Project-X: Agent Development Guidance

July 21, 2025

This document provides detailed guidance for the Project-X team to develop the eight agents constituting the MVP of the agent-based architecture. Each agent is designed to operate within a LangGraph environment, leveraging pre-trained Ollama models. The descriptions include the agent's role, goal, inputs, outputs, and recommendations for configuring Ollama parameters and crafting effective prompts. A general agent structure is also provided to ensure seamless integration into the LangGraph pipeline.

1 Agent Architecture Overview

Each agent is a modular component in a LangGraph workflow, processing inputs from the previous agent (or user) and producing outputs for the next. Agents use pre-trained Ollama models for natural language processing tasks, such as prompt parsing or result interpretation, and Python libraries (e.g., `faker`, `scipy`, `pycaret`) for data-related tasks. The general structure of an agent is illustrated below, designed for integration into LangGraph's graph-based execution.

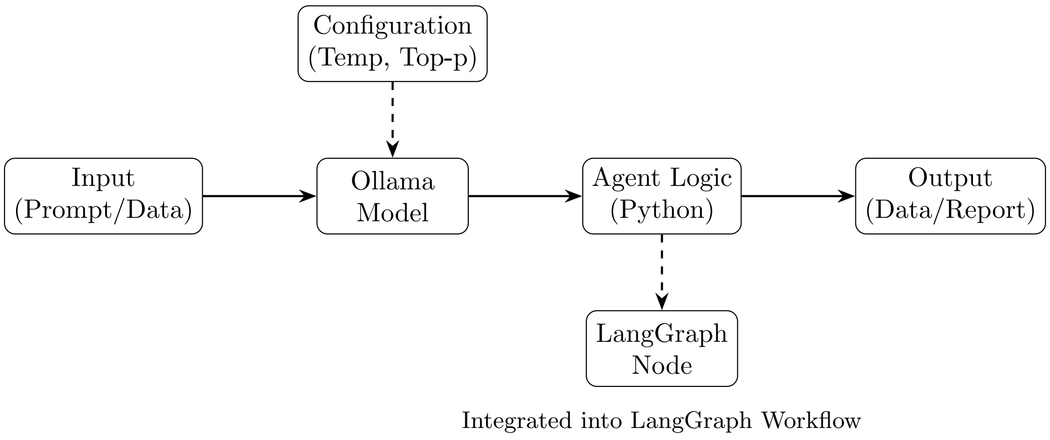

Figure 1: General Agent Structure

Each agent receives input (e.g., user prompt, dataset, or previous agent's output), processes it using an Ollama model (for NLP tasks) and Python logic (for data tasks), and produces structured output for the next agent. Configuration parameters (temperature, top-p) are set to balance creativity and determinism. The agent is encapsulated as a LangGraph node, enabling sequential execution and state management.

2 Agent Descriptions

2.1 User Prompt Parser Agent

Role: Parses user-provided prompts describing datasets or problem statements into structured inputs for downstream agents.

Goal: Convert natural language prompts into a JSON object specifying dataset characteristics or problem details, ensuring compatibility with the Code Generation for Synthetic Data Agent.

Inputs:

– **Source**: User-provided text prompt (e.g., "Create a dataset for fraud detection with columns: transaction_id (int), amount (float), country (string), is_fraud (bool)").

– **Format**: Plain text string.

– **Handling**: The agent uses an Ollama model to extract key details (e.g., column names, types, target variable) and validates the prompt for completeness (e.g., presence of a target variable).

Outputs:

– **Format**: JSON object, e.g., `{"columns": [{"name": "transaction_id", "type": "int"}, {"name": "amount", "type": "float"}, {"name": "country", "type": "string"}, {"name": "is_fraud", "type": "bool"}], "target": "is_fraud", "rows": 1000}`.

– **Usability**: Structured JSON ensures the Code Generation Agent can directly interpret column types and generate appropriate code. The output includes a default row count (e.g., 1000) if unspecified.

Ollama Configuration:

– **Temperature**: 0.3 (low to ensure deterministic parsing of structured information).

– **Top-p**: 0.9 (high to allow slight flexibility in interpreting varied prompt phrasings).

Prompt Engineering Suggestions:

– Use a clear instruction: "Extract dataset details (column names, types, target variable, row count) from the following prompt and output as JSON. If row count is missing, default to 1000."

– Provide examples in the prompt: "Example input: 'Dataset with user_id (int), purchase (float), date (string), churn (bool).' Example output: `{"columns": [...], "target": "churn", "rows": 1000}`."

– Include error handling: "If the prompt lacks a target variable, return an error message requesting clarification."

2.2 Code Generation for Synthetic Data Agent

Role: Generates Python code to create synthetic datasets based on user specifications, ensuring well-formed data for downstream agents.

Goal: Produce executable Python code using `faker` for tabular data or `scipy` for time series, tailored to the dataset description.

Inputs:

– **Source**: JSON object from the User Prompt Parser Agent.

– **Format**: JSON, e.g., `{"columns": [{"name": "transaction_id", "type": "int"}, ...], "target": "is_fraud", "rows": 1000}`.

