Vol.3 No.1 February 2026: 27-32
International Journal of Computer Science and Humanitarian AI (IJCSHAI)
e-ISSN: 3064-4372
DOI: 10.21512/ijcshaijournal.v3i1.15154

An End-to-End Architecture for Stock Market Prediction Integrating Mobile Application, Backend Services, and ML/DL Models

Abraham Kefas Wilham[1*], William[2], Sonya Rapinta Manalu[3]

[1-3]Mobile Application & Technology Program,
Computer Science Department, School of Computer Science,
Bina Nusantara University,
Jakarta, Indonesia 11480
abraham.wilham@binus.ac.id; william092@binus.ac.id;
sonya.manalu@binus.ac.id

***Abstract*—Prior research on stock market prediction has predominantly focused on algorithmic accuracy, leaving a significant research gap in the system-level realization required for real-world delivery. This paper addresses this disparity by presenting an end-to-end stock prediction delivery system that operationalizes trained machine learning models within a mobile-centric architecture. Unlike model-centric studies limited to offline evaluation, this work focuses on the rarity of system-level implementation. Market data are periodically ingested into a managed relational database, where predictions are generated using a fixed historical window and persisted for downstream access. A cross-platform mobile application serves as the primary user interface, providing structured access to historical prices, predictions, and accuracy metrics via backend APIs without local model inference. A key novelty is the implementation of an in-memory caching layer to optimize responsiveness for repeated mobile access. Experimental results demonstrate that this architecture significantly improves efficiency, reducing average API response times by approximately 94% from 817 ms to 48,7778 ms compared to direct database queries. These findings underscore the critical role of mobile-oriented system design in bridging the gap between predictive modeling and practical deployment.**

***Keywords*—Stock Market Prediction; End-to-End System; Machine Learning Models; Backend Services; Mobile Application; Data-Driven Application**

I. Introduction

Stock market prediction has been extensively studied using machine learning and deep learning techniques due to the complex and non-linear nature of financial time-series data [1]. Prior research has explored numerous Machine Learning and/or Deep Learning model architectures, feature engineering methods, and training strategies in an effort to improve prediction accuracy and robustness. These studies primarily focus on model-level challenges, such as data gathering, the model creation, the model's sensitivity to market volatility, and the model's generalization across different stock profiles [2], [3].

In the author's previous work, the author have evaluated and compared machine learning and deep learning models for stock market prediction on the Indonesian Stock Exchange (IDX) using data from 46 publicly listed stocks, primarily comprising the LQ45 index, which represents the 45 most liquid stocks as of July 2025, along with the Indonesian Composite Index (IHSG) [3]. Although these studies are valuable for understanding predictive behavior under controlled experimental conditions, deploying such models in real-world applications requires considerations that extend beyond offline evaluation.

In real-world applications, predictive models are typically embedded within larger software systems that manage data ingestion, model execution, result storage, and user interaction. As a result, the research focus shifts from the question of whether a model can generate accurate predictions to how those predictions can be reliably delivered and consumed. Studies of real-world ML systems document the architectural and data-flow challenges of integrating predictive models into production infrastructure beyond model training itself [4] and highlight how key operational capabilities such as deployment automation, monitoring, and lifecycle management are critical to reliable model serving [5]. System-level considerations, such as backend architecture, data access strategies, and user-facing interfaces, thus play a crucial role in enabling the effective use of predictive models. Ultimately, these models are built to provide value to users or support decision-making for their beneficiaries [6].

Although stock market prediction has been widely studied, most prior research emphasizes model accuracy in offline settings, often overlooking system-level integration and deployment considerations. As a result, the practical realization of trained prediction models within fully integrated application environments remains underexplored.

Received: Jan. 8, 2026; received in revised form: Mar. 8, 2026; accepted: Mar. 18, 2026; available online: Mar. 30, 2026.

Corresponding: abraham.wilham@binus.ac.id

Thus, this paper addresses this system-oriented perspective by presenting the design and implementation of an end-to-end stock market prediction application. Rather than introducing new predictive algorithms, the work focuses on operationalizing trained models through a structured backend architecture and a mobile-based user interface. The proposed system emphasizes reliable delivery and accessibility of prediction results within an integrated application environment.

