Agentic AI-Driven Cloud-Agnostic Pipeline for Scalable Model Training and Edge Deployment

Pratham Jain*
Dept. of Computer Science and Engineering
IIIT Raichur
Raichur, India
cs21b1021@iiitr.ac.in

Priyanka Singh
Dept. of Computer Science and Engineering
IIIT Raichur
Raichur, India
priyanka@iiitr.ac.in

Anandan Arumugam
Bosch Global Software Technologies
(SX/ECT2-MM)
Bengaluru, India
Anandan.Arumugam@in.bosch.com

Vipin Pulikkal
Bosch Global Software Technologies
(SX/ECT3-MM)
Bengaluru, India
Vipin.Pulikkal@in.bosch.com

***Abstract*—This paper presents a modular, cloud-agnostic machine learning pipeline that integrates agentic AI for automatic model configuration, Apache Spark on Delta Lake for large-scale data processing, PyCaret for low-code model training, and ONNX/TFLite for edge-ready deployment. We demonstrate how Microsoft AutoGen agents can propose and refine configurations at scale, enabling the training for different products with multiple business use case each having multiple target variables models in parallel. Our approach addresses challenges in IoT-driven products, data lineage, and edge constraints, and uses Delta Lake for robust logging, metadata handling, and incremental training support. The solution incorporates ST Edge AI Developer Cloud for cross-compilation and Renode for hardware-in-the-loop testing, bridging the gap between cloud-scale training and embedded deployment.**

***Index Terms*—Agentic AI, AutoGen, Delta Lake, Apache Spark, PyCaret, ONNX, TFLite, Edge Deployment, MLflow, IoT, Data Lineage, STM32 Cube.AI, Renode**

I. Introduction

The accelerating adoption of Internet of Things (IoT) devices has generated vast volumes of sensor and log data. Modern consumer and industrial products increasingly embed machine learning (ML) models for tasks such as anomaly detection, predictive maintenance, and quality inspection. While cloud-based centralized servers can support training and inference, there is a growing need to run models directly on edge devices to reduce latency, preserve privacy, and operate under intermittent connectivity [1].

The evolution of data infrastructure—from monolithic data warehouses to modern lakehouses—has paralleled this shift. Traditional **data warehouses** (e.g., Teradata, Snowflake) provided structured storage but struggled with unstructured data and scale. The rise of **data lakes** (e.g., Hadoop, AWS S3) enabled raw data storage but lacked ACID transactions and schema enforcement. The **data lakehouse** paradigm (e.g., Delta Lake, Apache Iceberg) emerged to unify these approaches, offering transactional consistency, schema evolution, and interoperability with ML frameworks [2]. Delta Lake, in particular, has become pivotal for ML pipelines due to its open table format, time-travel capabilities, and seamless integration with Apache Spark.

However, deploying and maintaining hundreds of ML models across diverse edge hardware poses significant challenges in data lineage, version control, and incremental retraining, particularly when device resources for storage and compute are limited.

II. Problem Statement

Organizations face the following pain points:

- **In-memory storage limitations:** Traditional pipelines load intermediate data and model artifacts in local memory, leading to challenges in reproducibility and operational risk when scaling.
- **Lack of open table formats:** Without a unified, versioned storage system, tracking schema evolution, merges, and metadata becomes error-prone.
- **High maintenance overhead:** Storing runs and transformations locally is expensive to manage, vulnerable to data loss, and hinders cross-team collaboration.
- **Cost and accessibility:** Relying on proprietary or monolithic databases increases costs, whereas object storage (e.g., S3, Azure Blob) offers inexpensive, highly available storage but lacks built-in ML metadata support.
- **Data lineage and drift:** Continuous streams from IoT devices require time-travel, upstream merge handling, and tracking of feature provenance to detect data drift and enable incremental retraining.
- **Edge deployment constraints:** Models must be quantized and optimized (e.g., via STM32 Cube.AI, NanoEdge AI) to meet RAM, flash, and inference-time constraints on embedded hardware.