– **Handling**: The agent interprets the JSON to select the appropriate library (`faker` for tabular, `scipy` for time series based on keywords like "date" or "timestamp") and validates column types.

Outputs:

– **Format**: CSV file containing the synthetic dataset, e.g., `data.csv` with columns `transaction_id, amount, country, is_fraud`.

– **Usability**: The CSV is directly usable by the EDA Agent, with standard data types (int, float, string, bool) and a target column for PyCaret compatibility.

Ollama Configuration:

– **Temperature**: 0.5 (moderate to balance code accuracy with flexibility for library selection).

– **Top-p**: 0.85 (moderate to ensure relevant code snippets).

Prompt Engineering Suggestions:

- Use a directive: "Generate Python code to create a synthetic dataset based on the provided JSON. Use `faker` for tabular data or `scipy` for time series if 'date' or 'timestamp' is present."
- Include examples: "For {`"columns": [{"name": "id", "type": "int"}, {"name": "date", "type": "string"}]`}, output code using `scipy` to generate time series."
- Specify validation: "Ensure the code produces a CSV with the exact column names and types specified."

2.3 EDA Agent

Role: Conducts non-traditional exploratory data analysis to produce a detailed dataset description, assessing its suitability for the problem.

Goal: Generate a structured report on dataset characteristics, focusing on data types, target variable suitability, and domain relevance, without classical statistics.

Inputs:

- **Source**: CSV dataset from the Code Generation Agent or user-provided.
- **Format**: CSV file, e.g., `data.csv` with columns `transaction_id, amount, country, is_fraud`.
- **Handling**: The agent uses Python (e.g., `pandas`) to analyse column types, target variable distribution, and domain-specific patterns (e.g., country diversity for fraud detection).

Outputs:

- **Format**: JSON report, e.g., {`"columns": [{"name": "amount", "type": "float", "relevance": "high for fraud detection"}], "target_suitable": true, "notes": "Diverse countries enhance fraud detection"`}.
- **Usability**: The JSON report is consumable by the PyCaret Model Running Agent, providing context for model selection (e.g., classification for boolean targets).

Ollama Configuration:

- **Temperature**: 0.4 (low to ensure precise analysis).
- **Top-p**: 0.9 (high to allow nuanced descriptions of dataset value).

Prompt Engineering Suggestions:

- Use a clear task: "Analyse the CSV dataset and produce a JSON report describing column types, target variable suitability, and domain relevance. Avoid statistical metrics like correlations."
- Provide context: "For fraud detection, assess if columns like 'country' or 'amount' are relevant."
- Include examples: "Output example: {`"columns": [...], "target_suitable": true, "notes": "Sufficient samples for classification"`}."

2.4 PyCaret Model Running Agent

Role: Generates and executes PyCaret code to train a machine learning model, saving the model and its performance metrics/plots.

Goal: Automate model training for classification or regression tasks, storing the model and evaluation outputs for interpretation.

Inputs:

- **Source**: CSV dataset from the EDA Agent and JSON report from the EDA Agent.
- **Format**: CSV (e.g., `data.csv`) and JSON (e.g., {`"target": "is_fraud"`}).
- **Handling**: The agent uses the JSON to identify the target variable and problem type (e.g., classification), then runs PyCaret's `setup` and `compare_models`.

Outputs:

– **Format**: Saved model file (e.g., `model.pkl`), metrics JSON (e.g., {`"accuracy": 0.85, "auc": 0.90`}), and plots (e.g., `confusion_matrix.png`).

– **Usability**: The model and metrics are directly usable by the Model and Results Interpreter Agent for performance analysis.

Ollama Configuration:

– **Temperature**: 0.3 (low for precise code generation).

– **Top-p**: 0.8 (moderate to ensure standard PyCaret code).

Prompt Engineering Suggestions:

– Use a directive: "Generate PyCaret code to train a model on the provided CSV, using the target variable from the JSON. Save the model and metrics/plots."

– Include examples: "For {`"target": "is_fraud"`}, use `setup(data, target='is_fraud', session_id=123)`."

– Specify outputs: "Save the model as `model.pkl`, metrics as JSON, and plots as PNG files."

2.5 Model and Results Interpreter Agent

Role: Interprets PyCaret model metrics and plots, generating a context-specific performance report.

Goal: Provide a clear explanation of model performance (e.g., accuracy, AUC) in the context of the problem (e.g., fraud detection), combining model and pipeline insights.

Inputs:

– **Source**: Metrics JSON, plots, and CSV dataset from the PyCaret Model Running Agent; JSON report from the EDA Agent.

– **Format**: JSON (e.g., {`"accuracy": 0.85`}), PNG files, CSV, and EDA JSON.

– **Handling**: The agent uses Ollama to interpret metrics and plots, contextualising them with the EDA report (e.g., "High recall suitable for fraud detection").

Outputs:

– **Format**: JSON report, e.g., {`"performance": "Accuracy: 0.85, suitable for fraud detection due to balanced dataset", "recommendations": "Increase recall for stricter flagging"`}.

