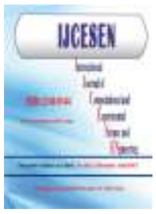
International Journal of Computational and Experimental Science and ENgineering *(IJCESEN)*

Vol. 11-No.3 (2025) pp. 5755-5767
http://www.ijcesen.com

Copyright © IJCESEN

ISSN: 2149-9144

Research Article



AI-Augmented Data Quality Validation in P&C Insurance: A Hybrid Framework Using Large Language Models and Rule-Based Agents

Shreekant Malviya[1]*, Vrushali Parate[2]

[1]Independent Researcher, Plano, TX, USA
* **Corresponding Author Email:** malviyashreekant@gmail.com - **ORCID:** 0009-0009-7229-657X

[2]Independent Researcher, Farmington Hills, MI, USA
Email: vrushali30109@gmail.com- **ORCID:** 0009-0007-5031-3581

Article Info:

DOI: 10.22399/ijcesen.3613
Received : 02 June 2025
Accepted : 30 July 2025

Keywords

Data Quality,
Large Language Models (LLMs),
Agentic AI framework,
Property & Casualty Insurance

Abstract:

The growing complexity and volume of data in Property & Casualty (P&C) insurance have intensified the need for robust, scalable, and intelligible data quality validation methodologies. Conventional rule-based validation systems provide transparency; however, they have challenges in adapting to evolving data and regulatory requirements. This paper addresses these challenges using a hybrid methodology that integrates Agentic AI, merging the precision of deterministic rule logic with the inferential prowess of large language models (LLMs). The architecture consists of modular agents—ProfilerAgent, LLMRuleAgent, RuleAgent, and SummaryAgent—each designated with a distinct role in a data quality pipeline, enhancing transparency, reusability, and scalability. Through the use of a locally hosted LLaMA model with Ollama, the system produces schema-aware YAML rules, verifies structured datasets, and creates natural language data quality issue summaries. An experimental evaluation with a real-world auto insurance claims dataset from Kaggle showed that the framework successfully identified schema mismatches, format problems, and semantic discrepancies without requiring human rule generation. The results indicate that the agentic architecture increases flexibility in resource-limited, compliance-focused settings. The research presents a new solution to the current controversy on automatic data governance in insurance by fusing explainable AI with operational reliability and developing a feasible solution for businesses, striking a balance between regulatory requirements and digital transformation initiatives. While experimented in the P&C context, the modular design enables straightforward adaptation to other domains such as retail and healthcare where similar data quality challenges exist.

1. Introduction

Property and Casualty (P&C) insurance depends on high-quality structured data for many high-stakes business decisions. The mission-critical decisions include underwriting, claims settlement, fraud identification, and risk analysis [1, 2]. Faulty or incomplete data, like missing policyholder data, inaccurate dates of incidents, or misclassified claim types, can cause defective risk assessment, regulatory and compliance breaches, and expensive organizational and operational inefficiencies. With the size and pace of information in modern insurance environments growing by the day, data accuracy, completeness, and consistency have become a technical requirement as much as a strategic one [1-4].

Conventional insurance companies have traditionally used rule-based data validation systems. Here, subject matter experts clearly define rules like NOT_NULL constraints, field-level domain value ranges, or correct date formats, and then these are implemented by SQL scripts or ETL pipelines. Even though these sorts of systems provide control and openness, they are not very scalable, have rigid logic, and are very expensive to operate with, especially at the time of schema evolution or regulatory change. As insurers evolve toward digital transformation efforts with more varied and changing data sources, this becomes more and more evident.

Recent progress in artificial intelligence, particularly large language models (LLMs) such as GPT and LLaMA, presents a new model for smart data validations. They can infer rules, identify semantic problems, and produce human-readable abstractions without the need for lots of manual setup [5, 6]. Yet LLM-based systems tend to reach stumbling blocks in production deployment owing to issues with traceability, control, and operational reliability.

This research suggests a hybrid data quality validation system that combines the auditability of traditional rule-based systems and the intelligence and responsiveness of large language models (LLMs) in order to overcome these challenges. The system is modular, agent-based with independent agents executing unique operations: profiling the data structures, rule construction, validation enforcement, and summarization. The method basically takes advantage of a locally deployed LLM (via Ollama) to carry out data privacy with the needed agility in order to enable scalability proofing. The method is specifically optimized to function effectively in resource-poor environments, typical with mid-sized insurance companies and legacy infrastructure deployments.

1.1 Contributions

This paper makes the following key contributions

- A Hybrid Validation Architecture which comprises of a new agent-based framework that combines rule-based and LLM-driven validation methods. This gives insurance data workflows a good mix of control and automation.
- LLM-Driven Rule Generation with Explainability to show how few-shot prompting with local LLMs may be utilized to make natural language summaries and field-specific validation rules without having to write any code.
- Modular Agentic Design loosely connects agents (ProfilerAgent, LLMRuleAgent, RuleAgent, SummaryAgent), which makes it easy to debug, add new features, and upgrade individual parts.
- Resource-Aware Deployment Strategy to ensure that validation is scalable and keeps privacy in contexts where cloud access is limited or there are regulatory restrictions by employing models hosted locally (via Ollama).
- Application to publicly available P&C vehicle insurance dataset to show how useful it is in real life, how well it diagnoses problems, and how well it can handle complicated insurance workloads.

2. Related Work

Data quality has been a long-standing issue in financial services and insurance industries because low-quality data can have significant downstream effects, such as incorrect pricing, erroneous risk modeling, and regulatory noncompliance [3, 4]. Conventional data validation techniques in such use cases have mostly been based on static rules implemented through SQL queries, schema enforcement tools, or Extract-Transform-Load (ETL) processes. These classic methods are recognized for their accuracy and reliability, particularly in controlled and regulated environments. Nonetheless, these technologies struggle to adapt to developing data structures or complex business processes without extensive human monitoring, rendering them unscalable in dynamic organizational environments [7]. Due to these shortcomings, the industry has been looking for AI- and ML-based solutions for evaluating the quality of data. Business software like IBM InfoSphere and Talend Data Quality have tools like statistical profiling, pattern recognition, and unsupervised anomaly detection that can find data anomalies without having to explicitly define the rules [8]. Some studies have also suggested using neural networks, decision trees, or clustering to identify outliers and inconsistency in large volume of insurance data [9-11]. Although these advanced methods show potential, they usually require a considerable amount of labeled training data and expert tuning. In addition to this, the lack of interpretability makes it difficult to deploy such a solution in highly regulated domain [12-14]. Despite technological progress, rule-based validation methods continue to prevail over operational workflows, particularly in the insurance sector [1, 2]. The predictable behavior, traceability, and regulatory friendliness render them attractive to workplaces that place a premium on adherence to the compliance [1, 2]. But conventional rule systems are confronted with imposed constraints i.e., lack of generalizability to novel patterns of data, rigidity across data sources, and restricted capacity to represent inter-field relationships [15]. The aforementioned issues have prompted an increasing interest in hybrid validation frameworks that amalgamate human-defined criteria with machine-learning diagnostics to enhance adaptability and automation [15-17].Recent advancements in large language models (LLMs), such OpenAI's GPT and Meta's OLLaMA, have created new opportunities in data quality automation. They display superiority in few-shot prompts, structured reasoned thinking, and natural language generation, supporting new applications such as automated rule generation, anomaly reasoning, and data validation task code generation [6], [18-20]. Experimental have established that LLMs can infer schema expectations, semantically diagnose issues, and create test cases under minimal supervision [5, 21]. However, there are issues with model hallucinations, prompt sensitivity, reproducibility, and integration issues with current data systems [22, 23].This paper here adds to the field by providing a modular agentic system that uses LLMs for automated rule generation and summarization without losing auditability and determinism of the classical rule-based validations. The suggested framework remedies numerous shortcomings inherent in both rule-based and solely AI-based systems by the purposeful addition of traceability and explainability into its structure. Its potential application to the property and casualty insurance industry makes it a plausible option for controlling data quality in real-world implementations, particularly where operational adaptability and regulatory adherence must be reconciled.

