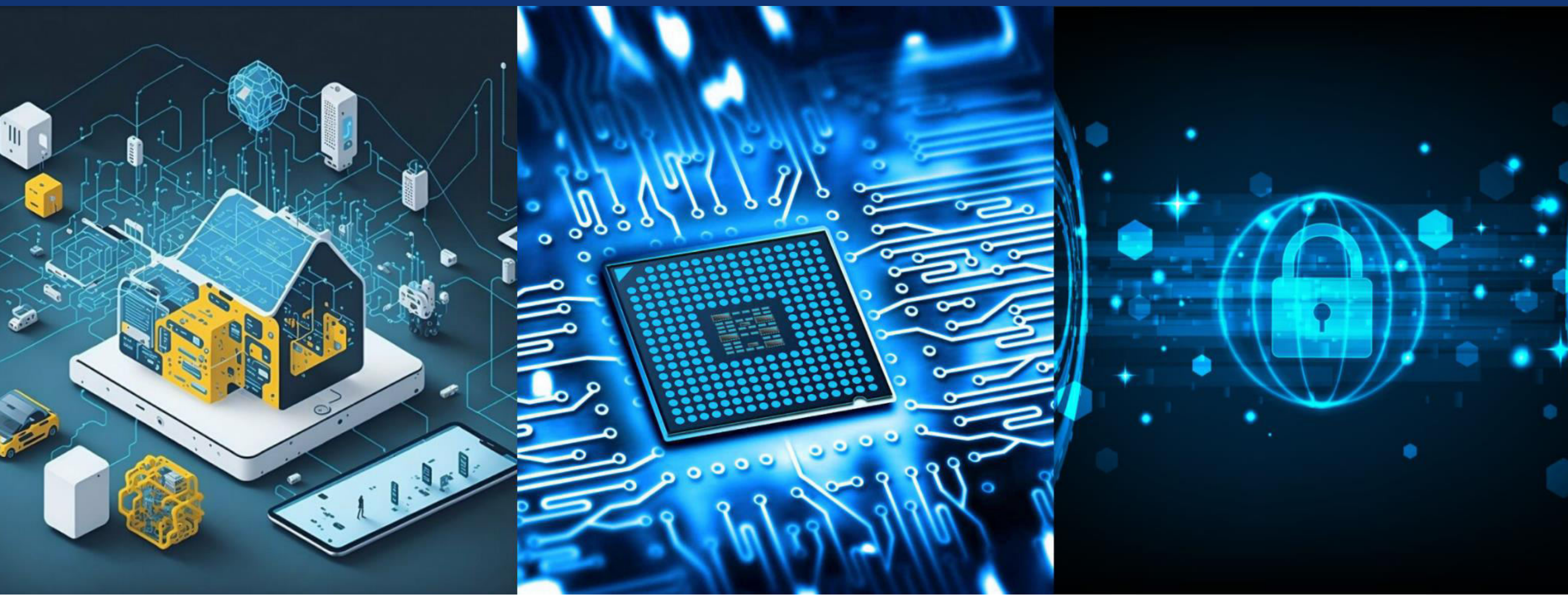
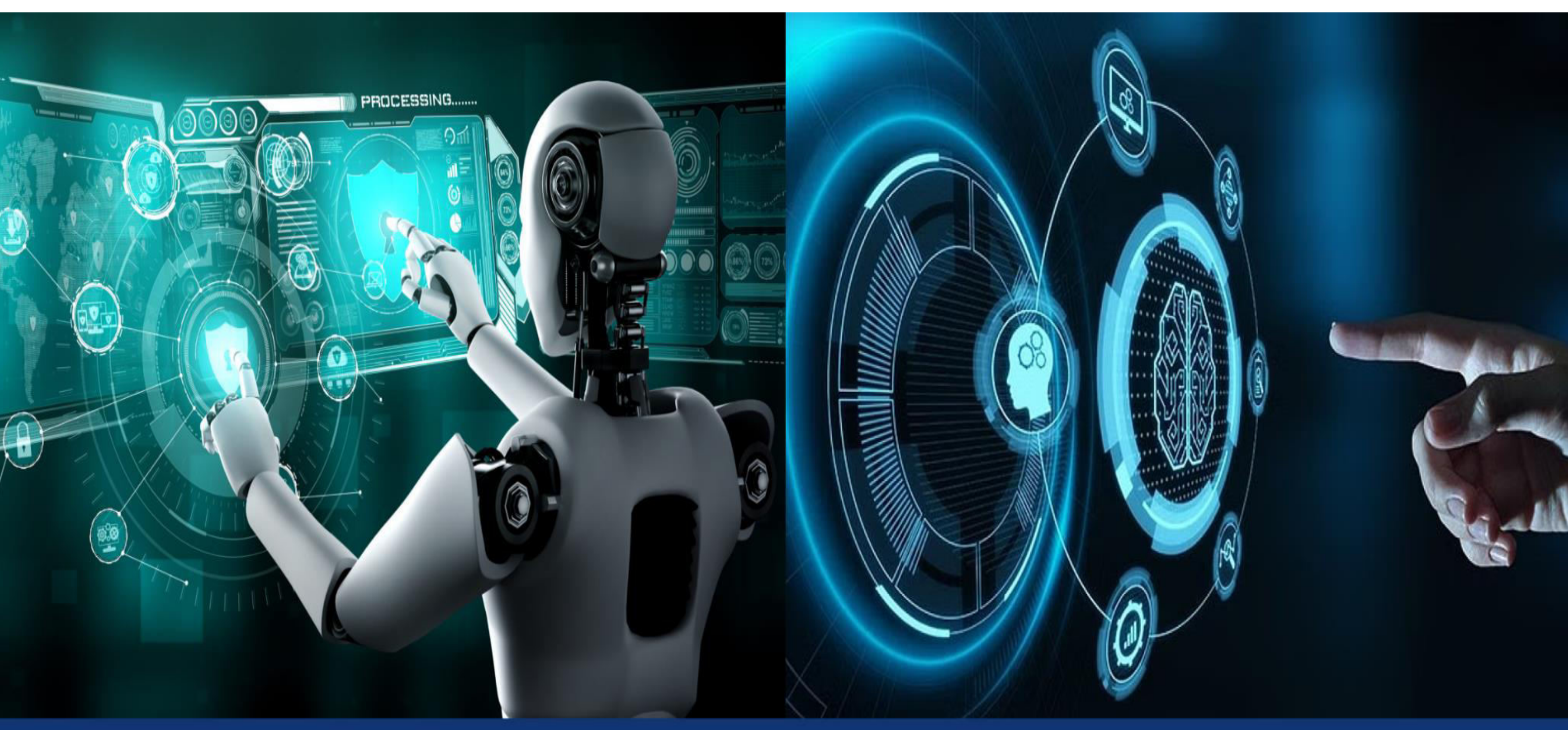
ISSN(O): 2320-9801
ISSN(P): 2320-9798



