LEVERAGING DEEP REINFORCEMENT LEARNING FOR REAL-TIME TRADING IN EMERGING MARKETS: INSIGHTS FROM NIFTY50

by

ANISHKUMAR DHABLIA

DISSERTATION
Presented to the Swiss School of Business and Management Geneva
In Partial Fulfillment
Of the Requirements
For the Degree

DOCTOR OF BUSINESS ADMINISTRATION

SWISS SCHOOL OF BUSINESS AND MANAGEMENT GENEVA

2022

LEVERAGING DEEP REINFORCEMENT LEARNING FOR REAL-TIME

TRADING IN EMERGING MARKETS:

INSIGHTS FROM NIFTY50

by

ANISHKUMAR DHABLIA

Supervised by

MONIKA SINGH

APPROVED BY

Apostolos Dasilas

Dissertation chair

RECEIVED/APPROVED BY:

Renee Goldstein Osmic

Admissions Director

Dedication

To my **teachers**—those who noticed the way numbers and systems lit up my eyes, who kept asking “why?” when an easier answer would have done, and who quietly steered me toward first principles when shortcuts tempted me. You let me wrestle with problems long enough to learn, not just to finish; you turned curiosity into craft and made rigor feel like kindness. Every page of this thesis carries your fingerprints: the habit of checking assumptions, the calm to rebuild when an idea breaks, and the courage to choose the harder, better path.

For seeing my technical acumen before I had a name for it - and for pushing me in the right direction, again - this work is yours as much as mine.

Acknowledgements

I am deeply grateful to my supervisor, **Monika Singh**, for steady guidance, honest critique, and constant encouragement throughout this work. I also thank the **examiners** for their careful reading and constructive comments.

Finally, to my family - for patience on long nights, quiet faith on hard days, and love that never wavered - thank you. This thesis is as much yours as mine.

Any remaining errors are my own.

ABSTRACT

LEVERAGING DEEP REINFORCEMENT LEARNING FOR REAL-TIME TRADING IN EMERGING MARKETS: INSIGHTS FROM NIFTY50

Stock trading is a complex decision-making problem influenced by market volatility, macroeconomic conditions, and investor sentiment. Traditional strategies, such as technical analysis and statistical models, rely on predefined rules and historical patterns but often struggle to adapt to dynamic markets. Reinforcement learning (RL) offers an adaptive approach by enabling trading agents to learn from past experiences and optimize decisions over time. This study applies Q-learning (QN), Deep Q-Network (DQN), and Double Deep Q-Network (DDQN) to intraday trading on NIFTY 50 stocks, evaluating performance based on total profit, risk-adjusted returns, and trade execution efficiency. The models were trained on four years of historical data and tested on one year to assess adaptability to real-world conditions. Results show that DDQN outperforms both QN and DQN, achieving the highest total profit (₹1,151,325), best Sharpe ratio (0.3450), lowest max drawdown (-1.12%), and highest trade accuracy (67.72%). DQN improves over QN but suffers from higher drawdowns due to Q-value overestimation, while QN struggles with profitability and risk control. These findings confirm that RL-based trading models can significantly enhance decision-making and profitability in algorithmic trading.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I:

INTRODUCTION

1.1 Background

Financial markets are highly dynamic, influenced by various factors such as economic policies, global events, and investor sentiment. Traditional stock trading strategies rely on either fundamental or technical analysis, both of which have limitations in adapting to rapidly changing market conditions. The introduction of machine learning (ML) and artificial intelligence (AI) in finance has led to significant advancements in algorithmic trading. Among these, reinforcement learning (RL) has emerged as a powerful tool for decision-making in complex and uncertain environments (Ansari et al., 2024).

1.1.1 Evolution of Algorithmic Trading

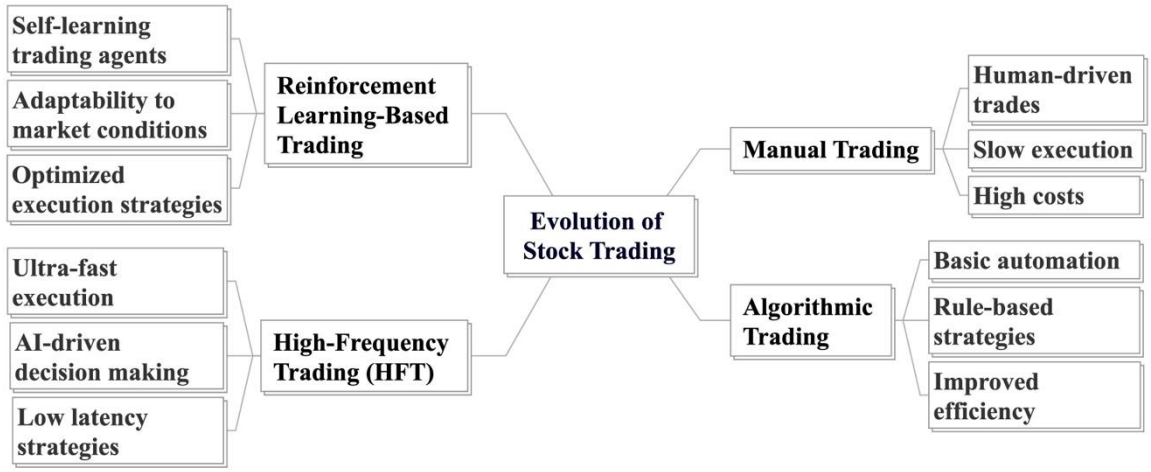

Figure 1: Evolution of Algorithmic Trading

Algorithmic trading has transformed financial markets by automating trade execution based on predefined rules. Early models used statistical approaches such as

moving averages and regression analysis, but these lacked adaptability to real-time market fluctuations. The rise of machine learning-based trading systems allowed for more dynamic strategies, leveraging historical price patterns and market indicators to predict future price movements (Awad et al., 2023).

1.1.1.1 From Manual Trading to High-Frequency Trading

- Manual trading involved human decision-making based on experience and market knowledge.
- The advent of electronic trading enabled faster trade execution, but decisions were still rule-based.
- High-Frequency Trading (HFT) emerged, utilizing algorithms to execute thousands of trades per second, reducing human intervention significantly.

1.1.1.2 Role of Machine Learning in Financial Markets

- Machine learning models, such as LSTMs and CNNs, have been widely applied to financial forecasting.
- Supervised learning models depend on labeled data but struggle with unseen market conditions.
- Reinforcement learning surpasses traditional methods by allowing the model to learn optimal strategies through trial and error (Byun et al., 2023).

1.1.2 Reinforcement Learning in Financial Markets

Reinforcement learning (RL) is an advanced decision-making framework where an agent interacts with an environment and learns from rewards and penalties. It is well-suited for financial markets, where traders must adapt to uncertain and dynamic conditions. RL-based models can self-improve over time, making them ideal for trading applications (Cui et al., 2023).

1.1.2.1 Why Reinforcement Learning for Stock Trading?

- Unlike traditional models, RL-based trading agents do not require predefined rules.
- RL continuously adapts to market conditions by optimizing trading actions (buy, hold, or sell).
- Advanced RL models such as Deep Q-Networks (DQN) and Double Deep Q-Networks (DDQN) have shown superior performance in handling market volatility (Choi & Kim, 2024).

1.1.2.2 Reinforcement Learning vs. Traditional Trading Strategies

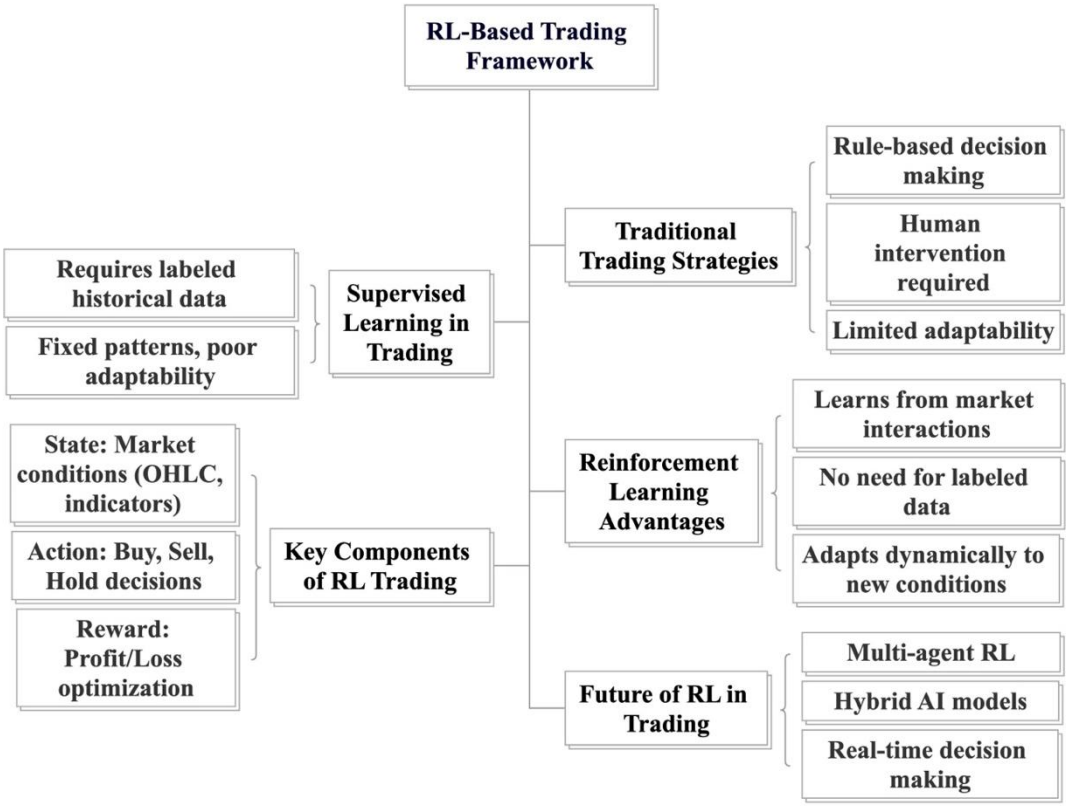

Figure 2: Reinforcement Learning Based Frameworks in Stock Market

Below table provides a comparison of different trading strategies, highlighting their approaches and limitations. It shows how traditional methods like technical analysis and statistical models struggle with adaptability, while reinforcement learning offers real-time learning capabilities, making it more suitable for dynamic market conditions.

