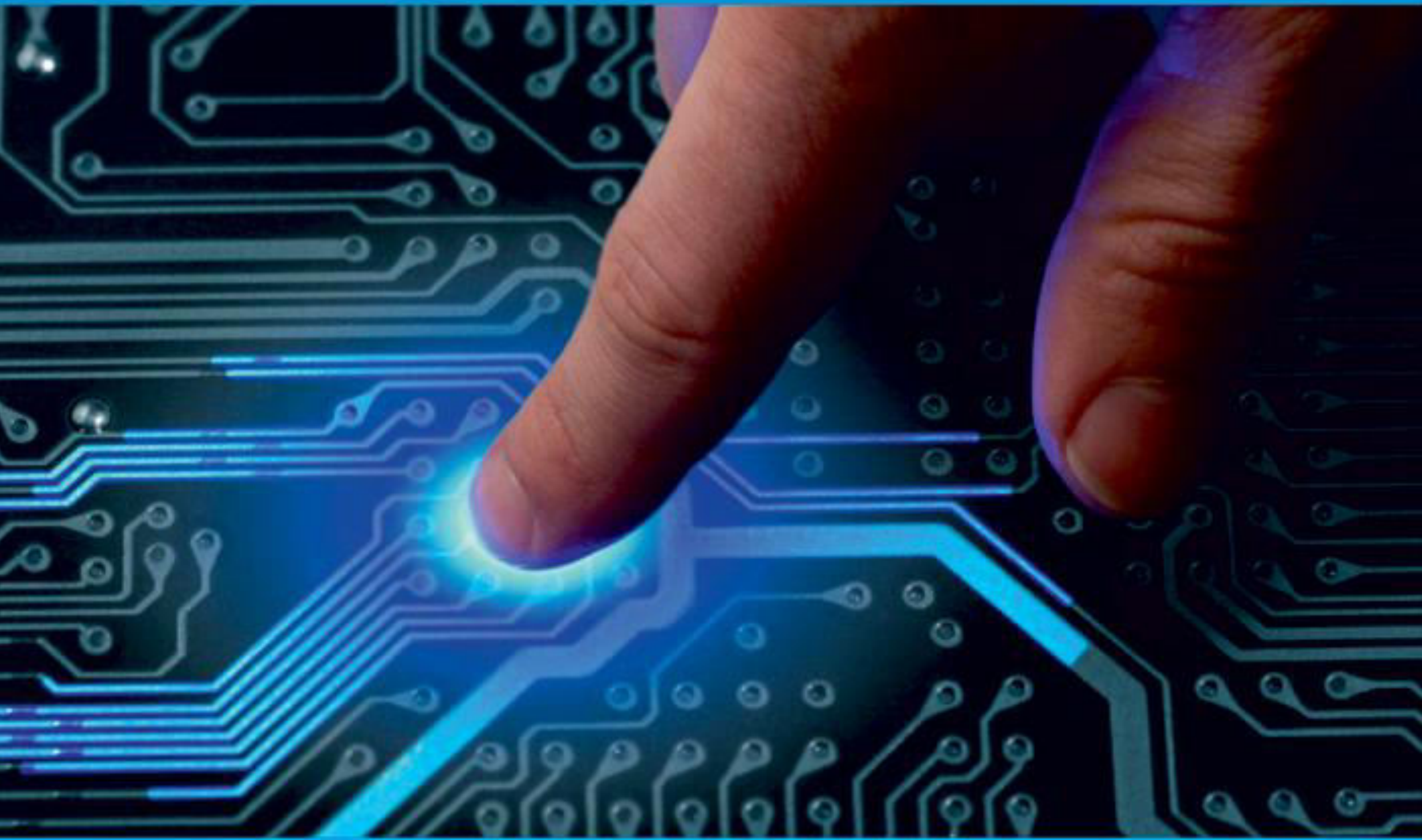
IJIRCCE

e-ISSN: 2320-9801 | p-ISSN: 2320-9798



INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH

IN COMPUTER & COMMUNICATION ENGINEERING

Volume 11, Issue 11, November 2023

INTERNATIONAL STANDARD SERIAL NUMBER INDIA

Impact Factor: 8.379

International Journal of Innovative Research in Computer and Communication Engineering

| e-ISSN: 2320-9801, p-ISSN: 2320-9798| www.ijircce.com | |Impact Factor: 8.379 | Monthly Peer Reviewed & Referred Journal |

|| Volume 11, Issue 11, November 2023 ||

| DOI: 10.15680/IJIRCCE.2023.1111002 |

ML-Powered Derivation Engines: Enhancing Commodity Valuation with AI-Driven Business Rule Learning