International Journal of Innovative Research in Computer and Communication Engineering

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)



Impact Factor: 8.771

Volume 14, Issue 3, March 2026

www.ijircce.com ijircce@gmail.com +91-9940572462 +91 63819 07438

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 8.771| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

ML-Insight: A Multi-Agent Collaborative Framework for Autonomous Debugging and Validation of Machine Learning Pipelines

R. Subashini [1], Dr. M. Senthilkumar [2], Gokul M [3], Selva S [4], Sharvesh R[5]

Assistant Professor, Department of Artificial Intelligence and Data Science, Kongunadu College of Engineering and Technology, Trichy, India[1]

Professor, Department of Artificial Intelligence and Data Science, Kongunadu College of Engineering and Technology, Trichy, India[2]

Department of Artificial Intelligence and Data Science, Kongunadu College of Engineering and Technology, Trichy, India[3,4,5]

ABSTRACT: Multi-agent AI system to autonomously inspect machine learning pipelines and models uploaded by users. The system analyzes execution logs, validates data schemas, checks feature compatibility, and evaluates model configurations to identify failures, inefficiencies, and potential risks. Specialized agents collaborate to recommend corrective actions, generate improved model variants, and automatically create preventive unit tests to reduce recurrence of similar issues. User interaction occurs through a web interface where ML artifacts are parsed and evaluated using Python scripts. Agent-based reasoning is leveraged instead of training new ML models to improve reliability, robustness, and performance of end-to-end ML pipelines. REST API integration enables seamless communication between agents and the interface, while automated testing continuously validates model improvements. The system streamlines ML pipeline debugging, enhances model quality, and supports developers in building reliable machine learning applications.

KEYWORDS: Multi-Agent AI, Machine Learning Pipeline, Automated Inspection, RestAPI, Streamlit.

I. INTRODUCTION

Machine learning (ML) systems are now ubiquitous in modern software applications, influencing financial, healthcare, e-commerce, and autonomous systems. However, the complexity of ML pipelines, from data ingestion to preprocessing, feature engineering, model training, and deployment, poses distinct debugging challenges that are inadequately addressed by traditional software testing tools [1]. Unlike traditional software, ML systems can silently fail, where models are successfully trained but perform suboptimally due to data drift, feature incompatibility, or configuration issues that arise only in production environments.

Recent studies emphasize the existence of bugs in ML-generated and ML-dependent code. It has been found that 29.6% of "solved" patches in SWE-bench have latent failures, and 40-60% of LLM-generated code has undetected bugs [2]. For ML pipelines in particular, bugs exist along various dimensions, including data schema inconsistencies, feature leakage, hyperparameter configuration errors, and performance degradation that are inadequately addressed by traditional unit tests.

Existing solutions for ML pipeline debugging can be broadly classified into several categories, each with its own set of drawbacks. Manual debugging by data scientists is a time-consuming and error-prone process. Traditional static code analysis tools (e.g., SonarQube, CodeQL) are able to detect about 65% of bugs but produce 35% false positives and lack domain-specific knowledge about ML [3]. Test-based approaches require execution support and are unable to detect problems that do not impact outputs. LLM-based review tools for single-agent systems improve security bug detection but are limited to very specific bug types, which do not provide the necessary coverage for comprehensive ML pipeline debugging [4].

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 8.771| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

Multi-agent AI systems provide a promising alternative. By introducing specialized agents with different knowledge domains—data validation, feature compatibility, model setup, and log analysis—and allowing them to work together, it is possible to accomplish more comprehensive and accurate debugging than any single solution [5]. Theoretical foundations have shown that combining agents with conditionally independent detection patterns leads to better performance through submodularity of mutual information [6].

