iJRASET
International Journal For Research in Applied Science and Engineering Technology



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** IV **Month of publication:** April 2026

DOI: https://doi.org/10.22214/ijraset.2026.79439

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue IV Apr 2026- Available at www.ijraset.com

TradeSense: An Explainable AI-Based Trading Decision Support System for Retail Investors

A. Pavani, S. Bhavesh, Y. Sumanth, Dr. Bharathi
[1, 2]*Students, Department of Artificial Intelligence and Data Science, Methodist College of Engineering and Technology, Abids, Hyderabad, Telangana 500001, India*
[3]*Assistant Professor, Department of Computer Science and Engineering, Methodist College of Engineering and Technology, Abids, Hyderabad, Telangana 500001, India*

Abstract**: Individual investors today face a significant imbalance. They have access to vast amounts of data, but the tools available to help them understand it are basic. Most retail trading platforms provide little more than historical charts and a few standard indicators. Users must manually piece together price signals, news events, and portfolio exposure without any structured support. This paper presents TradeSense, a decision support system that aims to bridge this gap by combining three areas of applied AI research: short-term price trend prediction, news-driven sentiment analysis, and retrieval-augmented natural language explanation. This all comes together in a single web application designed for non-institutional investors. TradeSense primarily relies on an XGBoost classifier, which estimates the likelihood of a stock's price trend continuing over five days. A FinBERT-based component processes recent financial news headlines, generating a sentiment index that users can compare with the quantitative prediction. A Retrieval-Augmented Generation (RAG) engine then combines these outputs into an easy-to-understand explanation, using a curated knowledge base of past analyses and indicator definitions. The system features a three-tier architecture, with a React frontend for user interaction, a Node.js/Express gateway for managing operations, and a Python/FastAPI backend for the analytical tasks. A lightweight portfolio module tracks user holdings and highlights concentration risk. Evaluations through historical backtesting, event-driven case studies, and structured software testing show that this integrated approach makes model outputs much more interpretable than presenting raw signals alone. However, real-world use would require significant improvements in data coverage and infrastructure.

Index Terms: Explainable AI, decision support systems, XGBoost, FinBERT, retrieval-augmented generation, financial sentiment analysis, algorithmic trading, retail investors.

I. INTRODUCTION

A. Motivation

Equity markets have become structurally complex. Prices not only reflect company fundamentals but also shifts in sentiment, algorithmic order flow, and macroeconomic signals that change faster than any individual can monitor. Professional asset managers tackle this challenge by using quantitative research teams and specialized machine learning systems. For retail investors, however, there is still a considerable gap in available tools. Most individual traders rely on applications that offer visual price history and a short list of classic technical indicators. Prediction, context, and explanation are often missing.

Research has made considerable progress in addressing each underlying issue independently. Gradient-boosted tree methods, especially XGBoost, perform well with structured financial feature sets and have been validated in risk modeling and short-term price forecasting tasks [1],[6]. Transformer-based language models, fine-tuned on financial text, notably FinBERT [3], extract reliable sentiment signals from news and analyst commentary. Recently, Retrieval-Augmented Generation frameworks have been shown to enhance the factual grounding of large language model outputs by incorporating retrieved context during inference [4]. What has not been fully explored is how to integrate these capabilities into a single system that retail investors can actually use—one that is fast, interpretable, and transparent about its limitations.

TradeSense seeks to create this system. The goal is not to outperform institutional models but to provide individual investors with the structured, explainable signals they currently lack. This paper outlines the design, implementation, and evaluation of a working prototype that combines prediction, sentiment, and natural language explanations, along with a basic portfolio management module.

©IJRASET: All Rights are Reserved

B. Problem Statement

The gap we are addressing is not mainly technical. The algorithms for forecasting short-term price trends and classifying news sentiment are well established. The gap lies in the architecture and user experience: these capabilities exist in academic prototypes or enterprise platforms, but they are not organized in a manner that individual traders can easily access or understand.

A useful retail system must excel in at least four areas. It should provide a short-term price forecast without requiring users to grasp how it was created. It needs to contextualize that forecast by presenting the current news environment related to the stock. It must connect both signals to the user's actual holdings. Finally, it must convey its reasoning in a language that does not expect a background in finance. TradeSense is designed around these four requirements.