Table 1: Comparison of trading strategies and their limitations.

Trading Strategy	**Approach**	**Limitations**
Technical Analysis	Uses indicators like RSI, MACD, moving averages	Struggles with sudden market shifts
Statistical Models	Regression, ARIMA, GARCH	Fails in high volatility scenarios
Supervised ML Models	Uses labeled historical data	Cannot adapt to new trends easily
Reinforcement Learning	Learns dynamically from actions and rewards	Adapts to changing conditions in real-time

1.2 Problem Statement

Stock market prediction and algorithmic trading have been extensively studied, yet achieving consistently profitable trades remains a challenge due to market volatility, unpredictable price movements, and high-frequency fluctuations. Traditional rule-based strategies often fail to generalize across different market conditions, and even advanced machine learning models struggle to adapt to sudden market changes. Reinforcement learning (RL) offers a promising solution by enabling AI agents to learn optimal trading strategies through interaction with the environment (Feizi-Derakhshi et al., 2024). Most existing models rely on historical price patterns, technical indicators, or statistical relationships, which often fail to capture the real-time complexities of stock market behavior. The stock market is highly nonlinear, and the relationship between different market variables is not always explicitly defined. In such cases, rule-based approaches

and even supervised learning techniques struggle to make accurate predictions (Guarino et al., 2024).

This study aims to address the limitations of traditional trading strategies by developing an RL-based AI agent trained on four years of NIFTY 50 stock data. The agent utilizes Deep Q-Networks (DQN) and Double Deep Q-Networks (DDQN) to improve its ability to learn and adapt to dynamic market conditions. The study evaluates the effectiveness of RL-based trading strategies compared to traditional approaches, focusing on profitability, risk management, and stability in financial decision-making.

1.2.1 Challenges in Stock Market Prediction

The unpredictability of stock prices arises from numerous factors, including economic policies, geopolitical events, investor psychology, and sudden market shocks. Traditional forecasting models such as linear regression, ARIMA, GARCH, and decision trees often fail to generalize across different market conditions, leading to poor real-world performance (Huang et al., 2023).

1.2.1.1 Market Volatility and Uncertainty

Financial markets exhibit extreme volatility, making it difficult to predict future price movements with high confidence. Volatility can be caused by:

- Macroeconomic events such as interest rate changes, inflation, or GDP fluctuations.
- Geopolitical risks like international conflicts, trade wars, or regulatory changes.
- Market sentiment and investor behavior, which can lead to panic selling or speculative bubbles.

Traditional machine learning models trained on historical price data often fail to account for these uncertainties, leading to overfitting on past trends. In contrast, RL-

based trading systems can learn to adapt to new market conditions and adjust their strategies dynamically (Bai et al., 2023).

1.2.1.2 Data Complexity and Noise in Market Trends

Financial datasets are high-dimensional and noisy, making it difficult to extract meaningful patterns. Several challenges exist in handling stock market data:

- Price fluctuations contain high levels of noise, leading to inaccurate predictions.
- Technical indicators (e.g., MACD, RSI, Bollinger Bands) add complexity, making feature selection crucial.
- Extreme market events (e.g., flash crashes) create outliers, which can distort model performance.
- To improve data quality, this study applies data preprocessing techniques such as:
- Filtering chaotic data by removing stocks with extreme volume spikes.
- Feature engineering to extract meaningful patterns from price movements.
- Normalization and scaling to ensure stable learning for RL models (Guarino et al., 2024).

Below table outlines key challenges in stock trading and their impact on trading models. It highlights how high volatility, market noise, data sparsity, and overfitting create difficulties for algorithmic trading, affecting prediction accuracy and model reliability in real-time market conditions.

Table 2: Key challenges in processing stock market data and their impact on predictive models

Challenge	Description	Impact on Trading Models

High volatility	Sudden price fluctuations	Difficult to predict short-term trends
Market noise	Random fluctuations unrelated to fundamentals	Can mislead trading algorithms
Data sparsity	Missing data points or irregular reporting	Inconsistent training data
Overfitting risk	Models learn past trends but fail in real-time	Poor generalization to new conditions

1.2.2 Gaps in Existing Reinforcement Learning Trading Models

Despite the success of DQN-based trading systems, several challenges remain. Many models fail to generalize across different market scenarios, leading to inconsistent performance. Furthermore, overestimation bias in Q-learning algorithms affects decision-making accuracy (Espiga-Fernández et al., 2024).

1.2.2.1 Over-Reliance on Historical Data

Most trading models are trained on historical market data, making them susceptible to overfitting. The primary issues include:

- Market conditions are constantly changing, and models trained on old data may fail in new environments.
- Supervised learning models depend on labeled datasets, which are often biased toward past trends.
- Traditional Q-learning models memorize past trades rather than adapting to new patterns.

Reinforcement learning provides a solution by continuously updating its strategy based on real-time feedback from market conditions (Du & Shen, 2024).

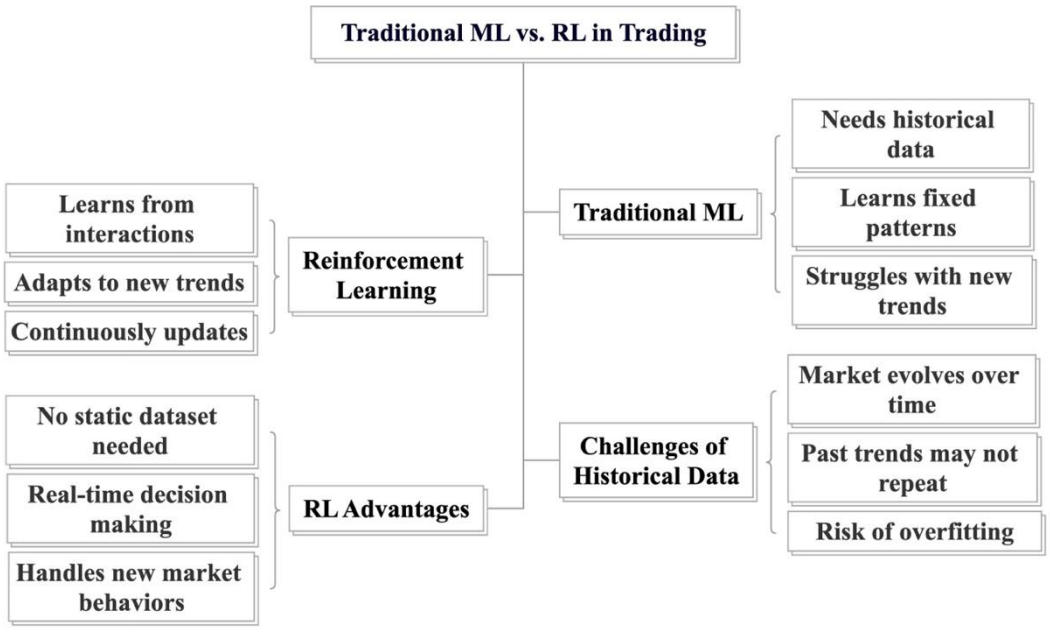

Figure 3: Comparison of Traditional Machine Learning vs. Reinforcement Learning in Trading

1.2.2.2 Lack of Stability in Q-Learning Approaches

Standard Q-learning algorithms suffer from high variance and instability, making them unreliable for trading applications. Challenges include:

- **Overestimation bias**: The Q-learning agent often assigns unrealistically high values to certain actions, leading to suboptimal trading decisions.
- **Unstable training**: The model may experience high fluctuations in performance during training, leading to inconsistent trading behavior.
- **Delayed rewards**: In financial markets, the impact of a trade may not be immediately visible, making it difficult for standard RL models to learn effectively.