Building upon this system-oriented perspective, this paper makes several key contributions. First, it presents the design of an integrated application architecture for delivering stock market prediction results in a stable and consumable manner. Second, it details the implementation of backend services and caching mechanisms that support efficient and scalable data access for mobile clients. Third, the paper provides an applied evaluation of system realization and discusses architectural trade-offs observed during the development of a prototype mobile application.

The remainder of this paper is organized as follows. Section II reviews related work on stock market prediction systems and supporting infrastructures. Section III describes the proposed system architecture and methodology. Section IV presents the results and discussion. Finally, Section V concludes the paper and outlines directions for future research.

II. Literature Review

A. ML/DL Models in Stock Prediction Applications

Machine learning and deep learning models used for stock market prediction commonly operate on historical financial time-series data and are typically trained and evaluated offline. These models range from traditional machine learning approaches to deep learning architectures designed to capture temporal dependencies in market behavior. The author's previous paper delves into the XGBoost, GRU, and LSTM models [3][7]. Due to the computational cost of model training and inference, as well as the sensitivity of predictions to data update frequency, such models are often integrated into applications through precomputed or periodically updated prediction outputs. These characteristics motivate system designs that separate model execution from user-facing components and emphasize efficient storage and delivery of prediction results [8].

B. Backend Infrastructure and Data Access Optimization for ML Applications

The deployment of machine learning models in real-world applications commonly requires supporting backend infrastructure to manage data storage, access, and service orchestration [8]. Backend-as-a-Service (BaaS) platforms have been increasingly adopted in research prototypes and applied systems due to their ability to simplify development through managed databases, authentication services, and built-in API support [9]. In this context, Supabase represents a modern open-source BaaS platform that provides relational database management and RESTful access mechanisms, making it suitable for rapid system development and experimental applications [10]. Efficient data access is a critical consideration in data-driven systems, particularly when prediction results are accessed repeatedly by client applications. Caching is a widely used architectural strategy to reduce database load and improve response latency. In-memory data stores such as Redis are commonly employed to cache frequently accessed data, enable faster retrieval, and improve system responsiveness. Prior studies and applied systems have demonstrated the effectiveness of caching mechanisms as part of scalable backend architectures [11], [12].

C. Mobile-Based Interfaces for Financial and Decision-Support Applications

Mobile interfaces play an increasingly important role in financial and decision-support applications by enabling users to access predictive information conveniently and in real time. Prior studies have presented mobile-based dashboards and visualization tools to deliver analytical results and support user decision-making in applied contexts. In these systems, the user interface primarily functions as an access layer to underlying analytical or predictive components rather than as a primary focus of methodological contribution [13]. Flutter has been widely adopted as a cross-platform mobile development framework in both research prototypes and applied systems, allowing a single codebase to support multiple platforms. It's used to facilitate the rapid development of functional interfaces that can consume backend services and display prediction results. In this context, the mobile interface primarily acts as an enabler for model utilization, emphasizing functional access to predictions rather than interface aesthetics. The Flutter framework itself enables the developer to ship code to Android, iOS, and Web. The determined platform of choice is stated in the methodology section [14].

III. Research Method

The methodology of this study is structured into three primary phases: requirement definition, system implementation, and a dual-category evaluation. This approach ensures that the system is not only technically sound but also optimized for end-user delivery.