C. Objectives

We established six specific goals when designing TradeSense:

- Build a fully integrated multi-tier web application that connects data ingestion, machine learning inference, sentiment classification, natural language explanation, and portfolio visualization into one workflow.
- Train and deploy an XGBoost classifier for five-day equity trend continuation, selecting features from standard technical indicators and evaluating the model using chronological data splits to avoid look-ahead bias.
- Integrate FinBERT as a news sentiment module that generates a sentiment index for each ticker, updated with each prediction cycle.
- Implement a RAG-based explanation engine that grounds generated text in retrieved past analyses and indicator descriptions, presenting outputs as interpretive guidance rather than investment advice.
- Provide a portfolio dashboard that calculates unrealized profit and loss, tracks asset allocation, and flags concentration risk with configurable thresholds.
- Conduct a structured evaluation of predictive performance, software reliability, and user-perceived interpretability, and identify the gaps that must be addressed for production deployment.

D. Contributions

This work offers three main contributions. First, it presents a concrete architecture for integrating XGBoost, FinBERT, and RAG into a single retail analytics product—an integration that, to our knowledge, has not been published as a complete working system before. Second, it proposes and implements a practical explainability layer that transforms numerical model outputs into natural language using retrieved context, easing the interpretative burden on non-expert users. Third, it provides an honest evaluation of the system's capabilities and limitations, including the conditions under which predictive accuracy declines and the infrastructure changes required to move from prototype to production.

II. RELATED WORK

A. Prediction Models for Short-Term Price Movements

Gradient-boosted trees have become a standard baseline for financial prediction tasks involving tables. Chen and Guestrin's XGBoost [1] introduced a scalable boosting framework that handles missing values directly and reduces overfitting through shrinkage and column subsampling. These features work well with noisy financial data. Subsequent studies have used XGBoost for stock volatility forecasting [6], credit risk classification, and short-term directional prediction, typically outperforming linear models and matching or surpassing more complex neural networks on structured data.

Sequence models provide a different approach. Long Short-Term Memory networks [7] can capture multi-step time dependencies that tree-based methods overlook. Hybrid designs that combine LSTM layers with self-attention have achieved strong directional accuracy on daily price data [8]. The trade-off here is interpretability. Boosted trees offer SHAP-based feature attribution [2], while LSTM models remain unclear to users and are more challenging to run with low latency. For systems that focus on explainability and real-time responses, XGBoost is the better option for the prediction module.

B. Sentiment Analysis in Financial Contexts

Early work on financial sentiment used lexicon-based methods. These methods involved word lists categorized as positive, negative, or neutral in a financial context. They often struggle with complex sentence structures or specialized jargon.

The introduction of BERT [9] changed this by allowing fine-tuning on labeled data specific to tasks, and Araci's FinBERT [3] adapted this approach for finance by pre-training on a large set of financial news and analyst reports before fine-tuning for sentiment classification benchmarks. FinBERT consistently outperforms both general-purpose transformers and traditional methods on financial sentiment tasks, making it the obvious choice for this module.

Recent research has moved toward entity-level and aspect-level sentiment. This approach acknowledges that an article may express different sentiments about various companies or topics [10]. While TradeSense does not currently use aspect-level analysis, this direction suggests a natural extension of the sentiment module that could improve accuracy for news stories covering multiple companies.

C. Retrieval-Augmented Generation

Large language models can generate fluent text but may deliver factually incorrect or contextually inappropriate outputs for areas outside their training. Retrieval-Augmented Generation addresses this issue by giving the model a retriever that brings up relevant documents during inference, which are then added to the generation prompt. Lewis et al. [4] established the main framework, and later work has applied it to open-domain question answering, customer support, and more recently, financial document analysis [5].

Xiao et al. [11] specifically studied RAG for forecasting financial time series, showing that conditioning a large language model on retrieved historical market summaries improved forecast quality during periods of regime change when compared to a purely parametric baseline. TradeSense adopts this principle but applies it to explanations rather than predictions. The RAG module retrieves past analyses and indicator descriptions to inform the generated text, preserving a clear separation between the quantitative model and the language model.

D. Identified Gaps

There are three significant gaps in the existing literature. First, prediction models are usually evaluated in isolation. Their accuracy is reported, but there's little focus on how to present their outputs in a way that fosters appropriate trust without exaggerating certainty. Second, sentiment analysis and price prediction are generally treated as separate issues, with limited research on integrated systems that present both to the same user at the same time. Third, RAG-based financial systems primarily emphasize document retrieval and question answering rather than contextualizing real-time point predictions on a retail dashboard. TradeSense is designed to address all three of these gaps.

