**AutoSynth: Intelligent Synthetic Data Generator – Agent Workflow Design**

**💡 Core Workflow and LLM Integration Strategy**

This document outlines the interactive design and modular agent responsibilities for AutoSynth, with a strong focus on how LLMs are embedded into the synthetic data generation pipeline.

**🎨 Input Phase: Use Case Declaration**

**User Input**: A clear use-case statement is provided *before* any schema or data generation begins.

* Example: "Generate synthetic hospital admission records for age analysis and seasonal trend study."

This input informs downstream agents (Schema, Generator, Validator) and guides LLM-based suggestions.

**🔹 Schema Agent**

**🔍 Inputs:**

* **Prompt** (e.g., domain description)
* **CSV file** (optional, mutually exclusive with prompt)

**🔠 Flow:**

1. If **Prompt**:
   * LLM generates an initial JSON schema.
   * System displays schema and asks:
     + "Are you satisfied?"
     + Options: Add column, Remove column, Modify column.
   * This loop continues until user clicks **"I'm Satisfied"**.
2. If **CSV**:
   * Parse columns and types via Pandas.
   * Convert to structured schema JSON.
   * Display for confirmation.

**🧼 Outputs:**

* Finalized Schema JSON object.

**🔧 Tech Stack:**

* Pandas, Pydantic, Pandera
* LLM (T5 / GPT) for prompt-to-schema

**🧱 Generator Agent**

**🔍 Inputs:**

* Schema JSON (from Schema Agent)
* Use-case description (global)

**✅ Parameters Asked:**

* Number of rows
* Generator model (CTGAN, TVAE, etc.)
* Seed data (optional upload)
* Conditional columns and values
* Random seed

🤖 For **each parameter**, user can:

* Enter manually
* Select "Let LLM choose" (auto-fill using use-case and schema context)

**🧼 Outputs:**

* Synthetic dataset (DataFrame or CSV)
* Generation metadata

**🔧 Tech Stack:**

* SDV (CTGAN, TVAE, CopulaGAN)
* PyTorch, Pandas
* LLM for auto-config suggestion

**🔢 Validator Agent**

**🔍 Inputs:**

* Synthetic dataset
* Schema JSON
* Constraints in **natural language**, Python, or SQL-style

**✅ Flow:**

1. Ask user to enter validation rules in *any format*.
2. LLM parses and converts rules into:
   * Pandera validation expressions
   * or Python functions
3. Validator applies all constraints
4. Displays a **validation report**:
   * Rows passed
   * Rows failed (with reasons)

User prompted: "Should I delete the failed rows?"

* Yes / No / Export failed rows separately

**🧼 Outputs:**

* Cleaned dataset
* Validation summary (pass/fail, deleted rows, constraints used)

**🔧 Tech Stack:**

* Pandera, Cerberus, custom validation engine
* LLM for rule parsing (natural language to logic)

**🔹 Summary of LLM Roles**

| **Agent** | **LLM Role** |
| --- | --- |
| Schema Agent | Prompt → JSON Schema, interactive schema editing |
| Generator | Auto-select best generator model, fill generation parameters |
| Validator | Natural Language → Validation Rules, resolve Python/SQL/NL constraints |

This design lays the foundation for a highly interactive, LLM-enhanced synthetic data generation system capable of handling custom user logic, domain awareness, and validation with minimal manual effort.

Next Step: Implement UI logic flow and backend functions for each phase.