To illustrate the scale, consider a business with 20 products, each having 5 use cases (classification, regression, time-series

forecasting, clustering, and computer vision) and 5 target variables per use case, resulting in 500 distinct model training tasks. Managing configurations, data pipelines, and model artifacts at this scale demands automated, robust, and cloud-agnostic solutions.

III. Proposed Solution

We propose a modular pipeline comprising:

A. Delta Lake on Object Storage

Replace in-memory and ad-hoc storage with Delta Lake in open table formats. Delta Lake provides:

- **Time-travel and versioning:** Query historical data and schema versions for reproducibility.
- **Upstream merges and metadata handling:** ACID transactions enable safe schema evolution and merge operations.
- **Scalable object storage:** Low-cost S3 or Azure Blob storage with Delta Lake allows centralized sharing and console/CLI access across teams [2].

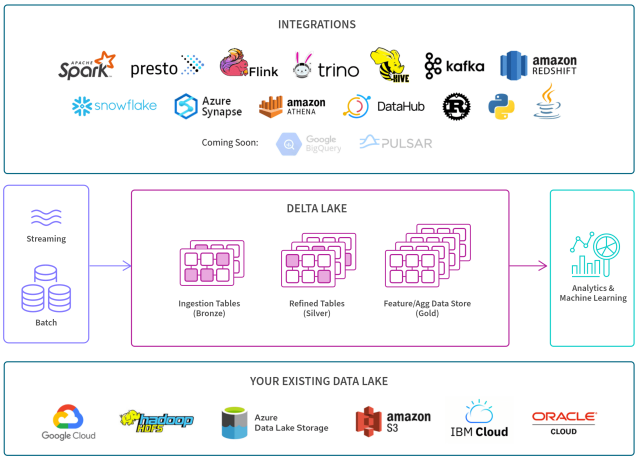

Fig. 1. Conceptual overview of Delta Lake architecture, showing data flowing from batch/streaming sources through Bronze (raw), Silver (refined), and Gold (aggregated/feature) tables for analytics and ML. Adapted from [14].

Delta Lake addresses the limitations of prior architectures (illustrated conceptually in **Figure 1**):

- **Data warehouses:** Rigid schemas, high costs, and poor support for ML workloads.
- **Data lakes:** Lack of transactions, inconsistent metadata, and "data swamp" risks.
- **Lakehouses:** Combines best of both worlds with schema enforcement, versioning, and ML-native features.

B. Apache Spark on Databricks

Use Apache Spark for distributed ETL and feature engineering:

- **Big data support:** Process large volumes via Spark SQL and DataFrame APIs.

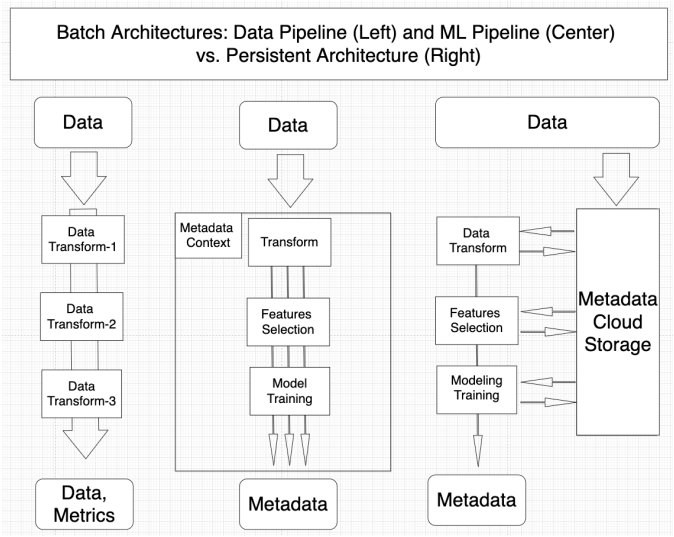

Fig. 2. Architectural comparison: Traditional Data Pipeline (Left) and ML Pipeline (Center) vs. Persistent Architecture with Cloud Storage (Right) used in our approach. The persistent architecture maintains metadata across pipeline stages, enabling reproducibility and efficient incremental updates.