III. PROBLEM FORMULATION AND SCOPE

A. Formal Problem Statement

We define the core task as follows. Given a user-specified stock ticker and the user's current portfolio status, the system must produce four types of output: (1) an estimate of the short-term trend with a probabilistic label; (2) a sentiment index based on recent news about that stock; (3) a natural language explanation that connects the prediction and sentiment to recent price behavior and the portfolio context; and (4) risk indicators at the portfolio level that cover allocation and concentration. All four outputs must be provided within a response time suitable for interactive use and should be understandable to users without formal training in quantitative analysis.

B. Scope Decisions

Several intentional constraints shape the current implementation. The system focuses only on publicly listed stocks. Price and volume data comes from the yfinance library, while news headlines are sourced via the Finnhub API—both are free-tier services with limitations discussed in Section VII. The prediction horizon is set at five trading days, a short enough window to be actionable but long enough to filter out daily fluctuations. The deployment model supports single-users with local JSON storage for portfolio data; there is no authentication layer, multi-tenancy, or cloud storage in this prototype.

Importantly, TradeSense is designed to serve as an analytical and educational tool. It does not connect with any brokerage, does not execute trades, and includes a clear disclaimer that its outputs do not constitute financial advice. This framework is not just a legal precaution; it reflects a genuine choice to keep the system focused on decision support rather than automated trading.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

IV. SYSTEM ARCHITECTURE

A. Three-Tier Design

We organized TradeSense around a traditional three-tier web architecture, separating the user interface, orchestration layer, and analytical computation into distinct services. This separation serves two purposes: it prevents long-running ML inference from blocking HTTP responses, and it allows for independent testing and replacement of each tier.

The presentation tier is a React single-page application. It displays the stock analysis dashboard, portfolio overview, sentiment summaries, and the natural language explanation panel.

Chart components visualize price history and computed indicators interactively. The orchestration tier is a Node.js/Express server that provides a RESTful API to the frontend. It manages an in-memory results cache to reduce redundant API calls, performs request validation, and handles all portfolio read/write operations. The analytical tier is a Python/FastAPI service that hosts the machine learning models, the FinBERT classifier, and the RAG pipeline, along with the data ingestion utilities.

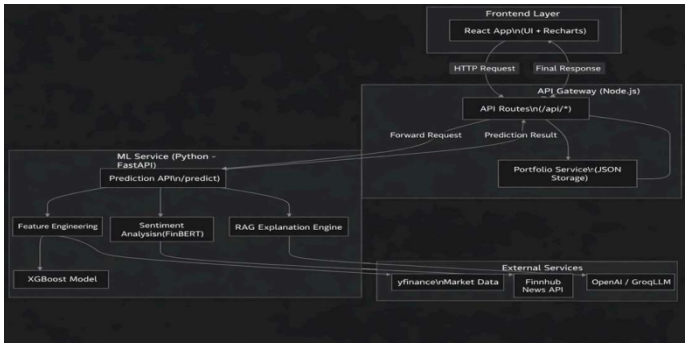


Fig. 1. TradeSense three-tier system architecture showing the Frontend Layer, API Gateway, ML Service, and External Services

B. Request Lifecycle

When a user submits a ticker for analysis, the React frontend sends a request to the Node.js layer, specifying which modules to invoke. Node.js first checks its cache; if there is a fresh cached result, it returns that immediately. If the cache is empty, Node.js sends sub-requests to the Python service. The Python service sequentially fetches updated price data, computes the feature vector, runs XGBoost inference, optionally fetches and classifies recent news headlines, and optionally calls the RAG module to generate an explanation. The Python service returns a structured JSON payload, which Node.js combines with the user's portfolio data before sending the final response to the frontend.

C. Technology Rationale

We chose XGBoost over a neural sequence model because of its speed and clarity: the tree model provides predictions in milliseconds, and its feature importance can be calculated quickly using SHAP [2]. We selected FinBERT since it is the most validated model for financial sentiment classification [3], and it is available as a pre-trained checkpoint via Hugging Face, making it easy to integrate without retraining. We picked FAISS as the vector store for the RAG module because it efficiently performs approximate nearest-neighbor searches on CPU without needing a dedicated vector database service. React and Node.js were chosen for their wide ecosystem support and the easy development experience they provide for this kind of data dashboard.

