Stock guard

A PROJECT REPORT

BY

TEAM NO. 8

Shubham Tandon (E23CSEU1730)

Vipin Yadav (E23CSEU1728)

A blue and red text on a black background

Description automatically generatedAbhay Pratap Singh (E23CSEU1720)

SUBMITTED TO

SCHOOL OF COMPUTER SCIENCE ENGINEERING AND TECHNOLOGY, BENNETT UNIVERSITY

GREATER NOIDA, 201310, UTTAR PRADESH, INDIA

April 2025

**TABLE OF CONTENTS**

[LIST OF TABLES vi](#_Toc85881407)

[LIST OF FIGURES vii](#_Toc85881408)

[LIST OF ABBREVIATIONS viii](#_Toc85881409)

[ABSTRACT ix](#_Toc85881410)

[1. INTRODUCTION 1](#_Toc85881411)

[1.1. Problem Statement 1](#_Toc85881412)

[2. Background Research 2](#_Toc85881413)

[2.1. Proposed System 2](#_Toc85881414)

[2.2. Goals and Objectives 2](#_Toc85881415)

[3. Project Planning 3](#_Toc85881416)

[3.1. Project Lifecycle 3](#_Toc85881417)

[3.2. Project Setup 3](#_Toc85881418)

[3.3. Stakeholders 4](#_Toc85881419)

[3.4. Project Resources 4](#_Toc85881420)

[3.5. Assumptions 5](#_Toc85881421)

[4. Project Tracking 5](#_Toc85881422)

[4.1. Tracking 5](#_Toc85881423)

[4.2. Communication Plan 6](#_Toc85881424)

[4.3. Deliverables 7](#_Toc85881425)

[5. SYSTEM ANALYSIS AND DESIGN 8](#_Toc85881426)

[5.1. Overall Description 8](#_Toc85881427)

[5.2. Users and Roles 8](#_Toc85881428)

[5.3. Design diagrams/ UML diagrams/ Flow Charts/ E-R diagrams 9](#_Toc85881429)

[5.3.1. Use Case Diagrams 9](#_Toc85881430)

[5.3.2. Class Diagram 10](#_Toc85881431)

[5.3.3. Activity Diagrams 11](#_Toc85881432)

[5.3.4. Sequence Diagram 12](#_Toc85881433)

[5.3.5. Data Architecture 13](#_Toc85881434)

[6. User Interface 14](#_Toc85881435)

[6.1. UI Description 14](#_Toc85881436)

[6.2. UI Mockup 14](#_Toc85881437)

[7. Algorithms/Pseudo Code 15](#_Toc85881438)

[8. Project Closure 16](#_Toc85881439)

[8.1. Goals / Vision 16](#_Toc85881440)

[8.2. Delivered Solution 16](#_Toc85881441)

[8.3. Remaining Work 16](#_Toc85881442)

[REFERENCES 17](#_Toc85881443)

LIST OF TABLES

Table Page

[Table 1: Goal and Objectives 2](#_Toc20994041)

[Table 2: Sample 2 3](#_Toc20994042)

[Table 3: Sample 3 4](#_Toc20994043)

[Table 4: Sample 4 4](#_Toc20994044)

[Table 5: Sample 4 5](#_Toc20994045)

[Table 6: Sample 6 5](#_Toc20994046)

[Table 7: Regularly Scheduled Meetings 6](#_Toc20994047)

[Table 8: Information To Be Shared Within Our Group 6](#_Toc20994048)

[Table 9: Information To Be Provided To Other Groups 6](#_Toc20994049)

[Table 10: Information Needed From Other Groups 7](#_Toc20994050)

[Table 11: Deliverables 7](#_Toc20994051)

[Table 12: Sample 12 8](#_Toc20994052)

LIST OF FIGURES

Figure Page

[Figure 1: Sample use-case diagram 9](#_Toc20994053)

[Figure 2: sample 2 11](#_Toc20994054)

[Figure 3: sample 3 12](#_Toc20994055)

[Figure 4: Sample 4 13](#_Toc20994056)

[Figure 5: Sample 5 13](#_Toc20994057)

[Figure 6: Sample 6 14](#_Toc20994058)

ABSTRACT

This project provides a strong and interactive means of identifying anomalies in stock market data using a deep learning trained autoencoder model. The goal is to help retail investors and students identify unusual trends, or manipulation of stock price activity, using lifetime historical stock price information. The system utilizes global publicly available stock data dynamically fetched from the Yahoo Finance API and implements a rolling normalization ideal for several overlapping time-window segments of closing price data. These time-window segments are then fed into a neural network-based autoencoder trained to effectively reconstruct normal stock activity. Reconstruction error will provide the basis for detecting anomalies with a dynamic threshold as determined by training loss statistics.