3. An Agentic System for AI-Augmented Data Quality

3.1 Agent Architecture

The AI based Data Quality (DQ) validation system presented in this paper is intended to be built on a modular, agent-based architecture that reflects the concepts of agentic AI systems. Under this architecture, independent AI agents cooperate, each responsible for a specific feature of the data quality process. Taking inspiration from multi-agent systems, the agents interact using clearly specified inputs and outputs, allowing for extensibility, transparency, and verifiable quality decisions. The design **[Figure 1]** is in sequential but loosely connected flow to facilitate additions or replacements of any one module without compromising the integrity of the overall system.The DQOrchestratorAgent is the prime coordination piece that initiates the pipeline, tracks execution, manages the temporal memory of outputs, and directs the order of agent interaction. The process starts with data ingestion and profiling, continues through rule creation, data validation, and summarization using AI assistance, and then stores output and summaries for analysis and reporting. This part briefly describes the role and interaction of each agent in the system and proceeds to respective roles and their contribution to a comprehensive, AI-powered data quality assessment.

The ProfilerAgent initializes the pipeline by reading and loading input data from a CSV file. It produces a Pandas Dataframe view of the data and then forwards it downstream. The agent acts as a entry point, providing an organized and coherent data representation to all downstream pieces.

After the dataset has been profiled, it is forwarded to the LMRuleAgentOllama. The agent utilizes a light-weight local large language model (e.g., LLaMA 3.2) and few-shot prompting **[Figure 2]** to reason and generate context-specific data quality validation criteria. Rules are strictly typed in YAML with restrictions applied to prevent markdown, ambiguity, or wrong formatting. The agent ensures dynamic rule generation and data-set awareness, preventing manual bottlenecks typically related to static rule definition. The internally generated YAML rules are also tested and filtered before being used.RuleAgent then applies the generated rules and compares the dataset to the inferred rules on a column-by-column basis. The adaptive nature effectively captures defective data, it records any deviations as issues and runs numerous quality checks, including not_null, positive_integer, date_format, and length(n). To maintain traceability, each identified issue is classified based on row identifiers and rule types.

DQMemory keeps track of each event in a structured JSON format which makes it a lightweight memory bank for the orchestrator that guarantees auditability and reliability. It also supports version control by creating timestamped output folders, and logs any activity that happens along the way, keeping track of outputs and transition states at all critical points, such as rule generation, validation, and summary.

Finally, the SummaryAgent produces a natural language summary of encountered data quality issues and potential solutions derived from the issue list by utilizing LLM to enhance mobility and analytical autonomy. By translating raw issues into an actionable and readable format, this agent bridges the gap between detection and understanding.The DQOrchestratorAgent regulates the precise sequence in which different agents communicate information. Through orchestration, it establishes a linear yet modular pipeline by ensuring that the output of one agent serves as the input for another. The rule generator derives its input from the profiler, while the rule agent interprets and verifies the YAML output. The orchestrator manages the flow and versioned output while consolidating and recording the recognized defects.The agentic architecture serves as a clear and scalable framework for AI-enhanced data quality evaluation where each agent has a distinct, abstract objective and employs standardized structures for information interchange, enhancing flexibility and resilience. This method expedites the identification and resolution of data problems while facilitating ongoing enhancements via agent optimization and retraining, establishing a robust framework for reliable data governance in property and casualty insurance and beyond.

4. Experimental Configuration in P&C Insurance Workload

This section outlines the establishment of the agentic data quality pipeline, and the resources employed to evaluate the agents. The objective of the experiment is to determine whether the agentic data quality pipeline can identify and describe issues inside real structured data.

4.1 Data Set Description

The experimental dataset is an open-source Auto Insurance Claims dataset sourced from Kaggle. This dataset represents Property and Casualty (P&C) insurance workloads and is structured in a tabular format, with variables like as policy_number, total_claim_amount, incident_date, age, and incident_location. The dataset consists of 40 columns, encompassing a varied assortment of categorical, numerical, and date elements that reflect multiple aspects of Property & Casualty (P&C) insurance claims. The columns are categorized into client information, policy details, event specifics, and claim analysis for property, automobile, and injury claims. The dataset was chosen for its relevance in tackling quality-related issues **[Table 1]** such as missing values, formatting errors, especially in date fields, and schema conflicts, which are prominent data quality challenges seen in insurance data pipelines.

4.2 Tools and Configuration

The test environment was established with the aid of a lightweight, modular skeleton that is intended to be deployed on low-resource devices. The technologies employed are directly aligned with the objectives of modularity, locality of interpretability, and data practitioner accessibility beyond high-performance cloud environments.

- Python 3.10 was the primary scripting language across the pipeline. It ensured necessary support for agent interaction management, I/O operations, and storing results in the intermediate step in an orderly and readable manner.
- Pandas was utilized by ProfilerAgent to read, explore, and alter the tabular structured data. The DataFrame operations expressed fast schema extraction, type inference, and record-level access required for rule-based validation.
- Ollama with LLaMA3.2 was utilized to execute both SummaryAgent and LLMRuleAgent. The local large language model was selected due to its memory conservative usage and CPU-friendly design and is especially well suited for resource-constrained environments. It is faster than the traditional cloud-based LLM APIs that need GPU acceleration and internet connectivity, and Ollama can be executed entirely offline and supports quick execution with model optimization for edge deployment. This renders it a suitable option for organizations with rigid infrastructure limitations, data localization demands, or cloud computing facility unavailability.
- YAML was used as the output syntax for representing validation rules produced by the language model. YAML's human-readability and simplicity enable simple reviewing, versioning, and manual editing of rules when required. Further, its support in configuration-driven systems makes it a perfect vehicle for conversion of LLM output to executable reason.

All agents were executed on a local development system equipped with adequate CPU and RAM resources, around 8 GB of RAM and a dual-core processor, without specialized hardware acceleration. This confirms the feasibility of implementing the complete pipeline in edge or offline settings, such as on-premises environments, laptops, or corporate compliance servers. The selection of an autonomous local, functional LLM was intentional, prioritizing data privacy and eliminating dependence on external services or proprietary APIs, thereby ensuring security and reproducibility of the system. This setup demonstrates the viability of utilizing agentic data quality systems in production-like environments where cloud computing is inaccessible or cost-prohibitive, hence advancing the implementation of AI-driven data governance across various businesses and organizations.

