## **AI-Driven Agentic Architecture: MCP + LLM + Multi-Agent System**

### **Vision**

We are building an **AI-first application** that unlocks **business value** by blending **cutting-edge LLM technology with agentic workflows** and robust backend systems. The aim is to:

* Embed AI **deeply into the product**
* Enable **natural language interfaces**
* Automate **complex, multi-step workflows**

### **System Overview**

**Natural Language Processing (LLM 3.5 and beyond)**:  
 While LLMs are excellent at interpreting **natural language**, they **do not inherently generate structured data**. To bridge this gap, we:

* Use **function-calling mechanisms** within the LLM
* Trigger backend functions to:  
  + **Structure the data**
  + **Execute business logic**
  + **Return actionable results**

### **Multi-Step Workflows**

Our system isn't "n-shot" (one-shot) capable for complex tasks. It executes **multi-step workflows** where each step contains:

* **Business Logic (Code)**
* **LLM Inference**

**Innovation**:  
 We delegate **workflow decision-making** to the **LLM**, allowing it to **determine the flow** dynamically and invoke the appropriate **tools and agents**.

### **Key Tools**

* **MCP (Multi-Channel Platform) Tool**:  
  + Bridges the gap between the **LLM and external systems**
  + Accesses multiple data sources, including **Postgres**
* **FastAPI + Pydantic AI**:  
  + High-performance API layer (FastAPI)
  + Structured data validation and parsing (Pydantic AI)

### **Prompting Framework**

* **System Prompt**:  
   Describes **what the system is designed to accomplish.**
* **User Prompt**:  
   Captures the **end-user's query**.

### **Why Multi-Agents?**

One LLM call isn't enough for complex queries—especially when multiple databases and data sources are involved. Hence, we build **agents** that:

* Handle **multi-turn conversations and iterative querying**
* Aggregate results from **different LLMs and databases**

**Example**:

* The **Decision Agent** determines the next action.
* Specialized agents access specific datasets via the **MCP server**.

## **MCP Architecture**

* **MCP Client**:  
   Invoked by the agent; connects to the **MCP server**.
* **MCP Server**:  
  + Interfaces with **Postgres** and other data sources
  + Supports **specialized servers** like the **Ledger MCP Server**
* **Agents**:  
  + LLM-driven
  + Decide flow → Call LLM → Call MCP client → Fetch & process data

### **Architectural Flow**

****Agents (Prompt-driven)

|

v

LLM (Decision-making)

|

v

MCP Server (Specialized Business Logic)

|

v

Postgres (Raw Data)

### **Difference: RAG vs. MCP + Agents**

| **Feature** | **RAG (Retrieval-Augmented Generation)** | **MCP + Multi-Agents** |
| --- | --- | --- |
| Data Access | Vectorized, subset search | Full data access (Postgres + other sources) |
| Retrieval Scope | Limited to **top-N results** | Deep, **specialized querying and processing** |
| Workflow | One-shot prompting | **Multi-step, iterative workflows** |
| Use of Agents | Typically none | **Yes (multi-agent architecture)** |
| Specialization | Generic search + ranking | **Business-specific logic and complex workflows** |
| Extensibility | Limited to dataset used | Connects **multiple systems and databases** |

### **Key Insights**

* **LLM Prompting Alone**:  
   Suitable for **simple, generalized tasks** but **limited** for deep business logic.
* **RAG Systems**:  
   Tailored to **specific datasets**, useful for **search and retrieval**, but:  
  + Must **partition and vectorize data**
  + Limited to **search/rank top N results**
  + Still **one-shot prompting**
* **MCP + Multi-Agent System**:  
   A **step ahead of RAG**—combines:  
  + **Specialist knowledge** of the database (beyond just search)
  + Support for **multiple data sources**
  + **Iterative, multi-agent querying**
* → Unlocks **advanced business logic execution**, and **dynamic, adaptive workflows.**

## **How to Build the MCP Server**

1. **Set up the Postgres database**.
2. **Build the MCP server** as a **FastAPI-based backend**:  
   * Implement business-specific endpoints
   * Design endpoints for **generic and specialized queries**
   * Use **Pydantic models** for data integrity
3. **Extend MCP server** into domain-specific servers (e.g., Ledger MCP Server).

## **How to Build LLMs on Top of the MCP Server**

1. **Design the LLM Prompt**:  
   * **System Prompt** defines what the LLM should achieve.
   * **User Prompt** captures the query.
2. **Implement Function Calling**:  
   * Enable LLMs to call backend functions dynamically.
3. **Wrap LLMs in Agents**:  
   * Use a **Decision Agent** to determine steps.
   * Build **specialized agents** for each data domain.
4. **Ensure Multi-Turn Capability**:  
   * Agents should handle **multi-turn querying** to fetch & refine results across multiple attempts.

## **Next Steps**

* Define **Agent APIs**.
* Implement **LLM prompts and workflows**.
* Build **MCP endpoints** for business logic.
* Set up **multi-agent orchestration**.

Let me know if you'd like **diagrams**, **sample code blocks**, or a **flowchart** to illustrate any of these points further! 🚀