The model architecture features a compact autoencoder neural network model designed for unsupervised anomaly detection providing robust applicability while significantly reducing dependence on labeled data. The user is prompted for stock ticker and for his/her user-specific start and end date ranges, facilitating user-specific interactive anomaly detection and visualization. The model calculates the reconstruction loss based across the windows of test data, noting any high degree of deviation expected stock behavior, and flags as an anomaly and visualizes on intuitive charts.

1. INTRODUCTION

The rapid growth of retail participation in financial markets has made stock trading more accessible than ever. With the proliferation of online brokerages, mobile trading apps, and real-time news, individual investors now have unprecedented access to financial data. However, this increased access has also exposed them to greater risks—particularly those stemming from market manipulation, pump-and-dump schemes, and flash crashes. Such events are difficult to detect in real-time and often go unnoticed until after significant damage has been done.

At the same time, the availability of historical stock data and open-source financial APIs has created new opportunities for researchers and developers to build intelligent systems that assist in understanding market behavior. Among these, unsupervised anomaly detection techniques have gained attention due to their ability to learn normal patterns and flag deviations without requiring labeled data. In particular, autoencoders—neural network models designed to reconstruct input data—are proving highly effective for identifying subtle anomalies in time-series datasets like stock prices.

This project is developed in response to this evolving landscape. It focuses on creating a practical, accessible, and modular system that uses autoencoders to detect anomalous behavior in Indian stock market data. It bridges the gap between academic research and real-world usability by allowing user-driven analysis through a simple interactive interface.

Key trends and motivations include:

* Increased algorithmic trading activity, raising concerns over transparency and fairness.
* Growing demand for explainable AI in finance, especially for anomaly detection.
* Limited tools available for individual investors to detect unusual stock behavior.
* High data availability through APIs like Yahoo Finance, enabling real-time and historical analysis.
* Autoencoders’ proven success in cybersecurity, fraud detection, and now increasingly in finance.
* By combining deep learning techniques with practical stock market datasets, this project aims to empower users with better insights into financial anomalies and suspicious market patterns.
  1. Problem Statement

Currently, retail investors and analysts in the Indian stock market do not have reliable tools to identify anomalous trading patterns or manipulative trades in real-time. While a large volume of historical OHLC (Open, High, Low, Close) stock data is available, there are no accessible, modular, and intelligent systems that can process these data to automatically flag unusual activity without a user's manual intervention. Agency stakeholders in established equity markets often have access to sophisticated analytical methods and tools, applied in combination with assurance from equity regulators. However, traditional anomaly detection approaches often struggle with the structural and non-stationarity complexities of financial time-series data. Also, most of the existing models are either too generic or designed for Western equity markets or Japan, so may not take into account certain idiosyncratic aspects of Indian equities. Consequently, instances of market manipulation, such as pump-and-dump scams, surging false volumes, or sudden price moves usually go undetected in public data, resulting in financial losses and diminishing trust in the market.

This project addresses this gap by developing an unsupervised anomaly detection framework using LSTM autoencoders, targeted to Indian stock market data. It will make use of stock-specific analysis, rolling-window processing, and realtime anomaly denotation, which will allow users to better understand when stocks are behaving suspiciously.

1. Background Research

Anomaly detection in financial markets is critical to detecting anomalous behaviours, suspicious activities, or potentially malignant activity which could manipulate the market. Chartists historically applied a rule-based approach to capture outliers using Bollinger Bands, while analysts frequently utilized Moving Averages and used statistical thresholds like Z-scores to assess the behaviour against that threshold, but the limitations of these methods are evident in our study of financial markets' highly dynamic, non-linear, data with periodic and temporal dependencies (Chandola et al., 2009).   
  
With the emergence of algorithmic trading, as well as the recent market pressures, there has been an increasing trend towards the utility of machine learning as a surrogate for anomaly detection. One of the early ML-based component were isolation forests (Liu et al. (2008)), which were one of the early ML approaches based on their computational efficiency for handling relatively high-dimensional datasets. They repeatedly isolate different samples and identify anomalies based on their isolation from their normal peers. Multiple financial applications (Kumar et al. (2020)) have shown its potential to detect anomalies, especially given a volume-price relationship.   
  
Isolation Forests and similar models do not retain the time-dependent structure of stock prices or models, which is one reason why LSTM (Long Short-Term Memory) Autoencoder models were introduced. These Deep Learning-based models are able to account for long-term dependencies in time series data. It takes sequences of normal behaviours and learns to reconstruct them. The reconstruction that diverges from what the model learned is deemed an anomaly, based on some measure of error (Zhao et al. (2017)). As an outside, when 검증 is positiv, least number of errors implies, minimal divergences when reconstructing sequences with incursions.

