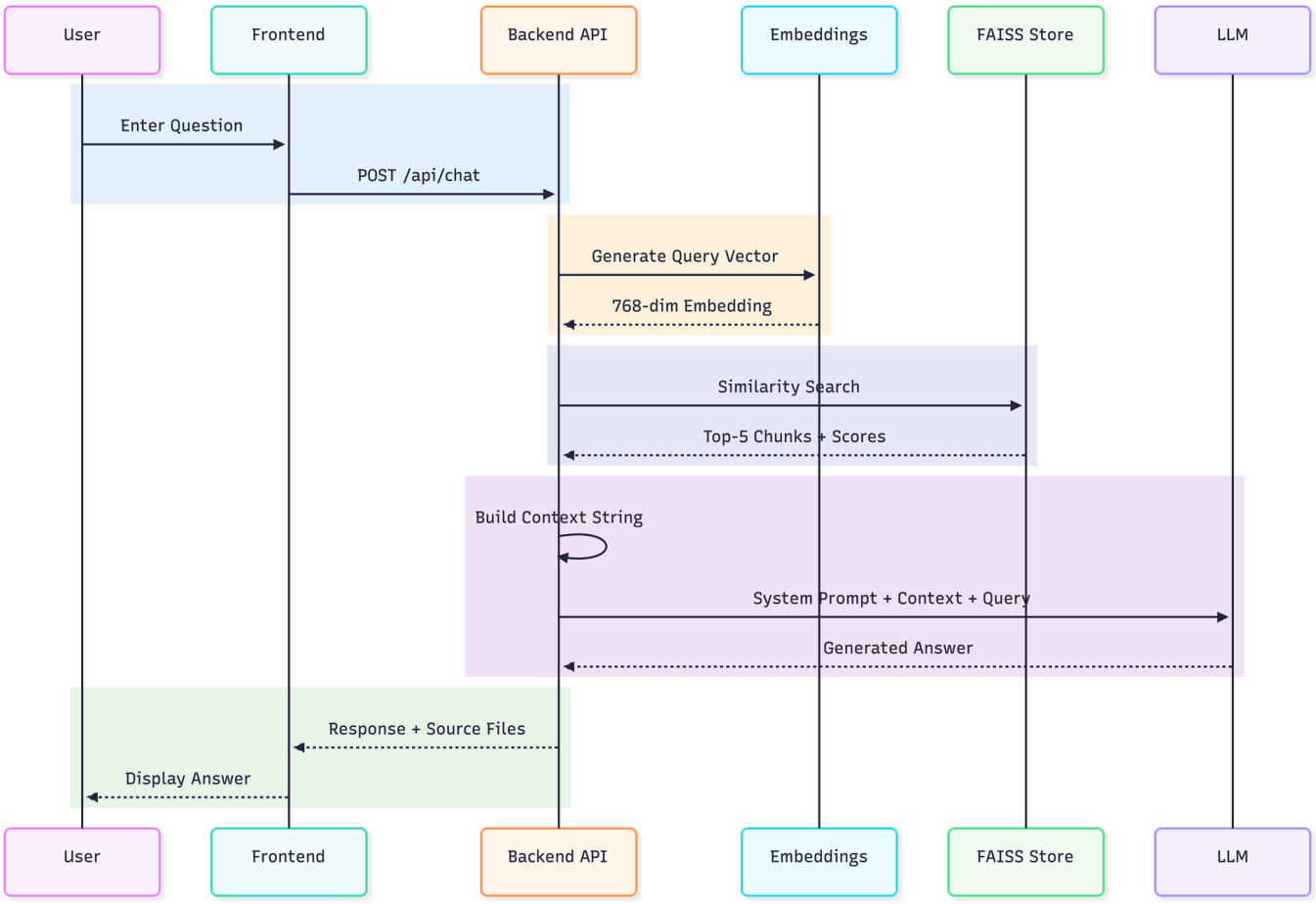
Document Upload Flow



Chat Query Flow (RAG Pipeline)



RAG Sequence Diagram



3. Context Construction Strategy

3.1 Text Chunking

The system employs a sliding window chunking strategy to preserve contextual continuity across document boundaries.

Parameter	Value	Rationale
Chunk Size	1000 characters	Provides sufficient semantic context while staying within LLM token limits
Chunk Overlap	200 characters	Prevents information loss at chunk boundaries (~20% overlap)
Top-K Results	5 documents	Balances context richness with latency and token constraints

3.2 Embedding Strategy

- Model: Google embedding-001
- Dimensions: 768
- Task Type: Retrieval-optimized embeddings for both documents and queries

The same embedding model is used for documents and queries to ensure vector space consistency.