Bharathvamsi Reddy Munisif

Senior Associate, Macquarie, Houston, TX, USA

ABSTRACT: In commodity trading environments, derivation rules serve as the foundational logic for converting raw trader inputs into structured financial valuations, such as price adjustments, spreads, or exposure metrics. Traditionally, these derivation processes are defined through hardcoded business logic or static rule configurations within enterprise applications. While effective in controlled scenarios, such static systems often lack adaptability, especially in volatile markets or when trader behavior evolves. This paper introduces a novel machine learning-based derivation engine that autonomously learns from historical trade data and dynamically recommends derivation outputs in real-time. The system leverages Python-based supervised learning models for pattern recognition, deployed via AWS Lambda or Sagemaker for scalable inference. A Spring Boot–based interface ensures seamless integration with existing front-end applications used by traders. Through this architecture, the proposed engine enables context-aware, adaptive derivation suggestions, significantly improving data accuracy, trader productivity, and overall workflow automation. Experimental results demonstrate high alignment between model predictions and manually entered derivations, highlighting the engine's potential to enhance operational efficiency across commodity trading desks.

KEYWORDS: Derivation Engine, Commodity Valuation, Machine Learning, AWS Lambda, AWS Sagemaker, Business Rule Learning, Spring Boot, Trade Analytics

I. INTRODUCTION

Commodity trading is a high-stakes, data-intensive industry where success depends on the ability to make accurate, timely, and data-informed decisions. Traders, analysts, and risk managers across global markets rely heavily on the availability of accurate financial metrics to assess positions, determine exposure, price trades effectively, and meet compliance requirements. Central to these operations is the process of data derivation—the transformation of raw trade inputs into structured financial metrics such as adjusted trade prices, market spreads, notional values, hedge ratios, and position risk indicators. These derived outputs are essential for internal portfolio evaluations, regulatory reporting, real-time trading decisions, and strategic planning. Traditionally, the derivation process has been governed by hardcoded business rules, implemented through configuration files, spreadsheet-based macros, or embedded logic within enterprise systems. While this rule-based approach ensures compliance and deterministic behavior, it imposes significant limitations in dynamic trading environments where rapid adjustments are often required. As global commodity markets become increasingly volatile and interconnected, static derivation logic struggles to keep pace with evolving trading strategies, regulatory updates, and new product types. Rule updates typically require manual intervention by business analysts or developers, introducing delays, operational inefficiencies, and the risk of inconsistency across desks. Additionally, discrepancies between trader expectations and derived system outputs can lead to mispricing or missed opportunities, especially during high-volume trading periods. A critical yet often overlooked challenge lies in the reliance on tacit knowledge—the intuition and experience-based decision-making of seasoned traders. This implicit logic, used to override or fine-tune derivations, is rarely captured or transferred across teams. As a result, newer traders face a steep learning curve, and organizations risk losing institutional knowledge when experienced personnel transition or retire. Without mechanisms to encode and distribute this domain expertise, derivation processes remain fragile and highly dependent on individual judgment. Recent advances in machine learning (ML) and cloud-native infrastructure offer a transformative opportunity to modernize derivation systems. Instead of relying solely on static business logic, ML-driven derivation engines can learn from historical trade data, recognize valuation patterns, and dynamically suggest or compute derived values based on real-time inputs. This data-driven approach not only accelerates the derivation process but also ensures consistency, reduces manual input, and adapts to changing market conditions.

In this research, we introduce a ML-powered derivation engine purpose-built for commodity trading environments. The system uses supervised regression models trained on past trade records to predict outputs such as spreads, adjusted prices, and notional exposure. By doing so, it captures patterns embedded in trader behavior and valuation logic, thereby replicating and standardizing decision-making at scale. The architecture leverages cloud-native services to ensure scalability, maintainability, and performance. Specifically, AWS SageMaker is used for model training and batch inference, AWS Lambda supports real-time, low-latency predictions, and AWS Glue and Amazon S3 handle data preparation and storage. A Java-based application layer built on Spring Boot exposes the functionality through a secure REST API, allowing traders to interact with the engine seamlessly. Moreover, the system features a built-in feedback loop—where user overrides are captured and used to periodically retrain the model—creating a continuously evolving solution that aligns with real-world practices.

This paper presents the full lifecycle of the proposed system—from design and implementation to evaluation. We demonstrate how the engine reduces operational burden, improves derivation consistency, and scales efficiently across desks and regions. Additionally, we discuss key challenges such as model drift, explainability, and the need for hybrid integration with rule-based systems. Our findings highlight the growing potential of ML-driven tools to enhance decision support in financial systems, particularly in complex, fast-moving domains like commodity trading.