* 1. Proposed System

The project consists of designing and implementing LSTM-based autoencoders for developing a real-time anomaly detection system, specifically for the Indian stock market. The objective is to identify suspicious stock behavior...manipulation patterns, price spikes, or abnormal behaviors. This will be done by analyzing historical data and analyzing real-time feeds to detect and alert anomalies. The approach differs from simple dashboards and analytics, because there will also be a learning component. Specifically, applying a training back-end system that will learn over-time, and then classify anomalies based on reconstruction error.

The fundamental premise is to provide a market participant a means to detect manipulation, reduce bad decision-making, and build trust in equity investing. An additional sellable feature of developing a tool is the platform is going to use a generalized version of a deep learning model that will be trained across many stocks in India, then support the each stock fine-tuning.

If implemented, the final results will be:

* Time-series data visualization of actual vs. the reconstructed prices.
* A marker based anomaly viewer.
* An explanation module that will be optional to breakdown each anomalies cause (future updates).

By embedding such a tool on existing investor platforms, retail investors will be provided an early warning system whereby they and analysts alike can be alerted to suspicious behaviors in the market that can aid in reducing their risk exposure and facilitating their decisions.

* 1. Goals and Objectives

The main aim of this project is the creation of a functional and intelligent anomaly detection system, called StockGuard, that detects indicators of stock market anomalies through deep learning. The system is utilizing a LSTM Autoencoder to locate abnormal stock price deviations that may represent different manipulation, or unexpected behavior.

As stated above, the main goal of this project is to build a solid anomaly detection model with the following designated goals:

Design A Model for Effective Anomaly Detection:

* Establish an LSTM Autoencoder to model normal stock behavior and identify deviations that represent anomalies.
* Train and evaluate the model with real stock market historical data.
* Design and Implement A User-Interactive Interface:
* Allow users without deep expertise, to input stock tickers and a chosen time range.
* Permit users to see the visual depiction of any anomalous movements by including plots comparing reconstructed versus actual.
* Permit Flexible Stock and Chosen Date Range:
* Allow users to look at different stocks, with user-selected time intervals.
* Allow users to filter anomaly detection by date windows, after 2025 as well.
* Design a Robust Backend Functionality:
* Easy to train and test with modular back-end logic.
* Easy to save trained models, and load them back in for making inferences while minimizing repeated re-training.
* Design a way to demonstrate interpretability visually:
* Plot sequences that are both normal and anomalous, to gain deeper insights.
* Use reconstruction error to plot a visible mark indicating anomalies.
* Lay the Foundation for Future Real-Time Detection
* Design the model so the detection could be easily expanded into real-time detection using live APIs.
* Look at performance optimizations to improve detections with full results shown.

1. Project Planning

The work on StockGuard was developed by following an iterative and modularly planned approach. The project featured three main areas of development—data preprocessing and modelling, backend API services, and frontend implementation of visualization. Each of the stocks being labelled is modelled with a separate LSTM autoencoder, using historical data retrieved using the Yahoo Finance API, and this model is built at the base of the logic of automating anomaly detection based on reconstruction error.

The backend was implemented using Flask, and included RESTful API services to access model training, anomaly and reconstruction visualisation services. To support a simpler use of a faster build time, the backend API services were coupled to use Vite for development work streams.

The frontend of StockGuard was developed using TypeScript and React to allow for a fast and interactive UI that had spaces for entering a stock ticker, date range, and entry stocks for the anomaly visualisation of original vs reconstructed stocks by detected anomalies. For development purposes, I built components as charts while including dynamic charts for user interactions and real-time feedback of my model performance.

The stakeholders for the project include new investors, those new to learning data science concepts, and more experienced traders looking for a platform that provides transparency and education for performance analysis of stock anomalies. Some resources I used through the planning stages of the project included yfinance (for the historical stock access), a GPU for sufficiently model training resources, and libraries such as TensorFlow, Matplotlib and Plotly for compute and visualisation. There were not really any major assumptions occurred.

* 1. Project Lifecycle

The project adopted an **iterative and modular development lifecycle**. Instead of a rigid waterfall approach, we opted for a flexible workflow where each major component (modelling, backend, frontend) was developed in isolation and integrated progressively. This allowed for parallel development and quick iteration based on observed anomalies, user interaction feedback, and model performance.

**Lifecycle Stages:**

1. **Requirement Gathering & Ideation** – Identified the need for a per-stock anomaly detection tool.
2. **Data Collection & Preprocessing** – Fetched historical OHLC (Open, High, Low, Close) data from Yahoo Finance via Yfinance API.
3. **Model Development** – Built a modular LSTM Autoencoder pipeline for time-series anomaly detection using sliding windows.
4. **Frontend Integration** – Created a web interface that accepts user input (stock ticker, date range), visualizes data and anomalies.
5. **Testing & Validation** – Compared anomalies with market events and ensured model accuracy using reconstruction error plots.
6. **Documentation & Final Touches** – Detailed the system architecture, model logic, and user guide for future development.
   1. Project Setup

To ensure a modular, scalable, and efficient development process, we made several key decisions regarding the architecture and tools used in StockGuard. The project was divided into three major components: **Frontend**, **Backend**, and **Model Training & Inference Pipeline**. Below are the primary decisions and configurations made:

**1. Frontend**

* **Framework**: React.js
* **Bundler**: Vite – chosen for its fast development and build speed.
* **Styling**: Tailwind CSS – used for rapid and responsive UI design.
* **Visualization**: Chart.js and custom D3-based components for plotting anomalies and reconstruction errors.
* **Routing & State Management**: React Router and Context API.