A. Requirement Definition and System Design

1) System Architecture Overview

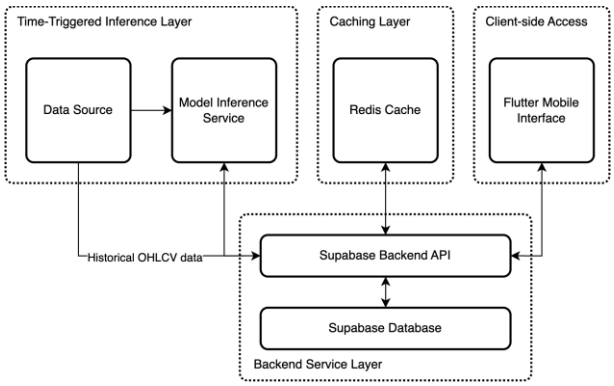

Figure 1. High-level architecture diagram

Figure 1 illustrates the high-level architecture of the proposed stock market prediction application. The system is structured to separate scheduled model inference from user-facing data access by organizing components into distinct logical layers. A time-triggered inference pipeline executes trained prediction models at predefined intervals using historical market data, after which the generated prediction results are stored in the backend database.

The backend service layer acts as the central access point for managing prediction data and coordinating communication between system components. Client applications interact exclusively with this backend layer to retrieve prediction results, manage authentication, and manage user stock preferences, while an in-memory caching mechanism is employed to reduce redundant database access and improve response efficiency for frequently requested data that originates from the model's inference. As a result, the system maintains a clear separation between model execution and client access.

2) *Prediction Model Deployment, Pipeline, and Scheduling Strategy*

The prediction pipeline adopts a scheduled, batch-based deployment strategy in which data collection, model inference, and post-processing are executed periodically as automated background tasks hosted on a remote execution environment. Market data collection is performed on a daily basis during trading days, where updated OHLCV data for selected stocks are retrieved and stored in the backend database. This approach ensures that the database maintains an up-to-date historical record of market activity, which serves as the input for subsequent prediction tasks. In parallel, a post-processing step is executed after data ingestion to associate newly observed market values with previously generated predictions and to update accuracy-related metrics accordingly.

Building on the findings of the author's previous work on a comparative study of machine learning and deep learning models for stock market prediction, this work adopts a single prediction model for system deployment. The Long Short-Term Memory (LSTM) model is selected due to its consistently higher forecasting accuracy observed across multiple evaluation metrics in the previous study. Although alternative models such as XGBoost demonstrated faster training and inference times, the latency difference was not significant in the context of the proposed batch-based prediction pipeline. Given that predictions are generated periodically and consumed asynchronously by users, inference speed does not represent a primary constraint. Consequently, model selection in this work prioritizes predictive accuracy over marginal computational efficiency, with the objective of delivering the most reliable prediction results to end users [3].

The LSTM model inference is executed on a periodic basis using a rolling historical window. For each stock, a fixed-length sequence of recent trading data is extracted from the database and used to generate predictions for the upcoming trading period. The resulting prediction outputs are stored in the backend database and made available to downstream services. By separating data ingestion, inference, and post-processing into scheduled tasks, the system ensures consistent prediction availability while avoiding direct coupling between model execution and user access. Meanwhile, client applications do not trigger model inference directly. Instead, prediction results are retrieved through backend APIs, with frequently accessed data served via the caching layer when available. This approach is done in hopes the prediction generation occur independently of user activity while supporting reliable access to prediction outputs.

3) *Backend-as-a-Service (BaaS), API Design, and Caching Strategies*

The backend component handle, process, and delegate data access and serves prediction results to both automated processes and client applications. It functions as an centralized orchestrator for the inference pipeline, persistent storage, the caching layer, and the mobile client, ensuring consistent and controlled access to prediction data. All interactions with the database and cached data are handled exclusively through backend services.

The backend exposes structured API endpoints that support two primary categories of operations. The first category serves internal system processes related to data ingestion, prediction generation, and evaluation, including storing prediction outputs, ingesting daily market data, and updating accuracy-related metrics. The second category supports user-facing access, enabling client applications to retrieve stock listings, view prediction results, access historical data, and manage lightweight interactions such as stock bookmarking.

