PV Monitoring Platform

MVP delivery for Ingestion, Normalization, and AI-First UX

Presentation by Thösam Norlha-Tsang

Introduction

Scalable PV Monitoring & AI Analytics

Scope: 10–20 PV dataloggers (different vendors, structures, quirks)

Task - design and implement:

1. Ingestion Layer
2. Abstraction & Cleanup Layer
3. Monitoring View + AI Interaction
4. Deployment

0. Road Map

Roadmap

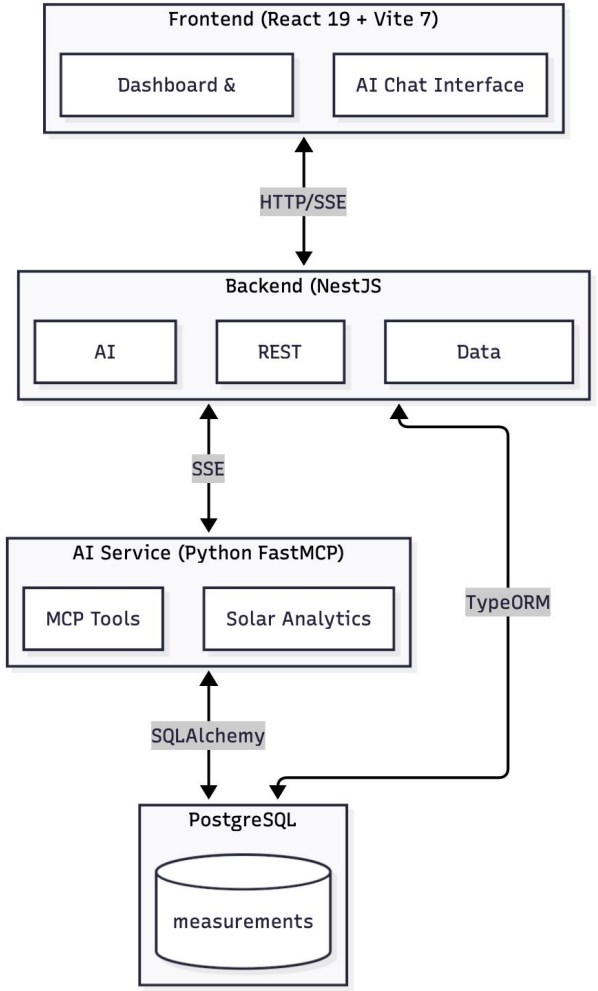
1. High-Level Architecture
2. The Data Flow
3. Handling Messy Data (Ingestion)
4. The AI Agent (LangGraph)
5. AI-First User Experience
6. Demo
7. Learnings
8. Q&A

1. High-Level Architecture

High-Level Architecture

Key Points:

- **Three-Tier Design**: Clear separation of concerns
- **Backend (NestJS)**: Type safety, modularity, robust API
- **AI Service (Python/FastMCP)**: Dedicated environment for data science tools (Pandas/NumPy)
- **Frontend (React + Vite):** Fast, reactive UX with shadcn/ui