V. METHODOLOGY

A. Data Preparation

For each stock in scope, we download several years of daily OHLCV data to capture multiple market situations, including trending, ranging, and high-volatility periods. We clean the raw series to address non-trading days, dividend adjustments, and split events. We then compute a feature set that includes simple and exponential moving averages over 5, 10, and 20 days; the Relative Strength Index (RSI); Bollinger Band width and position; average true range as a measure of volatility; and on-balance volume as a market participation indicator.

The prediction target is binary: a value of 1 if the closing price five sessions ahead is higher than the current close, and 0 otherwise. We apply a small neutrality band around zero to filter out days where the price movement is economically insignificant, which helps the model avoid fitting noise.

We normalize features using a standard scaler fitted only on training data, and the fitted scaler is saved alongside the model for consistent inference. We partition the data using expanding chronological windows to ensure no future information leaks into the training set.

B. XGBoost Model Training

We train an XGBoost binary classifier, tuning maximum tree depth, learning rate, and the L1/L2 regularization coefficients on a held-out validation segment. The model is evaluated on a final test window using directional accuracy, precision, recall, and the confusion matrix for both the continuation and reversal classes. We also stratify results by market volatility regime to see where the model performs best and worst—an important factor given the regime dependence noted in previous volatility forecasting work [6].
At serving time, the model outputs a class probability that maps to one of three user-facing labels: likely continuation, uncertain, and likely reversal, using thresholds set based on the historical confusion matrix.

C. News Sentiment Module

We retrieve recent news headlines and summaries for each queried ticker through the Finnhub API. After basic cleaning—removing HTML tags and shortening long descriptions—each text snippet is scored by FinBERT, which assigns probabilities across positive, neutral, and negative classes. We combine these scores into a single sentiment index by converting each prediction into a numeric weight (+1 / 0 / -1) and calculating a recency-weighted mean that gives less weight to older articles. The final index maps to one of three qualitative labels—bullish, neutral, bearish—which are shown alongside the prediction output. Presenting the sentiment index separately from the prediction, instead of merging it as a model feature, was a deliberate choice to keep the two signals independent, allowing users to see whether they support or contradict each other.

$$S_t = \frac{\sum_{i=1}^{n} w_i \cdot s_i}{\sum w_i}, \text{ where } w_i = e^{-\lambda(t-t_i)}$$

Fig. RWSI formula: Recency-Weighted Sentiment Index.

D. Retrieval-Augmented Explanation Engine

The RAG module works in three stages. In the indexing stage, after each analysis, we encode a structured summary of the case—ticker, date, key indicator values, XGBoost output, sentiment index, and, when available, the eventual five-day outcome—into a dense embedding vector using a sentence-transformer model and store it in FAISS. In the retrieval stage, when an explanation is needed, we encode the current context and use it to search the FAISS index for the most relevant past cases, along with relevant indicator definition snippets from a static knowledge base. In the generation stage, we submit a prompt template that combines the current numerical context, the retrieved cases, and the indicator descriptions to an LLM, with instructions to create a brief, evidence-based explanation that clearly acknowledges the model's limitations. The generated text is displayed in the dashboard under a clearly labeled "Analysis Summary" heading with a persistent advisory note stating that it is not investment advice.

E. Portfolio Analytics

User positions are recorded as ticker, quantity, and average cost basis. At analysis time, the module retrieves current market prices and computes per-position market value, unrealized P&L, and percentage allocation. A concentration heuristic flags any position whose allocation goes beyond a user-set threshold, surfacing the warning in the dashboard. By linking the stock being analyzed to the user's holdings, the system highlights when the model is expressing a view on a position that already represents a significant portion of the portfolio—something many retail charting tools do not address.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