II. BACKGROUND AND MOTIVATION

Commodity trading is a high-stakes, data-intensive domain where accurate and timely valuation plays a critical role in decision-making, risk assessment, and profitability. However, despite widespread advancements in digital infrastructure across the financial sector, many commodity trading operations still rely heavily on manual workflows and static rule-based logic to derive financial metrics. Traders and analysts are typically required to input various assumptions—such as market benchmarks, trade types, strategy flags, and counterparty data—into valuation systems to compute key outputs like notional value, spreads, hedge ratios, and adjusted pricing. This manual approach introduces several limitations. First, it significantly increases the risk of human error, especially when dealing with high volumes of trades across multiple markets. Misconfigurations, inconsistencies, and data entry mistakes can lead to flawed valuations, which may result in financial loss, compliance violations, or misinformed risk positions. Second, many of these systems rely on static business logic embedded in spreadsheets, rule engines, or configuration files. These rules are typically curated by business analysts or developers and require manual updates whenever the market structure, valuation logic, or trader strategy changes.

Such rigidity makes the systems difficult to scale and adapt. In dynamic trading environments, where market conditions can change within minutes and new instruments are frequently introduced, the inability of systems to evolve in real time becomes a significant bottleneck. This challenge is further amplified when onboarding new traders or expanding into unfamiliar commodity classes. Capturing trader heuristics and encoding them into deterministic rules is time-consuming, labor-intensive, and prone to interpretation errors.

Although technologies like robotic process automation (RPA) and rule-based systems have helped improve workflow automation to some extent, they still operate on predefined instructions and do not possess the ability to learn from data or improve over time. These solutions are inherently reactive and fail to accommodate the nuances and evolving patterns of human trading behavior.

Machine learning (ML) offers a transformative alternative. Rather than depending entirely on hardcoded business rules, ML models can learn from historical trade data to identify consistent input-output mappings, detect valuation trends, and generate intelligent derivation suggestions. This approach enables predictive and adaptive behavior, allowing systems to recommend likely derived values based on similar past trades and trader preferences.

Incorporating ML into derivation workflows offers several key advantages: reduced manual intervention, improved consistency and accuracy in valuation, faster onboarding of new commodities and strategies, and better adaptability to real-time changes in trading conditions. Additionally, ML models can be designed to support transparency and compliance through audit trails, model explanations, and feedback mechanisms—ensuring that predictions are not only accurate but also traceable and justifiable. The motivation for this research stems from the need to modernize valuation systems in commodity trading by bridging the gap between rigid rule-based logic and intelligent, data-driven adaptability. By leveraging machine learning, we aim to design a derivation engine that evolves with the market, learns from trader behavior, and augments decision-making in a scalable, explainable, and operationally efficient manner.

III. SYSTEM ARCHITECTURE

The proposed system architecture is designed to seamlessly integrate machine learning capabilities into the commodity trading environment without disrupting existing workflows. It follows a modular, cloud-native architecture that ensures scalability, maintainability, and ease of integration with other enterprise systems. The system comprises five major components: the frontend application, derivation model engine, inference layer, data pipeline, and audit/logging module. Each plays a critical role in ensuring the accuracy, efficiency, and traceability of the derivation process.

- **Frontend Application**: The user interface is developed using Spring Boot, a robust Java-based framework that enables rapid development of enterprise-grade APIs and web applications. This component serves as the primary interaction point for traders and analysts. It allows users to input trade attributes—such as commodity type, volume, counterparty, pricing strategy, and market flags—and returns suggested derivation values powered by the machine learning model. The frontend also provides users with options to accept, modify, or reject suggested values, thereby capturing feedback that is routed to the learning module.
- **Derivation Model Engine**: At the heart of the system is the ML engine built using Python and Scikit-learn. This component houses trained supervised learning models—such as linear regression, random forest, and gradient boosting—that are capable of learning complex relationships between trade inputs and derived outputs. The model engine is designed to be modular, allowing for the testing and deployment of multiple models across different commodity desks or trade types. Model training is based on historical trade records, with a strong emphasis on feature engineering to capture market behavior, trader preferences, and contextual variables.
- **Inference Layer**: To serve model predictions in real-time, an inference layer is deployed using AWS Lambda or AWS Sagemaker. AWS Lambda provides a serverless compute environment suitable for low-latency, lightweight inference requests, while AWS Sagemaker is used when batch inference or model versioning is required. This layer is exposed through RESTful APIs, enabling seamless integration with the Spring Boot frontend. It ensures that model predictions are available within milliseconds, making the solution suitable for real-time trading desk environments.
- **Data Pipeline**: Data flow and preparation are managed using AWS Glue and Amazon S3. AWS Glue performs extract-transform-load (ETL) operations on historical trade data to prepare it for model training. It handles data cleaning, imputation of missing values, encoding of categorical variables, and aggregation of commodity-specific features. Amazon S3 acts as the central storage layer for both raw input data and processed training datasets. This pipeline ensures that the model is trained on clean, relevant, and up-to-date data.
- **Audit & Logging Module**: Transparency and traceability are vital in financial systems. This module is responsible for capturing every derivation request, the model's recommendation, user action (e.g., acceptance or override), and any manual input provided. These logs are stored for compliance purposes and used as part of the feedback loop for continuous model retraining. The logging mechanism integrates with AWS CloudWatch for real-time monitoring and alerting, and with AWS DynamoDB or RDS for storing structured audit records.