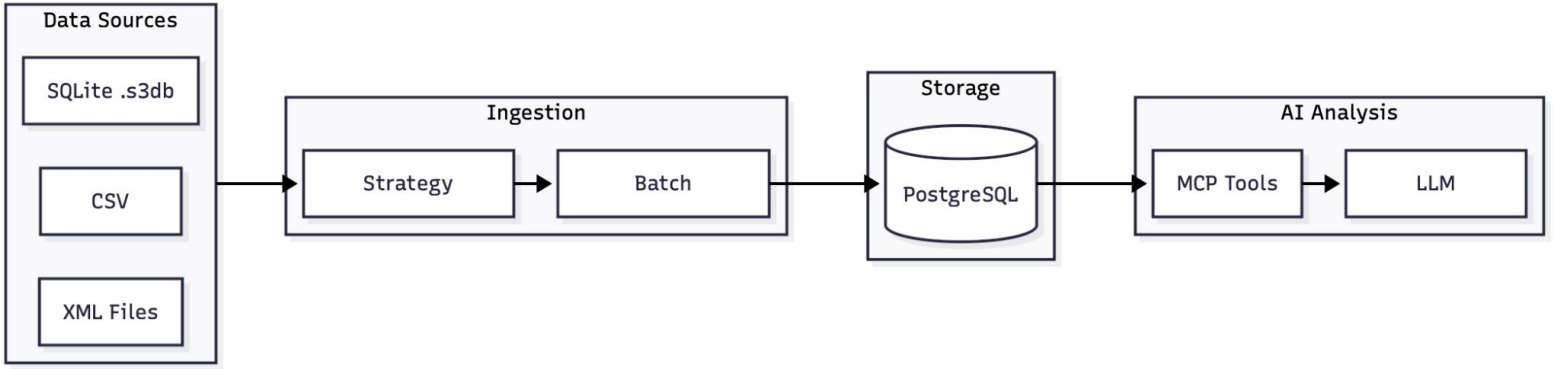


2. The Data Flow

The Data Flow

Key Points:

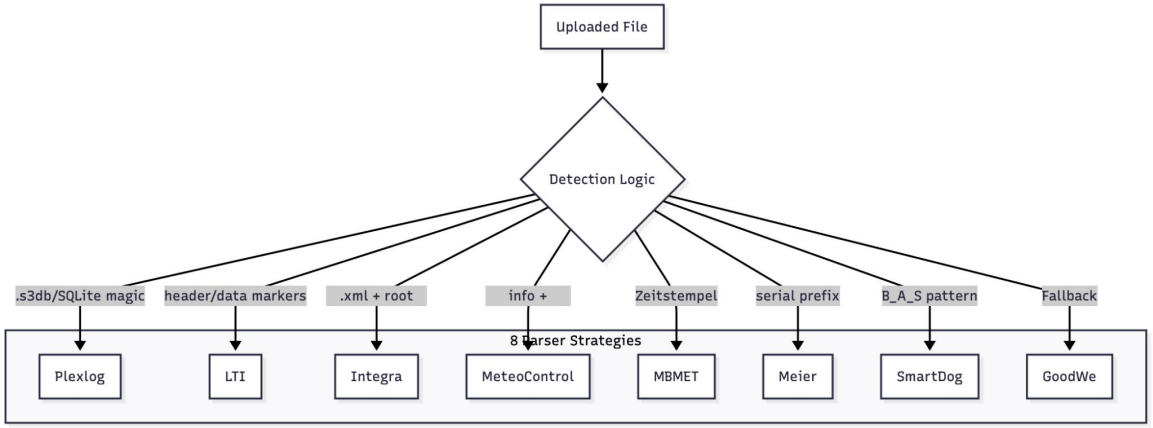
- Files ➔ Ingestion Pipeline ➔ Normalized Storage (PostgreSQL)
- User ➔ Chat Interface ➔ LangGraph Orchestrator ➔ Tools ➔ Database
- **Design Decision:** Real-time feedback via SSE (Server-Sent Events)



3. Handling Messy Data (Ingestion)

The Parsing Strategy

- **The Challenge:** 8+ formats (XML, SQLite, CSV, Text), inconsistent headers.
- **The Solution:** Specificity-First Strategy Pattern
  - Detects file type by content (magic bytes), not just extension
  - Easily extensible (Open-Closed Principle)



Data Abstraction & Storage

- **Design Decision: Hybrid Schema.**
  - **Columns:** "Golden Metrics" (Power, Energy) for fast SQL aggregation
  - **JSONB:** Metadata column for vendor-specific quirks
- **MVP Implementation:** Synchronous processing with `AsyncGenerator` (memory efficient)
- **Future Improvement:** Message Queue (RabbitMQ/BullMQ) for asynchronous bulk processing