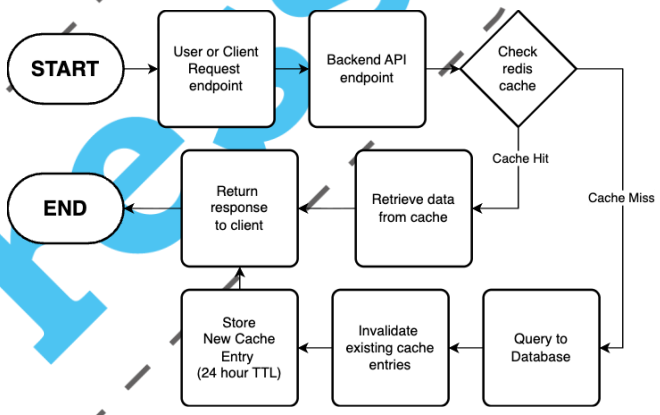

Figure 2. Backend request handling flow with caching and persistent storage.

Persistent data resides in a managed relational database, supplemented by an in-memory caching layer to optimize access. As shown in Figure 2, this layer addresses repeated client requests for identical prediction data within the same period. Stock prices and predictions are cached using a read-through strategy: the backend queries the cache first, falling back to the database only upon a cache miss. Entries are keyed by stock ID and date with a 24-hour expiration. To maintain accuracy, existing cache entries are invalidated and replaced whenever new data for a specific stock is requested.

4) *Mobile Application Implementation*

The mobile application was developed using the Flutter framework with GetX for state management and navigation, following an MVVM architectural pattern to separate presentation logic from business logic while enabling cross-platform support for Android and iOS. The implementation emphasizes functional access to stock prediction results rather than visual complexity [15].

The application serves as the user-facing layer of the end-to-end system and communicates with backend services to retrieve historical stock data and precomputed prediction results. Actual stock prices displayed in the application are based on closing prices to maintain consistency with the prediction model.

User authentication is supported through email-based login and Google Sign-In. Newly registered users are guided through an initial onboarding process to select preferred stocks, which are then stored in a personalized watchlist.

Core functionalities of the application include stock listing and search, industry-based stock filtering, watchlist management, and visualization of historical and predicted stock prices. Predictions are presented alongside historical closing prices in a combined chart, covering a five-day (one-week) forecasting horizon. The application also displays prediction accuracy in percentage form and provides access to historical prediction records. Prediction data are refreshed on a weekly basis. To improve usability, the application supports light and dark modes as well as localization that supports both English and Bahasa Indonesia language.

B. Functional and Performance Evaluation

The final phase validates the research objectives through functional and performance evaluations. The functional evaluation verifies the system's operational correctness using UI screenshots and sequence diagrams to demonstrate proper user experience and communication between the mobile client, backend APIs, and prediction engine. The performance evaluation measures technical efficiency by comparing API latency with and without the caching layer to demonstrate improvements in system responsiveness and data delivery speed.

IV. Result and Discussion

After a series of development sprints, the proposed system architecture and design described in the methodology were realized in the form of a functional prototype application. This section presents the outcomes of the implementation by outlining the realized system components and their integration within the end-to-end application.

A. System Realization and Application Features

Figure 3. UI of stock detail

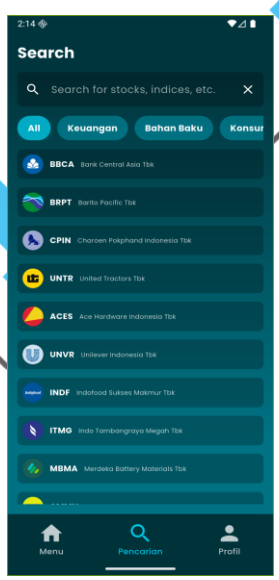

Figure 4. UI of list of stock

The proposed system was realized as a functional mobile application that serves as the user-facing interface for accessing stock prediction results generated by the backend prediction pipeline. The mobile client interacts exclusively with backend services and does not perform any local model inference, ensuring a clear separation between prediction generation and prediction consumption.

The application provides users with structured access to prediction outputs through a set of core features, including stock listing and search, industry-based filtering, and personalized watchlist management. Users can view historical stock prices alongside precomputed prediction results, which are presented in a combined chart to support short-term trend interpretation. Prediction results are displayed for a five-day forecasting horizon and are refreshed in accordance with the backend's scheduled update cycle.