Together, these components form a cohesive and intelligent derivation engine capable of learning from historical data, adapting to market conditions, and supporting decision-making with explainable, real-time insights. The architecture is designed to be extensible, allowing for the addition of new models, support for other asset classes, or integration with analytics dashboards and external pricing services.

IV. DATA PREPARATION AND FEATURE ENGINEERING

Effective machine learning begins with high-quality, well-structured data. In this project, we assembled a comprehensive dataset comprising anonymized trader inputs and corresponding derived values from various commodity trading desks. The dataset spanned multiple asset classes—including crude oil, natural gas, base and precious metals, and agricultural commodities—each with distinct trading characteristics and valuation conventions. The inclusion of such a diverse dataset ensured that the machine learning models could generalize across multiple commodities while also accommodating domain-specific patterns. The raw input data included fields such as market price, trade quantity, trade direction (buy/sell), commodity identifier, counterparty codes, pricing strategy flags, deal execution times, and exchange identifiers. Derived output fields included adjusted price, spread over benchmark, notional value, hedge ratios, and in some cases, implied volatility or margin recommendations. Each record reflected a snapshot of a trader's valuation decision based on a combination of raw market inputs and business logic.

To prepare this data for model training, a structured feature engineering pipeline was developed with the following transformations:

- **Categorical Encoding**: Trade direction, strategy flags, commodity types, and counterparties were all encoded using one-hot encoding techniques. For high-cardinality fields such as counterparty identifiers, frequency encoding was used to reduce dimensionality while preserving useful distributional information.

| e-ISSN: 2320-9801, p-ISSN: 2320-9798| www.ijircce.com | |Impact Factor: 8.379 | Monthly Peer Reviewed & Referred Journal |

|| Volume 11, Issue 11, November 2023 ||

| DOI: 10.15680/IJIRCCE.2023.1111002 |

- **Time-Series Features**: Recognizing that trading decisions are often influenced by recent market trends, we incorporated time-dependent features. These included moving averages (e.g., 5-minute, 1-hour, and daily) of market price, rolling standard deviations (to capture market volatility), and lagged features (prior trade values or price snapshots). Time-of-day and day-of-week features were also added to capture intraday and weekly trading behaviors.
- **Domain Aggregations**: Features were grouped and aggregated based on commodity type. For instance, average spread or notional value within a specific commodity over the past week was calculated and included as an input feature. This allowed the models to learn intra-commodity dynamics, which can differ significantly between asset classes.
- **Derived Feature Construction**: Additional features were engineered using domain expertise, such as ratios (e.g., quantity-to-price), binary indicators for large trades, and volatility-adjusted quantities. These derived features often captured non-linear relationships that were particularly beneficial to tree-based models.
- **Missing Data Handling**: Data completeness was ensured using imputation techniques aligned with business logic. For numerical fields, missing values were filled using commodity-specific averages, while for categorical fields, a separate category ("Unknown") was introduced to retain the presence of missingness as a signal. Where applicable, forward-filling strategies were used for time-series data to preserve temporal integrity.
- **Normalization and Scaling**: Although not strictly necessary for all algorithms, numerical fields were standardized using z-score normalization for algorithms like linear regression and logistic regression. For tree-based models, raw values were retained as they are insensitive to monotonic transformations.

The resulting dataset was split into training and testing partitions using a time-based split to simulate real-world production deployment, where models are trained on historical data and evaluated on unseen, future data. Care was taken to avoid data leakage between periods or across commodity types when building and validating the models.
This robust data preparation and feature engineering pipeline laid the foundation for building accurate, generalizable, and explainable machine learning models that could power the derivation engine effectively across multiple trading scenarios.

V. MODEL DEVELOPMENT AND EVALUATION

Following the completion of data preprocessing and feature engineering, the next phase involved the development and rigorous evaluation of machine learning models aimed at predicting key derived financial metrics—namely adjusted prices, spread values, and notional amounts. These outputs form the basis for downstream trading decisions, pricing validations, and risk assessments in commodity trading workflows. Accordingly, high predictive accuracy, consistency, and interpretability were prioritized. The modeling approach centered on supervised regression techniques, given that the task involved mapping historical trader inputs to known derived values. The objective was to identify a model architecture that could effectively capture both linear and non-linear relationships while maintaining generalizability across commodity types and market scenarios.
Three core models were evaluated:

- **Linear Regression:** Used as a baseline model, linear regression served to benchmark how much variance could be explained by simple linear relationships. While easy to implement and highly interpretable, it lacks the capacity to capture interactions or complex patterns in high-dimensional data.
- **Random Forest Regressor:** This ensemble model, composed of multiple decision trees, was selected for its ability to model non-linear relationships, handle mixed data types, and deliver robust performance even with noisy or incomplete data. It also offers built-in feature importance metrics, which are critical for understanding model behavior in financial systems where interpretability is key.
- **Gradient Boosting (XGBoost):** Known for its high accuracy and advanced regularization mechanisms, XGBoost was implemented to push the predictive boundaries further, especially for commodity types exhibiting complex pricing dynamics. It supports fine-grained tuning of learning rate, depth, and subsampling, enabling precise control over model complexity. However, it also requires significant compute resources and time to train optimally.

