STATISTICS, OPTIMIZATION AND INFORMATION COMPUTING
Stat., Optim. Inf. Comput., Vol. **15**, March 2026, pp 2086–2104.
Published online in International Academic Press (www.IAPress.org)

A Lambda Lakehouse Architecture Bridging Streaming and Batch Intelligence in Volatile and Scalable Financial Data Processing

Maryam MAATALLAH [1,*], Mourad FARISS [2], Hakima ASAIDI [1], Mohamed BELLOUKI [1]

[1] *LMASI, FPDN, Mohammed First University, Nador, Morocco*
[2] *ERCI2A, FSTH, Abdelmalek Essaadi University, Tetouan, Morocco*

Abstract The vast growth of digital financial market data necessitates new kinds of analytical infrastructure which can process large volumes of data continuously, while maintaining reliability for use over extended periods as part of a long-term historical processing requirement. Batch based platforms have difficulty meeting both these needs, whereas pure streaming platforms often sacrifice analytical consistency with respect to their analysis. To address this limitation our paper proposes a Unified Lambda-Lakehouse Architecture which allows Real-Time and Batch Processing to be performed together in a single, ACID compliant. Apache Kafka captures live Bitcoin markets and performs the real-time processing via Spark Structured Streaming, while the periodic storage of historical records and subsequent periodic reprocessing of those records is accomplished via Amazon S3. Ultimately both the real-time and batch processing paths converge at a Delta Lakehouse; thereby enabling schema enforcement, versioning, and time-travel queries. The proposed architecture places the emphasis on combining the Speed Layer, Batch Layer, and Serving Layer into a single operational workflow atop a transactional Lakehouse foundation. Advanced predictive models including LSTM, GRU, ARNN, and XGBoost are used to forecast Bitcoin prices at daily, hourly, and minute granularities. Results from experiments indicate that the LSTM model consistently produced the best results (RMSE = 2383.9, 539.3, 144.9) at the three respective levels.

Keywords Big Data, Lambda Architecture, Lakehouse, Delta Lake, Real-Time Analytics, Bitcoin Forecasting, GRU, ARNN, LSTM, XGBoost.

DOI: 10.19139/soic-2310-5070-3222

1. Introduction

Cryptocurrency exchanges generate large volumes of streaming data that are volatile, irregular, and rapidly evolving. At the same time, the rise of high-frequency trading activity has increased the need for data architectures capable of delivering both real-time insights and reliable historical analysis. Systems built mainly on traditional batch-processing, such as early Hadoop MapReduce, remain effective for large-scale computations but suffer from high latency that is incompatible with real-time-sensitive decision-making [1]. Conversely, purely streaming-oriented architectures excel in immediacy but often struggle to maintain analytical stability or to capture long-term dependencies within financial time series [2].

To address these limitations, the Lambda Architecture was proposed by Marz and Warren [1, 3] as a hybrid layer combining the two complementary paradigms of a Speed Layer for the low-latency stream processing and a Batch Layer for the accurate historical computation. They propose that their outputs would converge in a Serving Layer, providing unified and query-ready analytics [4, 6]. While useful, the Lambda introduces duplicated

*Correspondence to: Maryam MAATALLAH (Email: maryam.maatallah.d23@ump.ac.ma). LMASI, FPDN, Mohammed First University, Nador, Morocco.

ISSN 2310-5070 (online) ISSN 2311-004X (print)

processing pipelines and considerable maintenance costs because the batch and speed layers must remain aligned and reconciled over time [5, 7].

More recent developments in data engineering have provided the Lakehouse model, which provides a combination of the openness of data lakes and the transactional characteristics of data warehouses. Technologies such as Delta Lake, Apache Iceberg, and Hudi, provide ACID compliance, schema enforcement and time-travel queries allowing consistent storage of structured and semi-structured information [8]. In this ecosystem, the medallion layout has evolved into a standard Lakehouse design pattern for ensuring progressive data refinement and quality assurance. However, the layered structure is primarily focused on the organization and curation of the datasets and does not inherently provide a functional integration between streaming workloads and batch-processing pipelines [9].

This research outlines the Lambda-Lakehouse Architecture an approach that will combine the principles of the Lambda model and the Lakehouse ecosystem within a single operational architecture. The goal is not to recreate the Medallion hierarchical structure, which primarily focuses on progressively cleaning and organizing data, but instead to ensure that batch computations and streaming updates operate in a coordinated fashion using a transactional Delta Lake core. This system utilizes Apache Kafka and Zookeeper to manage the continuous ingestion [10, 11]; while Spark Structured Streaming [12] is used for real-time transformations; and Amazon S3 for scalable persistent storage [9]. The resulting data flow is then exposed via a lightweight Flask based web interface that allows users to monitor activity and visualize results as they evolve, creating a direct link between ingestion, processing, and analysis. To test how well the system can perform in forecasting tasks, we integrated deep learning and machine learning models: LSTM, GRU, ARNN, and XGBoost. These models were trained on Bitcoin price data from 2018-2025 [13], with observations available at daily, hourly, and minute resolutions. This range of temporal granularity permits the examination of both long-term trends and rapid market fluctuations [14, 15].

The rest of the paper is structured in the following way: Section 2 provides an overview of earlier work on hybrid data-processing architectures and financial prediction methods. Section 3 describes the proposed Lambda-Lakehouse framework and explains how its main components interact. Section 4 outlines the forecasting models and the evaluation criteria used in the study. Section 5 reports and interprets the experimental results. Finally, Section 6 summarizes the main contributions and points to possible directions for future work.

2. Related Work

The steady expansion of financial data has encouraged the design of architectures that can handle large historical datasets while still providing fast analytical responses. Early technologies most notably Hadoop MapReduce and its ecosystem: HDFS, Hive, Pig, were built with batch processing primarily for batch-oriented workloads, offering scalability and reliability for massive data analysis [1]. The most significant disadvantage of these systems was that although they provided scalable and reliable data storage, the delay (latency) involved with each system made it impossible to use them in environments that require immediate action based on fast-changing market conditions [2].

To solve some of the problems caused by these systems, Marz & Warren developed the Lambda Architecture [1, 3] which included a dual-layer architecture. The first layer, called the ”Batch Layer,” would take the entire historical dataset and run through it at the same time to find out what happened in the past. The second layer, called the ”Speed Layer,” would take the new data that had just come in and process it right away to make fast decisions about the current environment. Ultimately, the processed data from both layers would then be fed into a third layer called the ”Serving Layer” that would provide a consolidated view of all the information for consumers. This design brought many benefits, such as fault tolerance, scalability, and accurate views of the data in a distributed computing environment. Many different industries, such as smart farming [6], air-ground surveillance systems [2], etc., that require fast decision-making combined with deep analysis have used this model. However, in order to operate effectively, they need to maintain and synchronize both layers, which adds complexity to the operation, thus increasing the overall cost of operating the system [4, 5, 7]. To reduce the complexity of maintaining and

synchronizing the dual-layer model of Lambda, Kreps developed the Kappa Architecture [16], which suggests a single data stream that can be replayed to recreate previous states of the data whenever needed. This greatly reduced the amount of labor required to manage the system, as there were only two streams of data to manage. However, if the problem is dealing with a massive historical data set or a large volume of updates per unit of time (which is normal for financial markets), the cost of replaying the entire historical data set in order to get current state makes Kappa much less efficient.

Recently, Lakehouse technology has become increasingly popular because of its ability to combine the flexibility of data lakes with the reliability typically seen with data warehouses. Technologies like Delta Lake, Apache Iceberg and Hudi allow for ACID compliant transactions, schema evolution, and time-travel queries enabling both analytical consistency and scalability [8, 11]. With the rise of Lakehouse technology, the Medallion Architecture developed by Databricks, where data is organized into Bronze, Silver and Gold, has emerged as a template for incrementally improving data quality. While successful for structuring and refining datasets, the Medallion Architecture does not consider the integration of batch and real-time processing pipelines [8, 9].

