# 🚀 AI Agent Orchestrator for SEO & Analytics

**A Multi-Agent System that fuses Google Analytics 4 (GA4) traffic data with Technical SEO intelligence.**

This project is a high-performance AI backend built with **FastAPI**, **Pandas**, and **LLMs (LiteLLM)**. It features an intelligent Orchestrator that routes user queries to specialized agents, executes complex data fusion (SQL-style joins) between live APIs and static datasets, and handles large-scale data analysis with auto-profiling.

## 📂 Project Structure

```
spike-ai-builder/
├── agents/
│   ├── analytics_agent.py   # GA4 API Handler (Tier 1 Logic & Smart Retry)
│   ├── seo_agent.py         # Google Sheets Handler (Tier 2 Logic & Profiling)
│   └── __init__.py
├── utils/
│   ├── llm_client.py        # LiteLLM Wrapper with Exponential Backoff
│   ├── prompts.py           # System Prompts, Routing Rules & Guardrails
│   └── __init__.py
├── main.py                  # FastAPI Entry Point
├── orchestrator.py          # The "Brain" (Intent Routing & Multi-Agent Fusion)
├── deploy.sh                # Production Deployment Script (Linux)
├── deploy_windows.ps1       # Local Development Script (Windows)
├── seed_ga4_data.py         # GA4 Backfill Script ("Resourcefulness" Challenge)
├── test.py                  # Automated Test Suite (Tiers 1, 2, 3)
├── requirements.txt         # Project Dependencies
├── .env                     # API Keys (Not tracked in git)
├── .gitignore               # Ignorelist
├── credentials.json         # Google Service Account Key (Not tracked in git)
└── README.md                # Documentation
```

## 🏆 Hackathon Deliverables Achieved

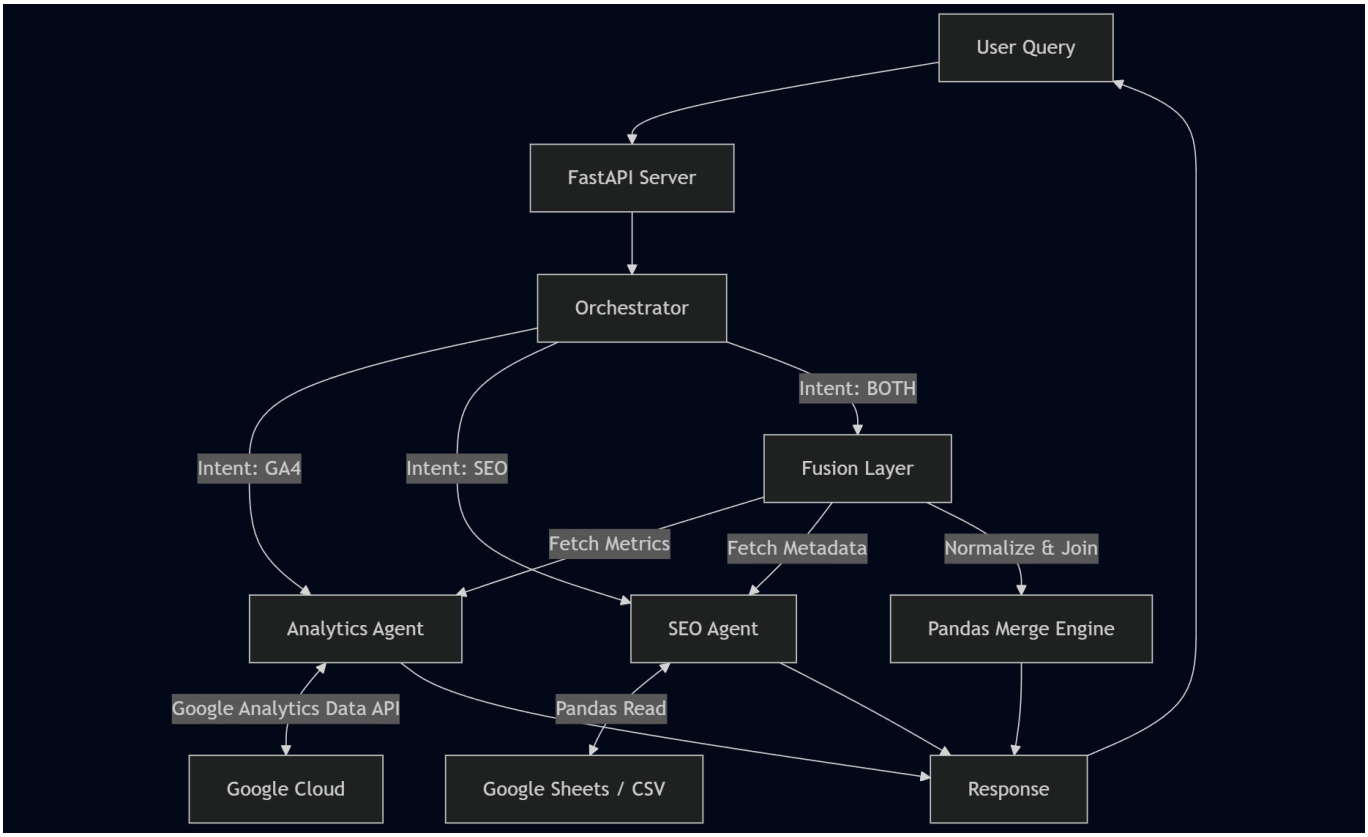
| Feature | Status | Description |
|---|---|---|
| **Tier 1: Intelligent Routing** | ☑ | Automatically detects if a query needs GA4, SEO, or BOTH. Routes to the correct Agent and specific Sheet Tab. |
| **Tier 2: Deep Analysis** | ☑ | Performs advanced Pandas filtering (`Indexability`, `Status Code`) and auto-profiles data (Counts & Breakdowns) before summarizing. |
| **Tier 3: Multi-Agent Fusion** | ☑ | **The "Flagship" Feature.** Fetches live traffic from GA4, joins it with Technical SEO data (Titles, Meta) by normalizing URLs, and delivers a unified insight. |

| Feature | Status | Description |
| --- | --- | --- |
| **Resourcefulness** | ☑ | Includes `seed_ga4_data.py` to backfill data into GA4 via Measurement Protocol (bypassing the need for a live website). |