**2. Backend**

* **Framework**: Flask – a lightweight Python web framework suitable for API endpoints and model handling.
* **Role**: The backend handles ticker-specific preprocessing, dynamic training of the LSTM Autoencoder model, and returns JSON responses with detected anomalies and model insights.
* **Endpoints**: /train, /predict, /explain, and /visualize.

**3. Model & Data Pipeline**

* **Language & Library**: Python with TensorFlow/Keras.
* **Data Source**: Yahoo Finance API (yfinance) – used to fetch lifetime OHLC (Open, High, Low, Close) data for Indian and global stocks.
* **Windowing Strategy**: Rolling windows with normalization to maintain temporal consistency.
* **Model**: LSTM-based Autoencoder for detecting anomalies based on reconstruction error.
* **Training**: Model is trained dynamically per user-selected stock to ensure specificity and avoid overfitting across multiple asset classes.

**4. Deployment & Local Hosting**

* **Frontend**: Hosted locally via Vite’s dev server or build artifacts.
* **Backend**: Hosted via Flask (local server) with endpoints connected to the frontend via Axios.
* **Model Checkpoints**: Stored in a saved\_models directory, specific to each stock.

|  |  |
| --- | --- |
| **#** | **Decision Description** |
| 1 | Frontend: React with TypeScript using the Vite bundler for a fast and modular UI |
| 2 | Backend: Flask (Python) for API development and anomaly detection model integration |
| 3 | Version Control: Git with GitHub repository for code management and collaboration |
| 4 | Modeling Framework: TensorFlow/Keras for LSTM Autoencoder-based anomaly detection |
| 5 | Deployment Plan: Local testing with potential cloud deployment for future scalability |
| 6 | APIs Used: yfinance for fetching live stock market data |
| 7 | Target Users: Tool designed for both beginner investors and advanced market analysts |

* 1. Stakeholders

This project involves multiple stakeholders, each playing a vital role in the development, evaluation, or eventual use of **StockGuard**. Identifying these stakeholders helps clarify responsibilities and expectations throughout the project lifecycle.

**1. Development Team**

* **Role**: Core designers and implementers of the system.
* **Responsibility**: Full-stack development, model implementation, testing, integration, and documentation.

**2. Project Instructor / Academic Supervisor**

* **Role**: Guides the direction of the project, ensuring academic and technical standards.
* **Responsibility**: Periodic evaluation, milestone approvals, and providing feedback on system design and implementation quality.

**3. End Users**

* **Categories**:
  + **Beginner Traders**: Individuals new to the stock market who can use StockGuard to understand market anomalies and learn about suspicious patterns.
  + **Experienced Investors**: Analysts or seasoned traders who seek deeper, model-driven insights into unusual market activity.
* **Responsibility**: Interact with the platform for market analysis and provide feedback for improvements.

**4. Testers**

* **Role**: Team members or external individuals assigned to test the tool across various use cases and datasets.
* **Responsibility**: Identify bugs, usability issues, and performance bottlenecks.

**5. Technical Support (Internal)**

* **Role**: Ensure smooth functioning during demos, presentations, and evaluations.
* **Responsibility**: Address runtime errors, maintain local deployment setup, and troubleshoot backend/frontend issues.

**6. Future Developers / Maintainers**

* **Role**: Individuals who may expand, scale, or refactor the project post-submission.
* **Responsibility**: Understand existing architecture and improve the tool for broader deployment or real-time usage.

**7. Evaluators / Jury Panel**

* **Role**: Academic or professional panel reviewing the final submission.
* **Responsibility**: Assess project outcomes based on innovation, correctness, relevance, and presentation quality.
  1. Project Resources

The successful development and execution of the **StockGuard** system relies on a combination of human, technological, and software resources. The following resources were identified as essential for the project:

**1. Human Resources**

* **Project Team Members**:
  + 3–4 undergraduate students with experience in full-stack development, machine learning, and data visualization.
* **Faculty Supervisor / Guide**:
  + Provides technical and academic guidance, approves milestones, and oversees overall progress.
* **Peer Testers / Volunteers**:
  + Assist in usability testing, feedback collection, and simulated user sessions.