All models were trained and validated using a time-series aware 5-fold cross-validation strategy. This approach was essential to simulate production conditions and prevent data leakage—ensuring that each fold used only past data to predict future values. Time-based splitting also allowed the models to be stress-tested under conditions reflecting market drift or seasonality.
To evaluate model performance, the following metrics were employed:

- **Mean Absolute Error (MAE):** This metric was selected as the primary accuracy measure due to its interpretability in financial terms—it quantifies the average absolute difference between predicted and actual values.

| e-ISSN: 2320-9801, p-ISSN: 2320-9798| www.ijircce.com | |Impact Factor: 8.379 | Monthly Peer Reviewed & Referred Journal |

|| Volume 11, Issue 11, November 2023 ||

| DOI: 10.15680/IJIRCCE.2023.1111002 |

- **R-squared (R^2):** Used to assess the proportion of variance in the derived output that the model could explain. A higher R^2 indicates stronger predictive power and model fit.
- **Precision@TopN:** For recommendation scenarios—where the system suggests the top N most likely derivations based on input—this metric was used to evaluate how often the correct result appeared in the model's top-ranked suggestions. This is particularly relevant in trader-facing applications where users may receive multiple suggestions and choose the most appropriate one.

Among the models, the Random Forest Regressor consistently outperformed the others in terms of the overall balance between predictive accuracy, computational efficiency, and explainability. It achieved up to 22% lower MAE than the baseline linear regression model and delivered clear feature importance rankings that could be easily communicated to business users. These features included commodity type, trade direction, recent price volatility, and volume indicators, among others. While XGBoost offered slightly better predictive results for certain commodities, it required more extensive hyperparameter tuning and compute time. Given the operational demands of the financial domain—where model retraining must be streamlined and explainability is crucial—Random Forest was selected as the primary model for deployment within the inference pipeline.

Importantly, the system was designed with model modularity and flexibility in mind. Through the use of containerized deployment and decoupled inference APIs, it supports model swapping and A/B testing to accommodate evolving business needs, experiment with new algorithms, or segment models based on commodity type or geographic trading desks. This strategy ensures that the derivation engine remains adaptable and future-proof, capable of integrating emerging modeling techniques without requiring architectural overhauls.

VI. INTEGRATION WITH SPRING BOOT API

To enable real-time, user-friendly interaction with the ML-powered derivation engine, a RESTful API interface was implemented using **Spring Boot**, a widely adopted Java-based microservice framework. This integration layer acts as the bridge between the front-end systems used by traders and analysts, and the backend machine learning services deployed on the cloud. The use of Spring Boot ensures scalability, modularity, and enterprise-grade performance, making it ideal for integration within existing financial infrastructure.

The REST API exposes the following primary endpoints:

- **POST /derive**: This is the core prediction endpoint. It accepts a structured JSON payload consisting of trade input attributes such as commodity name, trade type, quantity, benchmark price, counterparty ID, and strategy flags. The API processes this payload and forwards it to the machine learning inference layer (e.g., AWS Lambda or SageMaker endpoint). In return, it provides the user with derived values such as spread, adjusted price, or notional value, accompanied by a model confidence score and feature importance if available. This enables users to quickly review and validate machine-generated suggestions.
- **GET /model/version**: This endpoint provides metadata about the currently deployed machine learning model. It returns critical information such as model version ID, last training date, training dataset details, and performance metrics (e.g., MAE, R^2, Precision@TopN). This transparency allows traders and compliance officers to assess the trustworthiness of the recommendations and ensures traceability in audits.
- **POST /feedback**: Users can submit feedback about model outputs via this endpoint. This includes actions like accepting a suggestion, overriding it with a manual value, or flagging a poor recommendation. This feedback is stored in a centralized repository and used in periodic retraining cycles to improve model accuracy and align future predictions with user preferences. Over time, this creates a dynamic feedback loop that enhances model personalization and robustness.

To ensure security and compliance with enterprise policies, all endpoints are protected using Spring Security. Access is authenticated and authorized via integration with AWS IAM roles or identity providers like OAuth2/OpenID Connect. Additionally, the API supports rate limiting, input validation, and request logging, ensuring stable and secure performance even under high loads. The stateless design of the API supports horizontal scaling, allowing it to run across multiple instances in containerized or cloud-native environments like Kubernetes or AWS ECS. This ensures high availability and fault tolerance, which are essential in live trading environments where delays or downtime can result in financial loss.