4.3 Validation Scope

The validation scope in this study is purposefully designed around the characteristics and challenges inherent in the Auto Insurance Claims dataset used for experimentation. This structured tabular dataset, comprising over 40 fields such as policy_number, incident_date, total_claim_amount, incident_location, and fraud_reported, reflects the heterogeneity typical of Property and Casualty (P&C) insurance data pipelines. Accordingly, the agentic framework was tested to evaluate its performance in detecting rule violations, summarizing diagnostics, and maintaining transparency across diverse data fields **[Figure 3]** The modular pipeline comprises four sequential validation phases, each executed by a specialized agent:

1. **Schema Assessment**

   Conducted by ProfilerAgent, this phase ingests the raw CSV and constructs an initial schema snapshot by inferring column data types and identifying structural metadata. For instance, age and total_claim_amount are expected as positive integers, while incident_date and policy_bind_date are inferred as date fields. This baseline schema, derived directly from the experimental dataset, is essential for guiding downstream rule generation.

2. **Rule Generation**

   The LLMRuleAgentOllama applies few-shot prompting on a subset of the dataset—typically the first 5–10 rows—to generate column-specific validation rules in YAML format. Rules inferred during experimentation included:

   - not_null for critical fields such as fraud_reported and incident_severity
   - positive_integer for age, months_as_customer, and total_claim_amount
   - length(5) for zip codes like insured_zip
   - date_format: YYYY-MM-DD for date columns like incident_date and policy_bind_date
     These rules were tailored to the dataset schema, ensuring contextual relevance and automation in rule inference.

3. **Rule Application**

   The RuleAgent validates the entire dataset against the generated YAML rules. Each violation is logged with metadata such as row index, column name, and rule type. In this dataset, several validation issues were surfaced, including:

   - Incorrect date formats in incident_date (e.g., MM/DD/YYYY)
   - Null or invalid entries in age and total_claim_amount
   - Fixed-length mismatches in insured_zip. This stage transforms abstract validation logic into actionable insights.

4. **Summarization and Diagnostics**

   The SummaryAgent synthesizes a natural-language summary from the structured error logs. For example, it flagged that over 400 records in incident_date violated the expected YYYY-MM-DD format and recommended upstream ETL logic adjustments. The agent also highlighted recurring issues in age and fraud_reported, offering domain-specific suggestions like value imputation or format normalization.

Importantly, the scope of validation in this experiment is deliberately limited to issue detection and summarization. No automatic data correction techniques—such as imputation, transformation, or enrichment—were applied. This design choice supports the goals of explainability, auditability, and regulatory compliance, ensuring that outputs are transparent and interpretable by insurance domain experts, data stewards, and compliance officers.

4.4 Output Artifacts

The experimental pipeline produces three key artifacts for every dataset validation run, enabling traceability, reproducibility, and downstream auditability:

1. **generated_rules.yaml**
   YAML file in **[Figure 4]** contains the full set of validation rules inferred by the LLMRuleAgentOllama. Each rule is structured by column name and check type (e.g., not_null, positive_integer, date_format), serving as a declarative schema of expected data properties. The file facilitates transparency by exposing how semantic expectations are constructed for each field.

2. **issues.json**
   A structured JSON file **[Figure 5]** that enumerates all detected rule violations. Each entry includes the row index, column name, and the specific

failed check, as identified by the RuleAgent. This output provides a machine-readable diagnostic log that can be used for targeted inspection, alerting, or integration with data monitoring workflows.

3. **summary.txt**
 A human-readable textual report generated by the SummaryAgent **[Figure 6]**, summarizing key findings from the validation process. This includes an overview of columns with the highest error rates, common types of rule violations, and suggested remediation actions. It serves as a quick-reference guide for data stewards, analysts, or engineering teams. All output files are saved within a timestamped subdirectory following the format dq_outputs/YYYY-MM-DD_HH-MM-SS/. This structure enables versioning, supports longitudinal quality tracking, and ensures that each validation run remains reproducible for retrospective analysis or compliance reporting. Collectively, these artifacts demonstrate the agent-based system’s capacity to autonomously enforce, log, and communicate data quality expectations in real-world P&C insurance data pipelines.

5. Results and Observations

To gauge the effectiveness of the agent-based validation process, the experiment pipeline was run on the Auto Insurance Claims dataset. When applied to actual P&C insurance data, the process effectively generated actionable outcomes at every validation level, proving the system's interpretability and usability.A properly formatted YAML file called generated_rules.yaml was independently created by the LLMRuleAgent, using rules like strict date_format checks, positive_integer, and not_null. The agent's capability to learn schema-aware constraints was experimented with using these rules, which did not require them to be manually built and were highly in line with semantic expectations of the dataset's fields, these fields include incident_date, total_claim_amount, and age.

Several quality issues were detected and recorded in the issues.json file based on structure during the RuleAgent execution of the rules. Among the noteworthy trends of data quality violations were:

- Date Format Violations: The data in policy_bind_date and incident_date fields violated the date_format: YYYY-MM-DD test with very high frequency. These infractions point to incorrect date parsing on data importing, which could be caused by locale-dependent encoding (MM/DD/YYYY, for example) or legacy formatting.
- Null Failures and Integer Type: not_null and positive_integer constraints were broken in fields like total_claim_amount, age, and months_as_customer. They frequently resulted from non-numeric or null input, which undermine actuarial modeling and downstream analytics.
- Fixed-Length Check breachtrictions: In a small sample of rows, the insured_zip column, which was supposed to have adhered to a strict 6-character format, displayed length anomalies. Such anomalies point towards data consolidation inconsistencies or variant source system formatting.

SummaryAgent converted raw error logs into a readable summary and created a plain-text report (summary.txt) after validation. Highest error rate fields, most frequent rule-breaking, and recommended correction measures like date column reformatting, filling missing values by patching, or optimizing the upstream ETL logic were all reflected in the report. Technical and non-technical stakeholders are both able to act upon findings because of this human-readable output. One critical insight was how well and versatile LLM-based rule generation was. The few-shot prompting strategy, as opposed to conventional SQL-like rule definitions, enabled the system to create validation logic that was dataset-specific but transferable on other comparable insurance datasets. And problems. For every validation failure, the json artifact provided row-level traceability, which made debugging, auditing, and possibly integrating into data quality dashboards in the future easier.

