Problem Statement: Design an AI Agent for Automating Data Engineering Tasks

Objective

Design an AI agent that automates data engineering tasks by:

- Interpreting product backlog items.
- Generating ETL (Extract, Transform, Load) pipelines.
- Applying transformations.
- Delivering analytical outputs, such as scoring via a model.

The agent must ensure:

- Correct interpretation of backlog items.
- Optimal code generation (e.g., handling partitions).
- Data quality checks and error handling.
- Minimal manual intervention in the pipeline.

Key Requirements

1. **Backlog Interpretation**
    - Parse natural language backlog items to extract requirements, including source tables, transformations, and business rules.
    - Utilize **LangChain** and **AutoGen** for NLP task breakdown.
2. **Code Generation**
    - Auto-generate **Spark SQL** or **PySpark** code for ETL processes.
    - Suggest appropriate joins, filters, and aggregations.
    - Create **dbt models** or **Airflow DAGs** where applicable.
3. **Data Quality & Validation**
    - Integrate **Great Expectations** for automated data quality checks.
    - Flag anomalies and incorporate human-in-the-loop approval for validation.
4. **Model Integration**
    - Apply an **AutoML-generated scoring model** (e.g., for scoring applicants on loan repayment likelihood, ranging from 0 to 100).
    - Output results with metadata, including confidence scores and validation logs.
5. **Deployment & Version Control**
    - Automatically push generated code to **Git** or orchestrate via **Airflow**.
    - Include human approval hooks before deploying to production.

Possible Technology Stack

- **NLP & Agents**: GPT-4/Claude, LangChain, AutoGen.
- **ETL**: PySpark, dbt, Airflow.
- **Validation**: Great Expectations.
- **AutoML**: H2O, PyCaret, or custom MLflow pipelines.
- **RLHF**: Fine-tuning based on human feedback.

Submission Requirements

- **GitHub Repository**: Include agent code, sample backlog items, and demo outputs.
- **Short Report**: Explain design choices and limitations.

Hints

- **Modularity**: Break tasks into sub-agents (e.g., SQL generator, validator) for better manageability.
- **Fallbacks**: Use rule-based fallbacks to handle cost or LLM limitations.
- **Dataset Reference**: The implementation should consider a dataset for scoring applicants on their likelihood of repaying a loan, with scores ranging from 0 to 100.