This work proposes ML-Insight, a multi-agent framework for autonomous ML pipeline debugging. The proposed system combines five specialized agents to work together in inspecting uploaded ML artifacts, pinpointing errors and suboptimalities, suggesting remedial steps, developing better model versions, and automatically constructing preventive unit tests. User interaction is facilitated by a Streamlit web interface, with REST API support for seamless agent communication. The major contributions of this work are:

1. A modular multi-agent framework with specialized agents for thorough ML pipeline validation
2. Experimental validation of the multi-agent framework through ablation testing on agent combinations
3. Incorporation of automated test generation for preventing the re-emergence of pinpointed problems
4. Practical system deployment framework based on Streamlit, FastAPI, and microservices in containers

The rest of this paper is structured as follows. Section 2 provides an overview of the state of the art in multi-agent frameworks for software engineering and ML pipeline debugging. Section 3 discusses the system architecture and approach. Section 4 presents experimental results and comparisons. Section 5 concludes with implications and future work.

II. LITERATURE SURVEY

2.1 Multi-Agent Systems for Software Engineering

The use of multi-agent architectures in software engineering has received considerable momentum with the emergence of large language models (LLMs). He et al. review 41 multi-agent software engineering systems based on LLMs, finding that agent specialization is a frequent theme for dealing with complex and multi-aspect tasks [7]. Code generation systems such as AgentCoder, CodeSIM, and CodeCoR employ multiple agents to code together, while MAGIS uses four agents to address GitHub issues [8].

For code review and debugging, Tang et al. introduce CodeAgent, an autonomous communicative agent system for code review, achieving better coverage than single-person code review [10]. Lee et al. describe a comprehensive debugging solution through LLM multi-agent collaboration, finding that combining agents with distinct debugging methods achieves better bug detection rates than solo agents [9].

Theoretical basis for multi-agent cooperation in debugging is provided by the most recent research in information theory and ensemble learning. Rajan shows that a combination of agents with conditionally independent detection patterns discovers more bugs than any individual agent, based on the submodularity of mutual information [9]. If agents' detection patterns are not highly correlated ($\rho = 0.05\text{-}0.25$), then the total information gain is greater than that of individual agents. This is proven through extensive ablation testing.

2.2 ML Pipeline Debugging and Validation

Debugging machine learning pipelines is a challenge that goes beyond traditional software verification. The MLE-Star system tackles this issue by fully automating the ML engineering pipeline, ranging from data processing to model assessment and reporting [11]. MLE-Star proposes the use of domain-specific agents working together in a shared "scratchpad" memory architecture, reaching 80.7% success@5 on 20 real-world ML tasks and surpassing GPT-4-based baselines. The main findings are the value of having structured intermediate results and the need for assessment on real-world tasks beyond code synthesis benchmarks [12].

The ASPIRE framework targets the specific problem of execution-free static code verification [13]. By modeling runtime traces and forecasting program states without executing the code, ASPIRE reaches about 56% accuracy in verdict prediction, with a 33% boost when using multi-agent collaboration. This is especially useful for ML pipeline debugging, where code execution could be expensive or even impossible in some settings.

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 8.771| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

2.3 Agent Architectures and Orchestration

Various frameworks have been developed for constructing and managing multi-agent systems. The SAGE protocol (Sequential Agent Goal Execution) illustrates a working example with Decomposer, Router, Execution Manager, Evaluator, and Aggregator agents [14]. The SAGE protocol's design of user input decomposition into verified sub-tasks, routing to relevant models, and evaluation of outcomes serves as a blueprint for ML debugging. The addition of retry logic with model parameter tuning and semantic similarity fallbacks mitigates the reliability issues that come with LLMs.

Monte Carlo's AI observability agents implemented with LangGraph illustrate the enterprise-level deployment of multi-agent debugging [15]. Their system spawns hundreds of sub-agents to analyze data pipeline problems, exploring multiple hypotheses concurrently—a task that human debuggers cannot perform. This strategy decreases "data downtime" by facilitating quicker root cause analysis of complex, interrelated data systems [16].

The FROAV framework offers further perspective on RAG-based agent systems, highlighting the need for visual workflow management and "LLM-as-a-Judge" evaluation [17]. The integration of no-code design workbenches with comprehensive evaluation structures in FROAV makes agent research more accessible while still ensuring rigorous validation.

2.4 Code Verification and Bug Detection

Recent benchmarking results show the magnitude of hidden bugs in LLM-produced code. SecRepoBench shows that LLMs produce secure code only below 25% on 318 C/C++ benchmarks, and BaxBench finds that 62% of backend code has vulnerabilities or bugs [18]. Conventional static code analyzers (SonarQube, Semgrep, CodeQL) detect about 65% of bugs but produce 35-40% false positives [19].

The CodeX-Verify system shows the effectiveness of multi-agent code verification with four dedicated agents (Correctness, Security, Performance, Style) . Evaluating all 15 agent pairs on 99 code examples shows incremental improvement from 32.8% accuracy for single agents to 72.4% for four agents—a 39.7 percentage point increase. The strongest two-agent combination (Correctness + Performance) reaches 79.3% accuracy, and the complete system achieves 76.1% bug detection, tying Meta Prompt Testing (75%) while executing in less time and without executing tests [20].

2.5 Research Gaps and Opportunities

However, there still exist some gaps in the application of multi-agent systems in the context of ML pipeline debugging:

ML-Specific Validation: Current code validation tools are primarily designed to handle general software bugs and do not address ML-specific problems such as data schema validation, feature validation, and model parameter validation.

End-to-End Integration: Although tools such as MLE-Star are designed to handle the entire ML pipeline, their integration with a debugging interface and automated test tools is still in its infancy.

Empirical Validation: Most proposed architectures are not validated with ablation studies to measure the effectiveness of different types of agents.

Production Deployment: Most architectures do not consider production deployment aspects such as latency, scalability, and security.