3.3 Context Assembly Process

1. Query Vectorization
   User queries are converted into 768-dimensional embeddings using embedding-001.
2. Similarity Search
   FAISS performs cosine similarity search over stored document vectors.
3. Ranking & Filtering
   The top-5 most relevant chunks are retrieved and filtered by session_id to ensure session isolation.
4. Context Formatting
   Retrieved chunks are assembled into a single context block with source attribution for transparency.

3.4 Prompt Engineering

```
System:
You are an AI assistant. Answer questions based only on the provided
context.
If the answer is not present in the context, clearly state that.
Context:
{retrieved_document_chunks}
User Question:
{query}
This ensures responses are derived only from retrieved documents and
not from the model's general knowledge.
```

The system uses a strictly grounded prompt template:

4. Technology Choices and Rationale

4.1 Backend Framework – FastAPI

- Async Support: Native async/await for non-blocking I/O
- Auto Documentation: OpenAPI schema generation
- Type Safety: Pydantic-based request/response validation
- Performance: ASGI-based, suitable for concurrent file and query processing

4.2 Vector Database – FAISS

- In-Memory Speed: Sub-millisecond similarity search at prototype scale
- Zero Infra Overhead: Embedded within the application process
- Index Type: IndexFlatIP with L2 normalization for cosine similarity
- Scalability Path: Can be replaced with Milvus, Pinecone, or Weaviate in production

4.3 Embedding Model – Google embedding-001

- Free API tier for development and demos
- High-quality semantic embeddings for retrieval
- Standard 768-dimensional vectors compatible with most vector databases

4.4 LLM – Dual Provider Support

Gemini 3 Flash (Cloud)

- Primary model for stable, low-latency responses
- Free tier availability
- Consistent quality

Ollama + Llama 3.2 (Local)

- Self-hosted inference

- Privacy-preserving (no external API calls)
- Offline capability

Automatic fallback is implemented if the local or remote model is unavailable.

4.5 Frontend – React + Vite

- Component-based, modular architecture
- Instant HMR and fast builds via Vite
- Modern tooling with ESM and ESLint

5. Key Design Decisions

5.1 Session-Based Document Isolation

Each user session maintains an isolated document context using a unique session_id.

Benefits:

- Multi-user support without document collision
- Clean conversational state management
- Simple session cleanup and reset

5.2 Unified Text Extraction Pipeline

All supported file types are normalized into plain text via a single extraction pipeline.

Advantages:

- Simplified chunking and embedding logic
- Consistent downstream processing
- Easier maintenance and extensibility

Supported Categories:

- Documents: PDF, DOCX, DOC, PPTX, PPT
- Spreadsheets: XLSX, XLS, CSV
- Code: Python, JavaScript, TypeScript, Java, C, C++, Go, Rust, PHP, Ruby
- Markup: HTML, CSS, JSON, XML, Markdown
- Media: Images (OCR), Audio (Speech-to-Text), Video (Audio extraction + STT)

5.3 Graceful Degradation

The system degrades gracefully when optional dependencies are missing:

- OCR silently skips if Tesseract is unavailable
- Audio processing is bypassed if SpeechRecognition dependencies are missing
- Ollama failures automatically fall back to Gemini

This ensures robustness across diverse environments.

5.4 Source Attribution

Every response includes references to source documents.

Benefits:

- User-verifiable answers
- Increased trust and transparency
- Clear traceability of information

6. Limitations

6.1 Local LLM Performance

The current implementation uses llama3.2:3b due to hardware constraints (M1 MacBook Air, 8GB RAM).

Recommendations for stronger systems:

- llama3.2:7b – Better reasoning and fluency
- llama3:70b – Production-grade quality (GPU required)

6.2 No Authentication

Authentication is not implemented in the current version.

Production recommendations:

- JWT-based authentication
- Persistent session storage (DB-backed)
- Rate limiting and abuse protection

6.3 OCR & Speech Recognition Accuracy

- OCR quality depends on image clarity and layout
- Audio transcription varies with accents, noise, and recording quality
- Complex PDFs (multi-column, scanned) may extract imperfectly

6.4 Large File Handling

Files are processed synchronously in memory.