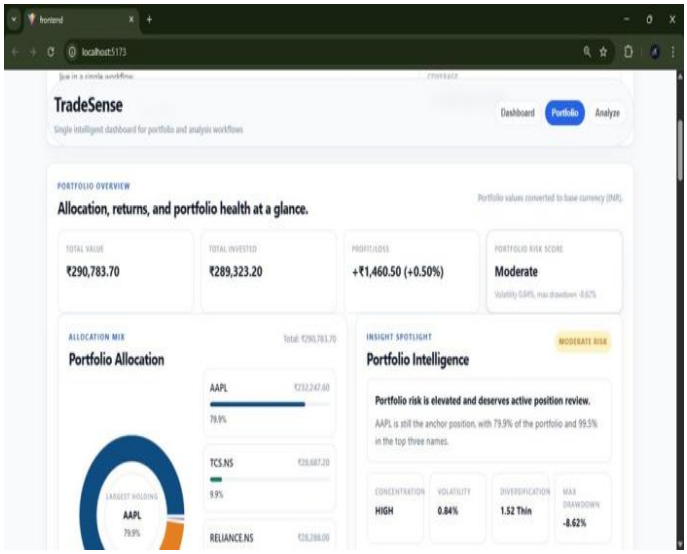


Fig. 3. TradeSense Portfolio dashboard showing allocation mix, profit/loss, portfolio risk score, and concentration warnings.

VI. EVALUATION

A. Testing Approach

We used a layered testing strategy. Unit tests cover the main Python functions for indicator computation, feature assembly, model inference, and sentiment aggregation. They also include the Node.js portfolio operations and API handlers. Integration tests check end-to-end request flows from the React frontend through the Node.js gateway to the Python service, ensuring response schemas, error codes, and caching behavior are correct.

Functional tests simulate complete user workflows: adding and removing portfolio positions, submitting valid and invalid tickers, toggling the sentiment and explanation modules, and inspecting the resulting dashboard state. Informal acceptance tests confirm that the system handles external API failures gracefully, returning meaningful error messages instead of crashing, and stays stable during continuous use.

B. Predictive Performance

We conducted backtesting on a set of liquid large-cap stocks over a multi-year period that included both trending and volatile market conditions. Five-day directional accuracy falls within the range reported by similar XGBoost-based stock prediction studies; however, we avoid making strong conclusions from these figures. Backtested accuracy on liquid stocks is necessary but not enough to determine real-world usefulness, and this figure changes significantly depending on the selected period and stocks. The regime-stratified analysis is more informative. It shows, as expected, that accuracy decreases noticeably around high-impact macro announcements and earnings releases, which are the events hardest to predict using technical features alone.

On the sentiment side, qualitative case studies around major earnings announcements confirmed that the FinBERT sentiment index aligns with intuitive human readings of headline tone in most cases. Occasionally, the model misclassifies sarcastic or highly hedged language, but these are exceptions. A more useful insight from the case studies is that the sentiment index and the XGBoost prediction often diverge—for example, a strong technical continuation signal may appear alongside a bearish news environment. Highlighting this divergence is valuable to a user who might otherwise see only one of the two signals.

C. User Experience Observations

Informal walkthroughs with test users showed that presenting the prediction probability, the sentiment index, and the natural language explanation together in one view is much easier to interpret than showing a raw probability score or a price chart alone. Users found the explanation panel particularly helpful for understanding why the model generated a given output, even when they disagreed with its conclusion. These observations are qualitative and cannot replace a formal user study, which we plan to prioritize in the future.

©IJRASET: All Rights are Reserved

VII. LIMITATIONS AND FUTURE WORK

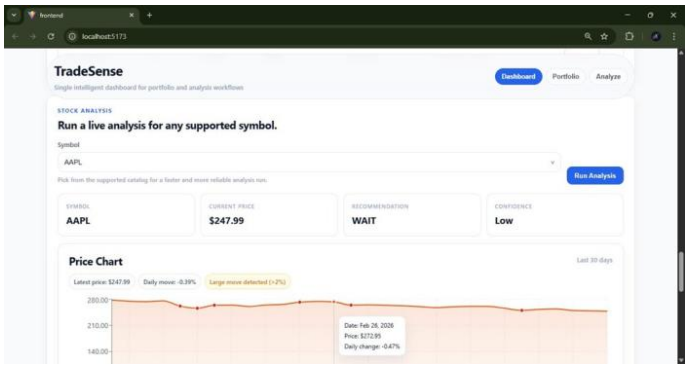


Fig. 2. TradeSense Stock Analysis dashboard displaying symbol lookup, current price, recommendation, confidence level, and interactive price chart

TradeSense is a research prototype, and several of its limitations are significant. We clearly document them here because we believe honest scoping is more beneficial to the research community than overstating what the current system can do.