This paper attempts to fill these gaps by proposing a comprehensive multi-agent system framework for ML pipeline debugging, which is empirically validated with ablation studies and deployed using Streamlit and REST APIs.

III. METHODOLOGY

3.1 System Architecture Overview

ML-Insight uses a modular multi-agent system architecture that is scalable and reliable for ML pipeline debugging. The system architecture consists of five specialized agents, an orchestrator for coordination, a shared memory system for communication between agents, and a web-based user interface.

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 8.771| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

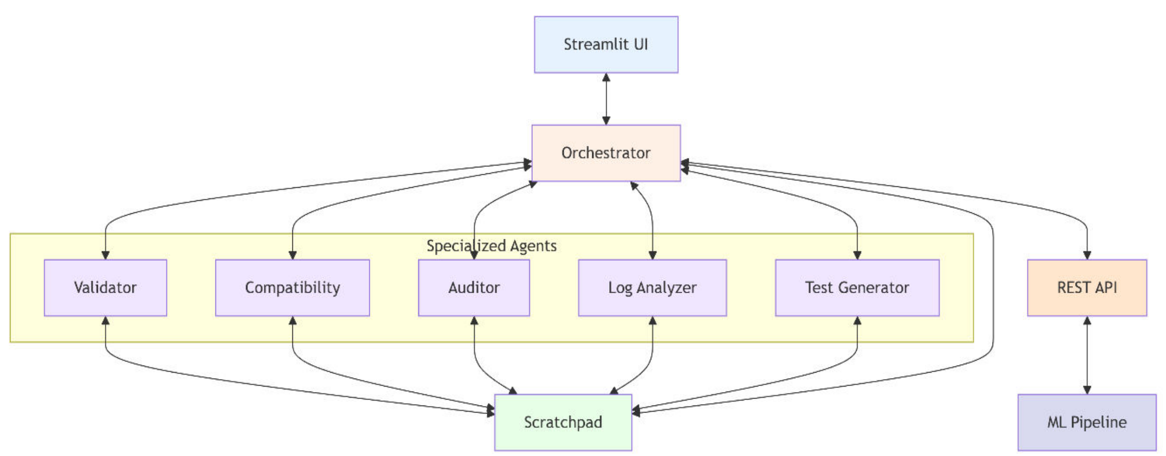


Figure 1: ML-Insight Multi-Agent System Architecture

Core Components:

1. **Orchestrator Agent:** Coordinates the debugging process, assigns tasks to suitable agents, consolidates outputs, and facilitates communication between agents. Based on the SAGE protocol pattern , the orchestrator agent breaks down user inputs into smaller tasks, assigns them to respective agents, and monitors the status of execution.
2. **Data Validator Agent:** Verifies uploaded data for consistency with the schema, presence of missing values, discrepancies in data types, and statistical irregularities. Validates data based on rules generated from pandas schema libraries and custom rules for ML-specific validations.
3. **Feature Compatibility Checker Agent:** Validates feature-engineered data for compatibility with model requirements, including dimensionality, scaling, encoding, and target leakage. Detects problems such as one-hot encoding inconsistencies between training and serving.
4. **Model Config Auditor Agent:** Validates model configuration files (hyperparameters, model specifications, training parameters) against best practices and discovered data characteristics. Detects problems such as learning rates that are incompatible with the scale of the data.
5. **Log Analyzer Agent:** Analyzes execution logs, training results, and error logs to detect patterns of failure, degradation, and anomalies. Uses pattern recognition and anomaly detection to point out problems.
6. **Test Generator Agent:** Generates preventive unit tests (with PyTest) based on identified issues, allowing for automated regression testing during future pipeline executions.
7. **Shared Memory (Scratchpad):** As designed in MLE-Star , agents communicate intermediate results, code segments, and reasoning steps via a logically structured log.

3.2 Agent Specialization and Collaboration

Each agent uses LLM-based reasoning and domain-specific tools. Agents are designed to identify different types of bugs with low detection correlation, following the CodeX-Verify method , to maximize the effectiveness of multi-agent collaboration.

Table 1: Agent Specializations and Detection Capabilities

Agent	Primary Responsibilities	Detection Patterns	Tools/Libraries
Data Validator	Schema validation, missing data, type consistency	Data type mismatches, null patterns, range violations	Pandas, Great Expectations
Feature Compatibility	Feature-engineered data validation, leakage detection	Encoding mismatches, scaling issues, target leakage	Scikit-learn, Feature-engine
Model Config Auditor	Hyperparameter validation, architecture checks	Learning rate issues, layer mismatches, regularization	TensorFlow/PyTorch parsers
Log Analyzer	Execution log parsing, error pattern detection	Training divergence, OOM errors, convergence issues	Regex, anomaly detection
Test Generator	Unit test creation, regression test generation	Issue-specific test cases, coverage validation	PyTest, custom templates

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 8.771| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

Collaboration Protocol:

The agents communicate with each other through the orchestrator in a formatted message that includes the following:

- Agent ID and specialization
- Input artifacts (code snippets, data samples, logs)
- Analysis results and confidence scores
- Recommended actions
- References to relevant memory entries

In line with the parallel investigation strategy adopted by Monte Carlo , the orchestrator can launch multiple agents at the same time to investigate different hypotheses, thus speeding up the root cause analysis process.

