Research Report

Query:

Technical Decisions & Trade-offs - Multi-Agent Research System

Generated: 2026-01-31 18:15

Technical Decisions & Trade-offs

Multi-Agent Research System

Documentation of architectural choices, trade-offs, and implementation decisions

---

1. Agent Provider Selection

Decision: OpenAI GPT-4 over Google Gemini

Context: The system was initially designed with three AI providers: Perplexity (search), Claude (analysis), and Gemini (validation/multimodal).

Change: Replaced Gemini with OpenAI GPT-4.

Reasoning:

- Gemini's billing model was complex and unpredictable
- OpenAI offers clearer pricing and usage tracking
- GPT-4 provides comparable multimodal capabilities
- Better API stability and documentation

Trade-offs:

- Gemini: 1M token context window vs OpenAI: 128K token context
- Gemini: Native video analysis vs OpenAI: Frame-based video analysis
- Gemini: Complex billing vs OpenAI: Predictable per-token pricing

Implementation: Created src/agents/openai.py mirroring the Gemini agent interface, updated configuration to use OPENAIAPIKEY, and modified the UI to reflect the new agent.

---

2. Homepage Architecture

Decision: Serve UI at Root Path

Context: Originally, the root path (/) returned JSON API information, while the UI was at /ui.

Change: Root path now serves the graphical research interface; API info moved to /api/info.

Reasoning:

- Users expect a web application at the root URL
- Reduces friction for first-time visitors
- API consumers can still access system info programmatically
- Follows convention of modern web applications

Backward Compatibility: The /ui endpoint remains functional for any existing bookmarks or links.

---

3. Document Export Implementation

Decision: fpdf2 + python-docx for Export

Context: Users requested ability to download research results as PDF and DOCX.

Alternatives Considered:

- WeasyPrint - HTML/CSS to PDF (rejected: heavy system dependencies)
- ReportLab - Low-level PDF generation (rejected: verbose API)
- fpdf2 - Lightweight pure-Python PDF (selected)
- pandoc - Universal converter (rejected: external binary dependency)

Selected Stack:

- PDF: fpdf2 - Pure Python, no system dependencies, simple API
- DOCX: python-docx - Standard library for Word documents

Implementation Details:

- Markdown converted to plain text for PDF (preserves structure)
- Markdown parsed and converted to DOCX styles (headings, lists, etc.)
- Metadata (query, sources, duration) included in both formats
- Files generated on-demand, not stored server-side

---

4. A2A Protocol Implementation

Decision: Custom A2A Protocol Layer

Context: Needed standardized communication between heterogeneous AI agents.

Design Choices:

- JSON-RPC 2.0 as the message format
- Agent Cards for capability discovery

- Skill-based routing for task distribution
- Async/await throughout for non-blocking operations

---

5. Streaming Architecture

Decision: Server-Sent Events (SSE)

Alternatives Considered:

- WebSockets - Bidirectional (rejected: overkill for one-way updates)
- Long Polling - Simple but inefficient (rejected)
- SSE - Native browser support, HTTP-based (selected)

Reasoning:

- Research progress is unidirectional (server to client)
- SSE works over standard HTTP (firewall-friendly)
- Automatic reconnection built into browsers
- Simpler than WebSocket for this use case

---

6. Configuration Management

Decision: Environment Variables + Admin Panel

Architecture:

- API keys stored in .env file (not committed)
- Runtime settings in data/settings.json
- Admin panel for non-sensitive configuration

Security Considerations:

- API keys never exposed to frontend
- Admin panel requires authentication
- Settings file has restricted permissions

---

7. Error Handling & Fallbacks

Decision: Graceful Degradation with Ollama Fallback

Strategy:

- Each agent initializes independently
- Failed agents don't prevent system startup

- Ollama provides local fallback when cloud APIs fail
- Provider priorities configurable per skill

---

8. Frontend Architecture

Decision: Vanilla JavaScript (No Framework)

Reasoning:

- Single-page application with limited complexity
- No build step required
- Faster initial load
- Easier deployment
- Demonstrates core web fundamentals

Libraries Used:

- marked.js - Markdown rendering (CDN)
- Native fetch API for HTTP requests
- Native EventSource for SSE

Summary of Key Decisions

- AI Provider: OpenAI over Gemini (Billing clarity)
- Homepage: UI at root (User experience)
- PDF Export: fpdf2 (No dependencies)
- Protocol: A2A/JSON-RPC (Agent interoperability)
- Streaming: SSE (Simplicity)
- Frontend: Vanilla JS (No build step)
- Fallback: Ollama (Offline capability)

Document Version: 1.0 | Last Updated: January 2026