The most significant architectural limitation is the single-user, local-first deployment model. Portfolio data is stored in a flat JSON file, and there is no user authentication. The system has not been load-tested for concurrent use. Moving to a production deployment would require migrating to an appropriate database, implementing authentication, and containerizing the analytical services for scalable deployment.

Data access is a related issue. Both yfinance and the Finnhub free tier impose rate limits that make the current approach unsuitable for a multi-user or high-frequency deployment. To achieve serious production use, we would need to integrate commercial data providers. The news coverage for smaller-cap stocks is also limited, meaning the sentiment module is most reliable for well-covered large-cap stocks.

On the modeling side, the current risk analytics are intentionally simple, using allocation percentages and a concentration flag. More advanced risk measures like Value-at-Risk, expected shortfall, and maximum drawdown are not implemented. The prediction model also has no way to detect regime changes, meaning its confidence estimates can be misleading during market disruptions. These are known weaknesses that need to be addressed before we can recommend the system for real financial decisions.

Future directions that seem most promising include extending the feature set with fundamental financial ratios and macroeconomic indicators [11], exploring hybrid prediction architectures that combine XGBoost's strengths in structured data with a sequence model for capturing temporal patterns [7],[8], enriching the RAG knowledge base with earnings call transcripts and regulatory filings, and conducting a formal user study to measure interpretability improvements more rigorously. A live paper-trading evaluation would also be a valuable addition, providing a measure of real-time performance that historical backtesting cannot replicate.

VIII. CONCLUSION

This paper presented TradeSense, a web-based decision support system aimed at making AI-driven market analytics available to individual investors. The system combines three components that are well-established in the research literature but rarely brought together in a single end-user product: an XGBoost classifier for short-term price trend prediction [1], a FinBERT-based news sentiment module [3], and a retrieval-augmented explanation engine [4] that transforms numerical outputs into plain-language summaries based on historical context.

The prototype shows that integration itself adds value. Presenting prediction, sentiment, and explanation together makes each component more useful than any one of them would be alone. The divergence between a technical signal and a sentiment reading, for instance, is only visible when both are displayed at the same time, and it is exactly this kind of contextual tension that can lead to more thoughtful decision-making.

We have been clear about the system's limitations. TradeSense is a decision aid, not a trading system. Its accuracy is limited by the quality and coverage of its input data, and its predictions should be seen as one input among many rather than as definitive signals. The value it provides is not predictive alpha but interpretive clarity. It gives retail investors a more structured, evidence-based way to consider a position than what a price chart and a news feed can offer alone. Building on this foundation to create a more capable, production-ready platform is a natural and achievable next step.