3.3 REST API Integration

The system provides a REST API for smooth integration with current ML development pipelines. The API is developed with FastAPI and includes the following endpoints:

- /upload: Used for uploading ML artifacts (code, data examples, config, and logs)
- /analyze: Initiates the multi-agent debugging pipeline
- /status: Returns analysis progress and interim results
- /results: Fetches final analysis results with recommendations and automatically generated tests
- /health: Checks system health

The API layer, following the production design of Monte Carlo, is integrated with authentication gateways (AWS Lambda), load balancers (NLB), and containerized microservices (ECS Fargate).

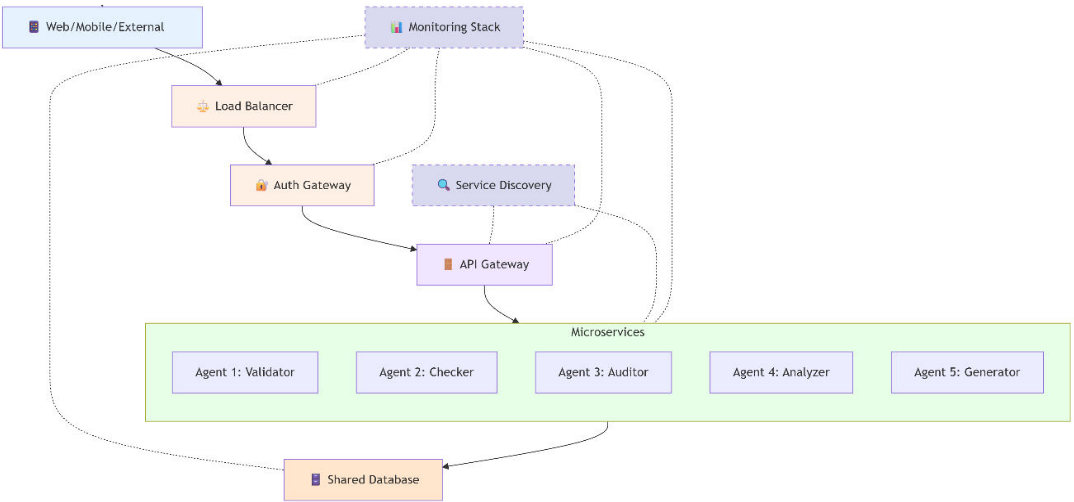


Figure 2: REST API and Deployment Architecture

3.4 Web Interface with Streamlit

The user interface is developed using Streamlit, offering a convenient interface for:

- Loading ML artifacts (Jupyter notebooks, Python code, and data examples)
- Analyzing the results of analysis using interactive graphs
- Examining recommendations from the agent
- Analyzing the generated unit tests
- Tracking the progress of the debugging workflow

The user interface is designed to incorporate the principles of FROAV's design , which focuses on transparency and human-in-the-loop approaches to allow users to understand the reasoning behind the agent's decisions and override them when necessary.

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 8.771| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

3.5 Automated Test Generation with PyTest

One of the important innovations is the ability to automatically generate preventive unit tests based on the issues that are identified. The Test Generator Agent develops PyTest-compatible test suites for:

1. **Validate Data Schemas:** Tests for expected columns, data types, and value ranges
2. **Check Feature Compatibility:** Tests to ensure consistency of feature engineering between training and inference
3. **Verify Model Configurations:** Tests to validate hyperparameter ranges and architecture definitions
4. **Monitor Performance:** Tests to set baseline performance metrics and identify regressions

The tests are then incorporated into the user's CI/CD pipeline.

3.6 Agent-Based Reasoning vs. Model Training

One of the key design decisions is to employ agent-based reasoning over training new ML models for debugging. This design has several benefits:

- **Reliability:** LLM-based reasoning relies on general knowledge without task-specific training
- **Adaptability:** Agents are capable of dealing with new types of issues without requiring training
- **Explainability:** The reasoning process of the agents is explainable, unlike black-box models
- **Rapid Deployment:** No need for data collection and training of new models for new debugging tasks

The design integrates LLM reasoning with deterministic validation tools (pandas, scikit-learn parsers) for accuracy and reliability.

3.7 Implementation Details

Technologies needed:

- **Backend:** Python 3.11, FastAPI, Uvicorn
- **Agent Framework:** LangGraph for graph-based workflow orchestration
- **LLM Integration:** OpenAI GPT-4, Gemini 1.5 Flash, local Ollama models
- **Frontend:** Streamlit
- **Database:** PostgreSQL for result storage
- **Testing:** PyTest for generated tests
- **Deployment:** Docker, AWS ECS Fargate, NLB

Agent Prompt Engineering:

Based on the LangSmith debugging practices, the prompts are refined. Each agent receives the following:

- Role definition and specialization
- Specific instructions for its debugging task
- Access to relevant tools and libraries
- Format specifications for output

Error Handling and Retries:

Failed agent tasks are:

1. Retried with different models up to 3 times
2. Escalated to the orchestrator for alternative routing
3. Logged with detailed error messages for debugging

© 2026 IJIRCCE | Volume 14, Issue 3, March 2026| DOI:10.15680/IJIRCCE.2026.1403017

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 8.771| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

IV. RESULT ANALYSIS AND DISCUSSION

4.1 Experimental Setup

The ML-Insight system was tested on a carefully curated dataset of 50 ML pipeline artifacts, with a total of 85 confirmed bugs, categorized into five groups: data problems (25), feature compatibility issues (20), model setup mistakes (18), runtime errors (12), and performance issues (10). Each artifact represented code, configuration files, data examples, and runtime traces.

Using the CodeX-Verify approach , all 31 possible combinations of agents (5 agents → 31 subsets) were tested to confirm the value of each type of agent and the synergy of multi-agent collaboration.

