Designing Data Quality Validation and Model Summarization Agents for an Agentic Clinical ML Workflow

1. Overview

This document describes two agents that extend an agentic clinical machine-learning workflow: a **Data Quality Validation Agent** and a **Model Output Summarizer Agent**. Both are implemented using LangGraph and LangChain, and are demonstrated on a heart-failure mortality prediction task with tabular clinical data.

The goal is to turn two often informal notebook steps data quality checks and model interpretation into first-class, reusable agents with clear contracts. The design is deliberately modular so these agents can be dropped into the broader workflow described in the Agentic Medical AI paper without rewriting existing components.

2. Workflow and Dataset

The prototype pipeline is a small LangGraph:
load_data → data_quality_validation → train_and_infer → model_output_summarizer → END

load_data reads a heart-failure dataset (Kaggle) into the state. There are two variants: a **clean** version and a **dirty** version where issues are deliberately injected (high missingness in key labs, negative values for non-negative measures, duplicates, and missing target labels). This gives a simple way to test whether the Data Quality Validation Agent behaves sensibly and whether its decision truly gates model training.

All information flows through a shared PipelineState object. More structured pieces, such as dataset profiles and model summaries, are Pydantic models. This makes the graph easy to inspect, log and extend, and aligns with the "typed state, explicit contracts" style in the paper.

3. Data Quality Validation Agent

Purpose.
This agent answers a simple but crucial question: *"Is this dataset safe and sensible for modeling, and why?"* It is deliberately model-agnostic and focuses only on the data.

Inputs and outputs.
The agent reads the dataframe from state and, when present, the target column name It writes back:

- a DatasetProfile summarizing global properties (rows, columns, duplicates) and per-column profiles;
- a DataQualityDecision with status ∈ {PASS, WARN, FAIL}, a markdown summary and a list of issues;
- a validation_status string that downstream nodes can branch on.

The markdown report is also saved to disk as data_quality_report_<run_id>.md for later inspection.

Internal logic.
The agent is a hybrid of rule-based checks, anomaly detection and LLM reasoning.

First, it performs deterministic profiling. For each column, it infers a coarse type (numeric, categorical, mixed), computes missingness and number of unique values, and, for numeric columns, calculates basic statistics (mean, standard deviation, min, max, quartiles). Outliers are flagged using an interquartile-range rule. Domain constraints are encoded for variables that should never be negative (ejection fraction, serum creatinine, serum sodium, age, time, etc.); negative values are treated as "impossible" and added to the issue list. At the dataset level, duplicates and target-column problems (missing labels, single-class outcomes, severe imbalance) are detected.

Second, the agent runs a **row-level anomaly detector** using Isolation Forest on numeric features This captures multivariate oddities that might not show up in per-column summaries. The agent records how many rows are flagged as anomalies and, if that fraction is large, adds a corresponding issue ("≈10% rows anomalous by Isolation Forest"), which we indeed see on the dirty variant.

Finally, all structured information is passed into an LLM using LangChain's structured output capabilities targeting the DataQualityDecision schema. The prompt asks the model to assign PASS/WARN/FAIL, and to produce a markdown report with fixed sections: Overview, Column-level checks, Row-level anomalies, Recommendation and Issues. This keeps the final answer both machine-readable and human friendly.

Integration.
The key integration point is validation_status. The train_and_infer node checks this value; if it is FAIL (for example, when half of serum_creatinine is missing and the target has NaNs), the node skips training and only passes along an empty metrics dict. This mirrors how, in the full agentic workflow, a failed upstream validation should halt or reroute the pipeline rather than silently training on bad data.

4. Model Output Summarizer Agent

Purpose.
This agent sits on the other side of the model and explanation step. It answers: *"Given this model and this data, what does the model do, how well does it perform, which features matter most, and how cautious should we be?"* It is designed to be reusable for different models, as long as they expose standard metrics and feature importances.

Inputs and outputs.
The agent expects three things in state:

- metrics: AUC, accuracy and F1 from the test set;

- top_features: a ranked list of features with SHAP-based importance scores;
- validation_status and the text summary from the Data Quality Validation Agent.

If metrics or feature importances are missing (because validation failed and training was skipped), the agent returns a short placeholder stating that no model is available. Otherwise, it asks an LLM (again via structured output) to produce a ModelSummary object with a title, executive summary, performance section, feature-importance section, data-quality section and recommendations. The summary explicitly states that feature importance is based on SHAP values and references concrete feature names such as time, ejection_fraction, serum_creatinine and age. It also folds in the data-quality verdict, for example recommending caution when anomalies are frequent or missingness is high. The resulting markdown is stored in the state and written to model_summary_<run_id>.md.