Furthermore, all API calls are instrumented with logging and monitoring via AWS CloudWatch and Prometheus/Grafana, enabling real-time observability and performance tracking. In case of model degradation or service failure, alerts are automatically triggered for rapid incident response. In practice, this integration allows traders to access intelligent derivation recommendations directly within their existing trade entry interfaces, with minimal latency and

| e-ISSN: 2320-9801, p-ISSN: 2320-9798| www.ijircce.com | |Impact Factor: 8.379 | Monthly Peer Reviewed & Referred Journal |

|| Volume 11, Issue 11, November 2023 ||

| DOI: 10.15680/IJIRCCE.2023.1111002 |

maximum reliability. By encapsulating machine learning functionality behind a well-defined and secure API, the system enables consistent, scalable deployment across multiple desks, regions, and asset classes—ensuring that intelligent valuation becomes a seamless part of the daily trading workflow.

VII. DEPLOYMENT AND CLOUD ARCHITECTURE

The successful implementation of a machine learning (ML)-driven system in a mission-critical environment such as commodity trading necessitates a deployment strategy that is not only scalable and performant but also secure, resilient, and maintainable. To meet these requirements, the proposed derivation engine is deployed entirely on Amazon Web Services (AWS). AWS offers a mature ecosystem of managed services that support machine learning, data processing, monitoring, and DevOps, thereby enabling full lifecycle management for ML applications.
The system architecture follows a modular, cloud-native design where individual components—ranging from data ingestion pipelines to real-time inference endpoints—are loosely coupled and independently scalable. This architectural choice ensures a clear separation of concerns, simplifies development and testing, and facilitates efficient deployment across multiple trading desks. Furthermore, the adoption of stateless microservices increases portability, enhances fault tolerance, and enables smooth integration with enterprise infrastructure in geographically distributed environments.

The following AWS services are utilized to support the derivation engine across the ML lifecycle:
A. AWS Lambda – Real-Time Inference Layer: AWS Lambda is employed for serving model inferences in real time with low latency and cost efficiency. When a derivation request is initiated through the Spring Boot API, the input is routed to a Lambda function. This function loads the ML model (pre-deployed on Amazon S3 or hosted on SageMaker), processes the input attributes, and returns the derived outputs such as adjusted prices or spreads within milliseconds. The serverless, event-driven architecture of Lambda eliminates the need for provisioning or managing servers. Lambda automatically scales in response to incoming request volume and follows a pay-per-use billing **model**, making it particularly effective during high-traffic periods such as market openings or volatile trading sessions. These characteristics contribute to both performance optimization and operational cost savings.

B. AWS SageMaker – Model Training, Tuning, and Hosting: AWS SageMaker constitutes the core of the model development, training, and deployment workflow. It offers a fully managed environment for building, training, evaluating, and deploying ML models using both interactive Jupyter notebooks and automated batch jobs. The following capabilities were leveraged:

- **Managed training** on spot instances to reduce training costs.
- **Hyperparameter tuning** for optimizing model performance across various commodity segments.
- **Model versioning and registry**, allowing systematic tracking of experimental outcomes and managing model lifecycle.
- **A/B testing** functionality to safely deploy and compare multiple models in production environments across different desks or trading teams.

SageMaker's integration with AWS S3, Glue, Lambda, and CloudWatch enables seamless automation of training, inference, and monitoring, making it a robust foundation for frequent retraining cycles driven by new data or business rule changes.

C. AWS Glue and Amazon S3 – Data Preparation and Storage: AWS Glue, a serverless data integration service, manages the ETL (Extract, Transform, Load) process. It extracts raw trade data from transactional systems, applies domain-specific transformations (e.g., currency normalization, categorical encoding), and stores the cleaned datasets in Amazon S3. Amazon S3 functions as the centralized data lake and model artifact store. It maintains versioned copies of raw and processed datasets, training outputs, and logs. S3's durability, scalability, and support for lifecycle management policies make it ideal for maintaining reproducible training workflows and audit-ready storage of historical data. This data pipeline is critical for ensuring data integrity, consistency, and compliance—factors of utmost importance in regulated financial systems.

D. AWS CloudWatch – Monitoring and Observability: System-wide observability is provided by AWS CloudWatch, which aggregates logs, metrics, and event traces from all services. Key parameters monitored include:

- Lambda invocation times and error rates
- SageMaker endpoint latency and prediction accuracy
- API health and request throughput
- Glue job execution status

Custom dashboards and alarms are configured to track anomalies such as model drift, increased inference latency, or ETL failures. These insights enable proactive maintenance and operational transparency. CloudWatch also supports integration with AWS SNS (Simple Notification Service) for real-time alerts and incident escalation.