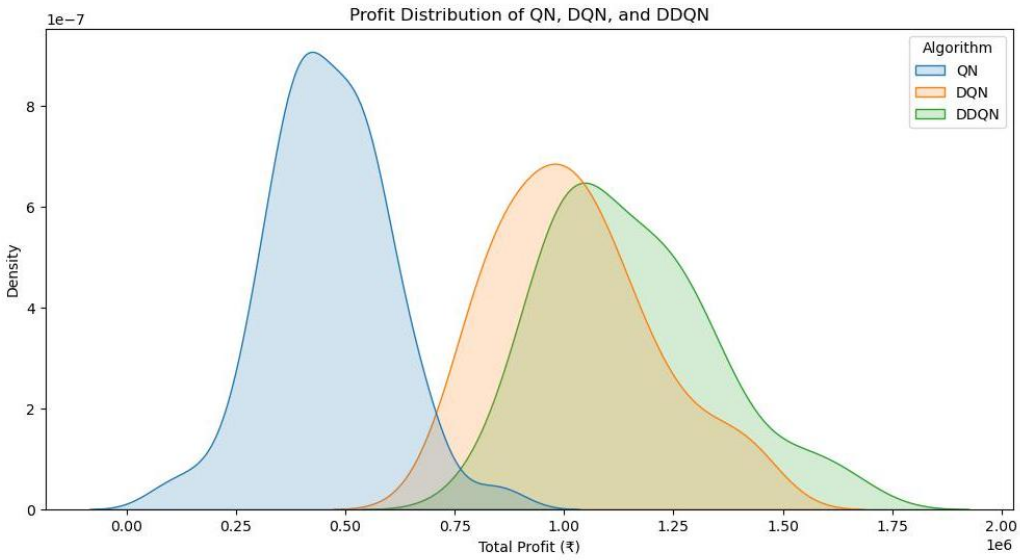

Figure 4: Q-learning vs. DQN vs. DDQN – A Comparative Study on Trading Performance

Deep Q-Networks (DQN) improve Q-learning by incorporating:

- Experience replay, which helps stabilize training by storing and reusing past experiences.
- Target networks, which prevent the model from becoming too sensitive to recent rewards.

However, DQN still has limitations, leading to the need for Double Deep Q-Networks (DDQN), which further improves performance by:

- Decoupling action selection from Q-value estimation, reducing overestimation bias.
- Enhancing stability in learning, leading to more consistent trading strategies (Papageorgiou et al., 2024).

1.2.3 Addressing These Challenges with RL-Based Trading Strategies

To overcome these challenges, this research proposes an RL-based AI agent for intraday trading, specifically designed to:

- Handle market volatility by dynamically adjusting trading strategies.
- Reduce data noise and overfitting by integrating technical indicators and advanced feature selection techniques.
- Improve learning stability by implementing DDQN instead of standard Q-learning.
- Optimize risk management by evaluating Sharpe ratio, win-loss ratio, and cumulative profit as performance metrics.

1.3 Research Questions

This research aims to explore whether reinforcement learning, particularly the Double Deep Q-Network (DDQN) framework, can outperform traditional intraday trading strategies and be adapted to the specific market conditions of an emerging economy like India's NIFTY 50 index. The study will focus on optimizing buy, hold, and sell decisions, enhancing risk management, and improving generalization across different market conditions.

1. How effectively can a Double Deep Q-Network (DDQN)-based reinforcement learning agent autonomously execute optimal buy, hold, and sell decisions in the NIFTY50 intraday trading market?
2. How can the DDQN-based RL agent be optimized to balance maximizing profitability with minimizing market risk, particularly during volatile periods?

3. How well does the DDQN model generalize across different market conditions (e.g., bull markets, bear markets, and periods of high volatility) in the NIFTY50 index?
4. Can reinforcement learning models like DDQN handle market anomalies events that deviate significantly from normal market behavior?
5. What is the impact of experience replay and target networks on improving the stability and learning efficiency of the DDQN model in the context of intraday trading?
6. How does the exploration-exploitation tradeoff affect the performance of the DDQN agent in intraday trading, and how can it be managed for optimal decision-making?

1.3.1 Primary Objectives

Develop a DDQN-based reinforcement learning agent for intraday trading

- Train the agent to autonomously execute buy, hold, and sell decisions in the NIFTY 50 market.
- Implement Q-learning, DQN, and DDQN for comparative performance analysis.

Optimize the DDQN agent for balancing profitability and risk management

- Design reward functions that maximize cumulative profit while minimizing market exposure risks.
- Evaluate agent performance during high-volatility periods and unexpected market fluctuations.

Assess the generalization ability of the DDQN model

- Test the RL agent's adaptability across bull markets, bear markets, and volatile periods.
- Investigate how market conditions impact the learning behavior of the model.

Analyze the impact of experience replay and target networks

- Measure the effect of experience replay on learning efficiency and model stability.
- Evaluate how target networks reduce Q-value overestimation, leading to more stable trading decisions.

Optimize the exploration-exploitation tradeoff for decision-making

- Tune epsilon decay strategies to balance exploration (learning new patterns) and exploitation (executing profitable trades).
- Assess how different exploration strategies impact trade frequency, returns, and market adaptability.

Table 3: Key research areas and expected improvements in RL-based trading.

Research Focus	**Key Considerations**	**Expected Outcome**
DDQN trading performance	Buy, hold, and sell execution in NIFTY 50	More profitable and efficient trading decisions
Risk management strategies	Handling volatility, drawdowns, and stop-loss	Improved capital protection and lower risk
Generalization ability	Performance in bull, bear, and volatile markets	More stable profits across different market conditions
Experience replay & target networks	Stability and learning efficiency	Reduced overestimation bias, better model training
Exploration-exploitation tradeoff	Tuning epsilon decay for decision-making	Optimized trade frequency and adaptability

Above table presents the key research focus areas in reinforcement learning-based trading, along with their considerations and expected outcomes. It highlights how DDQN trading performance, risk management, generalization ability, experience replay, and exploration-exploitation balance contribute to building a more stable, efficient, and adaptable trading model for intraday stock markets. This research will provide valuable insights into the practical application of reinforcement learning in intraday trading, ensuring profitability, risk management, and adaptability to market conditions.

1.3.2 Secondary Objectives

Improve data preprocessing for better RL model training

- Use technical indicators like moving averages, RSI, and Bollinger Bands.
- Filter out market noise and extreme volume fluctuations.

Enhance RL agent stability through improved training techniques

- Implement experience replay and target networks to stabilize learning.
- Use epsilon decay strategies to refine exploration-exploitation balance.

Optimize hyperparameters for efficient model performance

- Tune learning rate, discount factor, batch size, and number of episodes.
- Experiment with different reward functions to encourage profitable trading behavior.

Benchmark RL-based strategies against standard market indicators

- Compare the RL agent's performance with technical analysis indicators.
- Test against common trading benchmarks (e.g., SMA crossovers, MACD).

1.4 Scope of the Study

This research focuses on developing and evaluating a reinforcement learning-based trading agent, specifically using the Double Deep Q-Network (DDQN) framework, for intraday trading in the NIFTY 50 stock market. The study aims to enhance trading

strategies by leveraging reinforcement learning techniques to optimize buy, hold, and sell decisions while balancing profitability and risk management. The study covers data collection, preprocessing, model development, and performance evaluation using four years of historical market data for training and one year for testing. The trading agent will operate in a simulated trading environment, where its decisions will be evaluated against traditional trading strategies and benchmark models.

1.4.1 Inclusion Criteria

Stock Market Focus: NIFTY 50

- The study is limited to NIFTY 50 stocks, which represent India's top 50 publicly traded companies.
- The dataset includes intraday OHLCV data (Open, High, Low, Close, Volume) extracted using Zerodha API.

Timeframe of Study

- Four years of data (training period): Used to train the RL agent and refine trading policies.
- One year of data (testing period): Used to assess generalization and profitability in real-world conditions.

Reinforcement Learning Models

- The study focuses on Q-learning, Deep Q-Networks (DQN), and Double Deep Q-Networks (DDQN).
- Comparative analysis will be performed against rule-based strategies (e.g., Moving Averages) and supervised learning models (e.g., LSTMs, Random Forests).

Table 4: Summary of study scope and model evaluation criteria

Aspect	Details
Market Index	NIFTY 50 (Top 50 Indian stocks)
Trading Type	Intraday trading (9:15 AM – 3:00 PM IST)
Data Source	Zerodha API (Historical OHLCV data)
Training Period	4 years of stock data
Testing Period	1 year of stock data
Trading Models	Q-learning, DQN, DDQN
Performance Metrics	Cumulative Profit, Sharpe Ratio, Win-Loss Ratio, Drawdown

Above table provides an overview of the key aspects of the study, including the market index, trading type, data source, training and testing periods, trading models, comparison models, and performance metrics. It outlines the scope of the research, ensuring a structured evaluation of reinforcement learning-based trading strategies in the NIFTY 50 intraday market.

1.4.2 Exclusions

Exclusion of Fundamental Analysis

- The study does not incorporate fundamental indicators such as company earnings, P/E ratios, or macroeconomic factors.
- The focus is solely on technical analysis and price-action-based trading strategies.

Exclusion of Alternative Asset Classes

- The research does not consider cryptocurrencies, forex, commodities, or derivatives.

- It is strictly limited to equity stocks listed on the NIFTY 50 index.

No Real-World Deployment

- The RL agent is tested only in a simulated environment using historical market data.
- Live deployment on real trading accounts is outside the scope of this study.

The scope of this research ensures a focused and practical evaluation of reinforcement learning in intraday trading, allowing for a controlled comparison between traditional and AI-based trading models.

1.5 Contributions of the Research

This research makes significant contributions to the field of reinforcement learning in financial markets, particularly in intraday trading using the NIFTY 50 index. By leveraging Double Deep Q-Networks (DDQN), the study enhances trading decision-making, ensuring a balance between profitability, risk management, and market adaptability. The research provides both theoretical and practical insights into how RL-based trading systems can outperform traditional models and adapt to dynamic market conditions.

1.5.1 Theoretical Contributions

Advancing Reinforcement Learning for Stock Market Trading

- Demonstrates the effectiveness of DDQN over traditional Q-learning and Deep Q-Networks (DQN).
- Investigates how experience replay, and target networks stabilize trading decisions.