Research has continued to develop the understanding of Lakehouse systems. Azzabi et al. [17] provided a comprehensive review of Data Lake systems and explained how the principles of designing these systems have evolved toward Lakehouse designs. Similarly, Schneider [18] studied the technical basis for modern Lakehouse and pointed out the major technical demands of modern Lakehouse technology (ACID compliance, metadata management, and time travel query). Ait Errami [19] described the progression from Data Warehouse and Data Lake to Lakehouse technology and showed how this progression enables increased elasticity and improved schema management in large scale analytical workloads. Also, in a similar vein, Tagliabue and Greco [20] presented reproducible data pipelines built with Bauplan and Nessie, focusing on version control, replicability, and governance for distributed analytical environments.

Collectively, these papers show that the development of Lakehouse systems is maturing; however, they also illustrate a persistent shortcoming, there is relatively little research into how batch and streaming pipelines can be integrated into a single ACID compliant transactional layer specifically designed for financial data applications. Within the financial industry, other researchers have explored hybrid or cloud-based architectures for forecasting and trading-related tasks. For example, Fariss et al. [15] compared cloud and on-site environments for Forex predictions using machine learning methods, and highlighted the need for architectures that support both scalability and low latency.

Additionally, other researchers have employed Apache Kafka and Spark Streaming to process real-time market data, demonstrating that these technologies significantly enhance throughput and fault tolerance [10, 11]. Nevertheless, only a small number of frameworks attempt to integrate Lambda's dual-processing strategy and the governance characteristics of Lakehouse technology into a single, coherent solution ensuring consistency, scalability, and reproducibility. Simultaneously, developments in predictive modeling (notably LSTM, GRU, and XGBoost) have enabled financial forecasting by training models to capture complex, non-linear patterns in time series data. Studies that link large-scale data pipelines with deep learning forecasting methodologies [14, 15] have found considerable improvements in both accuracy and system performance. Nonetheless, these studies generally treat data engineering and prediction as distinct layers and do not propose an integrated architectural vision connecting them.

3. The Proposed Approach

The traditional data processing paradigm has limitations in regards to speed in high velocity domains (such as financial analysis) in terms of both the reliability/completeness of results generated through a batch system versus the timeliness/ability to react quickly required for trading. The architecture proposed here also addresses some of the same trade-offs identified in the literature; whereas the Kappa Architecture is able to support timely responses to the business environment, it does so at the cost of being inefficient in the calculation of large portions of historical data [21]. This limitation has led to the development of hybrid architectures which are attempting to combine the reliability/completeness of results provided by batch processes with the timeliness to react to changing conditions

provided by real-time data streams [2, 5]. The architecture proposed here supports the hybrid model by embedding the dual path processing capabilities of the Lambda Model [1, 3] into the governance and transactional capabilities of the Lakehouse Model [8]; in doing so, the architecture provides low latency analytics along with accurate, scalable, and effective historical data storage, effectively mitigating the limitations previously identified in prior research.

3.1. Comparison with Current Architectural Models

The Medallion Lakehouse builds its layers on top of each other with the lowest quality (Bronze) being the most raw, and then cleaning to Silver, and then to curating Gold; whereas this architecture will take a significantly different approaches by integrating batch and stream processing in a single layer of a transactional Lakehouse. This will provide coherent and real-time analytics over the full lifecycle of data. Table 1 shows the high-level conceptual differences between Lambda, Kappa, Medallion Lakehouse and the proposed architecture.

Table 1. Comparison Between Lambda, Kappa, Medallion Lakehouse, and the Proposed Architecture

Architecture	**Processing Layers**	**Storage Model**	**Historical Reprocessing**	**Data Governance**
Lambda [1]	Batch + Speed + Serving	Separate (HDFS + NoSQL)	Periodic	Limited schema control
Kappa [14, 16]	Single unified stream	Log replay	Continuous (stream replay)	Weak consistency
Medallion (Bronze-Silver-Gold) [8]	Multi-tier (data quality refinement)	Delta / Parquet	Optional	Strong schema + lineage
Unified Lambda-Lakehouse (Proposed)	Batch + Speed + Serving (integrated)	Delta Lakehouse	Automated	ACID + schema enforcement + time travel

3.2. Layered Architecture Design

The architecture (see Figure 1) has been designed using three processing layers:

- *The Speed Layer* is responsible for handling high volumes of real-time data processing. Apache Kafka serves as the central broker, and receives real-time Bitcoin trading feeds from Binance via the Binance WebSocket API [10, 11]. Spark Structured Streaming processes the feeds immediately and creates short term indicators, e.g., Simple Moving Average (SMA), Exponential Moving Average (EMA), Relative Strength Index (RSI) and Average True Range (ATR). In addition, the processing layer assigns a correct timestamp to each message and places messages into micro-batches to ensure that downstream analytic processing occurs consistently, and that real-time streaming data flows are synchronized.
- *The Batch Layer* provides long term analytical depth and provides a stable historical base. All raw and pre-processed data is stored in Amazon S3, which offers scalable and cost-effective cloud storage options [15]. Every so often, Spark batch jobs process large parts of the historical repository and provide additional capabilities such as trend detection, validation and aggregation. As a result of performing periodic batch recomputation, the historical repository remains current and complete, and does not affect the ongoing continuous processing occurring in the Speed Layer.
- *The Serving Layer* unifies the output of both streaming and batch processing using a single environment built atop Delta Lake. Using the transactional guarantee features, schema consistency and time travel features of Delta, the Serving Layer will allow analysts to view data at the exact point in historical time as desired [8]. AWS Glue manages metadata centrally, and AWS Athena allows analysts to query S3-based storage, and

Delta tables using standard SQL queries. The components of the Serving Layer work together to provide coherent analytical results and ensure that all real-time updates are consistently aligned with historical records.

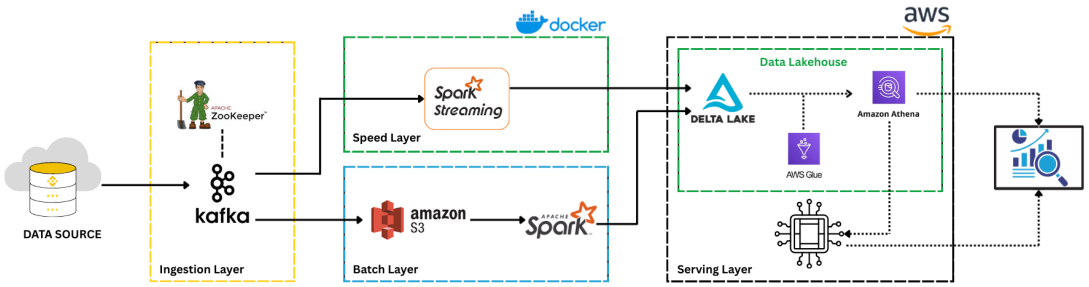

Figure 1. Structure of the Proposed Architecture Showing the Integration of the Batch, Speed, and Serving layers.

The architecture described above in Figure 1 consists of three interdependent layers (Speed, Batch and Serving) that allow it to support scalable real-time analytics as well as other types of analytics. The speed layer uses Spark Streaming along with Apache Kafka to rapidly process incoming data streams. The batch layer, which is based upon Spark and Amazon S3, processes the large amounts of historical data. When both layers have completed their processing of the data, they combine the data into the serving layer where it is stored using Delta Lake and AWS Glue. Once in the serving layer, the data may be queried using Athena and will be subject to all the ACID properties and schema governance associated with those technologies. Thus, through the combination of these different components of this architecture, users receive the same results regardless of the type of analysis that is performed and the system maintains the capability of efficiently processing large volumes of both real time and historical data.