6. Discussion

Experimental testing of the devised AI-supported data quality framework demonstrates exceptional strengths in scalability, flexibility, and explainability, especially in the context of Property and Casualty (P&C) insurance processes. The primary value added by the devised framework is that it can generate and apply validation rules dynamically on various datasets without human configuration. The system utilizes large language models (LLMs) to substitute the conventional, hard-coded SQL rules with context-specific prompts, formulating an elastic solution to the diverse data structures in different areas, policies, and time periods in the insurance sector. Beyond the insurance context, the modular design enables easy adaptation to data intensive domains with similar quality challenges such as retail, finance and healthcare. One of the most important advantages is the interpretability and traceability of the system. Each problem with the quality of the data can be linked to a specific record and is described in layman language by the SummaryAgent. This makes it hard to understand data logs into information that can be used by both technical and non-technical users. Additionally, YAML-based rule encoding allows human-in-the-loop review, which means that experts in the field can look over or change the rules that are being made before they are put into action.The agentic design enhances modularity by enabling each component from profiling to summarization to function independently. This modular architecture facilitates progressive maintenance, enhancements, and troubleshooting without necessitating alterations to the entire pipeline. This adaptability is crucial in the rapidly evolving insurance sector, where regulations are frequently revised and altered. Despite these advantages, some issues arose during the experiment setup, primarily due to the reliance on open-source LLMs and constrained computational resources:

- LLM model utilized: Rule accuracy is naturally tied to the accuracy of the LLM response. Minor discrepancies in quick build or absence of details regarding certain fields can lead to syntactically accurate but semantically inaccurate rules requiring human evaluation.
- YAML Parsing Sensitivity: Downstream processing may be impacted by parsing problems, the generated output by the model frequently is plagued by indentation errors or utilizes unsupported constructs. Such yaml errors unless preprocessed have the potential to disrupt dependent processes.
- LLM Hallucinations: The model at times proposed rules for missing fields in the data, indicating a high likelihood of LLM hallucinations. This requires additional testing to verify schema conformance prior to applying the rules.
- Comparison with SQL based system: SQL-based data quality systems are accurate, auditable, and predictable but typically are not scalable or flexible. The agentic approach gives some control in exchange for enhanced efficiency, faster onboarding, and the possibility of allowing rules to be executed on similar datasets. A blend of AI-driven rules and SQL logic written by experts may be the best and most ideal solution in a production environment.

Moreover, the inadequate infrastructure and the lightweight characteristics of the implemented local LLM restricted system throughput and resilience. The implementation of resource-efficient architecture, albeit advantageous for on-premises and privacy-sensitive applications, resulted in compromises regarding output dependability and generation latency. Small prompt variations particularly, can result in formatting differences or irregular rule coverage. These constraints imply that using more powerful or higher-level models—with better structural coherence and context handling—would substantially enhance overall performance and decrease the need for human verification or post-processing.
In summary, the approach is successful in illustrating how LLM-powered, agent-based systems are able to tighten data quality processes in sophisticated industry like insurance. While the current implementation validates the conceptual soundness of the approach, enhancements to model robustness, output consistency, and validation robustness under certain conditions will be beneficial to real-world scalability and production quality.

7. Future Work

To increase the scalability, robustness, and coverage of the existing framework, a number of improvements are envisioned. While rule-based verification of structured insurance data sets by the existing system is sufficient, there will be improvements in the future aimed at improving its readiness for work and analytical depth.

- Use of incorporated statistical anomaly detection methods alongside deterministic rule-based verification. Unusual data, or outliers, within valid ranges but that don't follow normal patterns may be detected by methods like z-score analysis, clustering, or isolation forests to add to the foundation. These techniques detect subtle problems with data quality that rule-based tests like not_null or positive_integer would otherwise ignore, such as fraud, data drift, or outlier reporting.
- Semantic accuracy in rules generated by LLMs will be enhanced by incorporating domain ontologies and insurance taxonomies into the rule generation process. Hallucinations can be avoided, and syntactically correct rules in accordance with industry standards can be generated by the system through structured knowledge of types of coverage, regulatory rules, and life cycle phases of claims. Particularly in traceable and domain-correct

regulated environments, this will increase confidence.

- Even though the existing paradigm is best suited for structured tabular data, untested data that has never been validated can be accessed in unstructured sources like adjuster remarks, claim reports, or customer conversations. To retrieve, analyze, and validate content from different domains, future releases could have Natural Language Processing (NLP) capability. Use cases are the identification of missing contextual entities (i.e., the where or when of an accident), ithe dentification of contradictory or incomplete descriptions, and the confirmation of sentiment congruence with claim outcomes.
- Support for integration with real-time streaming sources like Apache Kafka or Spark Streaming must be enabled in order to proceed with operational deployment. When fresh information is being digested, low-latency real-time checking would be facilitated with the integration of light-weight agents into such systems. With such a paradigm shift from batch to streaming architecture, in usage such as fraud prevention, claims adjudication, and risk analysis, proactive error identification is facilitated, downstream initiation of erroneous data is minimized, and business demands for rapid conclusions are fulfilled.

In addition to refining the existing system in the insurance scenario, these upgrades will allow it to be implemented in bigger commercial environments where operational efficiency, data reliability, and transparency are of critical importance.

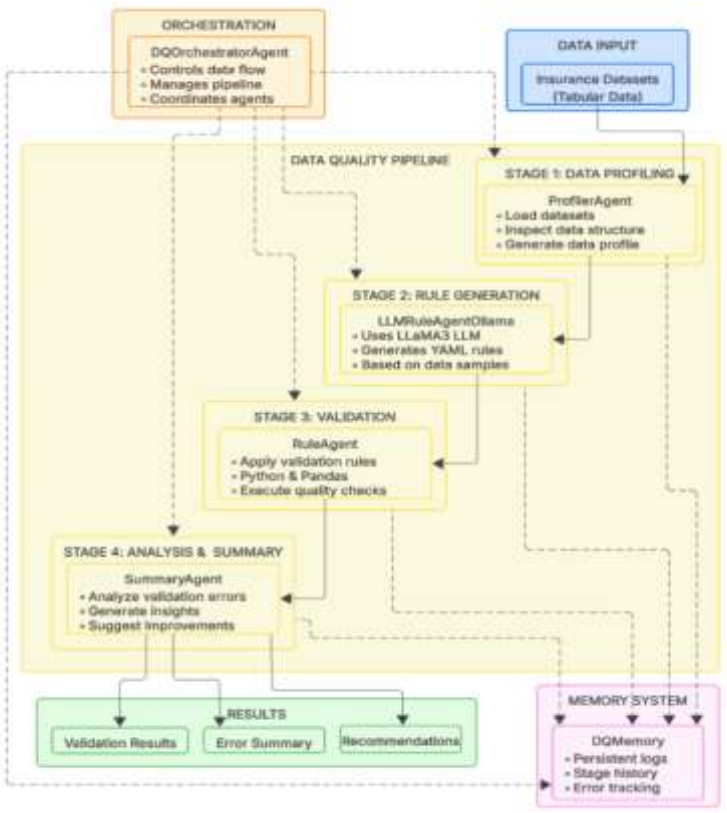
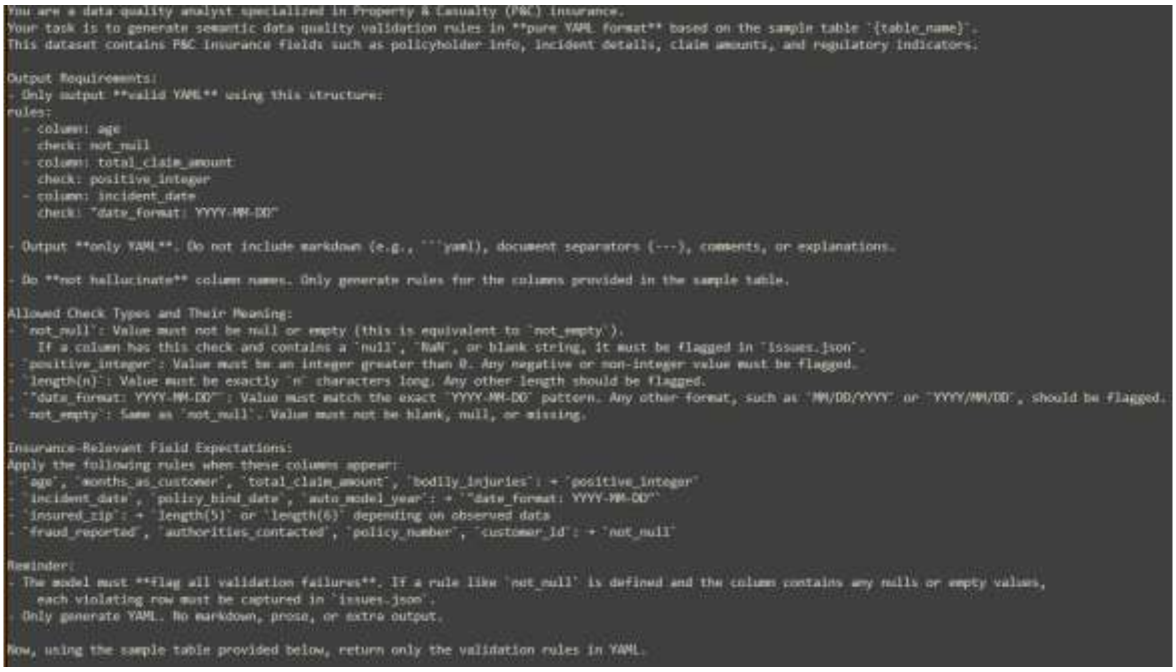