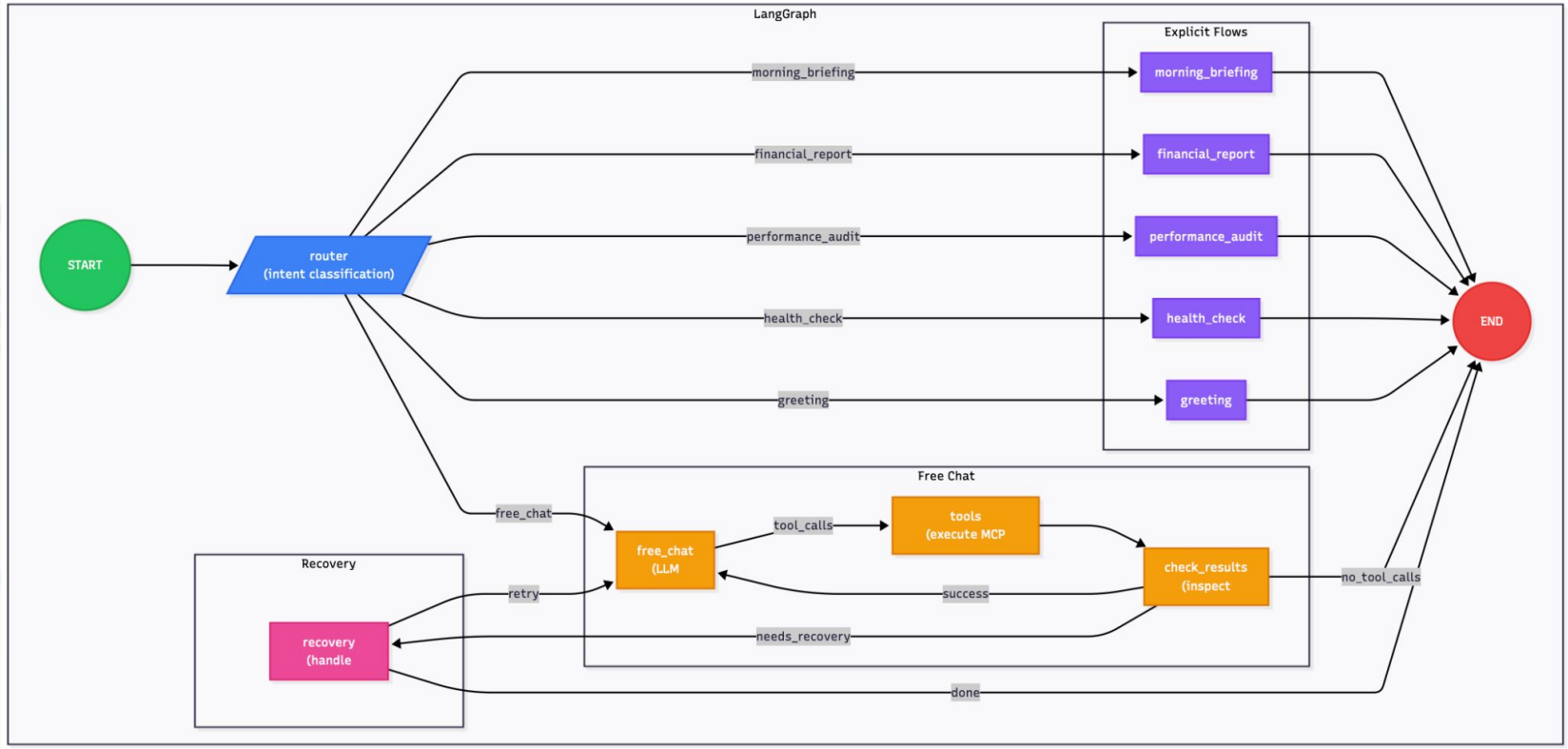
MEASUREMENTS		
string	loggerId	PK
timestamp	timestamp	PK
varchar	loggerType	
float	activePowerWatts	
float	energyDailyKwh	
float	irradiance	
jsonb	metadata	
timestamptz	createdAt	

4. The AI Agent (LangGraph)

Orchestration vs. Chatbot

- **The Concept:** Deterministic Graphs over probabilistic chaos
- **Structure:**
  - **Explicit Flows:** Morning Briefing, Financial Report (Strict steps)
  - **Free Chat:** LLM Loop (Flexible fallback)