This hybrid architecture has several benefits that distinguish it from prior hybrid architectures. First, the architecture decreases operational complexity by allowing batch and streaming data processing to take place on the same foundation of Delta Lake, eliminating the need for separate pipelines and solving the long-standing problem of synchronizing the results of batch and stream processing. Second, the architecture enhances the consistency and reliability of the system's results by utilizing ACID transactions and rigorously enforcing schemas; thereby, ensuring that the system produces trust worthy data for the entire length of the processing life cycle. Third, by using cloud-based services (S3, Glue, and Athena), the system can dynamically adjust how much storage and compute resources are used to process a workload; thus, providing the scalability needed to support varying workloads. Fourth, the architecture supports analyzing data at multiple levels of granularity (minute, hour, day), consequently, users can analyze both real-time and historical patterns of data. Fifth, the architecture represents a complete end to end workflow where ingestion, computation, storage and visualization all occur in a single environment.

4. Predictive Modeling and evaluation metrics

4.1. Methodology and Datasets

The data used in our experiment is based on historical prices of Bitcoin for over seven years that is from January 1, 2018 to May 16, 2025. The three different types of data resolution have been used: the daily dataset with 2,693 entries; the hourly dataset with 64,487 entries; and the minute level dataset with 3,841,850 entries [13]. During this time the minimum value of the price of Bitcoin was $3,211.72 while the maximum price was $106,143.82 as a result of strong growth of the Bitcoin price and its high volatility. In addition to the date of each record (the timestamp), we also know all six parameters of each transaction: the opening price of the transaction, the highest price of the transaction, the lowest price of the transaction, the closing price of the transaction and the trading

volume of the transaction. All data used has been previously processed by the Batch Layer in Spark before it was modeled: first, normalization of the data through Min-Max scaling; second, segmentation of the data into training (80%) and testing (20%) parts. This processing scheme is consistent with previous research on prediction of financial time series using deep learning [2, 14, 15].

Table 2. Characteristics of the Bitcoin Price Dataset

Dataset	**Records**	**Time Resolution**	**Description**
Daily	2,693	1 day	Long-term trend analysis
Hourly	64,487	1 hour	Balanced mid-term dynamics
Minute	3,841,850	1 minute	High-frequency volatility

4.2. Predictive Models

Four types of forecasting models were selected to evaluate the performance of the proposed system over multiple time horizons. These include an LSTM model and a GRU model as two types of Recurrent Neural Networks that can be used to identify and analyze trends within financial data that are volatile. The GRU model has been identified as a less resource-intensive option when compared to the LSTM model. The ARNN (autoregressive neural network), a model that identifies the linear components of financial price movements and captures the non-linear variations in those movements, was also evaluated. Finally, an XGBoost model was chosen to serve as a more traditional machine learning baseline. All four models were run separately on the daily, hourly, and minute datasets; the models were implemented in the same Lakehouse environment to ensure consistency in the evaluation process.

4.3. Evaluation Metrics

The performance of the forecasting models was assessed using three common error indicators: MAE, RMSE, and MAPE. These measures were selected because they are easy to interpret and are routinely used in financial time-series studies to compare predictive accuracy [22].

$$\mathrm{MAE} = \frac{1}{n}\sum_{t=1}^{n}|y_t - \hat{y}_t| \tag{1}$$

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{t=1}^{n}(y_t - \hat{y}_t)^2} \tag{2}$$

$$\mathrm{MAPE} = \frac{100}{n}\sum_{t=1}^{n}\left|\frac{y_t - \hat{y}_t}{y_t}\right| \tag{3}$$

4.4. Multi-Scale Forecasting and Visualization

Bitcoin trading is highly volatile and has shown large increases in speculative trading over two periods: 2021 and in the beginning of 2023 (see Figure 2). The total number of trades of Bitcoins reached values exceeding 700,000 units per day; these represent extreme speculative trading in this asset. In 2024 and 2025, the trading volume declined significantly and is far less liquid than previously indicating a less speculative environment in Bitcoin markets. The changes in the trading volume show that there are multiple market regimes operating within the Bitcoin markets which should be taken into consideration when developing predictive modeling techniques.

The next figure (Figure 3) illustrates how the closing price of Bitcoin was distributed throughout the entire study period. As indicated by the histogram, it was skewed to the right; most of the closing price observations were

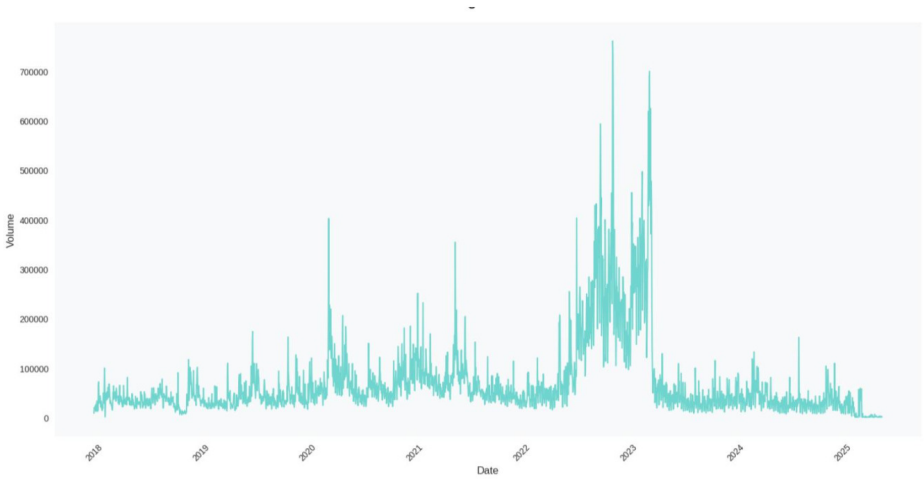

Figure 2. Bitcoin Trading Volume Over Time.

between \$0 and \$20,000 during the initial years of 2018-2020. However, after 2020, the closing price observations shifted upward. From 2021 through 2023, most closing price observations fell within the range of \$40,000-\$60,000, and from 2023 through 2025, closing price observations extended well above \$100,000. The shifting distribution of closing prices demonstrates the increasing value of the cryptocurrency, and also the increased volatility of the cryptocurrency. Additionally, the combination of the distribution of the closing price and the previous volume trends illustrate that Bitcoin markets fluctuate frequently and greatly. Therefore, such volatility must be addressed in the development of predictive models, especially predictive models that function at different frequencies i.e., daily, hourly, or minute-level frequency.

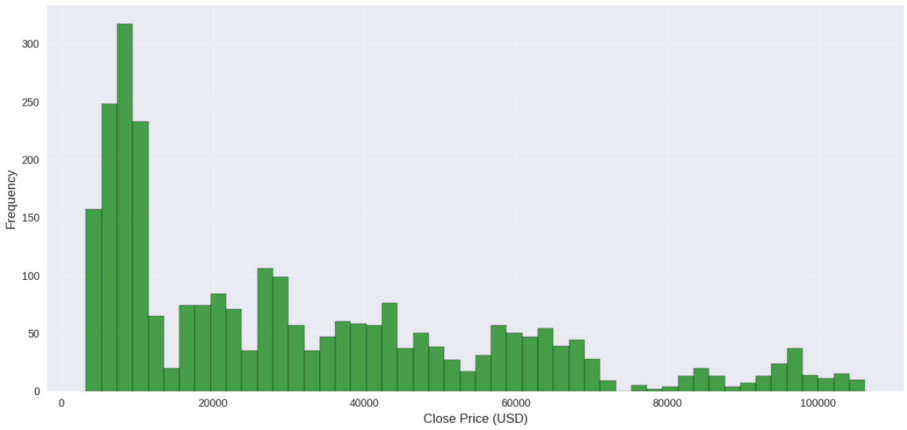

Figure 3. Distribution of Bitcoin Close Prices.

The following scatter plot demonstrates that most of the data points fall toward the bottom of the trading volume range (0–100,000) and the price range (\$0 – \$40,000); however, as trading volume and price increase, fewer and fewer observations occur. This indicates that the relationship between trading volume and price decreases as both values approach their upper limits. The visual and statistical representations assist in illustrating the behavior of the market, the market's volatility, and the underlying trends of the market that must be evaluated when developing predictive models and optimizing them for various time resolutions.