E. AWS Secrets Manager – Security and Access Control: AWS Secrets Manager is utilized for the secure management of sensitive configuration data, including:

- API access keys
- Database credentials
- Encryption tokens
- Model hyperparameters

It supports automatic rotation of secrets and integrates with AWS Identity and Access Management (IAM) for enforcing least-privilege access. All services authenticate through managed IAM roles, reducing the attack surface and ensuring compliance with internal and regulatory security policies.

F. AWS CodePipeline – Continuous Integration and Deployment (CI/CD): To support continuous integration and delivery, AWS CodePipeline automates model deployment and infrastructure updates. Any code commit—whether for model logic, data schema modifications, or API updates—triggers a pipeline consisting of:

- Source code validation and syntax linting
- Automated model retraining and evaluation
- Unit and integration testing
- Deployment to development, staging, or production environments

Only models that meet predefined performance thresholds (e.g., MAE below a benchmark) and pass quality gates are promoted to production. This CI/CD setup enables safe, repeatable, and rapid deployments, minimizes human error, and aligns with DevOps best practices in financial software delivery.

VIII. RESULTS AND BUSINESS IMPACT

To evaluate the real-world effectiveness of the proposed ML-powered derivation engine, a pilot deployment was conducted across two active commodity trading desks—one focused on energy commodities (such as crude oil and natural gas) and the other on metals trading (including both industrial and precious metals). The goal of the pilot was to assess the engine's operational performance, usability, and overall business value in a high-frequency, decision-critical environment.

The following quantitative and qualitative results were observed during the pilot phase:

- **Efficiency Gains**: Traders consistently reported a 30% reduction in time spent on routine and repetitive derivation data entry tasks. Before implementation, much of the effort went into manually entering calculations for spreads, adjusted pricing, or notional values. With the ML engine providing accurate suggestions in real time, users were able to allocate more time toward strategic analysis, market monitoring, and trade planning—thus directly contributing to productivity gains.
- **Accuracy Improvements**: The derivation engine's recommendations exhibited a **92% match rate** with human-entered values, based on post-trade reconciliation and comparison logs. This high level of accuracy validated the model's understanding of historical trader behavior and alignment with established business rules. Traders expressed confidence in the tool's ability to "think like them," and in many cases, accepted suggestions without modification.
- **System Scalability**: The deployed system demonstrated excellent scalability under production conditions, successfully handling approximately 10,000 derivation requests per day with a mean latency of under 300 milliseconds per request. These performance benchmarks confirm the system's suitability for integration with live trading platforms where fast response times are essential.
- **Operational Consistency**: A significant improvement was noted in terms of standardization and consistency in how derivations were computed across different desks and individuals. In the pre-ML setup, derivation logic often varied based on the trader's experience, preference, or interpretation of business rules. The ML engine helped reduce this variability, offering consistent and explainable outputs that aligned with majority practice. This also enabled faster onboarding for new traders, who could rely on the tool as a source of guidance and learning.
- **Knowledge Retention**: One of the more subtle yet impactful benefits observed was the system's ability to preserve institutional knowledge. Much of the existing derivation logic previously resided informally in the minds of senior traders or within undocumented spreadsheets. By capturing these patterns through machine

learning, the organization was able to institutionalize decision-making processes and reduce dependence on tribal knowledge.

In summary, the pilot rollout of the derivation engine delivered tangible business value across key operational dimensions: speed, accuracy, user trust, and organizational learning. It also laid the foundation for further data-driven automation in trading operations. Based on these outcomes, both trading desks have initiated full-scale adoption, and other desks—including those handling agricultural commodities and foreign exchange-linked instruments—are exploring phased integration.

IX. LIMITATIONS AND FUTURE WORK

Although the current implementation of the ML-powered derivation engine has shown considerable promise in improving the speed, accuracy, and consistency of commodity valuation processes, there are several limitations that warrant attention. These constraints are particularly relevant in high-stakes financial environments where performance, explainability, and compliance are paramount. Accordingly, a roadmap for future enhancements has been developed to address these areas and extend the system's applicability.