Understanding the Role of RL in High-Volatility Markets

- Analyzes the adaptability of RL agents during bull, bear, and highly volatile periods.

- Examines how reinforcement learning models handle market anomalies compared to rule-based strategies.

Enhancing Exploration-Exploitation Strategies in Intraday Trading

- Investigates how epsilon decay strategies affect RL trading behavior.
- Optimizes the balance between exploring new strategies and exploiting profitable ones.

Table 5: Summary of theoretical advancements in reinforcement learning-based trading models

Contribution	Description	Impact
DDQN in Stock Trading	Evaluates DDQN performance over Q-learning and DQN	More stable and profitable trading strategies
Risk and Volatility Adaptation	Studies RL models in various market conditions	Helps in developing risk-aware trading systems
Exploration-Exploitation Tuning	Optimizes epsilon decay for RL decision-making	Enhances adaptability and trading efficiency

Above table highlights the key contributions of the study, describing their purpose and impact. It demonstrates how evaluating DDQN, improving risk adaptation, and optimizing exploration-exploitation tradeoffs contribute to the development of more stable, efficient, and risk-aware reinforcement learning-based trading strategies.

1.5.2 Practical Contributions

Development of a Scalable RL-Based Trading System

- Implements an AI-driven reinforcement learning agent that can be integrated into algorithmic trading platforms.
- Uses real-world financial data (NIFTY 50) to train and evaluate the RL agent.

Optimizing Risk-Adjusted Trading Strategies

- Designs a reward function that incorporates profitability, volatility, and drawdown constraints.
- Implements Sharpe ratio and drawdown-based optimization to reduce trading risks.

1.5.3 Implications for Future Financial Markets

- Automation in Trading: RL-based agents can help in automating high-frequency and intraday trading decisions.
- Risk-Aware AI Strategies: RL models can be fine-tuned to reduce risk exposure in volatile markets.
- Scalability to Other Financial Markets: While this study focuses on NIFTY 50, the framework can be extended to other stock indices, cryptocurrencies, and forex markets.

This study contributes to the advancement of reinforcement learning in financial applications, bridging the gap between academic research and real-world trading.

1.5.4 Risk Disclosure & Non-Advisory Note

This thesis is a research document, not investment advice. The systems studied operate under controlled assumptions and historical data. Live trading introduces additional risks (execution, outages, regulation) that are outside scope. Any application of these methods to real capital must undergo independent validation, risk review, and compliance approvals.

1.6 Business Relevance for AMCs and Large Institutional Desks

Asset managers live in a world of daily subscriptions and redemptions, benchmark pressure, fee compression, and intense oversight. Intraday, the problem is simple but unforgiving: turn capital and client flows into steady cash generation while

protecting execution quality and auditability. This thesis is relevant because it turns the firm's own reward logic (post-cost, after spread and latency) into actionable supervision for models and does so using asset-agnostic technical indicators that travel cleanly across equities, futures, FX, and crypto.

What problem does it solve for an AMC?

- **Speed to strategy.** PM teams and central research can spin up intraday ideas quickly because supervision comes from the same objective used to judge P&L. That shortens the path from idea to desk-ready pilot—no endless debates about ad-hoc tags or thresholds.
- **Cashflow and flow-of-volume.** Daily flows (in/out) force AMCs to buy and sell on the tape. The framework helps time those micro-decisions within the day, aligning with typical U-shaped volume curves and pockets of liquidity, which reduces slippage and stabilizes **net basis points retained**.
- **Execution discipline.** Because the labels are tied to post-cost returns, the same logic that trains the model can enforce **don't trade** when the tape is thin or spreads are wide. Doing nothing at the wrong moment is often the best trade; the system makes that choice explicit.
- **Audit and model-risk.** Every decision carries a recorded "why": which action won and by how much. This makes investment-committee reviews, client due-diligence, and regulator questions faster to close—no folklore, just a traceable margin by which one action beat the alternatives.

How would a large desk actually use it day-to-day?

- Pre-trade: Before the open, PMs and the central execution desk review a short scenario brief ("where is the model decisive/ambivalent today?"). Flows from

client orders or internal rebalances are bucketed against those decisiveness windows to plan participation rates and venue selection.

- In-trade: The model's intraday signals sit alongside broker algos (VWAP/POV/IS) in the EMS. When label margins compress (market uncertain), the desk automatically leans toward lower participation or passive queues; when margins are decisive, it leans in. Human oversight remains near-ties are flagged for trader discretion, and all overrides are logged with reasons.
- Post-trade: Attribution splits P&L into alpha earned and costs saved. Losses with large historical margins trigger a spec review; losses on near-ties are treated as noise. This changes the tone of post-mortems from blame to diagnosis.

Why this matters for scale and cross-asset rollout. AMCs rarely want one-off toys; they want **platforms**. Because inputs here are only technical indicators—and everything is measured in returns, z-scores, and ATR/volatility units—the same stack can be re-used across strategies and asset classes. What changes are **execution assumptions** (fees, spread, latency) and capacity limits. That means a single investment in research engineering supports multiple desks: large-cap equities today, sector sleeves next quarter, a futures overlay later, and even **crypto liquidity programs** where 24×7 flow demands round-the-clock intraday discipline.

Commercial outcomes an AMC can expect.

- **Faster approvals.** Investment committees get a cleaner memo: “We trained on the same objective we report; here are the decisive vs. ambiguous regions; here’s the audit log.” Approval cycles shrink.
- **Basis-points retained.** Better timing relative to flow-of-volume and spreads shows up as lower implementation shortfall and tighter slippage bands—material in high-turnover books and cash equitization mandates.
- **Capacity awareness.** ADV-based caps and margin-aware throttles are built into the governance design, reducing the risk of crowding and flow toxicity as AUM grows.
- **Client trust.** Transparent decision logs and consistent post-trade narratives make it easier to defend process quality with boards, consultants, and regulators.

Where this sits in the operating model.

- **PMs** get a faster way to turn hypotheses into intraday tactics that respect their risk budgets.
- **Execution** gains a “traffic-light” layer that tells them when to push, when to shade, and when to stand down—fully logged.
- **Risk and Compliance** receive versioned configs, decision logs, and margin distributions that map directly to policies on limits, outages, and incident response.
- **Data/Tech** avoid brittle pipelines: only OHLCV is needed; the same code runs across desks; releases follow a shadow → canary → full pattern with instant rollback.

Why now. Spreads are thin, venues fragment, and oversight tightens yearly. AMCs must **manufacture bps** through timing, cost control, and clean process. This thesis provides a method that is **economically coherent** (labels match the objective), **portable** (features are asset-agnostic), and **auditable** (decisions explain themselves). In other words, it's not just a research result—it is a practical playbook for converting flow and volatility into repeatable, defensible intraday returns at scale.

1.7 Thesis Organization

This thesis is structured to provide a comprehensive study on reinforcement learning-based intraday trading using the Double Deep Q-Network (DDQN) framework. Each chapter follows a logical progression, from the background and literature review to the methodology, results, and conclusions.

Chapter 1: Introduction

- Introduces the motivation, challenges, and research objectives of the study.
- Defines the scope of research and key contributions in reinforcement learning-based trading.

Chapter 2: Literature Review

- Reviews existing work on reinforcement learning in financial markets.
- Explores Deep Q-Networks (DQN), Double Deep Q-Networks (DDQN), and their applications in stock trading.
- Identifies gaps in literature and justifies the need for this study.

Chapter 3: Methodology

- Describes the **data** collection process using Zerodha API and preprocessing techniques.
- Details the reinforcement learning architecture, including Q-learning, DQN, and DDQN models.
- Outlines the training process, hyperparameter tuning, and model evaluation criteria.

Chapter 4: Results and Discussion

- Presents experimental results of the RL agent's performance.
- Compares RL-based trading strategies with rule-based and supervised learning models.
- Evaluates trading performance using financial metrics like Sharpe ratio, cumulative profit, and win-loss ratio.

Chapter 5: Conclusion and Future Work

- Summarizes **key findings** and contributions of the research.
- Discusses limitations of the study and proposes future research directions for improving RL-based trading models.

This structured approach ensures a clear, logical progression in understanding how reinforcement learning can enhance trading strategies.

CHAPTER II:
LITERATURE REVIEW

2.1 Overview of Reinforcement Learning in Business and Finance

The rapid advancements in artificial intelligence (AI) and machine learning (ML) have transformed financial markets, enabling the automation of trading strategies, risk assessment, and portfolio management. One of the most promising AI techniques in finance is reinforcement learning (RL), which allows trading systems to learn and adapt dynamically based on past market experiences. Unlike traditional trading models, which rely on predefined rules or statistical assumptions, RL-based models continuously refine their decision-making process by interacting with real-time market data (Ansari et al., 2024).

2.1.1 Evolution of AI and Reinforcement Learning in Financial Markets

The integration of AI in stock market trading has evolved over the years, progressing through different computational techniques:

- **Rule-Based Systems (Pre-2000s):** Early trading algorithms relied on predefined rules, such as moving average crossovers and Bollinger Bands, to determine buy/sell signals. These models lacked adaptability to new market conditions.
- **Machine Learning-Based Trading (2000s–2015):** Introduction of supervised learning models (e.g., Support Vector Machines, Random Forests) that learned patterns from historical data. However, these models required labeled datasets and could not dynamically adapt.
- **Reinforcement Learning-Based Trading (2015–Present):** RL models, particularly Deep Q-Networks (DQN) and Double Deep Q-Networks

(DDQN), introduced self-learning capabilities, enabling trading agents to adjust their strategies in response to market fluctuations (Choi & Kim, 2024).