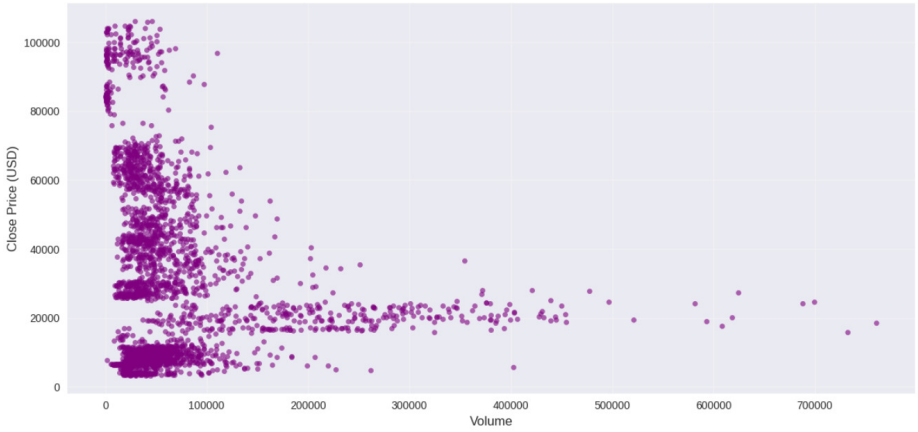

Figure 4. Relationship between Volume and Close Price.

These statistics and patterns are a good starting point for creating forecasting models because they indicate the underlying trends, volatility, and trader behavior that will influence which models are selected or modified for use at each of the three granularity levels.

Each of the forecasting models developed was outputted using Spark Structured Streaming sinks to a Delta table so the predictions could be viewed as they were being made. Figures 5- 7 compare the predicted versus the actual closing prices of Bitcoin over the course of daily, hourly and minute-long intervals.

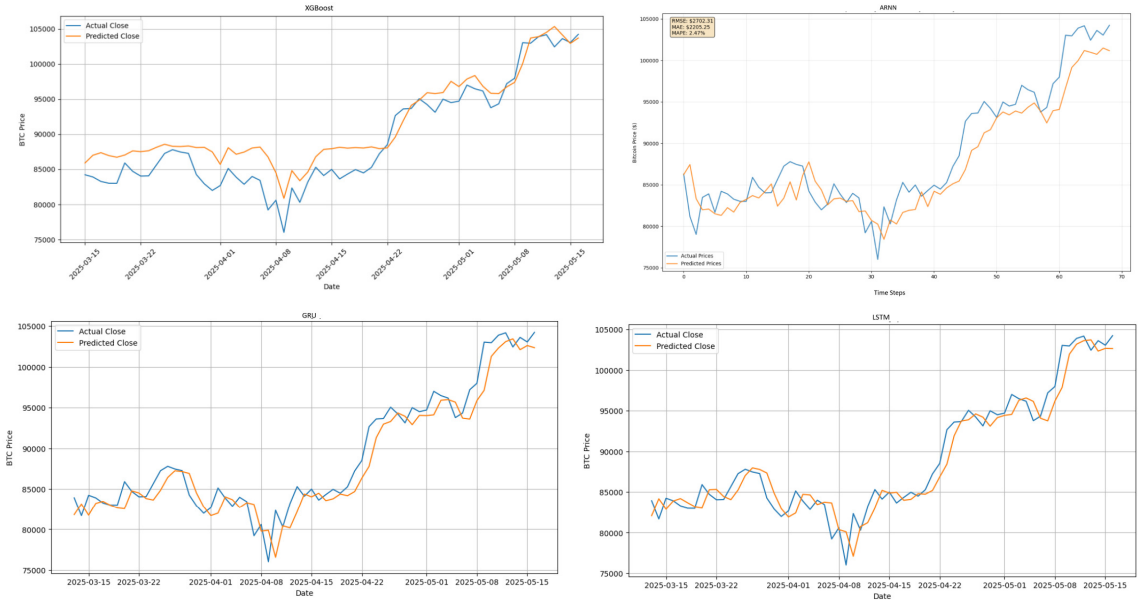

Figure 5. Comparative Analysis of True and Predicted Daily Bitcoin Closing Prices Using Machine Learning and Deep Learning Models.

LSTM is clearly the best model for predicting Bitcoin's daily closing prices (Figure 5), as it closely follows the overall direction of the market and the sudden spikes in the price of Bitcoin. The GRU model does nearly as well as LSTM, providing a good balance between smoothness and responsiveness. The ARNN model provides a reasonable approximation of the real value of Bitcoin, but generally falls behind when there is an increase in

volatility. XGBoost generally approximates the longer term trends of the market, but is poor at rapidly responding to sudden changes in the price of Bitcoin.

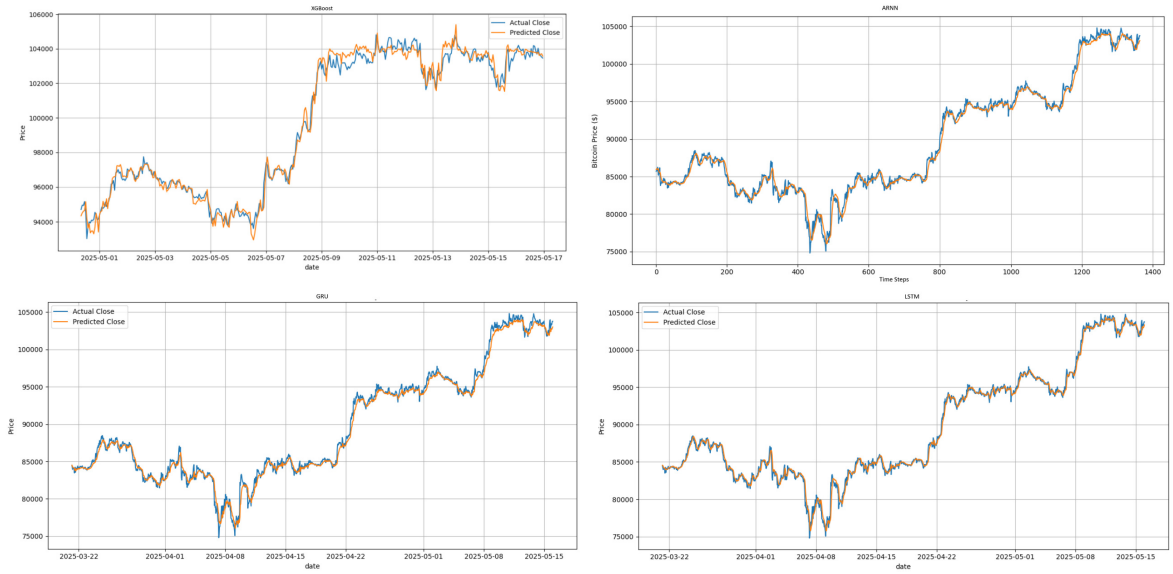

Figure 6. Comparative Analysis of Actual and Predicted Hourly Bitcoin Closing Prices Using Machine Learning and Deep Learning Models.

Both the LSTM and GRU models perform the best in terms of prediction accuracy at the hourly interval (Figure 6), providing a very close representation of the historical prices of Bitcoin and both capturing short-term price fluctuations as well as the long-term trends of the market. Although it does not perform as accurately as LSTM or GRU at the hourly interval, the ARNN model still performs well with only slight deviation from the historical prices of Bitcoin, although this deviation becomes slightly greater when the market becomes highly volatile. The XGBoost model provides reasonable predictions, but tends to have larger gaps than either the LSTM or GRU models during periods of rapid price movement.

The best performing models at the minute interval are the LSTM and GRU models (Figure 7), which are able to track the historical prices of Bitcoin with significant precision and respond quickly to sudden changes in the price of Bitcoin while maintaining the relative smoothness of their predictions. The ARNN model also performed well in terms of predictions, and captured many of the short-lived price movements; however, the accuracy of the ARNN model dropped off slightly when the market shifted direction rapidly. XGBoost can follow the long-term trends of the market, but is unable to adjust to the high frequency of volatility and noise present in minute-by-minute data. Overall, the proposed system has been shown to be well-suited for forecasting across multiple time scales, combining high frequency stream processing with deep sequence modeling into a single, unified system.