– **Usability**: The report is consumable by the Report Formatter Agent for final output.

Ollama Configuration:

– **Temperature**: 0.6 (moderate to allow interpretive flexibility).

– **Top-p**: 0.9 (high to capture nuanced explanations).

Prompt Engineering Suggestions:

– Use a clear instruction: "Interpret the provided model metrics and plots, using the EDA report for context. Output a JSON report with performance summary and recommendations."

– Provide examples: "For {`"accuracy": 0.85`}, output: {`"performance": "High accuracy indicates robust model"`}."

– Include context: "For fraud detection, prioritise recall over precision."

2.6 Business Rule Agent

Role: Applies and interprets business rules on the dataset, producing a modified dataset and rule summary.

Goal: Formalise and apply business rules (e.g., "Flag transactions from country X as suspicious") and describe their impact in the problem context.

Inputs:

- **Source**: CSV dataset from the PyCaret Model Running Agent and a rule prompt from the user.
- **Format**: CSV (e.g., `data.csv`) and text prompt (e.g., "Flag transactions where country in ['X', 'Y'] as suspicious").
- **Handling**: The agent formalises the rule into Python code (e.g., `df['suspicious'] = df['country'].isin(['X'` `'Y']))` and applies it.

Outputs:

- **Format**: Modified CSV (e.g., `data_ruled.csv` with new column `suspicious`) and JSON rule summary (e.g., {`"rule": "Flag countries X, Y", "impact": "10% transactions flagged"`}).
- **Usability**: The modified CSV and summary are usable by the Report Formatter Agent.

Ollama Configuration:

- **Temperature**: 0.4 (low for precise rule formalisation).
- **Top-p**: 0.85 (moderate to ensure accurate code generation).

Prompt Engineering Suggestions:

- Use a directive: "Formalise the provided rule prompt into Python code and apply it to the CSV. Output the modified CSV and a JSON summary of the rule's impact."
- Include examples: "For 'Flag amount ¿ 1000', output code: `df['flag'] = df['amount'] > 1000`."
- Specify validation: "Ensure the rule is applied to all rows and the summary includes affected row percentage."

2.7 Configuration Agent

Role: Allows users to adjust parameters for model training or rule application via a text-based interface.

Goal: Provide a simple mechanism to update configurations (e.g., PyCaret model type, rule thresholds) for flexibility in the pipeline.

Inputs:

- **Source**: User-provided configuration prompt.
- **Format**: Text, e.g., "Use random forest for PyCaret" or "Set fraud threshold to 500".
- **Handling**: The agent parses the prompt to generate a configuration JSON, validating for supported options (e.g., PyCaret algorithms).

Outputs:

- **Format**: JSON configuration, e.g., {`"pycaret_model": "rf", "fraud_threshold": 500`}.
- **Usability**: The JSON is usable by the PyCaret Model Running or Business Rule Agent to adjust their behaviour.

Ollama Configuration:

- **Temperature**: 0.3 (low for precise parsing).
- **Top-p**: 0.8 (moderate to ensure valid configurations).

Prompt Engineering Suggestions:

– Use a clear task: "Parse the configuration prompt and output a JSON with supported parameters for PyCaret or business rules."

– Provide examples: "For 'Use random forest', output: {`"pycaret_model": "rf"`}."

– Include validation: "Reject unsupported parameters and request clarification."

2.8 Report Formatter Agent

Role: Standardises outputs from the pipeline into a unified format for API delivery.

Goal: Combine EDA, model performance, and rule application outputs into a cohesive report, suitable for API integration.

Inputs:

– **Source**: JSON reports from EDA, Model and Results Interpreter, and Business Rule Agents; CSV from Business Rule Agent.

– **Format**: Multiple JSON files and CSV.

– **Handling**: The agent aggregates inputs into a single structured output, ensuring consistency.

Outputs:

– **Format**: JSON report, e.g., {`"eda": {...}, "model": {...}, "rules": {...}, "summary": "Model and rules suitable for fraud detection"`}.

– **Usability**: The JSON is ready for API delivery, ensuring compatibility with Week 6's API Design and Test phase.

Ollama Configuration:

– **Temperature**: 0.5 (moderate for clear summarisation).

– **Top-p**: 0.9 (high for comprehensive reporting).

Prompt Engineering Suggestions:

– Use a directive: "Combine the provided JSON reports and CSV into a unified JSON report, summarising key findings."

– Include examples: "Merge {`"eda": {...}, "model": {...}`} into a single JSON with a summary field."

– Specify format: "Ensure the output is API-compatible JSON with clear sections."

3 Team Composition

Name	**Role**	**Agent**	**Comments**
Rafael	ML and APIs for Models, AI-Dev	***	
Andrés	Frontend, AI-Dev	***	
Carlos	Frontend, AI-Dev	***	Integrator
Mauricio	Backend, AI-Dev	***	
Johnattan	DevOps, AI-Dev	***	
Gastón	Tech Lead, AI-Dev	***	
Jan	AI–developer	***	
Enrique	AI–developer	***	

Table 1: Team Members and Assigned Agents