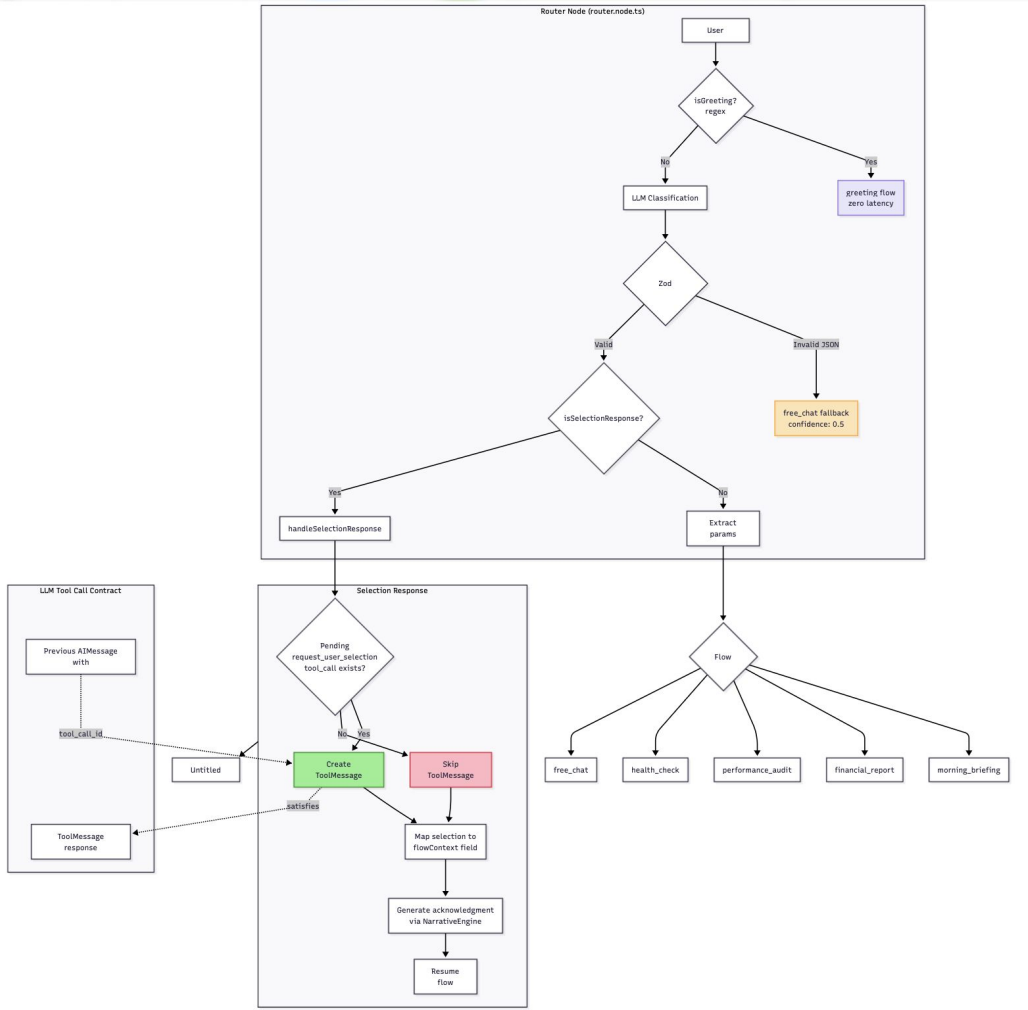
Orchestration vs. Chatbot



Intelligent Routing

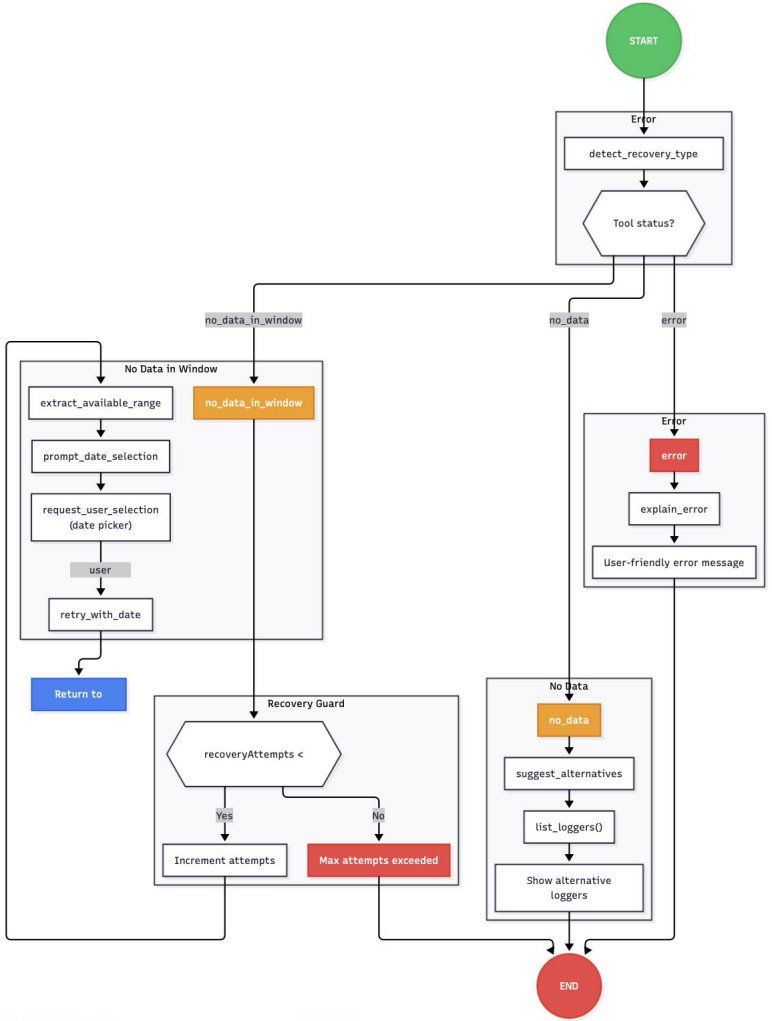
- **Optimization:** Two-Tier Classification
  1. **Regex (Tier 1):** Zero latency for greetings/patterns
  2. **LLM (Tier 2):** Semantic understanding for complex intents
- **Benefit**: Reduces cost and latency; increases reliability