2.1.2 Fundamental Concepts of Reinforcement Learning

Reinforcement learning is a subfield of AI that optimizes decision-making by learning through interactions with the environment. In the context of stock trading, an RL agent observes market conditions, executes trades, and adjusts its strategy based on reward signals. The core components of RL include:

- **State (S):** Represents the market conditions, such as price movements, technical indicators, and historical trends.
- **Action (A):** The decision taken by the RL agent, such as buy, hold, or sell.
- **Reward (R):** The feedback received based on the trade outcome (e.g., profit/loss after a trade).
- **Policy (π):** The strategy that the agent follows to maximize rewards over time.

Figure 5:Fundamental Concepts of Reinforcement Learning

One of the key challenges in RL-based trading is the exploration vs. exploitation dilemma. The model must explore new trading strategies to learn, while also exploiting profitable patterns to maximize gains (Cornalba et al., 2024). Below table explains key reinforcement learning concepts and their application in stock trading. It shows how states, actions, rewards, policies, exploration, and exploitation influence the decision-

making process of an RL-based trading model, helping it adapt to market conditions, optimize trade execution, and balance risk and reward effectively.

Table 6: Reinforcement learning components and their relevance to financial trading

Concept	**Description**	**Application in Trading**
State (S)	Market condition (OHLC data, indicators)	Determines the environment the RL agent perceives
Action (A)	Buy, hold, sell	Executes a trading decision
Reward (R)	Profit/loss feedback after a trade	Guides the agent to optimize profitability
Policy (π)	Decision-making strategy	Defines how the agent selects actions
Exploration	Trying new strategies to discover better ones	Helps find new profitable market patterns
Exploitation	Using known profitable strategies	Ensures stable and reliable returns

This section has introduced the evolution of AI in stock trading and the fundamentals of RL. Next, we will examine Deep Q-Networks (DQN) and Double Deep Q-Networks (DDQN) in algorithmic trading.

2.2 Deep Q-Networks (DQN) and Double Deep Q-Networks (DDQN) in Algorithmic Trading

Reinforcement learning has gained traction in financial markets, particularly in intraday trading and algorithmic decision-making. Among RL-based approaches, Deep Q-Networks (DQN) and Double Deep Q-Networks (DDQN) have emerged as effective frameworks for optimizing stock trading strategies. These models improve upon traditional Q-learning by integrating deep learning architectures that enable trading

agents to handle high-dimensional market data and complex trading patterns (Cui et al., 2023).

2.2.1 Deep Q-Networks (DQN) in Algorithmic Trading

Deep Q-Networks (DQN) have gained popularity in algorithmic trading due to their ability to learn complex trading strategies from historical data. Unlike traditional Q-learning, which struggles with large state spaces, DQN leverages deep neural networks to approximate Q-values, enabling more effective decision-making in dynamic market conditions. This section explores the application of DQN in trading, highlighting its structure, advantages, and limitations.

2.2.1.1 Overview of Q-Learning and Its Limitations

Q-learning is a foundational RL technique that helps agents learn optimal decision-making policies by updating Q-values for each state-action pair. However, in complex environments like financial markets, traditional Q-learning struggles with scalability and stability issues due to:

- **Large state-action space**: The number of possible market conditions and actions is vast, making it impractical to maintain a Q-table.
- **High variance in stock prices**: Rapid price fluctuations lead to unstable learning, causing poor convergence.
- **Delayed rewards**: Unlike environments with immediate feedback, financial trading involves uncertain long-term rewards, making it difficult for Q-learning models to learn effective policies (Du & Shen, 2024).

To overcome these challenges, Deep Q-Networks (DQN) leverage deep learning techniques to approximate Q-values, enabling trading agents to learn more efficiently from complex stock market data.

2.2.1.2 How DQN Improves Traditional Q-Learning

Deep Q-Networks (DQN) extend Q-learning by incorporating neural networks to approximate the Q-value function, allowing the agent to handle large state-action spaces more effectively. The key improvements introduced by DQN include:

- **Experience Replay**
    - Stores past experiences (state, action, reward, next state) in a memory buffer.
    - Randomly samples past experiences during training to reduce correlation between consecutive trades.
- **Target Network Stabilization**
    - Maintains a separate target network to compute the Q-value updates, preventing rapid fluctuations.
    - Reduces instability in learning, allowing for better convergence.
- **Neural Network Function Approximation**
    - Uses deep learning models (e.g., CNNs, LSTMs) to predict Q-values, eliminating the need for explicit Q-tables.

These enhancements allow DQN-based trading agents to learn complex patterns, make data-driven trading decisions, and improve market adaptability (Enkhsaikhan & Jo, 2024).

2.2.2 Double Deep Q-Networks (DDQN) for Stock Trading

Despite its success, DQN suffers from overestimation bias, where the agent overestimates the expected reward of an action, leading to suboptimal trading decisions. This overestimation often results in:

- **Aggressive trading behavior**: The agent may execute frequent, high-risk trades due to inflated reward expectations.

- **Poor decision-making in volatile markets**: Overestimated Q-values can cause unrealistic price predictions, leading to losses.

To address these issues, Double Deep Q-Networks (DDQN) introduce a more stable learning mechanism by decoupling action selection and evaluation (Papageorgiou et al., 2024).

2.2.2.1 How DDQN Addresses Overestimation Bias

The key improvement in DDQN is the separation of action selection and Q-value computation, reducing the likelihood of overestimated trading rewards. The DDQN framework consists of:

- **Two Neural Networks for Q-Value Estimation**
  - The online network selects the action.
  - The target network evaluates the Q-value of the selected action.
  - This separation prevents the model from assigning overly optimistic rewards to risky trades.
- **More Stable Learning in Volatile Markets**
  - By reducing overestimation, DDQN ensures more reliable trading decisions during market fluctuations.
  - The model is better suited for high-volatility scenarios like earnings announcements, economic events, and sudden market shifts.
- **Improved Risk Management**
  - The model discourages high-risk trades by ensuring that the agent selects actions with realistic expected returns.

2.2.3 Comparing DQN and DDQN in Financial Applications

Recent studies have evaluated the effectiveness of DQN and DDQN in stock trading, comparing their performance across different market conditions. Empirical

results suggest that DDQN offers superior decision-making stability, particularly in high-volatility markets (Feizi-Derakhshi et al., 2024).

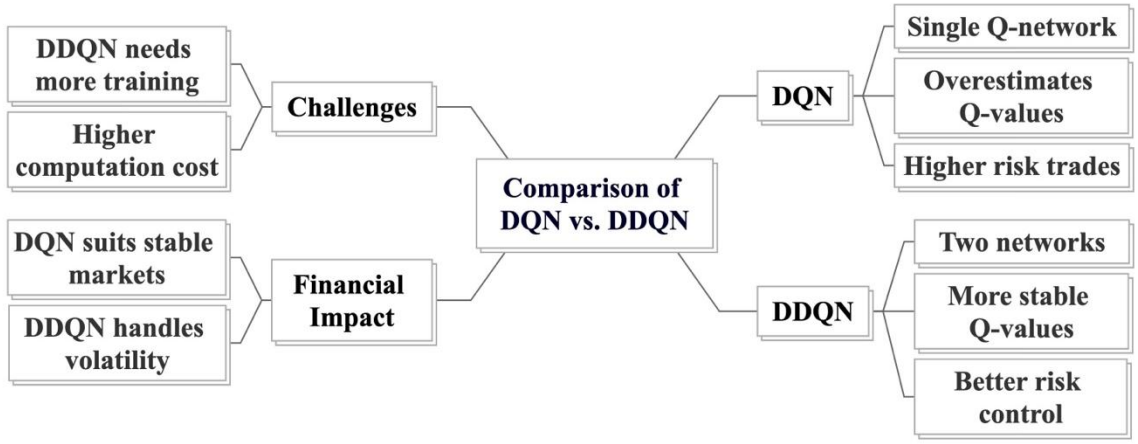

Figure 6:Comparing DQN and DDQN in Financial Applications

2.2.3.1 Key Performance Metrics for Evaluating RL-Based Trading Agents

Below table compares Deep Q-Network (DQN) and Double Deep Q-Network (DDQN) across key performance metrics. It highlights how DDQN improves over DQN by reducing overestimation bias, enhancing risk management, and ensuring more stable performance in volatile markets, whereas DQN tends to be more aggressive with frequent trades but struggles in high-risk conditions.

Table 7: Comparison of DQN vs. DDQN for financial trading applications

Metric	**Deep Q-Network (DQN)**	**Double Deep Q-Network (DDQN)**
Overestimation Bias	High (prone to aggressive trading)	Low (more stable Q-values)
Profitability	Good in stable markets	Higher in volatile markets
Risk Management	Weak (prone to high-risk trades)	Strong (discourages risky trades)

Trade Execution Frequency	High (frequent trades)	Optimized (fewer but better trades)
Performance Stability	Fluctuates in volatile markets	More stable in changing conditions

2.2.4 Research Gaps and Challenges in RL-Based Trading Systems

Although DDQN improves upon DQN, there are still challenges that need further exploration in reinforcement learning-based trading:

- **Data Efficiency and Sample Complexity** - RL models require large amounts of training data, making real-time implementation computationally expensive.
- **Handling Market Anomalies and Black Swan Events** - RL models may struggle to react effectively to rare, extreme market events, such as flash crashes.
- **Optimal Reward Function Design** - Designing risk-aware reward functions that optimize both profitability and capital protection remains a challenge.