5. Experimental Results and Discussion

5.1. Experimental Setup

The experiments carried out in this study were performed within the Unified Lambda-Lakehouse Architecture introduced earlier. All computations were run locally on a standard personal computer equipped with an Intel Core i7-1165G7 (11th generation, 4 cores / 8 threads, 2.80 GHz), 16 GB of RAM. The system operated under Windows 11 (64-bit), an Intel Iris Xe graphics unit, and an NVMe solid-state drive.

All model development and all associated data processing occurred on a virtualized environment which used Kafka for ongoing ingestion of new data, Spark Structured Streaming for all streaming data processing and Delta

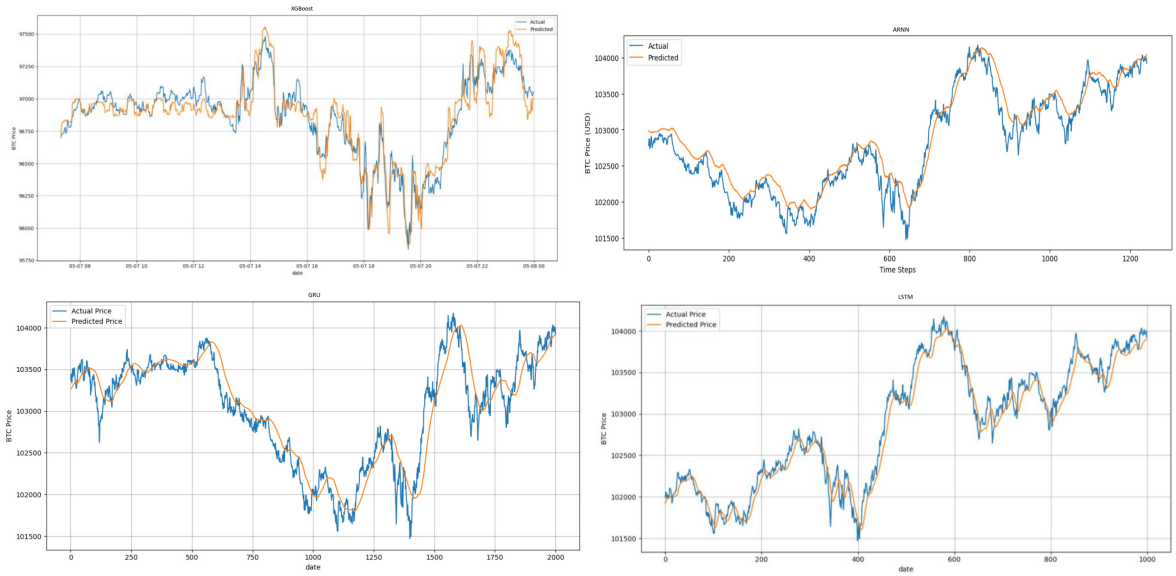

Figure 7. Comparative Analysis of Actual and Predicted Minute-Level Bitcoin Closing Prices Using Machine Learning and Deep Learning Models.

Lake as the unified data repository. All new data processed by this environment will be persisted via Amazon S3. Using such an environment has allowed for both reproducibility and isolation of each component involved in this data pipeline. In addition, the neural forecasting models (ARNN, LSTM, and GRU) were implemented using TensorFlow (version 2.13). Furthermore, XGBoost was implemented separately using the scikit-learn–compatible API provided by the XGBoost library (version 1.7). A random split of the full set of the Bitcoin dataset time-series data, at the level of daily, hourly and minutes from 2018 to 2025, resulted in the establishment of a test size of 20% and a train size of 80%. In order to optimize performance of the neural networks' hyperparameters, each model was optimized utilizing early stopping methods and adaptively adjusting the learning rates. The neural architectures utilized are comprised of two hidden layers that contained 64 units, used the activation function ReLU, had a dropout of .2 and were trained using the Adam optimizer at a learning rate of .001. Each of these architectures was then evaluated based upon the standard metric of MAE, RMSE and MAPE as defined by equations (1) to (3).

5.2. Forecasting Results

The LSTM model was able to achieve the smallest error in relation to all time scales analyzed in Table 3; this is a result of its excellent capabilities in identifying sequential behavior and non-linear relationships that exist throughout highly volatile markets. The differences in forecasting capability among the models were most evident at the minute level of analysis, where the price of assets fluctuates the most. The GRU model performed very similarly to LSTM in terms of accuracy; however, it had slightly larger errors which resulted from GRU's efficiency and lower computational requirements. Both ARNN and XGBoost struggled to adapt to changes in the market; however, they still performed reasonably well. Regardless of the model used, the average absolute percentage error remained under 2% demonstrating that all models provided stable forecasts that were also extremely accurate. Figures 5 through 7 show the correlation between the actual and forecasted closing prices of Bitcoin. At the daily level, both LSTM and GRU modeled the overall trends in the market well, while XGBoost mostly captured the overall trend in the market. However, when looking at more detailed time resolutions (hourly and minute data) the forecasted curves generated by LSTM were smooth and much more accurate than those of the other models, therefore, validating its use for generating high frequency financial forecasts.

Based on the above, the proposed architecture allows for a dynamic model selection strategy based upon the adaptive selection of the best performing model at each respective temporal granularity. Models can be evaluated

Table 3. Models Performance Comparison

Granularity	Model	RMSE	MAE	MAPE
Daily	*LSTM*	2383.90	1725.25	1.86%
	GRU	2471.04	1827.14	1.96%
	ARNN	2702.31	2205.25	2.47%
	XGBoost	3043.51	2538.37	2.98%
Hourly	*LSTM*	539.29	370.28	0.42%
	GRU	648.09	466.03	0.52%
	ARNN	691.98	483.65	0.55%
	XGBoost	1263.75	902.72	1.07%
Minute	*LSTM*	144.92	119.68	0.12%
	GRU	194.65	138.49	0.14%
	ARNN	226.69	196.45	0.20%
	XGBoost	275.74	210.63	0.22%

using the rolling performance metric of choice (i.e., RMSE, MAE) and the model that has the highest recent performance can be selected automatically for future predictions. For example, if a particular temporal granularity requires a low frequency stream, then tree-based models would be more appropriate. Conversely, for high frequency streams, recurrent neural network models would be more applicable. The adaptive model selection process can be added to the serving layer of the proposed architecture and can be updated at regular intervals; therefore, the proposed architecture will automatically adjust to changes in the input data streams without requiring manual intervention.

5.3. Unified Architectural Performance Evaluation

In order to adequately assess the operational behavior of the proposed architecture we conducted an analysis of the architecture's performance in terms of three main criteria: End-To-End Latency, Batch Processing Efficiency, and Resource Utilization. The three criteria above are representative of how well the system can produce "real-time" prediction models, recalculate historical data, and operate under typical workstation limitations. System-Level Performance in terms of all evaluation criteria.

The results from the evaluation reported in Table 4 demonstrate that the system responds appropriately across all types of workload; i.e., 1,000 messages/s yielded a mean latency of 82ms (118ms @p95), and at 50,000 messages/s, the mean latency was 244ms and 309ms @p95. Similarly, batch processing performance followed this same trend with the Daily (2.6k rows), Hourly (64k rows), and Minute level datasets (3.8m rows) being recomputed in 2.1s, 6.4s, and 39.8s respectively. In addition to the previously mentioned trends, Resource utilization remained under control. At 50,000 messages/s, streaming workloads had CPU utilization ranging from 52% to 68% and Memory utilization ranging from 48% to 62%. Additionally, at the same rate, write-throughput using Delta Lake ranged from 70mb/s to 110 mb/s. It was to be expected that batch jobs would require additional resources than their streaming counterparts, however the resource utilization for both were below those of a 16 GB workstation. Overall, the results presented here clearly support the efficiency and stability of the architecture when subjected to common operating conditions.