In addition to prediction values, the application presents model accuracy metrics in percentage form and allows users to access historical prediction records for selected stocks. To ensure consistency with the underlying prediction models, all displayed historical prices are based on daily closing values. The application also supports user authentication via email and third-party sign-in, as well as basic personalization through an onboarding process that captures preferred stocks. Figures 3 and 4 present the key user interface screens used to demonstrate system realization, namely the stock detail page with graphical visualization and the stock list and search interface. Visual assets and icons used in the application were obtained from publicly available sources and are used solely for illustrative purposes.

B. Prediction Delivery Workflow Validation

This section validates the end-to-end workflow through which prediction results are generated, stored, and delivered to client applications. Rather than evaluating predictive accuracy, the focus is on verifying that prediction outputs produced by the backend pipeline are consistently accessible to users through the mobile application and delivered in accordance with the intended system design. The model predictive capabilities are already mentioned in the previous research work [3].

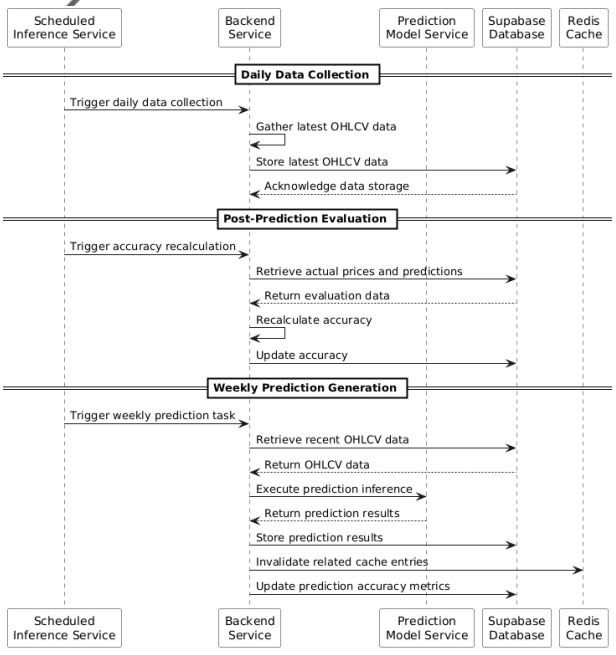

Figure 5. Sequence of scheduled prediction generation and storage.

As illustrated in Figure 5, prediction generation and prediction consumption are decoupled through a scheduled execution pipeline. Inference is performed periodically by a backend service independent of user interaction, and the

resulting outputs are persisted for subsequent access. The delivery workflow was validated by confirming the availability of updated prediction results following each scheduled inference cycle. Once new predictions are stored, they become accessible through the backend API without additional processing. Cache invalidation mechanisms ensure that outdated entries are replaced upon update and allows users to consistently receive the most recent available results within each prediction interval. The workflow also supports post-prediction evaluation by recalculating prediction accuracy after new market data become available. Updated accuracy metrics are stored and exposed alongside prediction results, enabling ongoing assessment of model performance. Overall, the validated workflow demonstrates that the system reliably operationalizes trained prediction models into a stable and repeatable delivery pipeline for end users.

C. Data Access Strategy Evaluation

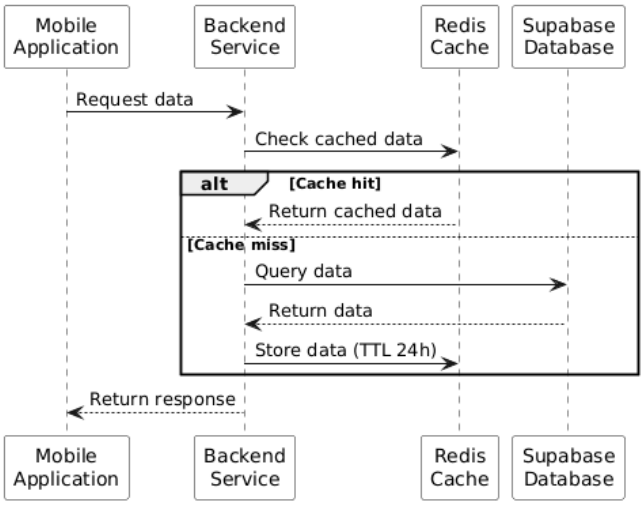

Figure 6. Sequence of prediction retrieval with cache validation.