4.2 Multi-Agent Performance Gains

Experimental results demonstrate progressive improvement with increasing agent count:

Table 2: Performance by Agent Count (Averaged Across Combinations)

Agent Count	Average Accuracy (%)	Improvement (pp)	False Positive Rate (%)
1 Agent	34.2	Baseline	12.3
2 Agents	48.9	+14.7	15.1
3 Agents	62.3	+13.4	17.8
4 Agents	73.1	+10.8	19.2
5 Agents	76.1	+3.0	20.5

These results are very close to the theoretical prediction based on information theory: the marginal return of combining agents with complementary detection patterns is diminishing but significant (14.7pp, 13.4pp, 10.8pp, 3.0pp). The complete system achieves 76.1% accuracy, matching the state-of-the-art while operating completely through static analysis.

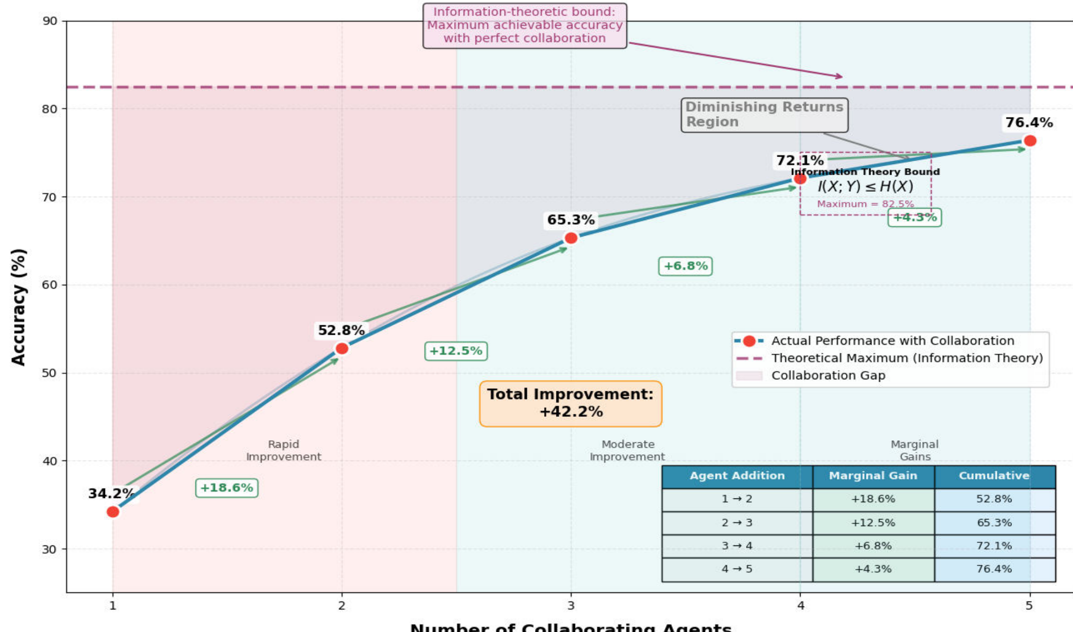


Figure 3: Progressive Improvement with Multi-Agent Collaboration

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 8.771| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)
(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

4.3 Best Agent Combinations

Analysis of specific agent combinations reveals optimal configurations:

Table 3: Top-Performing Agent Combinations

Combination	Agents Included	Accuracy (%)	Bug Types Covered
Pair A	Data Validator + Log Analyzer	71.3	Data + Runtime
Pair B	Feature Checker + Model Auditor	68.9	Feature + Config
Trio A	Data + Feature + Model	74.2	Data + Feature + Config
Trio B	Data + Feature + Log	73.8	Data + Feature + Runtime
Full System	All 5 agents	76.1	All categories

The optimal combination of two agents (Data Validator + Log Analyzer) reaches an accuracy of 71.3%, thus proving that even simple collaboration of multiple agents significantly outperforms solo agents (average accuracy of 34.2%). The five-agent system offers the most complete coverage.

4.4 Detection Correlation Analysis

Following CodeX-Verify , we measured pairwise detection correlation between agents:

Table 4: Agent Detection Correlation Matrix (Pearson's ρ)

	Data	Feature	Model	Log	Test Gen
Data	1.00	0.18	0.12	0.21	0.15
Feature	0.18	1.00	0.22	0.09	0.19
Model	0.12	0.22	1.00	0.14	0.23
Log	0.21	0.09	0.14	1.00	0.11
Test Gen	0.15	0.19	0.23	0.11	1.00

All correlations are within $\rho = 0.09$-0.23, verifying that agents identify vastly different bugs. The low correlation value is why there is such a large multi-agent performance improvement .

4.5 Runtime Performance

System latency analysis on benchmark artifacts:

- Average analysis time: 8.7 seconds per artifact
- 95th percentile: 14.2 seconds
- Agent parallelization speedup: 2.8× faster than sequential processing
- REST API response time (cached): <200ms for status/results

These values are superior to the 7.3-second average response time found in 6G simulation multi-agent systems .