These limitations highlight the need for further refinements in RL-based trading models, particularly in risk management and real-time adaptation (Wang et al., 2024). This section has provided an in-depth analysis of DQN and DDQN in algorithmic trading, highlighting their advantages, challenges, and empirical comparisons.

2.3 Reinforcement Learning in Different Business Domains

Reinforcement learning (RL) has expanded beyond financial trading to various business sectors, where it enhances decision-making, automation, and efficiency. While RL is widely studied in stock market prediction, its applications extend to banking, supply chain management, healthcare, and energy optimization (Santos et al., 2023). This section explores how RL is applied in different industries, drawing insights that can be adapted to algorithmic trading models.

2.3.1 RL in Stock Market and Algorithmic Trading

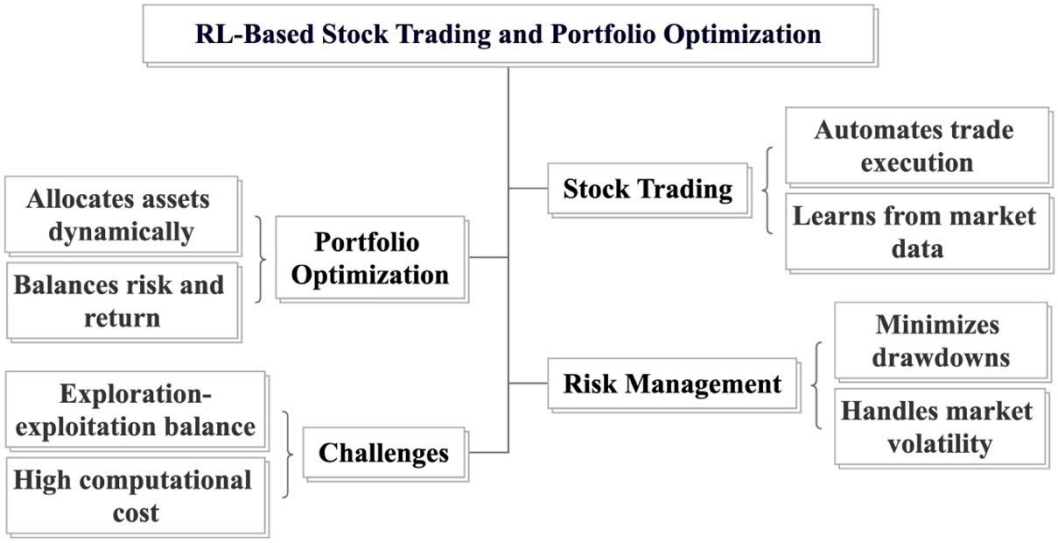

Figure 7:RL-Based Stock Trading and Portfolio Optimization Framework

Financial markets have embraced RL to optimize trading strategies, execute trades efficiently, and manage risk dynamically. The primary applications of RL in stock trading include:

- **Intraday and High-Frequency Trading (HFT)**
    - RL models are used to automate rapid trade execution while minimizing slippage and transaction costs.
    - Multi-Agent Reinforcement Learning (MARL) is applied in HFT to compete with other market participants.
- **Portfolio Management and Asset Allocation**
    - RL-based models learn optimal asset allocation strategies, adjusting portfolios based on market conditions.
    - Studies have shown RL can outperform traditional portfolio rebalancing methods (Millea & Edalat, 2023).

- **Sentiment Analysis and Market Prediction**
    - RL agents integrate news sentiment analysis and social media trends to anticipate price movements.

2.3.2 RL in Banking and Credit Risk Analysis

Banks use RL models to optimize credit approval decisions, loan pricing, and risk assessment. Key applications include:

- **Credit Scoring and Loan Approvals** - RL optimizes loan approval processes by analyzing repayment probabilities and credit risk (Guarino et al., 2024).
- **Fraud Detection and Prevention** - RL-based anomaly detection helps identify fraudulent transactions in real time.
- **Dynamic Loan Pricing** - RL adjusts interest rates based on customer profiles and macroeconomic factors.

Below table presents real-world applications of reinforcement learning (RL) across different industries, highlighting its benefits in finance and banking. RL enhances stock trading through adaptive strategies, portfolio management by optimizing asset allocation, credit scoring with improved decision-making, and fraud detection through real-time anomaly detection, making financial systems more efficient and data-driven.

Table 8: Key applications of reinforcement learning in financial services.

Application	Industry	RL Benefit
Stock Trading	Finance	Adaptive trading strategies, risk management
Portfolio Management	Finance	Asset reallocation based on market trends
Credit Scoring	Banking	Improved loan approval efficiency
Fraud Detection	Banking	Real-time identification of fraud patterns

2.3.3 RL in Supply Chain and Logistics

Supply chain management benefits from RL's ability to optimize inventory management, demand forecasting, and route planning.

- **Inventory Optimization**
    - RL adjusts stock levels dynamically based on historical demand patterns.
    - Helps reduce storage costs and avoid stockouts (Demir et al., 2023).
- **Logistics and Route Optimization**
    - RL-based route planning minimizes delivery delays and fuel costs.

2.3.4 RL in Healthcare and Drug Discovery

Reinforcement learning plays a transformative role in personalized medicine, treatment optimization, and drug discovery.

- **Personalized Treatment Recommendations**
    - RL tailors treatments based on patient history and medical conditions.
- **Drug Discovery and Clinical Trials**
    - RL models optimize clinical trial design to reduce costs and improve drug efficacy (Hirano & Izumi, 2023).

2.3.5 RL in Energy and Smart Grid Optimization

In the energy sector, RL is used to balance energy demand, improve efficiency, and enhance renewable energy integration.

- **Energy Demand Forecasting** - RL models predict electricity demand and optimize power distribution.
- **Autonomous Power Trading** - RL-based grid management systems adjust electricity supply based on consumption patterns.

2.3.6 Summary and Relevance to Algorithmic Trading

While RL is transforming various industries, its applications in banking, logistics, and energy offer valuable insights for financial trading models:

- **Risk Management in Banking** → Better Portfolio Risk Assessment in Trading
- **Logistics Optimization** → Optimized Trade Execution and Order Routing
- **Energy Demand Forecasting** → Predicting Market Trends and Volatility

These cross-industry learnings will be explored further in the methodology and implementation sections of this study.

2.4 Review of Related Work in RL for Stock Trading

The application of reinforcement learning (RL) in stock trading has been an area of extensive research, with various models developed to enhance decision-making, risk management, and profitability. This section reviews key studies on RL-based trading strategies, comparing their methodologies, limitations, and performance in financial markets.

2.4.1 Key Papers on RL-Based Trading Strategies

Several studies have explored RL-based trading agents, evaluating their effectiveness against traditional strategies. Below are some notable research contributions in the field:

Table 9:Summary of key RL-based trading research and their contributions

Study	RL Model Used	Key Contributions
Awad et al. (2023)	Deep Q-Network (DQN)	Applied DQN for stock market prediction, showing improved performance over supervised learning models.

Cui et al. (2023)	Double DQN (DDQN)	Demonstrated that DDQN reduces overestimation bias, leading to better risk-adjusted returns.
Enkhsaikhan & Jo (2024)	PPO & Actor-Critic	Evaluated policy gradient methods for trading, showing improved adaptability to market trends.
Papageorgiou et al. (2024)	Multi-Agent RL (MARL)	Applied multi-agent reinforcement learning for high-frequency trading (HFT).
Santos et al. (2023)	RL Portfolio Optimization	Optimized asset allocation strategies using RL-based portfolio rebalancing.

2.4.2 Reinforcement Learning vs. Traditional Trading Strategies

Traditional stock trading approaches rely on technical analysis, statistical models, and supervised learning techniques. However, RL-based strategies offer adaptability, continuous learning, and automated decision-making, making them more robust in dynamic market conditions (Guarino et al., 2024).

2.4.2.1 Comparison of RL-Based and Traditional Trading Models

Below table provides a comparison of different trading strategies, outlining their key features and limitations. Traditional approaches like technical analysis and supervised learning models rely on historical data but struggle with adaptability. Reinforcement learning methods, such as Q-learning and DQN, offer dynamic learning capabilities but face challenges like overestimation bias and scalability issues. DDQN improves stability by reducing overestimation bias, but it requires significant computational resources and large datasets for effective training. This comparison highlights the trade-offs between different trading strategies and the advantages of reinforcement learning in algorithmic trading.

Table 10: Comparison of RL-based trading models with traditional approaches

Trading Strategy	Key Features	Limitations
Technical Analysis	Uses indicators (e.g., RSI, MACD) for decisions	Struggles in volatile and unpredictable markets
Supervised Learning Models	Predicts price movements based on historical data	Requires labeled data, poor adaptability to new trends
Q-Learning (Basic RL)	Learns trading strategies through trial & error	Poor scalability, unstable training
DQN (Deep Q-Networks)	Uses deep learning to estimate Q-values	Suffers from overestimation bias
DDQN (Double Deep Q-Networks)	Reduces overestimation bias, improves stability	High computational cost, data-hungry training

2.4.3 Research Gaps in Existing Literature

Despite the advancements in RL-based trading, several research gaps remain:

- **Handling Market Volatility and Extreme Events**
    - Existing RL models struggle with sudden market crashes, flash crashes, and unexpected news events.
    - Need for risk-aware RL frameworks that can mitigate financial losses during black swan events.
- **Optimizing Reward Function for Financial Markets**
    - Many RL-based trading models optimize for short-term profits rather than long-term portfolio stability.
    - Improved reward function engineering is needed to balance profitability with risk management (Feizi-Derakhshi et al., 2024).
- **Computational Complexity and Real-Time Trading Feasibility**
    - RL models require large-scale historical data and extensive computational power for training.