**2. Hardware Resources**

* **Personal Laptops** (4 units):
  + Minimum 8 GB RAM, multi-core processor, with support for local model training and testing.
* **Cloud or Lab Systems (Optional)**:
  + In case of resource-intensive training tasks, cloud-based virtual machines or college lab systems may be used temporarily.

**3. Software Resources**

* **Programming Languages**:
  + Python (for model training and backend API with Flask)
  + JavaScript (React.js frontend with Vite and Tailwind CSS)
* **Libraries & Frameworks**:
  + TensorFlow / Keras: Deep learning model implementation
  + Flask: Backend API
  + yFinance: Stock data extraction
  + Matplotlib / Plotly: Data visualization
  + Scikit-learn / NumPy / Pandas: Data preprocessing and analysis
* **Development Tools**:
  + Visual Studio Code (VSCode)
  + Jupyter Notebook (for model exploration and testing)
  + GitHub (version control and collaboration)
  + Postman (API testing)
* **Frontend Build Tools**:
  + Vite (for fast React builds and bundling)
  + Tailwind CSS (for UI styling and responsiveness)

**4. Data Resources**

* **Stock Market Data Source**:
  + Yahoo Finance via yfinance API, offering OHLC and volume data for selected Indian stocks.

**5. Other Resources**

* **Internet Connectivity**:
  + Stable access for data fetching, version control sync (Git), and collaboration tools.
* **Presentation Tools**:
  + Microsoft PowerPoint / Google Slides
  + Screen recording and video editing tools for demo preparation

|  |  |  |
| --- | --- | --- |
| **Resource** | **Resource Description** | **Quantity** |
| Database Server | A database server provided by the sponsoring company. | 1 |
| Capstone Team | Our team of students who will be the primary developers of the project. | 4 |
| Jim Somebody | The mentor who will be able to provide us with technical assistance. | 1 |
| Mac Workstation | An OS X workstation with X Code for developing the OS X version of the software. | 1 |
| Android Phone | An Android phone to be used as test hardware for the mobile version of the software. | 2 |

* 1. Assumptions

The development and deployment of the **StockGuard** project are based on the following key assumptions:

1. **Data Availability**  
   It is assumed that stock market data (historical OHLC and volume) will remain accessible via the Yahoo Finance API (yfinance) throughout the development period.
2. **Stable Internet Access**  
   Team members will have consistent access to the internet to facilitate collaboration, data fetching, API calls, version control (GitHub), and cloud services (if needed).
3. **Tool and Library Support**  
   All chosen software libraries (e.g., TensorFlow, Flask, React, yFinance) and development tools (e.g., VSCode, Postman, Jupyter Notebook) will continue to be supported and free for academic use during the project duration.
4. **Team Availability**  
   All team members are expected to remain actively involved throughout the project timeline, including development, testing, documentation, and presentation phases.
5. **Hardware Sufficiency**  
   The personal computing devices used by team members are assumed to be capable of handling all model training, testing, and frontend/backend development workloads locally.
6. **Faculty Support**  
   It is assumed that the assigned faculty mentor will be available for periodic reviews, approvals, and feedback sessions as outlined in the academic schedule.
7. **Timeframe Adherence**  
   The academic calendar will remain unchanged, and no unexpected delays (e.g., exam rescheduling, institutional holidays) will interfere with the project schedule.
8. **User Environment Compatibility**  
   The final product is assumed to be accessed by users with modern web browsers that support JavaScript, React, and interactive chart visualizations without compatibility issues.

|  |  |
| --- | --- |
| **#** | **Assumption** |
| A1 | It is assumed that stock market data (historical OHLC and volume) will remain accessible via the Yahoo Finance API (yfinance) throughout the development period. |
| A2 | All team members are expected to remain actively involved throughout the project timeline, including development, testing, documentation, and presentation phases. |
| A3 | The academic calendar will remain unchanged, and no unexpected delays (e.g., exam rescheduling, institutional holidays) will interfere with the project schedule. |
| A4 | The final product is assumed to be accessed by users with modern web browsers that support JavaScript, React, and interactive chart visualizations without compatibility issues. |

1. Project Tracking
   1. Tracking

The **StockGuard** project was tracked and managed using a combination of modern development tools and collaborative platforms to ensure transparency, version control, and efficient progress monitoring. Below are the key components of our tracking system:

1. **Source Control and Version Management**
   * **Platform Used**: Git and GitHub
   * **Repository Access**: The entire project, including frontend, backend, and model scripts, was version-controlled through a private GitHub repository.
   * **Usage**: Branching strategies were employed to separate development, feature additions, and bug fixes. Pull requests were reviewed and merged systematically to maintain code quality and avoid conflicts.
2. **Project Management and Task Tracking**
   * **Platform Used**: GitHub Projects
   * **Usage**: A Kanban-style board was created within GitHub to assign and track tasks across different phases such as Research, Data Preprocessing, Model Development, Frontend/Backend Integration, Testing, and Documentation.
   * **Status Updates**: Issues were regularly created, updated, and closed to reflect real-time progress.
3. **Bug Tracking and Issue Logging**
   * **Platform Used**: GitHub Issues
   * **Usage**: Bugs encountered during development and testing were logged using GitHub’s Issues feature, categorized by severity and assigned to relevant team members. Fixes were linked to commits and pull requests for traceability.
4. **Testing and Regression Management**
   * **Tools Used**: Custom Python scripts and Jupyter Notebooks
   * **Regression Testing**: Key modules—such as the LSTM autoencoder, normalization pipeline, and data visualization components—were tested iteratively.
   * **Storage**: Testing scripts and logs were stored in dedicated subdirectories within the GitHub repository.
5. **Documentation and Communication**
   * **Collaboration Platform**: Google Docs and WhatsApp
   * **Usage**: Design discussions, progress updates, meeting notes, and formal documentation drafts were maintained on Google Docs. Regular team syncs and quick decisions were made via WhatsApp.

|  |  |  |
| --- | --- | --- |
| **Information** | **Description** | **Link** |
| Code Storage | Project code will be stored in SVN repository. | Link |
| Bug Tracking | Bug tracking will be done with Trac. | Link |
| Project Documents and Assignments | Weekly reports, specification and design documents, etc. will be stored in our SVN repository. | Link |
| Continuous Integration | Continuous integration will be done with Jenkins. | Link |
| Regression Testing | Regression testing will use JUnit unit tests and Jenkins. | Link |

* 1. Communication Plan

Effective communication was critical throughout the StockGuard project development to ensure that all team members and stakeholders were kept informed and aligned. The communication plan outlined below describes the types of communications that will take place, how often they will occur, and who will be responsible for them, as well as the platforms used.

**1. Internal Team Communication**

* **Purpose**: Day-to-day coordination, progress updates, technical discussions, and issue resolution.
* **Participants**: All developers, data scientists, and UI/UX contributors.
* **Medium**: WhatsApp group for quick updates, Google Meet for virtual meetings, and GitHub for code-related communication.
* **Frequency**: Daily informal communication, with structured weekly review meetings.

**2. Communication with Project Advisor / Instructor**

* **Purpose**: Progress reporting, milestone approvals, and academic guidance.
* **Participants**: Entire project team and assigned faculty advisor.
* **Medium**: Email for formal updates and meeting scheduling; in-person or virtual meetings for milestone reviews.
* **Frequency**: Biweekly formal updates, milestone-specific meetings.

**3. Documentation and Knowledge Sharing**

* **Purpose**: Maintain shared understanding and archive decisions made during development.
* **Participants**: All team members and instructors (for review).
* **Medium**: Google Docs for collaborative editing; GitHub Wiki for technical documentation.
* **Frequency**: Ongoing updates as tasks are completed or revised.

**4. Presentation and Demonstrations**

* **Purpose**: Showcase progress and obtain feedback at key development stages.
* **Participants**: Project team, advisor, faculty panel, and peer audience.
* **Medium**: In-person demo sessions, recorded video walkthroughs, PowerPoint presentations.
* **Frequency**: At designated checkpoints—mid-semester review, final evaluation, and exhibition day.

**5. Feedback and Evaluation**

* **Purpose**: Gather constructive feedback and assess project performance.
* **Participants**: Faculty advisor, external evaluators, peers.
* **Medium**: Google Forms (for surveys), verbal feedback during presentations.
* **Frequency**: During and after formal presentation events.
  1. Deliverables

Table 11: Deliverables

|  |  |
| --- | --- |
| **#** | **Deliverable** |
| 1 | Study results: Literature review and comparative study on anomaly detection methods in stock markets. |
| 2 | Code: Complete source code for frontend (React + Vite), backend (Flask API), and model training (LSTM Autoencoder). |
| 3 | Test and test results: Unit and integration test cases for core modules, along with logs of model evaluation and reconstruction error analysis. |
| 4 | Build process documents: Instructions to build the project locally including environment setup and dependency management. |
| 5 | Install process documents: Steps for deployment and installation of the system in a new environment. |
| 6 | Administrator or user manual : User guide explaining how to use StockGuard, interpret graphs, and query specific stocks. |
| 7 | Postmortem document: Retrospective report covering project challenges, what went well, and lessons learned. |
| 8 | Final report: Final project deliverables including the PowerPoint presentation, 3-minute video walkthrough, and final sprint summary. |

1. SYSTEM ANALYSIS AND DESIGN
   1. Overall Description

The objective of this project is to use deep learning for time-series anomaly detection in financial markets. More specifically, the goal is to identify anomalous or suspicious activity in stock price changes. The central system is based on an LSTM Autoencoder architecture, and this style is one of the more effective ways to account for temporal dependencies and to detect sometimes minor deviations from "normal" patterns in stock market data. The populated user interface allows the user to enter a single stock ticker (e.g., INFY.NS) and date range, and then system pulls historical OHLC data for processing via a rolling window normalization, and then report anomalies based on reconstruction errors that the model introduces into the system.