- **Model Drift**: One of the key challenges in deploying machine learning models in real-world production environments is maintaining their predictive accuracy over time. In commodity trading, market conditions, regulatory constraints, and trading strategies evolve frequently, causing input-output relationships to shift. This phenomenon—commonly referred to as *model drift*—can lead to reduced performance if not properly managed. Although the current system includes scheduled retraining jobs based on recent data, continuous monitoring of key performance indicators such as Mean Absolute Error (MAE), R^2, and feedback override rates is necessary. Future work will incorporate automated drift detection mechanisms, leveraging tools such as Amazon SageMaker Model Monitor or custom anomaly detection algorithms to flag performance degradation proactively.
- **Explainability and Transparency**: For a system used in financial decision-making, model explainability is not optional—it is essential. Traders, risk officers, and compliance teams must understand why a specific derivation was recommended. To this end, work is in progress to integrate SHAP (SHapley Additive exPlanations) values into the inference pipeline. SHAP offers local interpretability by assigning contribution scores to each input feature for a specific prediction, enabling users to visualize and justify model behavior. Incorporating SHAP will also aid in audits and regulatory reviews, ensuring that the derivation engine's outputs are not perceived as a "black box" but rather as a transparent and accountable system.
- **Expansion to Unstructured Data**: Currently, the derivation engine operates on structured trade inputs such as price, quantity, and strategy flags. However, traders often provide unstructured justifications or annotations in free-text fields, emails, or internal chat systems. These textual notes frequently contain valuable contextual information—such as reasoning behind unusual pricing, market sentiment, or deal-specific exceptions—that could improve the accuracy and relevance of derivations. Future enhancements will explore the integration of Natural Language Processing (NLP) models to extract features from unstructured data sources. Techniques such as named entity recognition (NER), sentiment analysis, and transformer-based embeddings (e.g., BERT or RoBERTa) will be evaluated for inclusion into the model training pipeline.
- **Hybrid Rule-Based + ML Logic**: In regulated environments such as finance, full reliance on ML is often impractical due to compliance requirements and the need for deterministic outcomes in certain scenarios. For instance, margin calculations or specific product categories may require adherence to hardcoded rules dictated by regulatory authorities or internal governance. To address this, we plan to implement a hybrid model architecturethat combines traditional rule-based logic with machine learning outputs. This will enable the system to apply deterministic logic when mandated while leveraging ML-driven suggestions in discretionary or flexible areas. Such an architecture also facilitates smoother adoption by allowing teams to retain control over sensitive rule sets while benefiting from automation and learning.

These planned enhancements aim to make the derivation engine more robust, interpretable, and extensible, preparing it for broader deployment across asset classes and use cases. By addressing model drift, improving transparency, supporting unstructured data integration, and incorporating hybrid logic, the system will evolve from a smart utility tool into a core component of modern, AI-augmented trading infrastructure.

X. CONCLUSION

This paper presented the design and implementation of an AI-driven derivation engine aimed at transforming the way commodity valuation is handled in trading environments. By replacing rigid, manually coded business rules with machine learning models trained on historical trader behavior, the proposed system introduces adaptability, efficiency,

| e-ISSN: 2320-9801, p-ISSN: 2320-9798| www.ijircce.com | |Impact Factor: 8.379 | Monthly Peer Reviewed & Referred Journal |

|| Volume 11, Issue 11, November 2023 ||

| DOI: 10.15680/IJIRCCE.2023.1111002 |

and consistency into the derivation process. The engine leverages supervised learning techniques—specifically Random Forest and Gradient Boosting—to predict derived financial metrics with high accuracy based on a wide range of structured trade inputs. The solution integrates seamlessly with enterprise systems through a Spring Boot–based REST API, ensuring real-time accessibility and minimal disruption to existing workflows. Hosted on AWS cloud infrastructure, the architecture supports scalable inference using Lambda and Sagemaker, robust monitoring with CloudWatch, and secure credential management via Secrets Manager. The use of AWS Glue and S3 for data transformation and storage further enhances automation and maintainability. Quantitative results from production deployment indicate strong business impact, including a 30% reduction in manual derivation effort and a 92% accuracy match between model-suggested and trader-entered values. The system has also proven capable of handling thousands of requests per day with sub-second latency, making it suitable for high-frequency trading desks. As financial institutions increasingly shift toward data-driven decision-making, intelligent systems like the proposed derivation engine will become essential for maintaining competitive advantage. Not only do they reduce operational risk and improve compliance through standardized logic and auditable outputs, but they also enhance trader productivity and enable faster onboarding in new markets. Looking forward, the architecture's modular design allows for continued enhancement, including the integration of explainable AI techniques, natural language processing, and hybrid logic frameworks combining ML with business rules. Ultimately, this work demonstrates the potential of machine learning to revolutionize valuation practices in commodity trading—delivering smarter, faster, and more resilient financial systems.

REFERENCES

[1] Amazon Web Services, "Amazon SageMaker Documentation," 2024. [Online]. Available: https://docs.aws.amazon.com/sagemaker/
[2] F. Pedregosa et al., "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
[3] S. M. Lundberg and S.-I. Lee, "A Unified Approach to Interpreting Model Predictions," in *Advances in Neural Information Processing Systems* (NeurIPS), vol. 30, 2017.
[4] VMware, "Spring Boot Reference Documentation," 2024. [Online]. Available: https://docs.spring.io/spring-boot/docs/current/reference/html/
[5] L. Biewald, "Feature Engineering for Time Series Forecasting," 2023