5.4. Scalability and Resilience Evaluation

In addition to the performance measurements, we analyzed behavior when the ingestion rate increased and responses to planned failures. Those two dimensions have been important for demonstrating the reliability of a streaming/batch combined architecture, as shown in the results listed in Table 4.

Table 4. Experimental Performance Evaluation of the Unified Lambda–Lakehouse Architecture

Metric Category	Condition / Scenario	Measured Value	Interpretation
End-to-End Latency	1,000 msg/s	82 ms (avg), 118 ms (p95)	Real-time responsiveness
	10,000 msg/s	139 ms (avg), 177 ms (p95)	Stable under moderate load
	50,000 msg/s	244 ms (avg), 309 ms (p95)	Supports high-frequency ingestion
Batch Processing Time	Daily dataset (2.6K rows)	2.1 s	Fast historical reprocessing
	Hourly dataset (64K rows)	6.4 s	Efficient mid-scale processing
	Minute dataset (3.8M rows)	39.8 s	Complete rebuild in under one minute
Resource Utilization	CPU (streaming @ 50K msg/s)	52–68%	Balanced multi-core load
	Memory (streaming)	48–62%	Stable usage, no swapping
	Delta Lake write throughput	70–110 MB/s	Consistent NVMe performance
	CPU (batch jobs)	72–88%	Expected under heavy computation
	Memory (batch jobs)	65–78%	Fits within 16 GB RAM
	Simulated S3 read throughput	95–140 MB/s	Good sequential scan speed
Scalability	Kafka ingestion up to 50K msg/s	Linear throughput	Demonstrates horizontal scaling
	Spark micro-batch duration	< 400 ms	Maintains streaming responsiveness
Fault Tolerance	Kafka broker failure	Leader re-election: 2.3–3.1 s; no message loss	Replication ensures availability
	Spark executor failure	Checkpoint recovery: 1.4–1.9 s; no lost or duplicated batches	Streaming continuity preserved
	Delta Lake interrupted commit	Rollback < 500 ms; no corrupted files	ACID guarantees consistent state

- **Scalability to Increasing Ingestion Rates:** The system was tested at 1000 msgs/sec., 10,000 msgs/sec., and 50,000 msgs/sec. in all cases, the average time for spark micro-batches remained below 400 ms and no backlog accumulation occurred; additionally, the rate of throughput increased nearly linearly with increasing message ingestion rate indicating that when proper parameters are used to partition data and configure micro-batch execution, horizontal scaling is achievable.
- **Resiliency/Fault Tolerance:** We evaluated the following fault-tolerant mechanisms:
  Kafka replication: A broker failure resulted in automatic election of a new leader within 2.3 - 3.1 sec., with no message losses.

Spark checkpointing: Streaming was recovered from executor failure in 1.4 - 1.9 sec., without any missing or duplicate batches.
Delta lake ACID recovery: An interrupted commit was rolled back in less than 500 ms, restoring last valid snapshot without corrupted files.

These results confirm that the architecture provides continuity, and ensures integrity of the data even if individual components fail.

5.5. Data Governance and Model Management

The Unified Lambda-Lakehouse Architecture treats data governance as an intrinsic part of providing a reliable and repeatable platform for streaming and batch data processing. Data governance will be provided by utilizing Delta Lake's ability to provide transactional guarantees and version all data sets systematically. Because each new ingestion or data manipulation creates a new committed state in the Delta transaction logs, the system maintains a complete history of every modification made to the data set; therefore, at any given point in time, a previous version of the data set can be accurately reconstructed. Therefore, any analytical study or model training can be replicated using exactly the same data snapshot that was used when the study/model was initially created and is critical to enabling audits.

The architecture is designed around a layered Bronze-Silver-Gold data storage model. The Bronze layer holds raw ingested data in a stable format (Schema S0). The Silver layer cleanses and normalizes the data while holding onto the original schema. This allows data to be maintained throughout the ingestion and preparation phases. The Gold layer adds derived features and allows for controlled schema evolution (Schema S1) so that new features can be added without affecting the downstream applications. A schema validation mechanism prevents invalid or incomplete data from being stored, while schema evolution ensures that any changes to the structure of the data will be versioned over time. This will allow models to be safely re-trained as the feature space expands without losing the ability to reproduce the results historically.

Model management is also closely tied to the data versioning approach described above. Each model is linked to the specific version of the data from the Delta Lake dataset that was used to train that model. As long as there is a clear association between the model behavior and the data that was used to train that model, then model behavior can always be traced back to its source data. While the current implementation of the unified lambda-lakehouse architecture focuses on versioned model artifacts, the architecture has been designed to support the integration of a model registry such as MLflow to store hyperparameters, evaluation metrics, and model lifecycle stages. With this level of integration, the transparency and governance will be extended from data ingestion to model deployment.

5.6. Deployment and Operational Considerations

The Unified Lambda-Lakehouse Architecture has been designed with an emphasis on the ability to deploy this architecture in practice, as well as its operational feasibility. The experimental analysis was carried out in a local environment; however, there are no limitations to where it can be deployed, it can be used in either on-premise, cloud based, or hybrid environments. This design provides for each module to be deployed and operated independently. Therefore, in addition to the modularity, if a user wishes to use their own resources (e.g., the need to accommodate specific hardware) it will also allow them to do so.

In addition to accommodating the needs of users to adapt to different infrastructures, the architecture also supports both container-based deployments as well as scalable resource management solutions. In order to support the operational requirements of users that require standardization across all systems (i.e., the ability to observe what is happening in the system at any point), the architecture is also compatible with standard monitoring and logging mechanisms, allowing for system-wide observation as well as long term maintainability. Additionally, the unification of the Lakehouse foundation eliminates the operational overhead associated with the duplicate data and avoids the duplicate processing pipeline. A list of the key operational and deployment considerations for this architecture are listed in Table 5.

Table 5. Deployment and Operational Aspects of the Unified Lambda–Lakehouse Architecture

Aspect	Related Components	Operational Considerations
Deployment flexibility	Kafka, Spark, Delta Lake	The architecture can be deployed on-premises, in cloud environments, or in hybrid settings, as it does not rely on infrastructure-specific assumptions.
Component modularity	Kafka, Spark, Delta Lake	Each component can be deployed and managed independently, allowing gradual integration and easier adaptation to infrastructure constraints.
Container-based deployment	All components	Container technologies facilitate reproducible deployments, simplify configuration, and improve portability across different environments.
Resource management	Spark, Kafka	Resource management platforms support workload distribution, scalability, and fault isolation, especially in large-scale or dynamic deployments.
Monitoring	Streaming and batch pipelines	Monitoring tools can be used to track ingestion throughput, processing latency, and system stability during operation.
Logging and maintenance	All components	Logging mechanisms support debugging, auditing, and long-term maintenance of the system.
Operational overhead	Storage and computation layers	The unified Lakehouse design reduces operational overhead by limiting data duplication and avoiding redundant processing pipelines.

5.7. System Achievements and Implementation Insights

Beyond just measuring the accuracy of forecasts, these experiments demonstrated the performance of the proposed architecture to handle large volumes of streaming as well as batch processing. The use of both the Speed Layer (built with Spark Structured Streaming and Kafka) and the Batch Layer (using Spark batch jobs and AWS S3), along with a common, transactional, unified structure for storing data (via the Delta Lake technology), resulted in a scalable and real-time system with the ability to enforce ACID properties, maintain schema integrity and provide perfect alignment between real-time and historical data.

In addition to minimizing the delay from when data is ingested to when models are served, this integrated design provided an environment where all analyses were consistently represented regardless of the time resolution. Lastly, it provided an environment that was reproducible so large-scale reprocessing and incremental updates could occur simultaneously without duplicating effort or losing consistency.

To make this system usable for both experts and non-experts alike, a user-friendly interface was built using the Flask framework. This lightweight web interface is intended to allow users to visualize in real time how they are interacting with the proposed forecasting system as well as interact with it. The interface supports observing financial indicators at various scales of time as well as observing the predictions produced by the system.