The front-end application was built using React and styled with Tailwind CSS, and bundled as a Vite app to increase development speed and performance. The backend application was developed in Flask and facilitates data ingestion from Yahoo Finance through yfinance library, the data preprocessing, model training, anomaly detection logic and at the end visualizing the identified anomalies on price-time plots and allowing users to click on the identified anomaly to see the explanation for the why the model thought that point was unusual given user-defined parameters. Our intention is to provide novice and very experienced traders with insights into the occurrence of unusual market activity to allow for more data-driven choices and an overall more transparent market.

* 1. Users and Roles

Table 12:

|  |  |
| --- | --- |
| **User** | **Description** |
| Developer | |  | | --- | |  |  |  | | --- | | A team member responsible for building and maintaining the StockGuard system. This includes developing the React frontend, Flask backend, implementing the LSTM Autoencoder model, and integrating all components into a cohesive anomaly detection pipeline. | |
| Beginner Trader | A novice user of the stock market who uses StockGuard to understand what constitutes unusual behavior in stock prices. They rely on simplified visualizations and auto-generated explanations to learn about potential manipulation or market anomalies. |
| System Administrator | Backend components and scripts that run independently to fetch live stock data, normalize it, train the LSTM Autoencoder, detect anomalies, and generate model outputs without needing manual intervention. These agents ensure smooth automation throughout the system workflow. |

* 1. Design diagrams/Architecture/ UML diagrams/ Flow Charts/ E-R diagrams
     1. Product Backlog Items

| **User Story #** | **User Story** |
| --- | --- |
| 1 | As a beginner trader, I want to visualize anomalies on a stock chart so that I can identify unusual behavior in price movement. |
| 2 | As an experienced investor, I want to view reconstruction error graphs so that I can interpret how the model perceives anomalies. |
| 3 | As a user, I want to input a stock ticker and select a date range so that I can analyze specific periods of interest. |
| 4 | As a developer, I want to train an LSTM Autoencoder model on historical data so that the system can detect anomalies accurately. |

* + 1. Architecture Diagram

A diagram of a process

AI-generated content may be incorrect.

* + 1. Use Case Diagram

A diagram of a diagram

AI-generated content may be incorrect.

1. User Interface
   1. UI Description

Our project contains an interactive and fully functional web-based user interface (UI) built in ReactJS, styled with Tailwind CSS, and set up with Vite to allow for the fastest development builds. The UI is designed to allow users to interact with the system in a low-friction manner while still being highly responsive in real-time and across devices.

Users simply start by supplying a stock ticker symbol in standard format (e.g. INFY.NS) and selecting a custom date range. The UI utilizes a backend processing request made by the user and displays:

A Price vs Time graph with points to indicate anomalies detected.

An option to see the reconstruction error over time.

A feature that explains the anomaly marker when clicked on the graph, with respect to “Why this anomaly?”, based on logic derived from deviations present in the trained model.

The interface is equipped to address both beginner traders, who will respond positively to the visual stimuli and explored resolutions to the query, and the experienced investor, who may glean greater insight into recurring and deviated trends based on the anomaly detected. Overall, the user interface is core to StockGuard's overall aim to make market anomaly detection easy to understand, yet interactive and insightful.

UI Mockup

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

1. Algorithms/Pseudo Code OF CORE FUNCTIONALITY

The system starts with gathering user input, such as a stock ticker symbol (TCS.NS) along with the date range with a start and end date for the analysis dates). The system retrieves historical stock price data on the specified ticker from the Yahoo Finance API and converts all timestamps of the data to the IST time zone.

Then for the pre-processing, a rolling mean (with a 4-observation window) is applied to the price data to smooth out the short-term fluctuations in the price. The data is then split into a training set (which includes observations prior to 2022) and test set (2022 onwards).

Next, in preparation for the model, a suitable number of fixed windows are created from the price series with a defined sliding window size (for example, window size of 30) and a step size (for example, step size of 5). Each of the sliding windows are normalized with the MinMaxScaler, and each of the normalized sliding windows with their indices (corresponding to the date) were stored.

Next, an LSTM Autoencoder model is defined. This architecture includes a small encoder (e.g. a Dense layer of 16 followed by a Dense layer of 8), and then the decoder (e.g. Dense layers of 16 and 30). The model is compiled with the Adam optimizer and Mean Squared Error as the loss function.

The model is trained on the training dataset only, and during training, early stopping is used to monitor validation loss to help reduce overfitting. Once training is completed the model is saved to disk.