- **Out-of-memory operations:** Efficiently handle data partitions and caching.
- **Incremental transformations:** Time-travel and MERGE INTO operations for upserts and incremental loads.
- **Interoperability:** Convert Spark DataFrames to Pandas using Apache Arrow for PyCaret training [3].

C. Agentic Configuration with AutoGen and Multi-Agent Frameworks

Leverage Microsoft AutoGen agents alongside LangChain and CrewAI to assist users in model configuration:

- **AI-driven configs:** Agents propose algorithms and hyperparameters based on product/use-case metadata [4]. LangChain enables dynamic chaining of LLM prompts (e.g., "Suggest hyperparameters for a time-series model"), while CrewAI orchestrates multi-agent workflows for collaborative refinement.
- **Human-in-the-loop:** Allow expert review and refinement via CLI or minimal UI. AutoGen's conversational agents enable iterative feedback loops.
- **Auditability:** Log all suggestions and final configurations in Delta tables. Tools like Hydra and OmegaConf enforce structured YAML/JSON schemas.

1) Agentic AI for Model Selection: To address the challenges of manual model selection, we integrate a specialized Model Selection Agent within our architecture:

- **Problem Statement:** Model selection traditionally requires significant manual effort from data scientists to experiment with various algorithms and hyperparameters. This process is time-consuming, inconsistent, and depends heavily on individual expertise. An autonomous agent can continuously evaluate and deploy optimal models without human intervention.

- **Agent Architecture:** The Model Selection Agent operates as a specialized component positioned between the Feature Store and Model Training components, maintaining:
  - *Model Registry:* A catalog of available model architectures (Random Forests, XGBoost, Neural Networks, etc.)
  - *Performance Evaluator:* Measures model performance using multiple metrics (AUC, precision/recall, F1-score)
  - *Decision Engine:* Uses reinforcement learning to optimize model selection based on historical performance
  - *Hyperparameter Tuner:* Automatically configures models for optimal performance
- **Execution Flow:** The agent receives training data, selects candidate models based on data characteristics, performs hyperparameter optimization using Bayesian approaches, evaluates across multiple metrics, and selects the best-performing model for deployment. Over time, it learns which models perform best for different data patterns [12].

2) Agentic AI for Feature Selection & Suggestion: Feature engineering remains a largely manual process requiring domain expertise. Our architecture incorporates an intelligent Feature Selection Agent:

- **Problem Statement:** Automatically identifying relevant features and suggesting new derived features can significantly improve model performance while reducing development time. This is particularly valuable across the 500 distinct model training tasks in our use case.
- **Agent Architecture:** The Feature Selection Agent integrates with our data pipeline through:
  - *Feature Evaluator:* Assesses feature importance using techniques like mutual information, SHAP values, and permutation importance [13]
  - *Feature Generator:* Creates new features through transformations, aggregations, and combinations
  - *Domain Knowledge Base:* Stores patterns of useful features for specific domains
  - *Optimization Controller:* Balances feature set size and model performance
- **Execution Flow:** The agent analyzes raw data and current feature sets, evaluates existing features for relevance, generates candidate feature transformations, tests these against performance metrics, and recommends features to retain, remove, or add. It continuously learns patterns of effective features across different use cases.

D. Model Training with PyCaret and Distributed Orchestration

Utilize PyCaret for low-code ML, coupled with Ray or Kubeflow for parallel execution:

- **Simplified API:** `setup()`, $\texttt{compare}_m odels(), and$

E. Edge Export and Optimization

Convert and optimize models for embedded deployment using ST Edge AI Developer Cloud and Renode:

- **Format conversion:** Export to ONNX or TFLite. STM32 Cube.AI optimizes models for STM32 microcontrollers via graph pruning, quantization, and memory allocation [8].
- **Quantization:** Apply 8-bit quantization via ONNX Runtime or TensorFlow Lite to minimize RAM and flash usage [6], [7]. The ST Edge AI Developer Cloud benchmarks models on real hardware (e.g., STM32H7) and generates deployment-ready C code.
- **Hardware testing:** Use Renode, an open-source simulation framework, to test models on virtual STM32 targets before flashing to physical devices. Renode supports cycle-accurate emulation of peripherals (e.g., sensors, GPIO) [9].
- **Integration with NanoEdge AI:** Further optimize models for inference-time constraints on microcontrollers.