The interface (as shown in Figures 8– 9) has two sections; one section is live visualizations of Bitcoin's closing price as well as four technical indicators: Simple Moving Average (SMA), Exponential Moving Average (EMA), Relative Strength Index (RSI), and Average True Range (ATR). The second section of the interface contains summary statistics, a chart that shows a distribution of Buy-Sell, and a Forecasting Module, which allows users to see what predicted trend may occur for each horizon of minutes, hours, or days.

By creating a front-end interface that is accessible to all users while making the system's analytical back-end available to those who have the skills to interpret the output from the back-end, the dashboard provides a way to

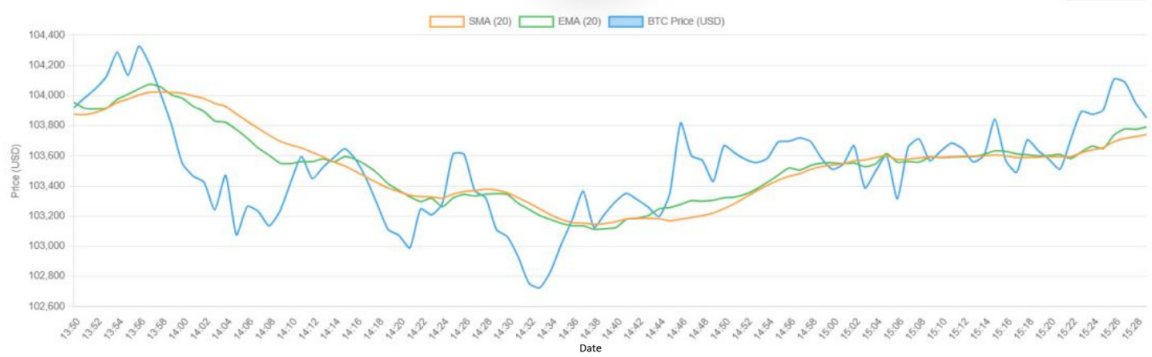

Figure 8. Price Monitoring with SMA/EMA in Streaming.

Figure 9. Link for Price and Technical Indicators Predictions.

make the system practical for all types of users, while increasing the usability and interpretability of the system's results in real world financial applications.

As illustrated in Figures 10– 12, the platform can generate forecasts at the daily, hourly, and minute scale, each with a 15-step prediction horizon. These outputs highlight the flexibility of the architecture, which can support both long-term market analysis and high-frequency trading needs. The strong agreement between the predicted curves and the real price movements shows the reliability of the LSTM and GRU models, especially at the more granular levels where market volatility tends to be most pronounced.

5.8. Architectural Discussion

The Lambda architecture has a high level of architectural complexity (both in terms of operation and analysis). It uses two separate pipelines: a speed layer, for low latency, and a batch layer, for higher precision. There are challenges associated with managing these two layers, including potential for high synchronization costs (especially in distributed environments), and possible temporary inconsistencies between real-time and batch computed results. The Kappa architecture simplifies the problem by providing a single streaming layer that can replay log data; however, it is not ideal for those that need to process large amounts of historical data for extended periods of time. The proposed approach unites the best features of both worlds: the rich analytical capabilities of the Lambda model, and the streamlined organizational structure of Lakehouse systems. By adding ACID transactions, schema control, and time travel capabilities via Delta Lake, the architecture significantly reduces the common

Figure 10. Price Predictions: Day-Level, Future Step 15.

Figure 11. Price Predictions: Hour-Level, Future Step 15.

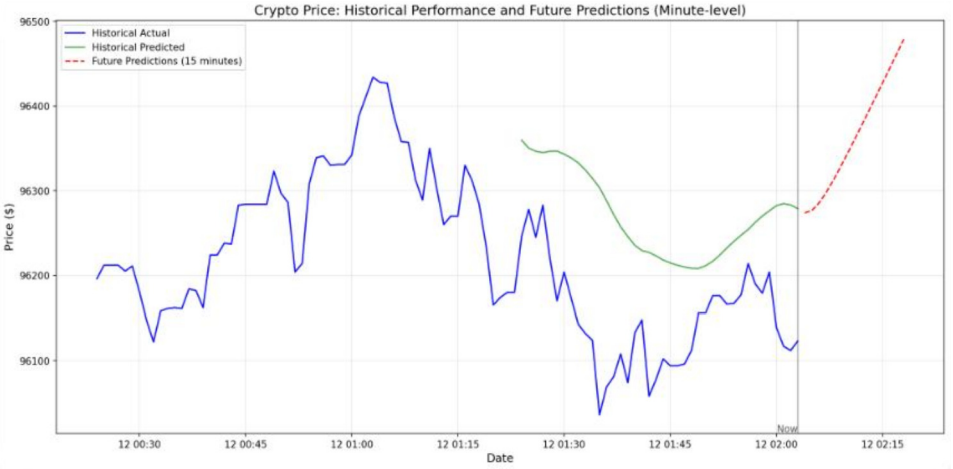

Figure 12. Price Predictions: Minute-Level, Future Step 15.

coordination and maintenance problems that occur in Lambda-style systems. Ultimately, the approach provides a more balanced solution between performance, reliability, and management cost.

While there are many similarities between the proposed architecture and other architectures such as Lambda and Kappa, the architecture has many implications for robustness and longevity. First, since the architecture utilizes a single, transactional Lakehouse core, it minimizes the redundancy associated with multiple pipelines (speed vs. batch) and thus minimizes the likelihood of coordination problems arising when attempting to run both types of computation at the same time. Second, the design will make it easier to maintain and update the system, and minimize the likelihood of the kinds of inconsistencies that commonly arise when using multi-layer architectures. Third, the proposed architecture does not restrict the user to utilize only one type of model or to utilize only a limited number of temporal resolutions. Rather, the architecture is flexible and enables users to deploy and evaluate different models based upon their specific needs and the specifics of their data and market conditions. Given the dynamic nature of financial markets, the ability to adapt to evolving statistical characteristics over time, and to use models that are well-suited to particular timescales, this flexibility is very desirable. Lastly, due to the modular design of the ingestion, processing, storage, and serving components, the architecture is very easily extendable. Therefore, new data feeds, models, or financial instruments can be added to the system without having to completely redesign the architecture. Thus, the proposed architecture represents a sustainable architecture for financial analytics systems that have to operate reliably, yet continuously adapt to changes in the nature of the data, and changes in the analytics requirements.

In conclusion, the experiments demonstrate that the proposed architecture is able to integrate the low-latency streaming capabilities of real-time processing with the high-accuracy batch analytic capabilities of Delta Lake into a single scalable system. Furthermore, the deep learning models, especially LSTM, provide high levels of predictive accuracy, even under extreme volatility in the markets. Integrating real-time processing, unified storage, and advanced forecasting techniques provides a solid foundation for the development of next generation financial analytics, and demonstrates the feasibility of developing adaptive, multi-asset forecasting systems that can operate efficiently in highly volatile market environments.

6. Conclusion and Future Works

The study presents a new unified architecture for addressing limitations of existing big-data architecture designs in real time applications of financial data analytics by merging the two-tiered nature of the lambda model with a lake-house architecture to integrate batch and stream operations to a single operational unit, thus providing both timely, accurate analytics and low latency operation. The architecture integrates several technologies such as Apache Kafka for high throughput ingestions, Spark Structured Streaming for real-time computations, AWS S3 for scalable storage and Delta Lake for ACID compliant data management and Flask-based to create a lightweight dashboard for visualization and real-time user interaction to enhance usability and provide real-time monitoring of the system.