Intelligent Routing



Resilience & Recovery

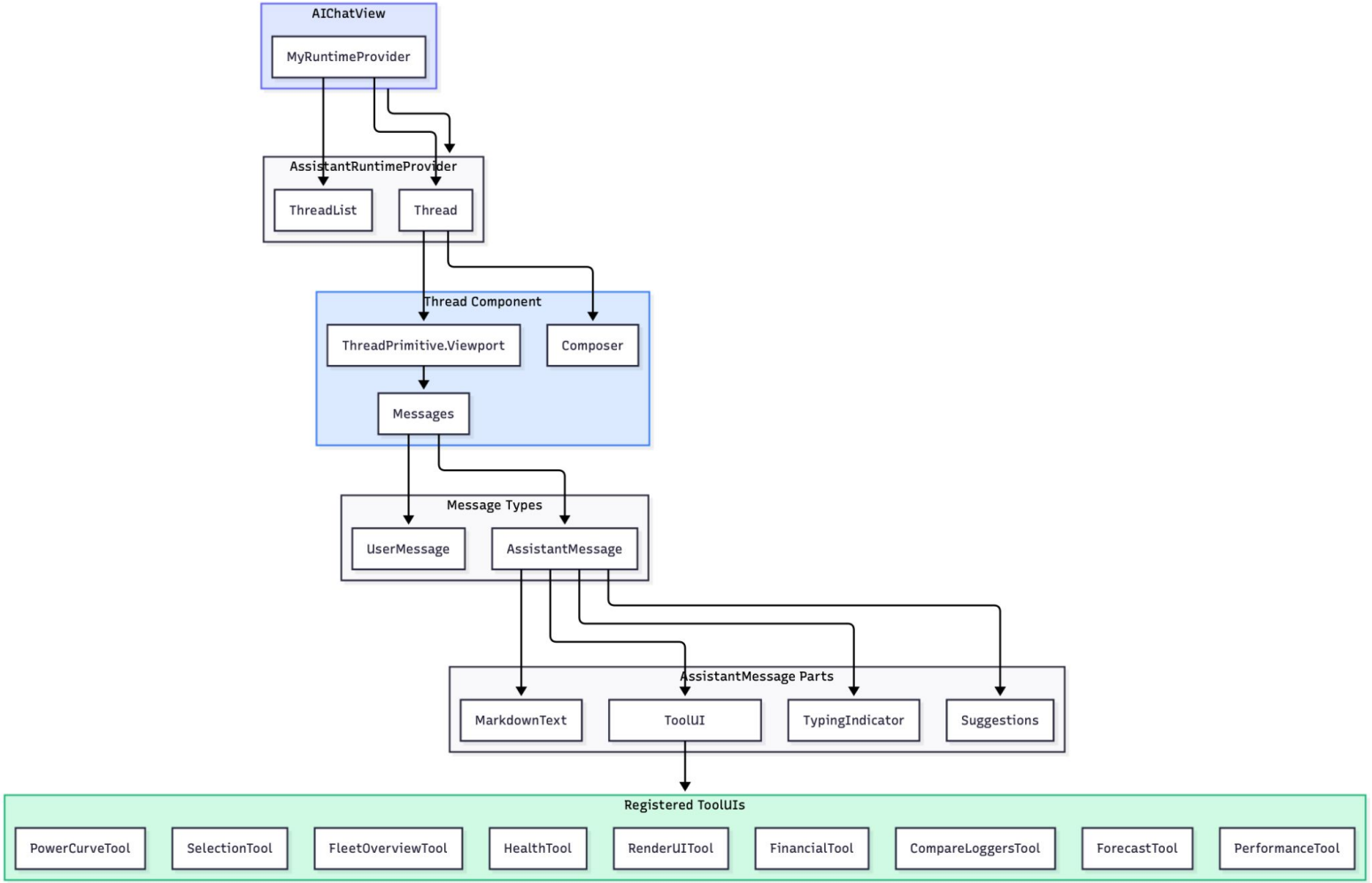
- **The Problem: LLMs hallucinate when data is missing**
- **The Solution:** Standardized Tool Status Codes (`no_data_in_window`, `error`)
  - Agent triggers **Interactive Recovery** (e.g., rendering a Date Picker)
  - Prevents "I don't know" dead-ends