Figure 1. *System Architecture Diagram*

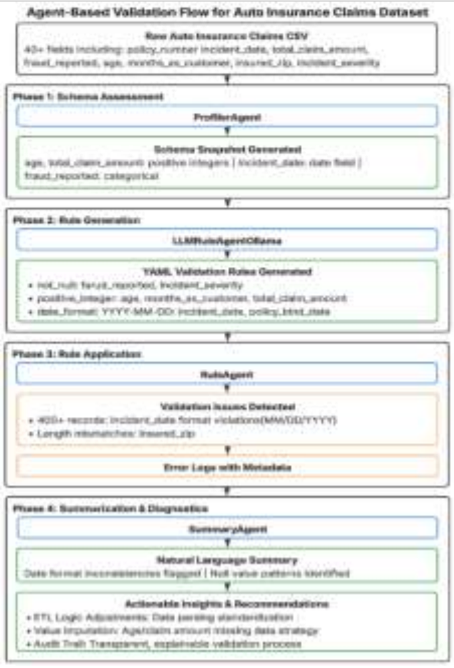


```
You are a data quality analyst specialized in Property & Casualty (P&C) insurance.
Your task is to generate semantic data quality validation rules in **pure YAML format** based on the sample table `{table_name}`.
This dataset contains P&C insurance fields such as policyholder info, incident details, claim amounts, and regulatory indicators.

Output Requirements:
- Only output **valid YAML** using this structure:
rules:
  - column: age
    check: not_null
  - column: total_claim_amount
    check: positive_integer
  - column: incident_date
    check: "date_format: YYYY-MM-DD"

- Output **only YAML**. Do not include markdown (e.g., ```yaml), document separators (---), comments, or explanations.

- Do **not hallucinate** column names. Only generate rules for the columns provided in the sample table.

Allowed Check Types and Their Meaning:
- `not_null`: Value must not be null or empty (this is equivalent to `not_empty`).
    If a column has this check and contains a `null`, `NaN`, or blank string, it must be flagged in `issues.json`.
- `positive_integer`: Value must be an integer greater than 0. Any negative or non-integer value must be flagged.
- `length(n)`: Value must be exactly `n` characters long. Any other length should be flagged.
- `"date_format: YYYY-MM-DD"`: Value must match the exact `YYYY-MM-DD` pattern. Any other format, such as `MM/DD/YYYY` or `YYYY/MM/DD`, should be flagged.
- `not_empty`: Same as `not_null`. Value must not be blank, null, or missing.

Insurance-Relevant Field Expectations:
Apply the following rules when these columns appear:
- `age`, `months_as_customer`, `total_claim_amount`, `bodily_injuries`: → `positive_integer`
- `incident_date`, `policy_bind_date`, `auto_model_year`: → `"date_format: YYYY-MM-DD"`
- `insured_zip`: → `length(5)` or `length(6)` depending on observed data
- `fraud_reported`, `authorities_contacted`, `policy_number`, `customer_id`: → `not_null`

Reminder:
- The model must **flag all validation failures**. If a rule like `not_null` is defined and the column contains any nulls or empty values,
    each violating row must be captured in `issues.json`.
- Only generate YAML. No markdown, prose, or extra output.

Now, using the sample table provided below, return only the validation rules in YAML.
```

Figure 2. *Prompt Used by LLMRuleAgent to Generate Rules*

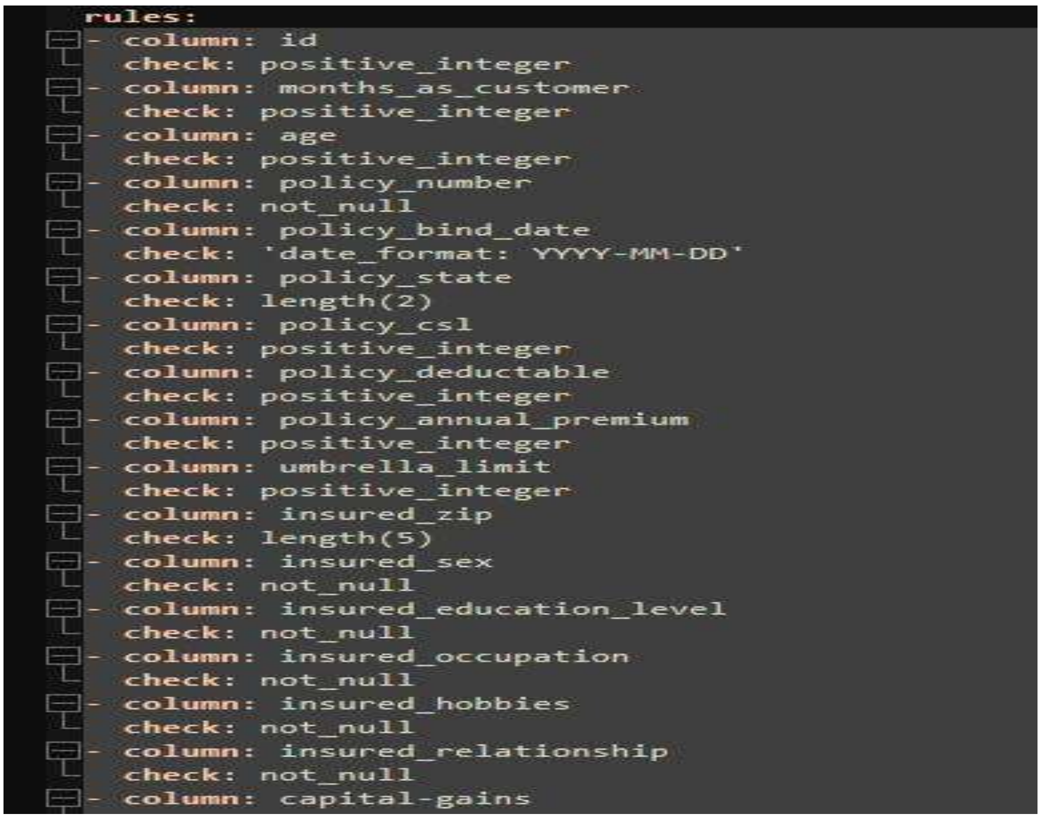
Table 1. *Auto Insurance Dataset Columns by Potential Anomalies*