F. Logging and Incremental Retraining

Maintain comprehensive metadata:

- **Experiment table:** Store product/use-case/target/config/metric entries in a Delta table. MLflow tracks lineage across training runs, while Delta Lake's change data feed identifies new data partitions.
- **Data lineage:** Record feature table versions and transformation steps for drift detection. Apache Atlas provides cross-system metadata governance.
- **Incremental training:** Trigger retraining only for products with new data using Delta Lake time-travel and MERGE operations. Prefect orchestrates conditional workflow branches.

IV. Methodology: Pipeline Implementation

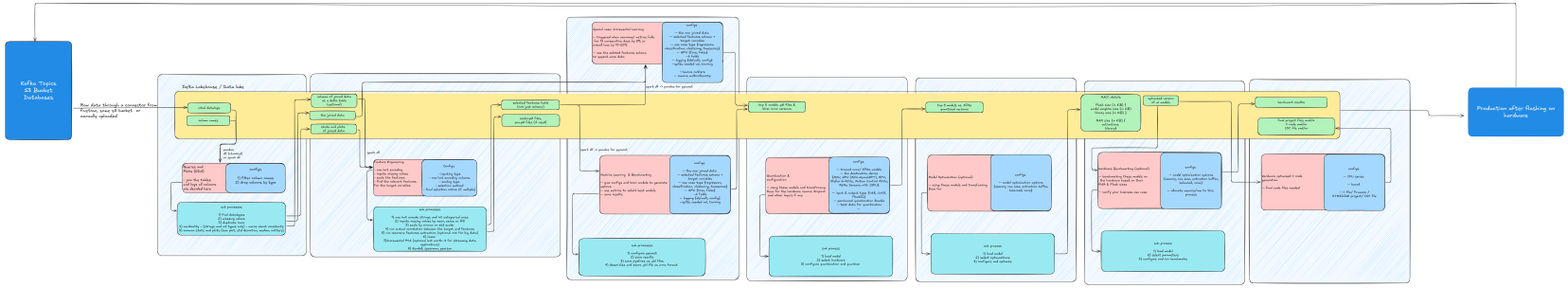

Fig. 3. Overall system architecture illustrating the flow from data ingestion through agentic configuration, distributed training, and edge deployment.

The proposed pipeline follows a structured methodology, depicted in the overall system architecture diagram (**Figure 3**), from data ingestion to edge deployment:

A. Data Ingestion and Preparation

- **Data Sources:** Raw data originates from various sources like Kafka topics, S3 buckets, or traditional databases.
- **Ingestion:** Data is ingested into the Delta Lakehouse using connectors like Fivetran, direct S3 uploads, or other custom connectors. Initial metadata, such as column names and raw data logs, are stored as Delta tables.
- **Exploratory Data Analysis (EDA):** Initial data exploration involves joining tables and selecting relevant columns. Sub-processes include identifying data types, handling missing values and duplicates, calculating cardinality, and generating summary statistics and plots (e.g., box plots, standard deviation). Configuration involves defining column filters and drops. The output includes potentially versioned joined data tables and EDA artifacts.

B. Feature Engineering

- **Transformation:** This stage applies various feature engineering techniques based on configuration. Sub-processes include one-hot encoding for categorical features, imputation of missing values (using mean, median, or other strategies), feature scaling (MinMax or StandardScaler), correlation analysis (Pearson, Spearman, Kendall), and optional techniques like Recursive Feature Elimination (RFE), lag feature generation for time-series, and Principal Component Analysis (PCA).
- **Configuration:** Key configurations include specifying imputation types, columns for one-hot encoding, scaling methods, feature selection strategies, and target ratios for selection methods.
- **Output:** The stage produces a Delta table containing the engineered features and potentially separate files defining the feature set schema.