The data access workflow employed by the system is illustrated in Figure 6 using a sequence diagram. The diagram depicts the interaction between the mobile application, backend service, caching layer, and persistent database during a typical data request. Upon receiving a client request, the backend first checks the in-memory cache for existing entries. If a cache hit occurs, the requested data are returned immediately without database access. In the case of a cache miss, the backend queries the database, returns the result to the client, and stores the retrieved data in the cache with a fixed 24-hour expiration period. This workflow underpins the comparative evaluation presented in Table 4.1, which measures response time differences between cached and non-cached access when retrieving stock details and prediction across different stock tickers.

Table 1 . Response Time Comparison with and without Caching

Query No.	Ticker	Without Caching (ms)	With Caching (ms)
1	ANTM.JK	1458	49
2	INDF.JK	*575*	55
3	AMMN.JK	754	44
4	ADRO.JK	1098	37
5	ASII.JK	949	55
6	MAPA.JK	481	52
7	INKP.JK	807	59
8	BMRI.JK	750	51
MIN		481	37
MAX		1458	59
AVG		817	48.7778

The comparison considers two scenarios: (1) direct database access without caching and (2) access through an in-memory caching layer with a fixed time-to-live (TTL). As illustrated in Table 1, the first request for a given stock and date results in a cache miss and requires a database query, while subsequent requests within the TTL period are served directly from the cache.

The results indicate a clear reduction in redundant database queries when caching is enabled. While initial access latency remains comparable due to database retrieval, repeated requests exhibit significantly lower response times under the cached configuration. This behavior is observed consistently across multiple query iterations, demonstrating improved responsiveness for frequently accessed prediction data.

In addition to latency improvement, the caching strategy contributes to system stability under repeated user requests by reducing database load and minimizing variability in response times. Overall, the evaluation confirms that integrating a lightweight caching layer effectively enhances data access efficiency without altering application-level functionality, making it suitable for serving prediction results in read-heavy client environments.

V. Conclusion

This paper extends prior work on stock market prediction by shifting the focus from model-level benchmarking to system-level realization. While earlier studies emphasized model accuracy and comparative performance, this work demonstrates how a selected prediction model can be operationalized within an end-to-end architecture that integrates scheduled inference, backend services, caching mechanisms, and a mobile application for prediction delivery.

The system design prioritizes reliability, predictability, and architectural simplicity. Scheduled, batch-based inference was selected to ensure stable prediction generation without introducing computational overhead tied to user activity. This approach is suited to short-term forecasting horizons and allows prediction results to be generated independently of client requests. An in-memory caching layer improves the overall efficiency by reducing redundant database access and improving response time for frequently requested data. Client-triggered inference is intentionally restricted to maintain clear separation between prediction generation and consumption and to prevent resource contention.

These design decisions involve trade-offs. While scheduled inference limits responsiveness to sudden market changes, it provides a controlled execution environment and consistent prediction availability. Similarly, caching improves performance at the cost of potential short-term data staleness, reflecting a balance between data freshness and computational efficiency. Overall, the architecture favors system stability and simplicity over real-time complexity.

Despite the successful realization of the proposed system, several limitations are acknowledged. The use of scheduled,

batch-based inference limits responsiveness to sudden market changes, and fixed data sources and update frequencies may reduce timeliness during periods of high market volatility. System evaluation primarily emphasizes functional validation and architectural feasibility rather than large-scale stress testing or concurrent user scenarios. In addition, caching performance is assessed under controlled access patterns, and the mobile application remains a prototype that has not undergone formal user testing to evaluate usability, user satisfaction, or decision-support effectiveness. Advanced features such as personalization, explainability, and automated model retraining are also not yet implemented, with model updates currently performed manually.