5. AI-First User Experience

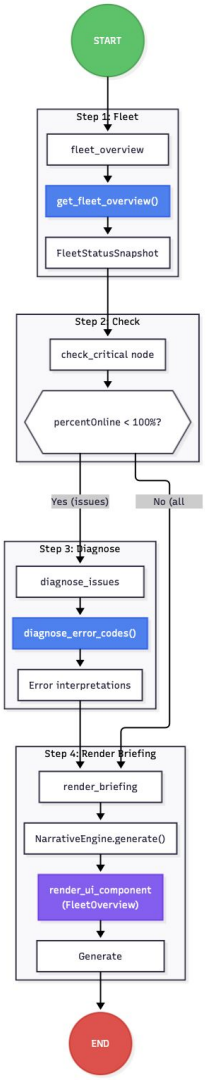
The "Glass Box" Approach

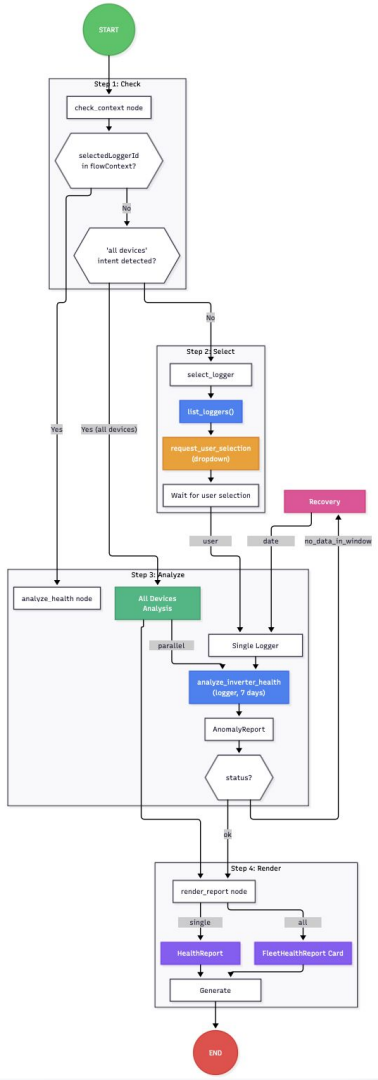
- **Philosophy:** Don't hide the work. Show the user *what* the AI is doing.
- **Tool Categories:**
  - **Hidden:** Heavy math/data fetching (shown as status indicators)
  - **Visible**: Charts, Selection Prompts, Reports (Rendered Inline)


AIChatView
MyRuntimeProvider
AssistantRuntimeProvider
ThreadList
Thread
Thread Component
ThreadPrimitive.Viewport
Composer
Messages
Message Types
UserMessage
AssistantMessage
AssistantMessage Parts
MarkdownText
ToolUI
TypingIndicator
Suggestions
Registered ToolUIs
PowerCurveTool
SelectionTool
FleetOverviewTool
HealthTool
RenderUITool
FinancialTool
CompareLoggersTool
ForecastTool
PerformanceTool