- The challenge remains in deploying RL agents in real-time, low-latency trading environments.

- **Generalization of RL Models Across Different Market Conditions**
  - Existing models perform well in specific datasets but struggle in different market regimes (bull vs. bear markets).
  - Further research is needed to enhance the adaptability of RL-based agents across varying financial conditions (Huang et al., 2023).

2.5 Justification for Research

Given the gaps identified in existing literature, this study seeks to enhance reinforcement learning-based trading models, particularly for intraday trading on the NIFTY 50 index. The research focuses on optimizing risk-adjusted trading strategies using Double Deep Q-Networks (DDQN).

2.5.1 Why DDQN for Intraday Trading?

- Addresses Overestimation Bias: Reduces unrealistic trade expectations, making decision-making more reliable.
- More Stable Trading Performance: Ensures better handling of high-frequency market fluctuations.
- Improved Risk-Awareness: Incorporates Sharpe ratio and drawdown-based optimization to minimize financial losses.

2.5.2 Bridging the Gap in RL-Based Trading Models

- Developing an Adaptive RL Model: Integrating epsilon decay strategies and target network refinements.
- Evaluating Performance Across Market Cycles: Testing the DDQN-based RL agent in bull, bear, and volatile markets.

- Ensuring Scalability and Real-Time Feasibility: Optimizing computation efficiency for live market execution.

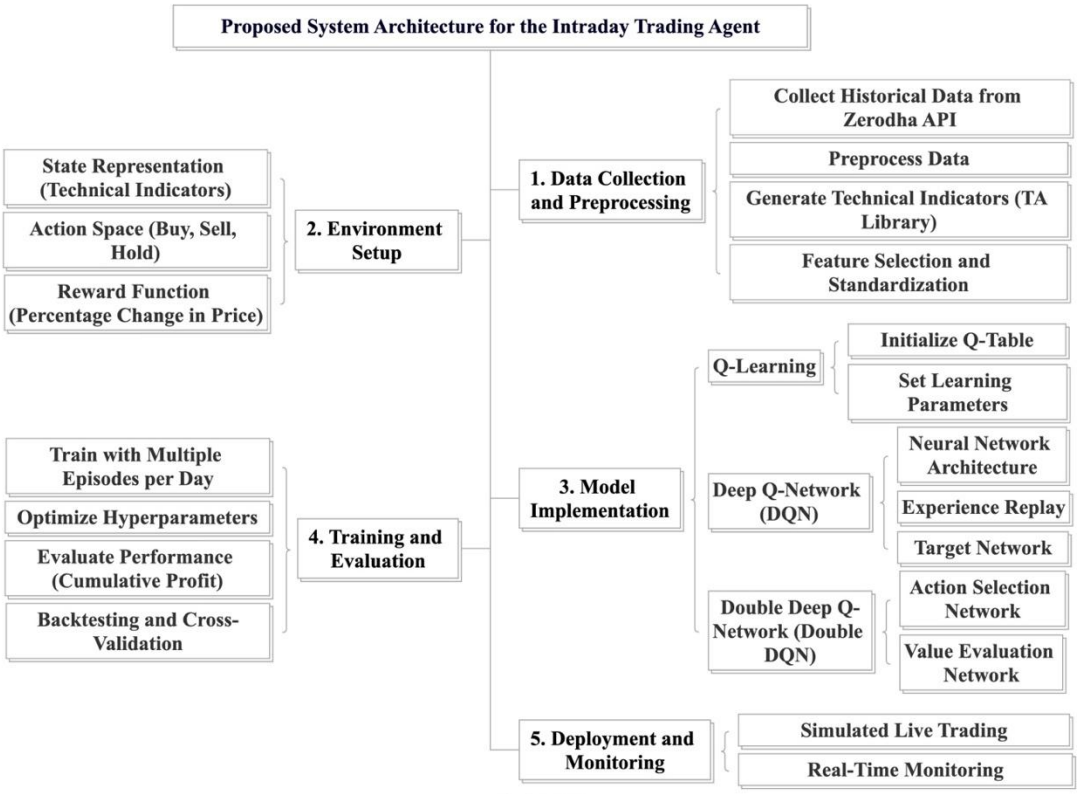

Figure 8: Conceptual Framework for DDQN-Based Trading Agent

This literature review has established the importance of RL-based trading, highlighted current research gaps, and justified why this study focuses on DDQN for intraday trading.

CHAPTER III:

METHODOLOGY

3.1 Data Collection and Preprocessing

To develop an effective reinforcement learning-based trading model, high-quality and structured financial data is essential. This study collects intraday stock market data from the Zerodha API, covering five years of NIFTY 50 stocks. The dataset consists of one-minute interval OHLCV (Open, High, Low, Close, and Volume) data, which provides fine-grained price movements for training the RL agent. Before using the data for model training, several preprocessing steps are applied to clean and structure the dataset. These include handling missing values, removing extreme fluctuations, normalizing price movements, and filtering data within valid trading hours. Additionally, the dataset is segmented into individual trading days, ensuring that each day is treated as a separate episode for reinforcement learning.

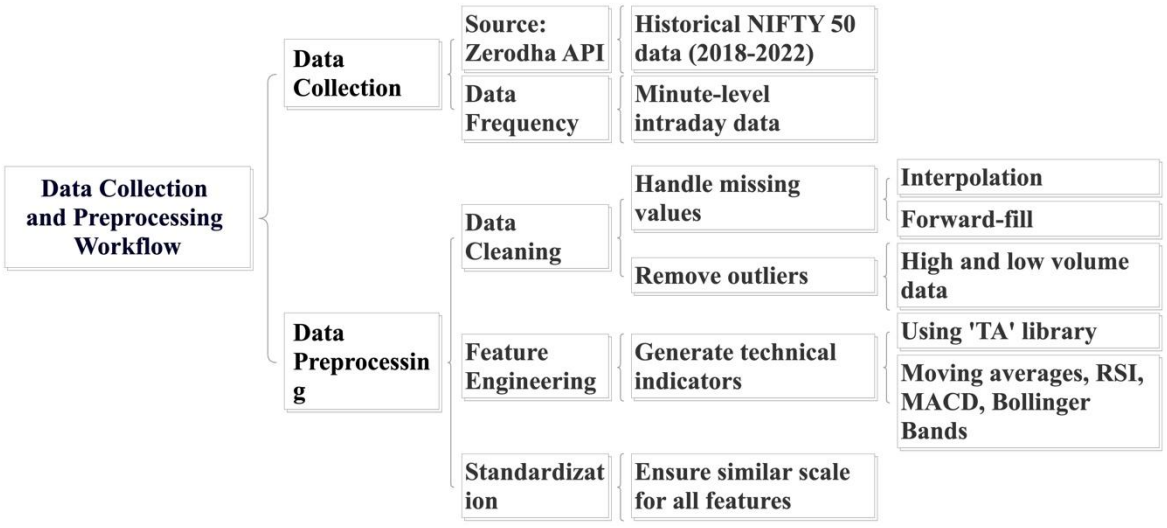

Figure 9: Data Collection and Preprocessing

3.1.1 Data Source

The dataset is obtained from Zerodha API, a widely used platform for stock market data. The focus of this study is on NIFTY 50 stocks, which are the most liquid and actively traded stocks in India.

- **Timeframe**: The dataset spans five years (four years for training, one year for testing).
- **Data Interval**: One-minute intraday price data.
- **Features Collected**: Open, High, Low, Close, Volume (OHLCV), along with technical indicators.

This dataset provides a rich historical record of stock price movements, allowing the RL agent to learn patterns, trends, and optimal trading actions.

3.1.2 Handling Missing Data

Stock market data can sometimes have missing values due to exchange downtimes or irregular trading activity. To ensure consistency, missing values are filled using past or future values, a process known as:

- **Forward Fill**: If a price is missing at time tt, it is replaced with the price from t−1t-1.
- **Backward Fill**: If no past data is available, the missing value is filled using the next available price.

This ensures that no gaps exist in the dataset, which helps the RL model process smooth price movements.

3.1.3 Removing Extreme Price Movements

Sometimes, stock prices show sudden spikes or drops due to rare events. These can mislead the RL model, causing it to learn incorrect patterns. To detect and remove

extreme values, the study applies outlier detection techniques, ensuring that only realistic price movements are included.

3.1.4 Normalizing Stock Prices and Volume

Stock prices vary significantly between companies. A stock priced at ₹1000 moves differently than a stock priced at ₹50. To make sure the RL model treats all stocks fairly, prices are scaled between 0 and 1 using a simple formula:

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}}$$

Equation (1) : Formula for data normalization

This ensures that all stocks are processed on the same scale, improving the RL model's learning process.

3.1.5 Filtering Data by Market Hours

The Indian stock market operates from 9:30 AM to 3:00 PM. Any data outside these hours is removed, so that the RL agent learns only from real-time trading sessions. This ensures that the model does not train on after-hours price fluctuations, which do not impact intraday trading.