The experimental results utilizing Bitcoin indicated the system was able to perform the entire pipeline from ingestion to prediction using a unified process. Of the evaluated models, the LSTM network produced the highest performance at each level of granularity, achieving both low values of root mean squared error (RMSE) and mean absolute percentage error (MAPE), even when the data exhibited turbulent patterns. Results from the GRU model were comparable in terms of performance but required less computational power than the LSTM model, and results from the ARNN and XGBoost models provided useful baseline measures for evaluating trend analysis. The results indicate that the use of a unified architecture resulted in both increased computational efficiency and consistent data management while providing a solid foundation for conducting deep-learning driven forecasting under volatile conditions. Additionally, the study illustrates how advanced predictive models can be paired with modern data engineering tools to create a single architecture. From a conceptual perspective, the proposed architecture integrates hybrid architecture models in an ACID governed environment to bridge between the need for real-time streaming data and the need for historical data analysis. From a practical perspective, the proposed architecture provides a

scalable and reproducible workflow that can be used in various analytical environments that require high frequency data and reliable predictions.

Future development will focus on extending the scope of the study to additional financial data sets, including foreign exchange (Forex) and equity markets, in order to validate the generality of the proposed architecture beyond a single asset class. This extension will enable a systematic evaluation of the framework across markets exhibiting diverse statistical characteristics, ensuring that its performance is not specific to cryptocurrency data. Furthermore, the architecture will be enhanced to support multi-asset forecasting, allowing concurrent analysis of cryptocurrency, Forex, and equity data within a unified processing pipeline. This capability will facilitate the investigation of cross-market dynamics and inter-asset relationships without requiring modifications to the underlying system design.

In addition to expanding the scope of the study to include additional data sets and developing the system to support multi-asset forecasting, future work will also focus on improving the adaptability of the predictive layer through real-time model retraining and concept drift detection. Financial markets are constantly changing and changes in market regime or data distribution over time can negatively impact the effectiveness of forecasting models that do not adapt. To mitigate this issue, the proposed architecture can be augmented with mechanisms to monitor the accuracy of forecast outputs and detect shifts in data distribution during streaming execution. When significant drift is detected, controlled retraining of the forecasting model can be initiated using the latest available data in the Lakehouse and continuous inference can be maintained. The use of Delta Lake's versioning and reproducibility features allows researchers to validate the new version of the forecasting model against previous versions prior to deployment, thereby ensuring that the model evolution process is both traceable and reliable.

Concurrently, the predictive layer will be enhanced through the inclusion of transformer-based models such as the Temporal Fusion Transformer (TFT) and Informer, which have demonstrated the ability to learn long-range temporal dependencies and improve interpretability relative to traditional recurrent models. Other enhancements to the predictive layer will include automated alert generation and adaptable model selection strategies to further evolve the dashboard into a fully interactive decision-support system that can continuously learn and respond to the changing dynamics of financial markets.

REFERENCES

1. M. Kiran, P. Murphy, I. Monga, J. Dugan, and S. S. Baveja, *Lambda architecture for cost-effective batch and speed big data processing*, in Proc. of the IEEE International Conference on Big Data (Big Data), pp. 2785–2792, 2015, doi: 10.1109/BigData.2015.7364082.
2. M. U. Demirezen and T. S. Navruz, *Performance analysis of lambda architecture-based big-data systems on air/ground surveillance application with ADS-B data*, Sensors, vol. 23, no. 17, 2023, doi: 10.3390/s23177580.
3. J. Warren and N. Marz, *Big Data: Principles and Best Practices of Scalable Realtime Data Systems*, Simon and Schuster, 2015.
4. Z. Hasani, M. Kon-Popovska, and G. Velinov, *Lambda architecture for real time big data analytic*, in Proc. of ICT Innovations, pp. 133–143, 2014.
5. M. Gribaudo, M. Iacono, and M. Kiran, *A performance modeling framework for lambda architecture based applications*, Future Generation Computer Systems, vol. 86, pp. 1032–1041, 2018, doi: 10.1016/j.future.2017.07.033.
6. M. E. M. El Aissi, S. Benjelloun, Y. Lakhrissi, and S. El Haj Ben Ali, *A scalable smart farming big data platform for real-time and batch processing based on lambda architecture*, Journal of System and Management Sciences, vol. 13, no. 2, pp. 17–30, 2023, doi: 10.33168/JSMS.2023.0202.
7. J. B. Nkamla Penka, S. Mahmoudi, and O. Debauche, *A new kappa architecture for IoT data management in smart farming*, Procedia Computer Science, vol. 191, pp. 17–24, 2021, doi: 10.1016/j.procs.2021.07.006.
8. A. A. Harby and F. Zulkernine, *Data lakehouse: A survey and experimental study*, Information Systems, vol. 127, p. 102460, 2025, doi: 10.1016/j.is.2024.102460.
9. J. Yasmin, J. A. Wang, Y. Tian, and B. Adams, *An empirical study of developers' challenges in implementing workflows as code: A case study on Apache Airflow*, Journal of Systems and Software, vol. 219, p. 112248, 2025, doi: 10.1016/j.jss.2024.112248.
10. S.-A. Ionescu and A.-O. Radu, *Assessment and integration of relational databases, big data, and cloud computing in financial institutions: Performance comparison*, in Proc. of the International Conference on Innovations in Intelligent Systems and Applications (INISTA), pp. 1–7, 2024, doi: 10.1109/INISTA62901.2024.10683852.
11. T. P. Raptis, C. Cicconetti, and A. Passarella, *Efficient topic partitioning of Apache Kafka for high-reliability real-time data streaming applications*, Future Generation Computer Systems, vol. 154, pp. 173–188, 2024, doi: 10.1016/j.future.2023.12.028.
12. J. Gupta Nikhil and Yip, *Spark structured streaming: A comprehensive guide*, in *Databricks Data Intelligence Platform: Unlocking the GenAI Revolution*, Apress, pp. 409–429, 2024, doi: 10.1007/979-8-8688-0444-1_18.
13. N. Anugrah, *Bitcoin historical datasets (2018–2024)*, Kaggle, 2024, doi: 10.34740/KAGGLE/DS/6055749.
14. M. Maatallah, M. Fariss, H. Asaidi, and M. Bellouki, *An effective real-time comparative analysis of lambda and kappa architectures*, in Proc. of the International Conference on Circuit, Systems and Communication (ICCSC), pp. 1–6, 2025,

doi: 10.1109/ICCSC66714.2025.11135352.
15. M. Fariss, M. Maatallah, B. B. A. Y. Bay, H. Asaidi, and M. Bellouki, *Enhancing forex trading predictions with machine learning: Cloud and local performance evaluation*, in Proc. of the International Conference on Intelligent Computing in Data Sciences (ICDS), pp. 1–8, 2024, doi: 10.1109/ICDS62089.2024.10756412.
16. J. Kreps, *Questioning the lambda architecture*, Online article, 2014.
17. S. Azzabi, Z. Alfughi, and A. Ouda, *Data lakes: A survey of concepts and architectures*, Computers, vol. 13, no. 7, 2024, doi: 10.3390/computers13070183.
18. J. Schneider, C. Gröger, A. Lutsch, H. Schwarz, and B. Mitschang, *The lakehouse: State of the art on concepts and technologies*, SN Computer Science, vol. 5, no. 5, p. 449, 2024.
19. S. Ait Errami, H. Hajji, K. Ait El Kadi, and H. Badir, *Spatial big data architecture: From data warehouses and data lakes to the lakehouse*, Journal of Parallel and Distributed Computing, vol. 176, pp. 70–79, 2023, doi: 10.1016/j.jpdc.2023.02.007.
20. J. Tagliabue and C. Greco, *Reproducible data science over data lakes: Replayable data pipelines with Bauplan and Nessie*, in Proc. of the Workshop on Data Management for End-to-End Machine Learning (DEEM '24), ACM, pp. 67–71, 2024, doi: 10.1145/3650203.3663335.
21. D. Owczarek, *Lambda vs. kappa architecture: A guide to choosing the right data processing architecture for your needs*, Online article, 2022.
22. O. B. Sezer, M. U. Gudelek, and A. M. Ozbayoglu, *Financial time series forecasting with deep learning: A systematic literature review: 2005–2019*, Applied Soft Computing, vol. 90, p. 106181, 2020. doi: 10.1016/j.asoc.2020.106181.