Risks with large files (100MB+):

- High memory usage
- Long processing times
- Potential request timeouts

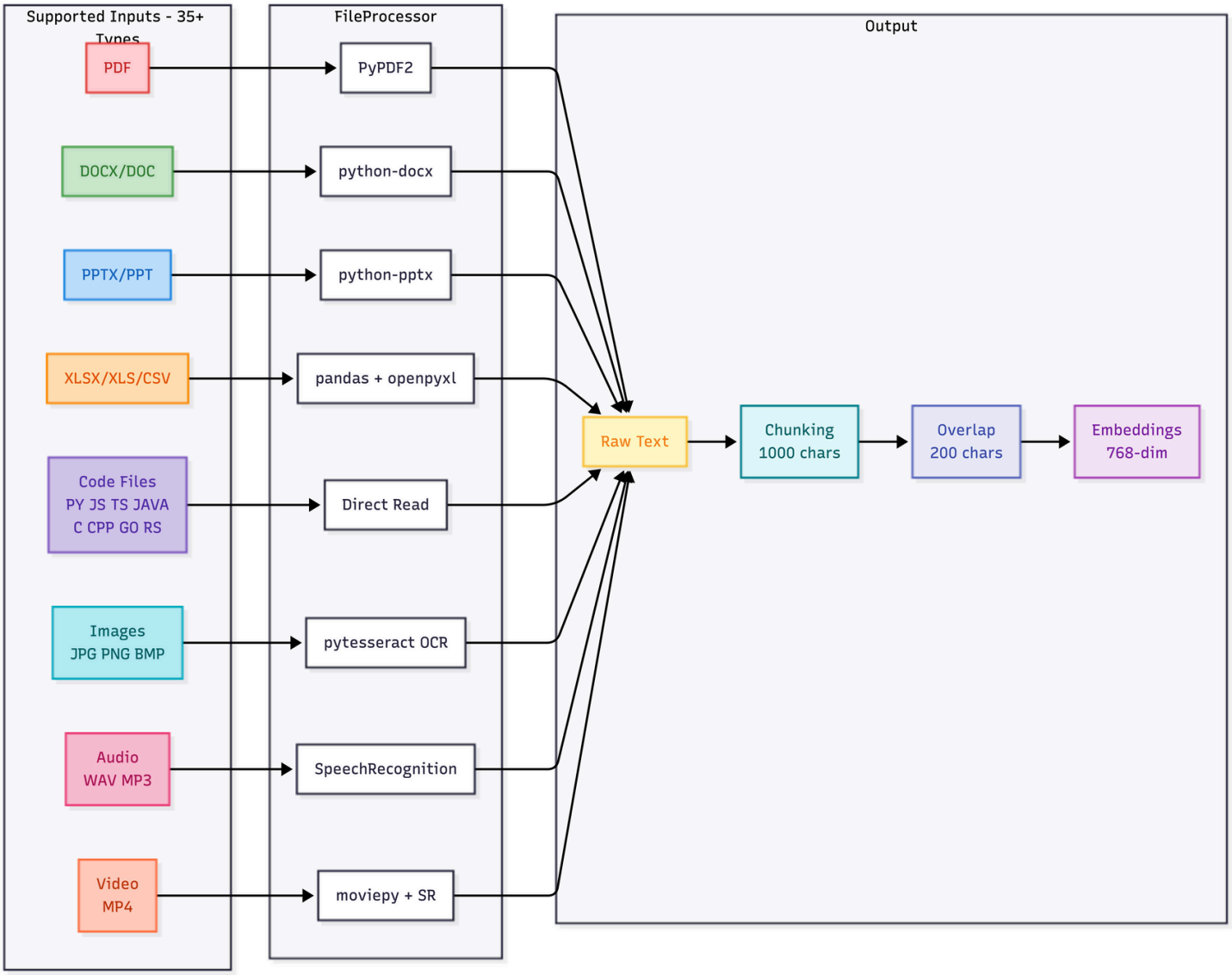
Production solution:

- Chunked file streaming

- Background processing with task queues

7. Data Engineering

7.1 File Processing Pipeline



7.2 Text Cleaning

- Remove excessive whitespace
- Normalize line endings
- Strip non-printable characters
- Preserve semantic structure (paragraphs, lists)

7.3 Chunk Metadata

Each stored chunk includes:

- text – Chunk content
- source – Original filename
- chunk_index – Position within source document
- session_id – User session identifier

This metadata enables filtered retrieval and accurate source attribution.

8. API Design

Method	Endpoint	Purpose
POST	/api/upload	Upload and process documents
POST	/api/chat	Query the system and receive RAG responses
GET	/api/models	List available LLM providers
GET	/api/supported-formats	List accepted file types
DELETE	/api/session/{id}	Clear documents for a session
GET	/health	Service health check

9. Dependencies

Backend (Python)

- fastapi, uvicorn – Web framework
- langchain, langchain-core – RAG orchestration
- google-generativeai – Gemini API and embeddings
- faiss-cpu – Vector similarity search
- pypdf2, python-docx, python-pptx, openpyxl, pandas – Document parsing
- pytesseract, opencv-python – OCR
- speechrecognition, pydub, moviepy – Audio/Video processing

Frontend (Node.js)

- react – UI framework
- vite – Build tool
- axios – HTTP client
- lucide-react – Icon library