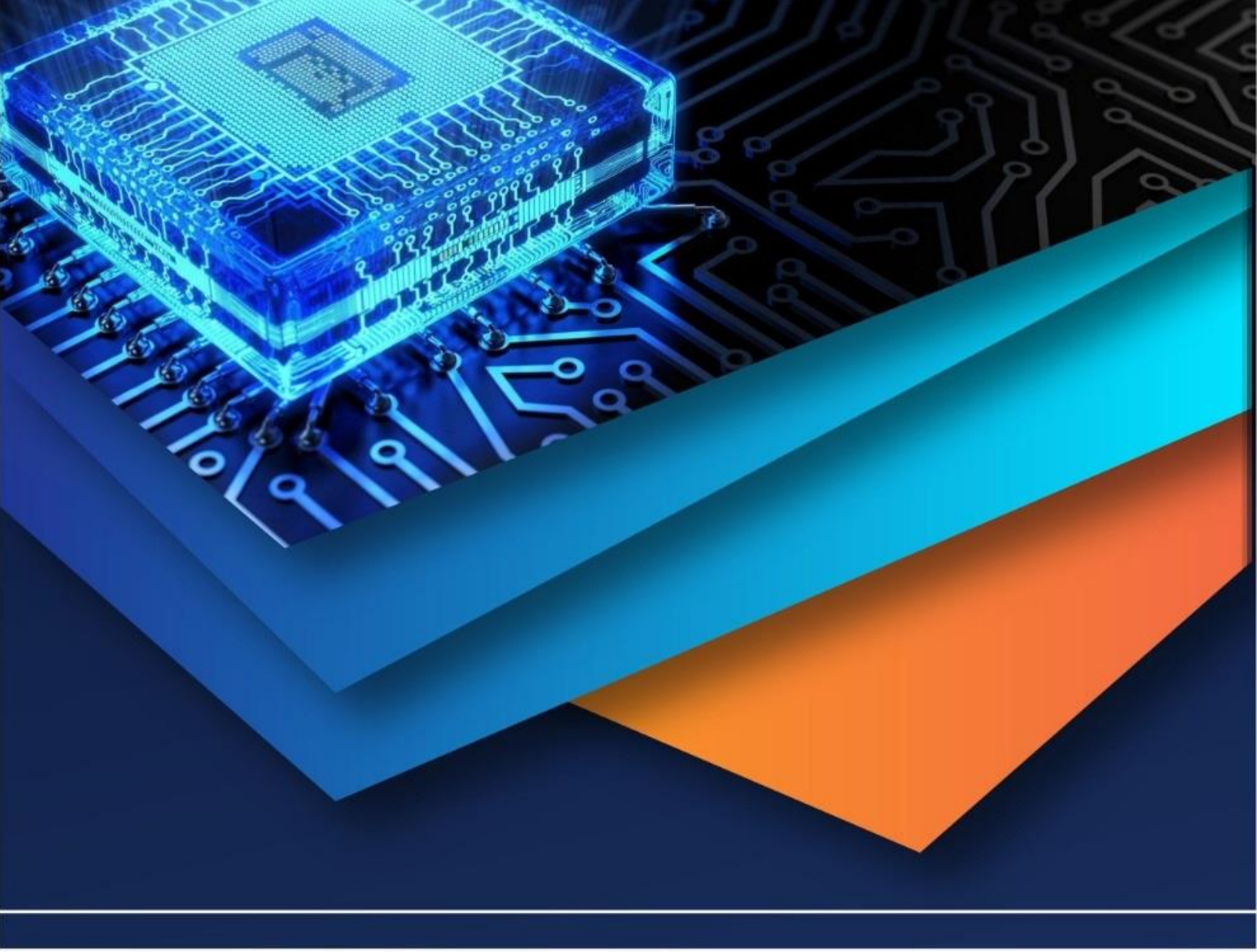
IX. ACKNOWLEDGMENT

The authors thank the Department of Artificial Intelligence and Data Science at Methodist College of Engineering and Technology, Hyderabad, for the support and academic environment that made this project possible. They also acknowledge the developers of the open-source tools on which TradeSense is built, including XGBoost, the Hugging Face Transformers library, FAISS, and the React and Node.js ecosystems.

REFERENCES

[1] T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," in Proc. 22nd ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining, 2016, pp. 785–794.
[2] S. M. Lundberg and S.-I. Lee, "A Unified Approach to Interpreting Model Predictions," in Advances in Neural Information Processing Systems (NeurIPS), 2017, vol. 30.
[3] D. Araci, "FinBERT: Financial Sentiment Analysis with Pre-trained Language Models," arXiv preprint arXiv:1908.10063, 2019.
[4] P. Lewis et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," in Advances in Neural Information Processing Systems (NeurIPS), 2020, vol. 33, pp. 9459–9474.
[5] FinRAG System, "Fin-RAG: A Retrieval-Augmented Generation System for Financial Documents," Technical Report, 2024.
[6] T. Sharma and S. Prasad, "Forecasting Stock Market Volatility Using XGBoost: A Time Series Analysis," in Proc. IEEE Intl. Conf. on Emerging Technologies, 2022.
[7] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," Neural Computation, vol. 9, no. 8, pp. 1735–1780, 1997.
[8] K. Pardeshi et al., "Stock Market Price Prediction: A Hybrid LSTM and Sequential Self-Attention Architecture," in Proc. IEEE ICDACS, 2023.
[9] J. Devlin et al., "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in Proc. NAACL, 2019, pp. 4171–4186.
[10] A. Sinha, S. Kedas, and R. Kumar, "SEntFiN 1.0: Entity-Aware Sentiment Analysis for Financial News," ACM Trans. Management Information Systems, vol. 14, no. 2, 2023.
[11] M. Xiao et al., "Retrieval-Augmented Large Language Models for Financial Time Series Forecasting," arXiv preprint arXiv:2502.05878, 2025.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)