## Architecture

The system follows a **Hub-and-Spoke** architecture:

1. **The Orchestrator:** The "Brain" that parses intent and manages the workflow.
2. **Analytics Agent (GA4):** Connects to Google Analytics Data API. Includes "Smart Retry" logic for invalid metrics.
3. **SEO Agent (Sheets):** Connects to Google Sheets (Screaming Frog exports). Includes "Fuzzy Column Matching" and "Auto-Type Conversion".
4. **Fusion Layer:** A Pandas-based logic block that normalizes URLs (stripping protocols/trailing slashes) to merge dataset A and B.




# Assumptions & Limitations

## 1. Google Analytics 4 (GA4)

- **Data Latency:** The GA4 Standard API (`runReport`) has a processing latency of 24-48 hours. Real-time data is not instantly available via this API.
    - *Mitigation:* For the demo, we use a "Mock Mode" or historical data to demonstrate functionality without waiting for processing.

- **Metric Compatibility:** We assume standard metrics (`activeUsers`, `screenPageViews`) are available. Custom metrics defined in the UI but not the API will cause errors.
    - *Mitigation:* The Agent includes "Smart Retry" logic to fallback to standard metrics if a query fails.

## 2. SEO Data (Screaming Frog / Sheets)

- **Static Export:** The system relies on a pre-exported CSV/Google Sheet. It does not crawl the website in real-time.
- **Column Naming:** We assume standard Screaming Frog column headers (e.g., "Address", "Title 1", "Status Code").
    - *Mitigation:* The `SEOAgent` uses fuzzy matching to detect URL columns even if headers vary slightly.
- **File Size:** The system loads the spreadsheet into memory.
    - *Mitigation:* We enforce a 500-row limit for the Hackathon demo to ensure performance on standard VMs. Production would require database chunking.

## 3. Data Fusion (Tier 3)

- **URL Matching:** We assume the GA4 `pagePath` (e.g., `/blog`) corresponds to the SEO `Address` (e.g., `https://site.com/blog`).
    - *Mitigation:* A robust normalization engine strips protocols (`https://`), domains, and trailing slashes to maximize match rates.

---

# 🛠️ Setup & Installation

### 1. Prerequisites

- Python 3.10+
- A Google Cloud Service Account (`credentials.json`) with access to GA4 and Sheets.
- An LLM Provider Key (LiteLLM/OpenAI/Gemini).

### 2. Installation

Clone the repository and install dependencies:

```
# Windows
python -m venv .venv
.\.venv\Scripts\Activate
pip install -r requirements.txt

# Linux / Mac
python3 -m venv .venv
source .venv/bin/activate
pip install -r requirements.txt
```

### 3. Configuration

Create a `.env` file in the root directory:

```
LITELLM_API_KEY=sk-your-key-here
SEO_SPREADSHEET_ID=your-sheet-id-here
```

Ensure `credentials.json` is placed in the **root** folder.

---

## 🚀 How to Run

### Option A: Production / Evaluator (Linux)

Use the provided deployment script which handles environment creation and `pip` issues automatically:

```
bash deploy.sh
```

*Server will start on port 8080 in background mode.*

### Option B: Local Development (Windows)

```
.\deploy_windows.ps1
```

*Or manually:* `uvicorn main:app --host 0.0.0.0 --port 8080 --reload`

---

## 🧪 Testing the Agents

You can use the included `test.py` script to verify all 3 Tiers.

```
python test.py
```

### Sample Queries Supported

#### 1. Analytics (Tier 1)

> *"How many active users did we have in the last 7 days?"*

#### 2. Technical SEO (Tier 2)

> *"Show me all pages with 404 errors."* *"Group pages by Indexability and give me a count."* (Uses Auto-Profiling)

**3. Fusion (Tier 3 - High Value)**

> *"What are the top 5 pages by views and what are their title tags?" (This triggers the Multi-Agent Merge logic)*

---

## "Resourcefulness" Challenge

**Goal:** Ingest data into GA4 without a live website.

I solved this by building `seed_ga4_data.py`.

- It uses the **GA4 Measurement Protocol API**.
- It generates synthetic traffic events.
- It sends them directly to Google's servers via HTTP POST, bypassing the browser.

**To run the backfill:**

```
python seed_ga4_data.py
```

*(Note: Data appears in "Realtime" dashboard instantly, but standard API takes 24-48h to process).*

---

## Robustness & Optimizations

- **Exponential Backoff:** The LLM Client handles rate limits (429 errors) by waiting 2s, 4s, 8s...
- **Smart Truncation:** Large text fields (like HTML content) are truncated to 100 chars to prevent Token Limit Exceeded errors.
- **URL Normalization:** The Fusion engine strips `https://`, `www.`, and trailing slashes (`/`) to ensure `site.com/blog` matches `/blog/`.
- **Safe Defaults:** If the LLM requests an invalid metric (e.g. `bounce_rate`), the Analytics Agent catches the 400 error and retries with standard metrics automatically.

---

## Authors


- **Chirag Gupta**
- **Kumar Ayush Aman**