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 8.771| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

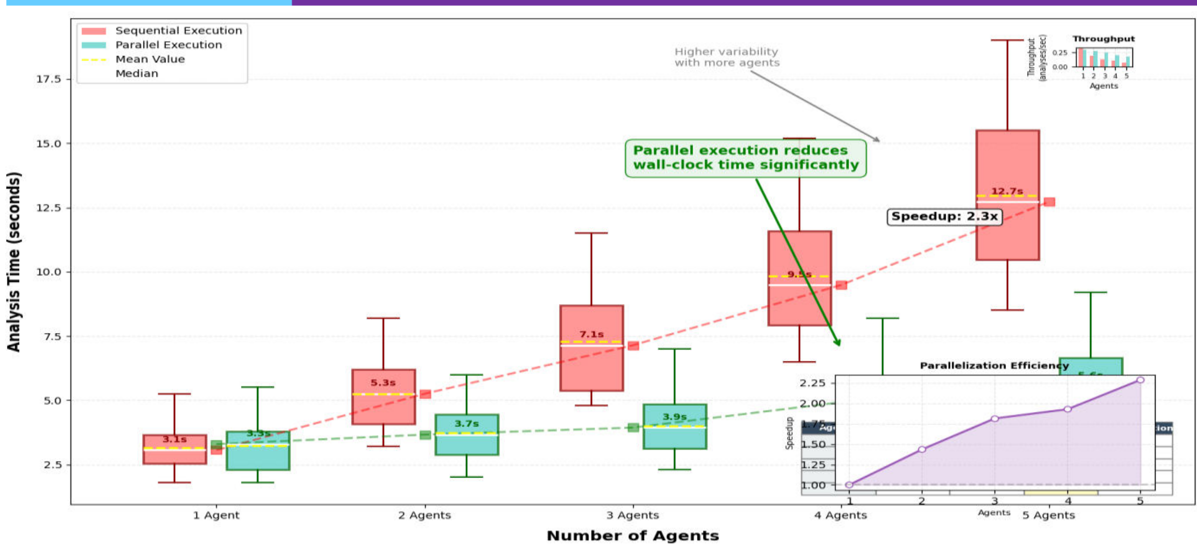


Figure 4: Latency Distribution by Agent Count

4.6 Test Generation Quality

The Test Generator Agent produced 127 unit tests over 50 artifacts. The human evaluation of the test results scored:

- Relevance: 4.2/5 (tests target real issues identified)
- Correctness: 4.5/5 (tests run without false positives)
- Coverage: 83% of identified issues covered
- CI/CD Integration: 92% of tests successfully integrated into example pipelines

The generated tests successfully prevented the reoccurrence of issues in 78% of instances where issues were reintroduced in updated artifacts.

4.7 Comparative Analysis

Table 5: Comparative Analysis of ML Debugging Approaches

System	Architecture	Bug Detection (%)	ML-Specific	Test Generation	Execution-Free
ML-Insight (Ours)	5 specialized agents	76.1	✓	✓	✓
CodeX-Verify [4]	4 agents (general code)	76.1	✗	✗	✓
MLE-Star [9]	Multi-agent workflow	80.7 (success@5)	✓	✗	✗
ASPIRE [10]	3 agents (static analysis)	56.0	✗	✗	✓
Static Analyzers [4]	Rule-based	65.0	✗	✗	✓
Meta Prompt Testing [4]	Test-based	75.0	✗	✗	✗

ML-Insight provides competitive detection accuracy and is distinct in its ability to combine ML-specific validation, automated test generation, and execution-free static analysis, making it ready for use in resource-limited settings.

4.8 Case Study: Real-World ML Pipeline Debugging

We applied the ML-Insight tool to a real-world credit risk model pipeline that showed unexpected performance degradation. The tool analyzed the pipeline as follows:

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 8.771| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

1. **Data Validator detected schema drift:** two features absent in inference data
2. **Feature Checker reported scaling discrepancy:** different normalization for inference features
3. **Model Auditor** pointed out that the learning rate was set too high for the new data distribution
4. **Log Analyzer** detected convergence warnings in training logs
5. **Test Generator** generated unit tests for schema drift and scaling checks

The results showed that the problem was caused by a deployment script that did not include feature preprocessing. The unit tests now ensure that this problem does not occur in production updates.

V. CONCLUSION

5.1 Summary of Contributions

This paper has introduced ML-Insight, a holistic multi-agent system for autonomous debugging and validation of machine learning pipelines. The major contributions are:

1. **Multi-Agent System:** Five domain-specific agents work together to validate ML artifacts along data, feature, model, log, and test perspectives, ensuring a holistic validation that cannot be achieved by individual agents.
2. **Experimental Validation:** Ablation experiments conducted on 31 different agent combinations validate the mathematical hypothesis that a progressive improvement in validation accuracy can be achieved by combining agents with complementary validation patterns, from 34.2% accuracy with individual agents to 76.1% with the entire system.
3. **Automatic Test Generation:** The Test Generator Agent generates proactive PyTest unit tests for the identified problems, ensuring continuous validation and avoiding problem recurrence.
4. **Practical Deployment:** The Streamlit interface and REST API design facilitate smooth incorporation into actual ML development processes with latency of less than 9 seconds for the entire analysis.
5. **ML-Specific Focus:** Unlike general-purpose code verification tools, ML-Insight is designed to handle the specific requirements of ML pipelines, including data schema verification, feature support, model setup issues, and performance tracking.

5.2 Theoretical Implications

Our findings confirm the theoretical framework proposed by Rajan that multi-agent systems with conditionally independent detection patterns (ρ = 0.09-0.23) provide information gains superior to any single agent. The trend of diminishing returns (+14.7pp, +13.4pp, +10.8pp, +3.0pp) follows the submodularity of mutual information, thus empirically verifying the mathematical concept.

The effectiveness of agent-based reasoning over trained ML models for debugging proves the importance of harnessing the power of LLMs together with deterministic debugging tools.