Explicit Flows UX

1. Morning Briefing
2. Health Check

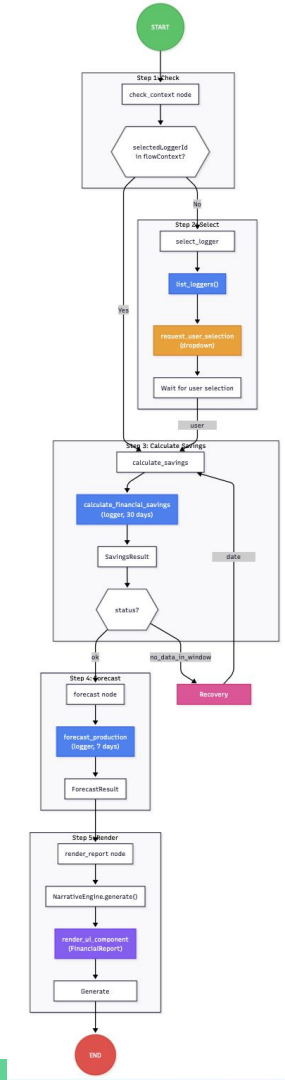


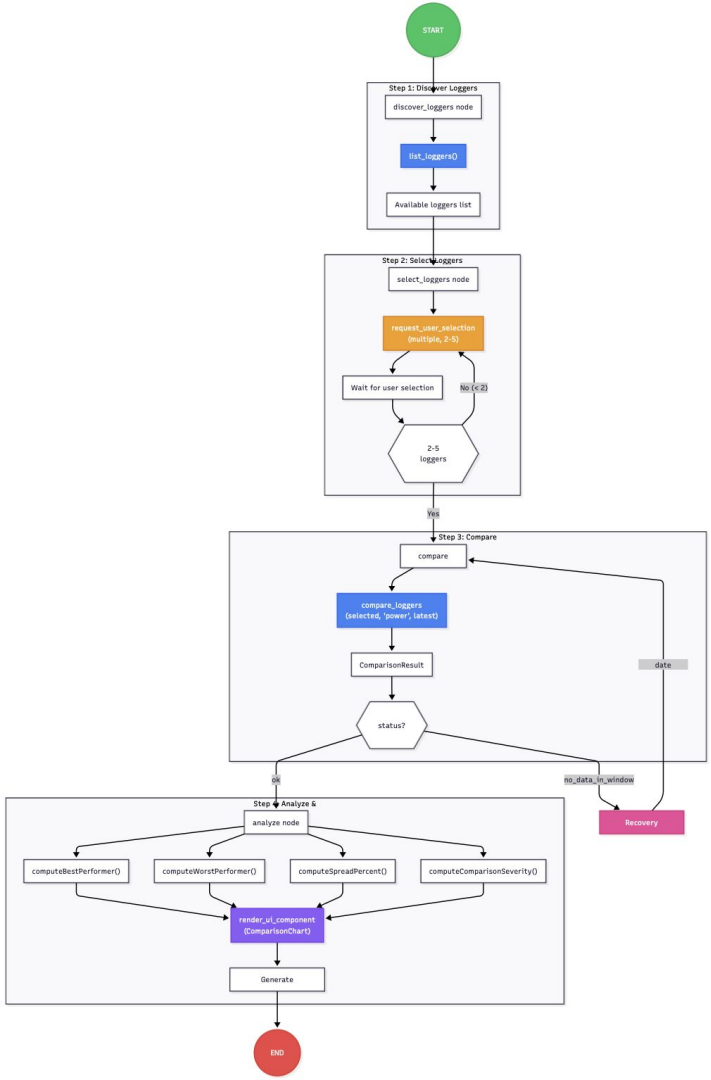


Explicit Flows UX

3. Financial Report

4. Performance Audit





6. Demo

Explicit Flows UX

Demo Checklist:

1. Ingest "messy" folder
2. Run "Morning Briefing" (See the graph action)
3. Trigger Error Recovery (Date Selection)
4. 4. Perform "Performance Audit" (Multi-logger comparison)

7. Learnings

Learnings: Technologies & Workflow

- **Acceleration:** Used Claude Code & MCP for rapid API integration
- **Quality:** SonarQube for code quality checks
- **Flexibility:** System supports swapping providers (Gemini, OpenAI, Ollama/Local) via simple ENV vars

8. Q&A

Thank you !

(and Q&A)