Anomaly Type	**Affected Columns**
Missing or Null Values	age, total_claim_amount, fraud_reported, property_damage, collision_type, witnesses, incident_severity, number_of_vehicles_involved
Incorrect Date Formats	incident_date, policy_bind_date
Inconsistent Lengths	insured_zip (e.g., not meeting fixed length of 5 characters)
Invalid or Unexpected Types	age, months_as_customer, total_claim_amount, policy_deductible, policy_annual_premium
Semantic Inconsistencies	insured_hobbies, insured_education_level, incident_type, authorities_contacted (e.g., unexpected categorical values or typos)
High Cardinality Identifiers	policy_number, incident_location, incident_city
Label Integrity Issues	fraud_reported (e.g., ambiguous values beyond expected binary 'Y' or 'N')



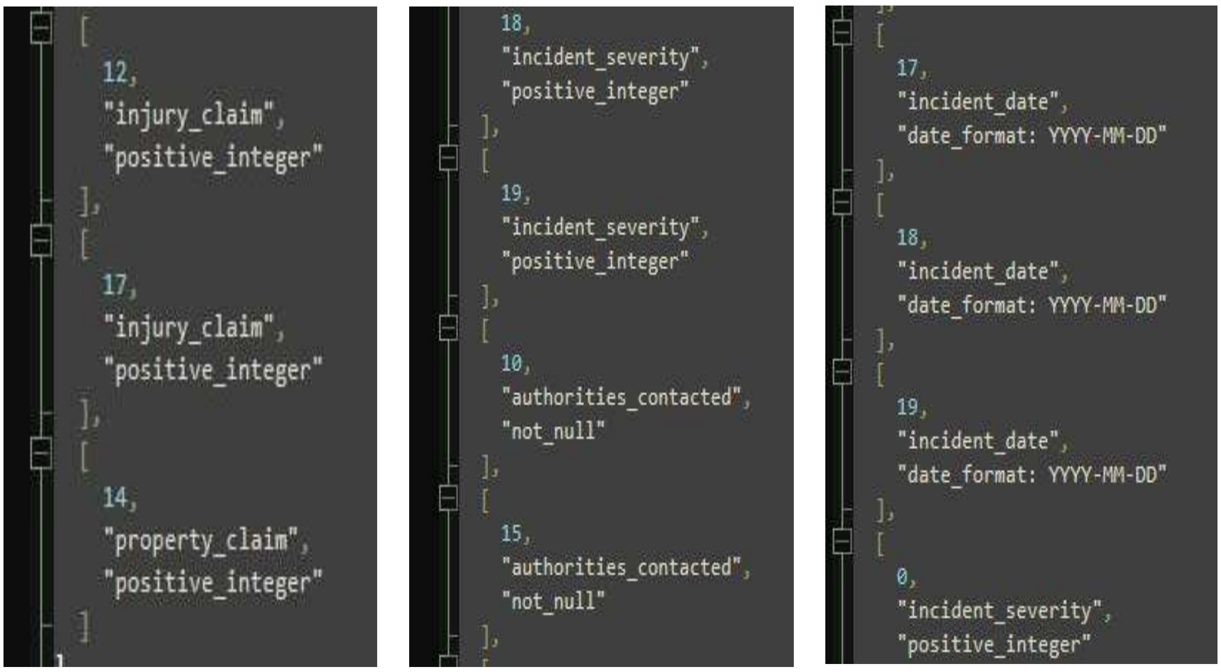
Agent-Based Validation Flow for Auto Insurance Claims Dataset
Raw Auto Insurance Claims CSV
40+ fields including: policy_number incident_date, total_claim_amount, fraud_reported, age, months_as_customer, insured_zip, incident_severity
Phase 1: Schema Assessment
ProfilerAgent
Schema Snapshot Generated
age, total_claim_amount: positive integers | incident_date: date field | fraud_reported: categorical
Phase 2: Rule Generation
LLMRuleAgentOllama
YAML Validation Rules Generated
• not_null: fraud_reported, incident_severity
• positive_integer: age, months_as_customer, total_claim_amount
• date_format: YYYY-MM-DD: incident_date, policy_bind_date
Phase 3: Rule Application
RuleAgent
Validation Issues Detected
• 400+ records: incident_date format violations(MM/DD/YYYY)
• Length mismatches: insured_zip
Error Logs with Metadata
Phase 4: Summarization & Diagnostics
SummaryAgent
Natural Language Summary
Date format inconsistencies flagged | Null value patterns identified
Actionable Insights & Recommendations
• ETL Logic Adjustments: Data parsing standardization
• Value Imputation: Age/claim amount missing data strategy
• Audit Trail: Transparent, explainable validation process

Figure 3. *Validation Flow over Auto Insurance Claims Dataset*



```
rules:
- column: id
  check: positive_integer
- column: months_as_customer
  check: positive_integer
- column: age
  check: positive_integer
- column: policy_number
  check: not_null
- column: policy_bind_date
  check: 'date_format: YYYY-MM-DD'
- column: policy_state
  check: length(2)
- column: policy_csl
  check: positive_integer
- column: policy_deductable
  check: positive_integer
- column: policy_annual_premium
  check: positive_integer
- column: umbrella_limit
  check: positive_integer
- column: insured_zip
  check: length(5)
- column: insured_sex
  check: not_null
- column: insured_education_level
  check: not_null
- column: insured_occupation
  check: not_null
- column: insured_hobbies
  check: not_null
- column: insured_relationship
  check: not_null
- column: capital-gains
```