Future work may address these limitations by exploring hybrid inference strategies that combine batch-based and on-demand prediction, as well as integrating real-time or streaming data pipelines to improve responsiveness. Further evaluation of system scalability and concurrency under higher workloads is required. Planned improvements also include more sophisticated cache management strategies, enhanced user interfaces with model explainability and confidence indicators, and the implementation of automated retraining and deployment pipelines supported by model monitoring and drift detection. In addition, structured user testing and feedback collection will be conducted to assess usability, interpretability, and practical value in real-world usage scenarios, thereby strengthening the system's robustness and applicability.

Author Contribution Statement

William and Abraham Kefas Wilham contributed to the development of the prototype, writing process of the paper, conduct experimental, data analysis, and data visualization. Sonya Rapinta Manalu provided supervision, guided the research direction, and conducted critical reviews of the manuscript.

Data Availability Statement

All data supporting the findings are provided within this article. The numerical values used in the case study are presented in the tables and figures, and no additional datasets were generated or analyzed.

References

[1] D. Kumar, P. Sarangi, R. Verma. "A systematic review of stock market prediction using machine learning and statistical techniques," in Materials Today: Proceedings, vol. 49, pp. 3187–3191, 2022.

[2] P. Chhajer, M. Shah, A. Kshirsagar. "The applications of artificial neural networks, support vector machines, and long–short term memory for stock market prediction," in Decision Analytics Journal, vol. 2, pp. 100015, 2022.

[3] William, A. K. Wilham, and S. R. Manalu, "Stocks and models: A comparative benchmark of machine learning and deep learning approaches for stock market prediction," Procedia Computer Science, vol. 269, pp. 1532–1545, 2025, doi: 10.1016/j.procs.2025.09.095.

[4] Cabrera, C., et al. "Machine Learning Systems: A Survey from a Data-Oriented Perspective," in ACM Computing Surveys, vol. 58, no. 5, pp. 1–38, 2025.

[5] A. Lima, L. Monteiro, A. Furtado. "MLOps: Practices, Maturity Models, Roles, Tools, and Challenges-A Systematic Literature Review," in ICEIS (1), pp. 308–320, 2022.

[6] Lwakatare, L., et al. "Large-scale machine learning systems in real-world industrial settings: A review of challenges and solutions," in Information and software technology, vol. 127, pp. 106368, 2020.

[7] R. Jain, R. Vanzara. "Emerging trends in AI-based stock market prediction: A comprehensive and systematic review," in Engineering Proceedings, vol. 56, no. 1, pp. 254, 2023.

[8] Crankshaw, D., et al, "Clipper: A Low-Latency online prediction serving system," in 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17), 2017, pp. 613–627.

[9] Paguthaniya, S., et al. "Integration of machine learning models into backend systems: Challenges and opportunities," in Journal of Computer Engineering (IOSR-JCE), vol. 26, pp. 33-39, 2024.

[10] Supabase, "Supabase Documentation," [Online]. Available: https://supabase.com/docs. Accessed: Dec. 2025.

[11] M. Privalov, M. Stupina. "Improving web-oriented information systems efficiency using Redis caching mechanisms," in Indonesian Journal of Electrical Engineering and Computer Science, vol. 33, no. 3, pp. 1667–1675, 2024.

[12] Redis, "Redis Documentation", [Online]. Available: https://redis.io/docs/latest/. Accessed: Dec. 2025.

[13] Z. Azzahra, R. Fa'rifah, S. Lathifah. "Empowering MSMEs with Data-Driven Insights: Mobile Sales Dashboard Application for MSMEs," in Jurnal Nasional Teknologi dan Sistem Informasi, vol. 11, no. 1, pp. 1–8, 2025.

[14] Flutter, "Flutter Documentation," [Online]. Available: https://docs.flutter.dev/. Accessed: Dec. 2025.

[15] I. Husain, P. Purwantoro, C. Carudin. "Analisis Performa State Management Provider Dan Getx Pada Aplikasi Flutter," in JATI (Jurnal Mahasiswa Teknik Informatika), vol. 7, no. 2, pp. 1417–1422, 2023.