C. Model Training and Benchmarking

- **Framework Integration:** Apache Spark DataFrames holding engineered features are efficiently converted to Pandas DataFrames using Apache Arrow for compatibility with PyCaret.

- **Training Process:** PyCaret's low-code API (`setup()`, $\texttt{compare}_{m}odels()$,

D. Edge Model Conversion and Optimization

- **Quantization:** Trained models are converted to edge-friendly formats like ONNX or TFLite. Configuration includes selecting the target model format, specifying the destination device (MCU, MPU, NPU characteristics), defining input/output data types (e.g., INT8, FP32), and providing test data for calibration if needed. Personalization options for quantization are available.
- **Optimization (Optional):** Further optimization can be applied based on target constraints like memory footprint, RAM usage, activation buffer size, or balanced performance. This involves loading the quantized model and applying selected optimization techniques.
- **Output:** This stage produces quantized and potentially further optimized model files (e.g., '.tflite').

E. Hardware Benchmarking and Code Generation

- **Benchmarking (Optional):** The optimized models can be benchmarked on target hardware (or simulators like Renode) to evaluate performance against RAM and Flash size constraints. Configuration involves specifying hardware parameters and memory allocation.
- **Code Generation:** Using tools like STM32 Cube.AI via the ST Edge AI Developer Cloud, hardware-specific C code is generated. Configuration includes specifying the target CPU series, firmware details, and associated project files (e.g., '.ioc' for STM32).
- **Output:** The final outputs are benchmark results, optimized C code, and necessary project files for deployment.

F. Deployment

- **Flashing:** The generated code and model artifacts are flashed onto the target edge hardware.
- **Production:** The device operates in production, running the deployed ML model.

V. Conclusion

Our solution, visually summarized in **Figure 3**, replaces in-memory storage with Delta Lake for robust versioning and metadata handling, uses Spark for scalable ETL, AutoGen for AI-driven configuration, and PyCaret for simplified training workflows, culminating in optimized edge deployments. By integrating ST Edge AI Developer Cloud for cross-compilation and Renode for hardware simulation, we bridge the gap between cloud-scale training and embedded execution. This addresses IoT-driven ML challenges and supports efficient, incremental model updates at scale.

References

[1] X. Chen et al., "Edge Computing for IoT: A Survey," *IEEE Access*, 2020.
[2] A. Armbrust et al., "Delta Lake: High-Performance ACID Table Storage over Cloud Object Stores," *PVLDB*, 2020.
[3] Y. Sun et al., "Apache Arrow: A Cross-Language Development Platform for In-Memory Data," *SIGMOD*, 2019.
[4] Microsoft AutoGen, https://github.com/microsoft/autogen, Accessed 2025.
[5] A. Rawat, "PyCaret: An Open Source, Low-Code Machine Learning Library in Python," *JOSS*, 2021.
[6] ONNX Runtime Quantization, https://onnxruntime.ai/docs/performance/quantization.html, Accessed 2025.
[7] TensorFlow Lite Quantization Guide, https://www.tensorflow.org/lite/performance/post_training_quantization, Accessed 2025.
[8] STMicroelectronics, "STM32Cube.AI Developer Cloud," https://stm32ai.st.com, Accessed 2025.
[9] Antmicro, "Renode - Open Source Simulation Framework," https://renode.io, Accessed 2025.
[10] M. Rocklin, "Ray: A Distributed Framework for Emerging AI Applications," *OSDI*, 2022.
[11] J. Baylor et al., "Kubeflow: Portable Machine Learning on Kubernetes," *SIGMOD*, 2021.

[12] J. Wang et al., ”Automated Machine Learning: State-of-the-Art and Open Challenges,” *IEEE Transactions on Knowledge and Data Engineering*, 2023.
[13] S. Lundberg and S. Lee, ”A Unified Approach to Interpreting Model Predictions,” *NIPS*, 2017.
[14] Qlik, ”What is Delta Lake?”, https://www.qlik.com/us/data-lake/delta-lake, Accessed: May 5, 2025.