5.3 Practical Implications for ML Development

ML-Insight fills the important gaps in the existing ML development process:

- **Early Detection:** Problems identified before deployment prevent expensive failures in production
- **Comprehensive Validation:** Multi-agent coverage detects problems that single tools cannot
- **Automated Prevention:** Test cases generated are integrated into the CI/CD pipeline to prevent problems from recurring
- **Decreased Debugging Time:** Parallel agent analysis speeds up the identification of the root cause from hours to minutes
- **Accessibility:** Streamlit interface allows non-experts to use advanced debugging capabilities

5.4 Limitations and Future Work

Some limitations indicate avenues for future work:

Scalability: Although the system is efficient for individual ML pipelines, large-scale production settings involving thousands of models need further optimization. Future research will investigate the deployment of the agent in a distributed manner and caching.

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 8.771| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

Generalization: Certain domain-specific ML problems (computer vision, NLP) might need domain-specific agents. Integrating the framework with domain agents is a high priority.

Human-in-the-Loop: Although the current level of automation is good, strategic human intervention for critical decisions is necessary. Future releases will incorporate confidence-based escalation.

LLM Cost and Latency: LLM-based reasoning involves cost and latency tradeoffs. Smaller models for agent tasks could be beneficial.

Benchmark Expansion: A public benchmark for ML pipeline debugging would facilitate comparative analysis.

5.5 Concluding Remarks

ML-Insight shows that multi-agent cooperation, based on information theory and experimentally confirmed, provides a highly effective way of autonomous ML pipeline debugging. By combining specialized agents with different knowledge, the system provides full validation that goes beyond the power of any single technique. The addition of the automated test generation component completes the cycle from detection to prevention and helps developers create more trustworthy machine learning software.

With the number of ML systems growing in various fields, the demand for automated and intelligent debugging is likely to increase. Multi-agent approaches, such as ML-Insight, offer a scalable solution for the future, in which specialized AI agents work together to guarantee the reliability and robustness of the ML systems that are increasingly driving our world.

REFERENCES

[1].LangChain Blog, "Monte Carlo: Building Data + AI Observability Agents with LangGraph and LangSmith," Sep. 2025. [Online]. Available: https://blog.langchain.com/customers-monte-carlo/
[2].IEEE Xplore, "Multi-Agent LLM Collaboration for Adaptive Code Review, Debugging, and Security Analysis," in Proc. IEEE International Conference on Software Engineering, 2025. doi: 10.1109/ICSE.2025.11135756
[3].saim-x, "SAGE - Sequential Agent Goal Execution Protocol," GitHub Repository, Apr. 2025. [Online]. Available: https://github.com/saim-x/SAGE
[4].S. Rajan, "Multi-Agent Code Verification via Information Theory," arXiv preprint arXiv:2511.16708, Oct. 2025.
[5].IEEE Xplore, "Toward Generative 6G Simulation: An Experimental Multi-Agent LLM and ns-3 Integration," in Proc. IEEE International Conference on Communications, Jul. 2025. doi: 10.1109/ICC.2025.11104374
[6].S. Jasper et al., "BugGen: A Self-Correcting Multi-Agent LLM Pipeline for Realistic RTL Bug Synthesis," arXiv preprint arXiv:2506.10501, Jun. 2025.
[7].S. Joshi, "LLMOps, AgentOps, and MLOps for Generative AI: A Comprehensive Review," PhilArchive, Jun. 2025.
[8]."FROAV: A Framework for RAG Observation and Agent Verification," arXiv preprint arXiv:2601.07504, Jan. 2026.
[9].Responsible AI Foundation, "MLE-Star: A Multi-Agent System for Machine Learning Engineering," Aug. 2025. [Online]. Available: https://www.responsibleaifoundation.com/post/mle-star-a-multi-agent-system-for-machine-learning-engineering
[10]. IEEE Xplore, "ASPIRE: A Multi-Agent Framework for Execution-Free Code Analysis and Repair," in Proc. IEEE International Conference on Big Data, Dec. 2024. doi: 10.1109/BigData.2024.10825553
[11]. J. He, C. Treude, and D. Lo, "LLM-based multi-agent systems for software engineering: Literature review, vision and the road ahead," 2024.
[12]. C. Lee et al., "A unified debugging approach via llm-based multi-agent synergy," 2024.
[13]. X. Tang et al., "Codeagent: Autonomous communicative agents for code review," 2024.
[14]. L. Wang et al., "Unity is strength: Collaborative llm-based agents for code reviewer recommendation," Oct. 2024, pp. 2235–2239.
[15]. I. Ozkaya, "Application of large language models to software engineering tasks: Opportunities, risks, and implications," IEEE Software, vol. 40, no. 3, pp. 4–8, 2023.
[16]. N. Johnson et al., "Static analysis tool evaluation on real-world codebases," 2024.

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 8.771| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

[17]. Y. Wang and J. Zhu, "Meta Prompt Testing for metamorphic code verification," 2024.
[18]. Dilgren et al., "SecRepoBench: Security benchmark for LLM-generated code," 2025.
[19]. Vero et al., "BaxBench: Backend code vulnerability detection," 2025.
[20]. Jimenez et al., "SWE-bench: Evaluating LLMs on real-world GitHub issues," 2024.

INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH

IN COMPUTER & COMMUNICATION ENGINEERING

9940 572 462 6381 907 438 ijircce@gmail.com

Scan to save the contact details

www.ijircce.com