Figure 4. *Generated Rules by LLM Rule Agent using llama 3.2*

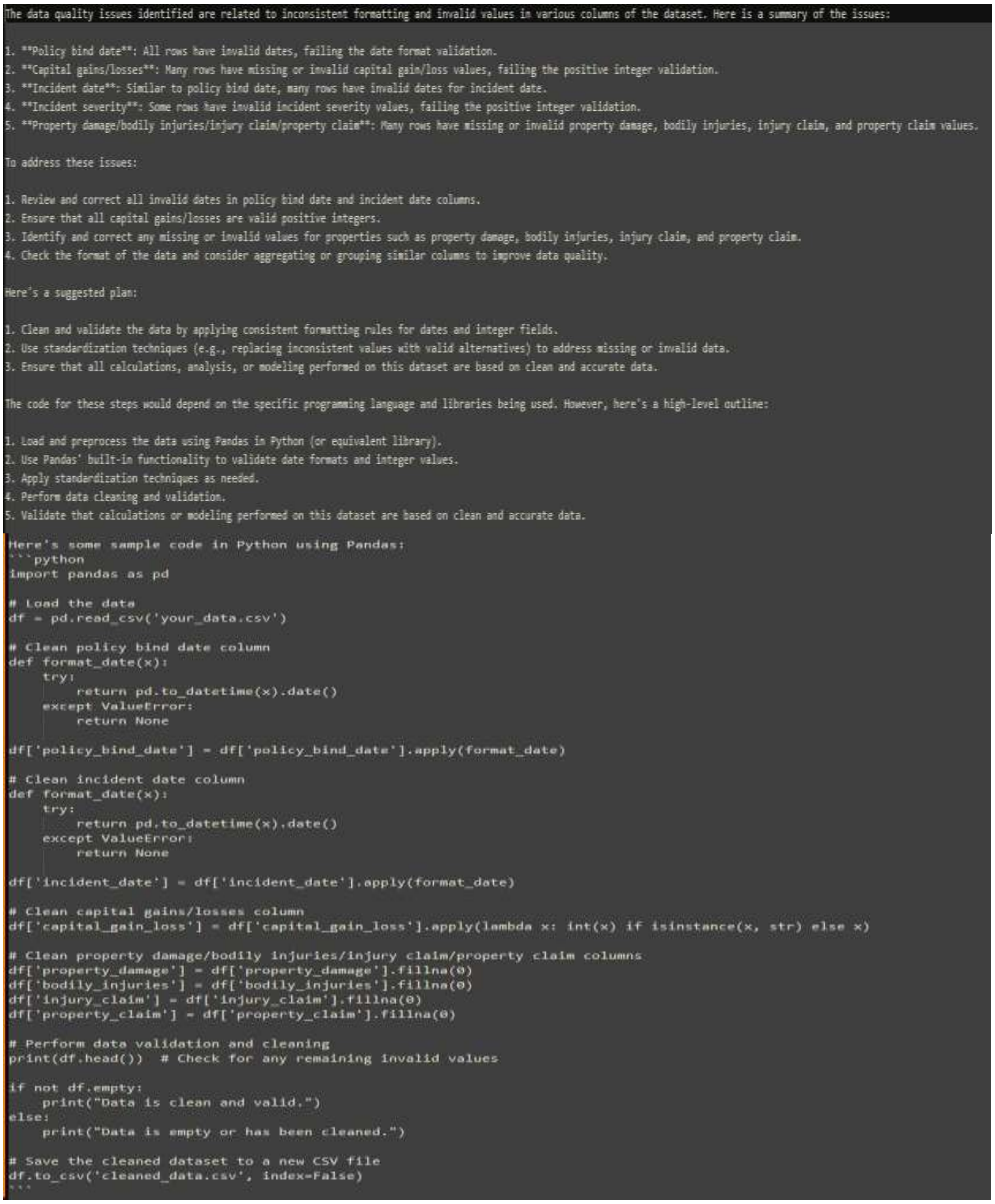


```
[
  12,
  "injury_claim",
  "positive_integer"
],
[
  17,
  "injury_claim",
  "positive_integer"
],
[
  14,
  "property_claim",
  "positive_integer"
]
]
```

```
  18,
  "incident_severity",
  "positive_integer"
],
[
  19,
  "incident_severity",
  "positive_integer"
],
[
  10,
  "authorities_contacted",
  "not_null"
],
[
  15,
  "authorities_contacted",
  "not_null"
],
[
```

```
],
[
  17,
  "incident_date",
  "date_format: YYYY-MM-DD"
],
[
  18,
  "incident_date",
  "date_format: YYYY-MM-DD"
],
[
  19,
  "incident_date",
  "date_format: YYYY-MM-DD"
],
[
  0,
  "incident_severity",
  "positive_integer"
```

Figure 5. *Structured JSON showing sample detected rule violation*



```
The data quality issues identified are related to inconsistent formatting and invalid values in various columns of the dataset. Here is a summary of the issues:

1. **Policy bind date**: All rows have invalid dates, failing the date format validation.
2. **Capital gains/losses**: Many rows have missing or invalid capital gain/loss values, failing the positive integer validation.
3. **Incident date**: Similar to policy bind date, many rows have invalid dates for incident date.
4. **Incident severity**: Some rows have invalid incident severity values, failing the positive integer validation.
5. **Property damage/bodily injuries/injury claim/property claim**: Many rows have missing or invalid property damage, bodily injuries, injury claim, and property claim values.

To address these issues:

1. Review and correct all invalid dates in policy bind date and incident date columns.
2. Ensure that all capital gains/losses are valid positive integers.
3. Identify and correct any missing or invalid values for properties such as property damage, bodily injuries, injury claim, and property claim.
4. Check the format of the data and consider aggregating or grouping similar columns to improve data quality.

Here's a suggested plan:

1. Clean and validate the data by applying consistent formatting rules for dates and integer fields.
2. Use standardization techniques (e.g., replacing inconsistent values with valid alternatives) to address missing or invalid data.
3. Ensure that all calculations, analysis, or modeling performed on this dataset are based on clean and accurate data.

The code for these steps would depend on the specific programming language and libraries being used. However, here's a high-level outline:

1. Load and preprocess the data using Pandas in Python (or equivalent library).
2. Use Pandas' built-in functionality to validate date formats and integer values.
3. Apply standardization techniques as needed.
4. Perform data cleaning and validation.
5. Validate that calculations or modeling performed on this dataset are based on clean and accurate data.
```

~~~
Here's some sample code in Python using Pandas:
```python
import pandas as pd

# Load the data
df = pd.read_csv('your_data.csv')

# Clean policy bind date column
def format_date(x):
    try:
        return pd.to_datetime(x).date()
    except ValueError:
        return None

df['policy_bind_date'] = df['policy_bind_date'].apply(format_date)

# Clean incident date column
def format_date(x):
    try:
        return pd.to_datetime(x).date()
    except ValueError:
        return None

df['incident_date'] = df['incident_date'].apply(format_date)

# Clean capital gains/losses column
df['capital_gain_loss'] = df['capital_gain_loss'].apply(lambda x: int(x) if isinstance(x, str) else x)

# Clean property damage/bodily injuries/injury claim/property claim columns
df['property_damage'] = df['property_damage'].fillna(0)
df['bodily_injuries'] = df['bodily_injuries'].fillna(0)
df['injury_claim'] = df['injury_claim'].fillna(0)
df['property_claim'] = df['property_claim'].fillna(0)

# Perform data validation and cleaning
print(df.head())  # Check for any remaining invalid values

if not df.empty:
    print("Data is clean and valid.")
else:
    print("Data is empty or has been cleaned.")