To perform anomaly detection, we reconstruct outputs of our training dataset and test dataset with the trained model. To do this we calculate the mean absolute error (MAE) between the original window and the reconstructed window, for both the training dataset sensors and the test dataset sensors. We establish an anomaly threshold by taking the mean MAE of the training set and adding one standard deviation.

To detect anomalies, we identify each test window in the dataset, and we compare the reconstruction error of that window to the threshold. If the error is greater than the threshold, the window is marked as an anomaly.

To allow user interaction, the system takes a custom date range and filters the windows to only contain those relevant to the selected date range. For all the detected anomalies in the provided date range, it plots the original input and the reconstructed signal, shading the plotted graph with any reconstruction error. If no anomalies exist in the date range, the user gets a message that says no anomalies exist in the date range. The detection threshold is printed, and the relevant visualizations are displayed for the date range.

1. Project Closure
   1. Goals / Vision

Our original vision for this project was to build a tool for detecting anomalies in the stock market using deep learning models and providing useful recommendations to end users via an easy-to-use interface. This original plan was to build an LSTM Autoencoder for time-series anomaly detection, along with a frontend to allow for interactive visualization and explanation of detected anomalies.

As we undertook the project, however, the vision transformed into a larger goal: to be able to build StockGuard, a full-stack, real-time anomaly detection system for the Indian stock market. The tool now not only detects potential market manipulation utilizing advanced reconstruction-based logic but also communicates that information via a user-friendly web interface. The system ultimately supports both novice and experienced traders by externalizing price behaviour; suspicious data patterns that are highlighted, and interpretations of anomalies, all supported by a specifically trained model for each stock. This final product is a testament to our vision for using deep learning together with stock market data to create financial data fairness in a usable format.

* 1. Delivered Solution

We intended to plan out a solution to build an end-to-end anomaly detection system for the Indian stock market that can identify and explain unusual price movements using deep learning. The basic concept was to use an LSTM Auto encoder to model normal stock price behavior, look for deviations from this modelling, and flag these behaviors as possible anomalies. It was to be a model that was also connected to a web-based interface for visualizations and user interactions.

The complete solution Stock Guard meets and expands upon this idea. It is a robust full-stack implementation with a Flask back-end used for data pre-processing, model training and anomaly detection, and front-end developed using React, which is styled with Tailwind CSS and uses Vite for speed optimization. Users can select a stock ticker, select a date range, and visualize their results in real time. The resulting price charts highlight any anomaly detection, and there is further inspection of the reconstruction errors and explanations of any detected anomalies through to changes in patterns. The tool should be easy enough for novices to use, while still making it flexible enough for more experienced traders to complete deeper analyses. In addition to the final delivery, we have also organized, and version controlled, all of the code, models and supporting documentation to allow for future extensibility.

* 1. Remaining Work

Although StockGuard provides an effective functional prototype of an anomaly detection system for stock market data, there are several next steps for further implementation and improvements. The current prototype system detects anomalies offline by reproducing algorithm-based behavior through historical batch detections only. One of the biggest improvements going forward will be detecting anomalies in real time, by integrating livestock data feeds, and also making sure the model inference works as quickly as it can.

Yet despite the LSTM autoencoder being useful in detecting anomalies, including context-aware modules (such as news sentiment analysis and macroeconomic variables) may improve the usability and accuracy of the anomalies outlined. Perhaps exploring APIs related to financial news feeds or social sentiment tracker services could be interesting modules to explore.

One aspirational feature for the dashboard would be to enhance it with filtration options, allow users to compare anomalies against other stocks, and give more detail on the anomalies explained within the plots. Time series plots would improve usability as well. Accepting user accounts and personalized watch lists would allow users to create unique notifications for live anomalies—and while these may take time, they would be useful features.

Finally, it would also be useful to benchmark, validate performance, and explain in detail all stocks tested with a broader array of stock data and timeframes to demonstrate any generalizability.

REFERENCES

* [**Manipulator Detection in Cryptocurrency Markets Based on Forecasting Anomalies**](https://ieeexplore.ieee.org/document/9502700)
* [**A Survey on Pump and Dump Detection in the Cryptocurrency Market Using Machine Learning**](https://www.mdpi.com/1999-5903/15/8/267)
* <https://acrobat.adobe.com/id/urn:aaid:sc:AP:f1505a72-d9f0-419e-bbea-e10c62655aa9>
* <https://scholarworks.lib.csusb.edu/cgi/viewcontent.cgi?article=1385&context=ciima|>
* <https://paperswithcode.com/dataset/pump-and-dump-dataset>
* <https://huggingface.co/datasets/sebdg/crypto_data>
* <https://www.blockchain.com/explorer/charts/n-payments>
* <https://docs.coinapi.io/market-data/how-to-guides/retrieve-and-analyze-crypto-order-book-data-using-a-cryptocurrency-API>
* <https://ieeexplore.ieee.org/document/9629409>