# Save the cleaned dataset to a new CSV file
df.to_csv('cleaned_data.csv', index=False)
```
~~~

Figure 6. *Sample Validation Output*

8. Conclusion

This paper introduces a hybrid Agentic AI system by integrating rule-based verification methods with the capabilities of large language models (LLMs) to facilitate data quality assurance for Property & Casualty (P&C) insurance. The system, based on an agent-based framework, modularly assigns work such as profiling, rule creation, enforcement, and summarization to autonomous agents that communicate through structured inputs and outputs. This approach guarantees transparency, traceability, and extensibility, so the system can provide dataset-specific validation rules automatically and parse results with little human intervention. Experiments on a real-world auto insurance claims dataset showed that the system can identify schema violations, semantic errors, and formatting errors in resource-constrained, on-premises settings with lightweight local LLMs such as LLaMA. SummaryAgent improved explainability through the process of transforming technical details into actionable layman narratives to guide business stakeholders and decision makers. The agentic design enables plug-and-play optimization of each component independently without pipeline breakage and ensuring scalability and flexibility as insurance data passes through. Despite limitations like LLM hallucinations and YAML parsing sensitivity, this Agentic AI system presents a strong case for coupling automation with regulation. It provides a solid basis for explainable, privacy-oriented, and domain-aware data quality in governed enterprise where responsibility and adaptability are paramount.

Author Statements:

- **Ethical approval:** The conducted research is not related to either human or animal use.
- **Conflict of interest:** The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper
- **Acknowledgement:** The authors declare that they have nobody or no-company to acknowledge.
- **Author contributions:** The authors declare that they have equal right on this paper.
- **Funding information:** The authors declare that there is no funding to be acknowledged.
- **Data availability statement:** The code and experimental data used in this paper are available on GitHub https://github.com/malviyashreekant/DQ_AGENT. The Auto Insurance Claims dataset sourced from Kaggle https://www.kaggle.com/datasets/buntyshah/auto-insurance-claims-data

References

[1] C. R. O. Forum, "Data Quality in the Insurance sector - Stocktaking and proposed way forward," The CRO Forum. Accessed: Jul. 17, 2025. [Online]. Available: https://thecroforum.org/data-quality-in-the-insurance-sector/

[2] G. Hall, M. Jones, K. Madigan, and S. Zheng, "DATA QUALITY MANAGEMENT IN THE P&C INSURANCE SECTOR".

[3] M. K. Borowicz, "The data quality problem (in the European Financial Data Space)," *Int. J. Law Inf. Technol.*, vol. 32, p. eaae015, Jun. 2024, doi: 10.1093/ijlit/eaae015.

[4] B. M. V. Bernardo, H. S. Mamede, J. M. P. Barroso, and V. M. P. D. dos Santos, "Data governance & quality management—Innovation and breakthroughs across different fields," *J. Innov. Knowl.*, vol. 9, no. 4, p. 100598, Oct. 2024, doi: 10.1016/j.jik.2024.100598.

[5] A. M. Astobiza, "The role of LLMs in theory building," *Soc. Sci. Humanit. Open*, vol. 11, p. 101617, Jan. 2025, doi: 10.1016/j.ssaho.2025.101617.

[6] A. Rath, "Structured Prompting and Feedback-Guided Reasoning with LLMs for Data Interpretation," May 03, 2025, *arXiv*: arXiv:2505.01636. doi: 10.48550/arXiv.2505.01636.

[7] M. P. J. van der Loo and E. de Jonge, "Data Validation," Dec. 21, 2020. doi: 10.1002/9781118445112.

[8] "Best Data Quality Tools for 2025: Top 10 Choices." Accessed: Jul. 14, 2025. [Online]. Available: https://www.adverity.com/blog/data-quality-tools/

[9] A. Groll, A. Khanna, and L. Zeldin, *A Machine Learning-based Anomaly Detection Framework in Life Insurance Contracts*. 2024. doi: 10.48550/arXiv.2411.17495.

[10] Kevin N. Shah, Sandip J. Gami, and Abhishek Trehan, "An Intelligent Approach to Data Quality Management AI-Powered Quality Monitoring in Analytics," *Int. J. Adv. Res. Sci. Commun. Technol.*, pp. 109–119, Dec. 2024, doi: 10.48175/ijarsct-22820.

[11] P. S. Dhoni, "Enhancing Data Quality through Generative AI: An Empirical Study with Data," Nov. 07, 2023, *Institute of Electrical and Electronics Engineers (IEEE)*. doi: 10.36227/techrxiv.24470032.

[12] D. Coquelin *et al.*, "Accelerating neural network training with distributed asynchronous and selective optimization (DASO)," *J. Big Data*, vol. 9, no. 1, Dec. 2022, doi: 10.1186/s40537-021-00556-1.

[13] The authors are with Universitas Indonesia, Indonesia, D. Maharani, H. Murfi, and Y. Satria, "Performance of Deep Neural Network for Tabular Data — A Case Study of Loss Cost Prediction in Fire Insurance," *Int. J. Mach. Learn. Comput.*, vol. 9, no. 6, pp. 734–742, Dec. 2019, doi: 10.18178/ijmlc.2019.9.6.866.

[14] S. Xie, "Improving Explainability of Major Risk Factors in Artificial Neural Networks for Auto Insurance Rate Regulation," *Risks*, vol. 9, no. 7, p. 126, Jul. 2021, doi: 10.3390/risks9070126.

[15] "Algomox Blog | Hybrid Approach: Pairing Threshold-Based Rules with AI/ML Techniques." Accessed: Jul. 17, 2025. [Online]. Available: https://www.algomox.com/resources/blog/hybrid_approach_pairing_threshold_based_rules_with_ai_ml_techniques

[16] "Why a Rules Based Plus a Machine Learning Hybrid Approach - 2021 | 1Spatial." Accessed: Jul. 17, 2025. [Online]. Available: https://1spatial.com/news/why-a-rules-based-plus-a-machine-learning-hybrid-approach-2021

[17] J. Oloyede and J. Owen, "Enhancing Data Quality and Integrity with AI: A Deep Learning Perspective Author: Joseph Oluwaseyi, Fajinmi

John," Feb. 19, 2025, *Social Science Research Network, Rochester, NY*: 5144205. doi: 10.2139/ssrn.5144205.

[18] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa, "Large Language Models are Zero-Shot Reasoners," Jan. 29, 2023, *arXiv*: arXiv:2205.11916. doi: 10.48550/arXiv.2205.11916.

[19] I. P. Carrascosa, "Zero-Shot and Few-Shot Learning with Reasoning LLMs," MachineLearningMastery.com. Accessed: Jul. 17, 2025. [Online]. Available: https://machinelearningmastery.com/zero-shot-and-few-shot-learning-with-reasoning-llms/

[20] G. Tao, "Autonomous AI-powered Anomaly Detection using Timeplus and DeepSeek-R1," Timeplus. Accessed: Jul. 17, 2025. [Online]. Available: https://www.timeplus.com/post/ai-anomaly-detection-deepseek

[21] "LLM-Powered Test Case Generation: Enhancing Coverage and Efficiency." Accessed: Jul. 17, 2025. [Online]. Available: https://www.frugaltesting.com/blog/llm-powered-test-case-generation-enhancing-coverage-and-efficiency

[22] "What are Models Thinking about? Understanding Large Language Model Hallucinations through Model Internal State Analysis." Accessed: Jul. 17, 2025. [Online]. Available: https://arxiv.org/html/2502.13490v1

[23] Tamanna, "Understanding LLM Hallucinations. Causes, Detection, Prevention, and Ethical Concerns," Medium. Accessed: Jul. 17, 2025. [Online]. Available: https://medium.com/@tam.tamanna18/understanding-llm-hallucinations-causes-detection-prevention-and-ethical-concerns-914bc89128d0