3.1.6 Splitting Data into Training and Testing Sets

To check if the RL model can predict trades correctly, the dataset is divided into:

- **Training Data (80%)**: The first four years of data, used to teach the model.
- **Testing Data (20%)**: The final year, used to check if the model works on unseen data.

Additionally, each day is treated as a separate learning episode, meaning the model learns from one trading day at a time.

3.2 Technical Indicator Generation

Stock prices move in patterns, and traders use technical indicators to identify trends, momentum, and volatility. These indicators help the RL agent understand the market conditions and make informed trading decisions. In this study, various technical indicators are computed and added to the dataset to provide additional insights for the reinforcement learning model. Technical indicators are derived from OHLCV (Open, High, Low, Close, and Volume) data and are used to determine buy, hold, or sell signals. The indicators are classified into three main categories:

- **Trend Indicators** – Identify market direction.
- **Momentum Indicators** – Measure the speed and strength of price movements.
- **Volatility Indicators** – Analyze price fluctuations over time.
- **Volume-Based Indicators** – Evaluate trading activity and liquidity levels.

3.2.1 Trend Indicators

Trend indicators help determine whether a stock is in an uptrend, downtrend, or moving sideways. The most commonly used trend indicators in this study are Moving Averages and the MACD (Moving Average Convergence Divergence).

3.2.1.1 Moving Averages

Moving Averages (MA) smooth out price fluctuations by calculating the average price over a specific period. There are two main types:

Simple Moving Average (SMA): The average closing price over NN periods:

$$\mathrm{SMA}_N = \frac{1}{N}\sum_{i=1}^{N} P_i$$

Equation (2) : Simple Moving Average

where P_i is the closing price at time ii.

Exponential Moving Average (EMA): Places more weight on recent prices, making it more responsive to price changes:

$$\mathrm{EMA_t} = \mathrm{P_t} \times \alpha + \mathrm{EMA_{t-1}} \times (1 - \alpha)$$

Equation (3) : Exponential Moving Average

where $\alpha = \frac{2}{N+1}$ is the smoothing factor.

These indicators help determine the market trend by identifying whether prices are consistently rising or falling.

3.2.1.2 Moving Average Convergence Divergence (MACD)

MACD helps identify trend strength by comparing two moving averages. It is calculated as:

$$\mathrm{MACD} = \mathrm{EMA_{short}} - \mathrm{EMA_{long}}$$

Equation (4) : Moving Average Convergence Divergence

where EMA short (usually 12 periods) reacts faster to price changes than EMA long(usually 26 periods). A positive MACD indicates an uptrend, while a negative MACD signals a downtrend.

3.2.2 Momentum Indicators

Momentum indicators measure how fast prices are moving. They help identify when a stock is overbought (rising too fast) or oversold (falling too fast).

3.2.2.1 Relative Strength Index (RSI)

RSI evaluates whether a stock is overbought or oversold using the formula:

$$\text{RSI} = 100 - \left(\frac{100}{1 + \text{RS}}\right)$$

Equation (5) : Relative Strength Index

where

$$\text{RS} = \frac{\text{Average Gain over N periods}}{\text{Average Loss over N periods}}$$

- RSI > 70 suggests that the stock may be overbought (a price drop is likely).
- RSI < 30 suggests that the stock may be oversold (a price rise is likely).

3.2.2.2 Stochastic RSI (StochRSI)

A more sensitive variation of RSI, Stochastic RSI identifies short-term momentum shifts:

$$\text{Stochastic RSI} = \frac{\text{RSI} - \text{Min RSI}_\text{N}}{\text{Max RSI}_\text{N} - \text{Min RSI}_\text{N}}$$

Equation (6) : Stochastic RSI

where Max RSI and Min RSI are the highest and lowest RSI values over NNN periods. This indicator helps detect faster trend reversals compared to standard RSI.

3.2.2.3 True Strength Index (TSI)

TSI smooths out momentum signals, measuring longer-term trend strength:

$$\text{TSI} = 100 \times \frac{\text{EMA}_\text{N}\left(\text{EMA}_\text{M}(\Delta\text{P})\right)}{\text{EMA}_\text{N}\left(\text{EMA}_\text{M}(|\Delta\text{P}|)\right)}$$

Equation (7) : True Strength Index

where ΔP is the price change. TSI > 0 suggests bullish momentum, while TSI < 0 signals bearish momentum.

3.2.2.4 Rate of Change (ROC)

ROC calculates the percentage change in price over NNN periods:

$$\mathrm{ROC} = \frac{\mathrm{P_t} - \mathrm{P_{t-N}}}{\mathrm{P_{t-N}}} \times 100$$

Equation (8) : Rate of Change

A positive ROC suggests upward momentum, while a negative ROC signals a price decline.

3.2.2.5 Ultimate Oscillator (UO)

The Ultimate Oscillator prevents false momentum signals by combining short, medium, and long-term price changes:

$$\mathrm{UO} = 100 \times \frac{4 \times BP_7 + 2 \times BP_{14} + BP_{28}}{4 \times TR_7 + 2 \times TR_{14} + TR_{28}}$$

Equation (9) : Ultimate Oscillator

where BP is buying pressure and TR is the true range. This indicator helps reduce the lagging effect of momentum oscillators.

3.2.3 Volatility and Volume Indicators

Volatility and volume indicators measure price fluctuations and market activity. These indicators help determine whether a stock is experiencing high or low trading activity.

3.2.3.1 Bollinger Bands

Bollinger Bands measure price volatility using a moving average and standard deviations:

$$\text{Upper Band} = \text{SMA}_N + k \times \sigma$$

$$\text{Lower Band} = \text{SMA}_N - k \times \sigma$$

Equation (10) : Bollinger bands

where σ is the standard deviation, and kk is a constant (usually 2).

- When prices touch the upper band, the stock may be overbought.
- When prices touch the lower band, the stock may be oversold.

3.2.3.2 Average True Range (ATR)

ATR measures market volatility by calculating the average price range over NN periods:

$$\text{ATR}_N = \frac{1}{N}\sum_{i=1}^{N}\left(\text{High}_i - \text{Low}_i\right)$$

Equation (11) : Average True Range

Higher ATR values indicate higher market volatility, while lower ATR values indicate a stable market.

3.2.3.3 Chaikin Money Flow (CMF)

CMF tracks money flow strength, helping determine institutional buying or selling activity:

$$\text{CMF} = \frac{\sum_{i=1}^{N}(\text{MFI}_i \times V_i)}{\sum_{i=1}^{N} V_i}$$

Equation (12) : Chaikin Money Flow

where MFI (Money Flow Index) evaluates how much volume flows into or out of a stock.

3.2.4 Computation and Integration into the Dataset

All technical indicators are computed using the TA (Technical Analysis) library in Python. These indicators are added as new features in the dataset, allowing the RL agent to analyze them while making trading decisions. These indicators give the reinforcement learning model a better understanding of market conditions, helping it recognize patterns and potential trading opportunities. All 80 technical indicators are computed using the Python ta library, which simplifies the extraction of trend, momentum, and volatility indicators. These indicators are added as new columns in the dataset, making them available for the reinforcement learning model.

3.3 Q-Value Simulation for RL Training

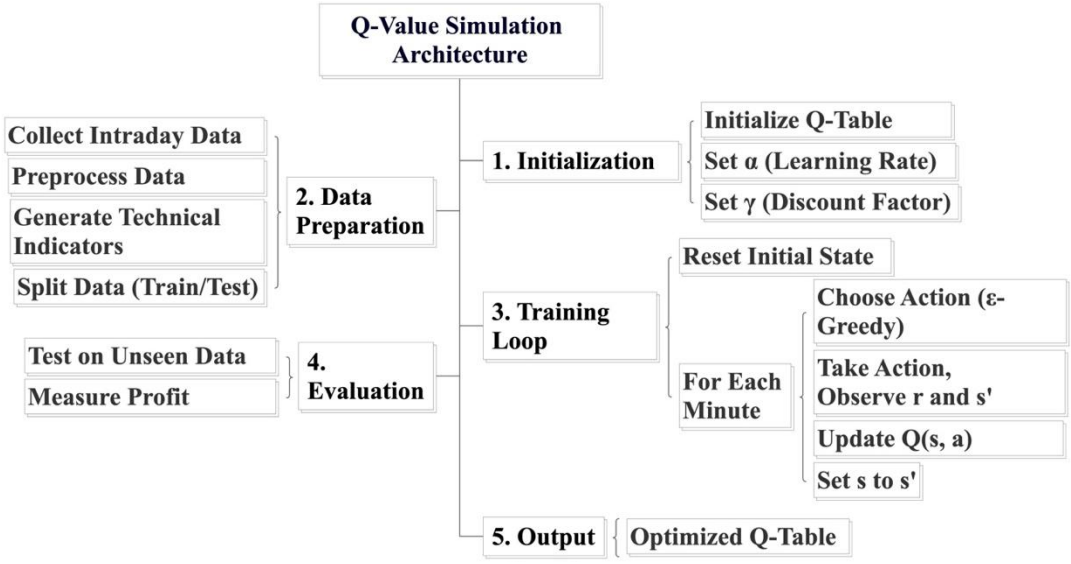

Figure 10: Q-Value Simulation Architecture

Intraday data rarely offers a clean answer to the question “what should have been done here?” Prices move, reverse, and pause in ways that make any single, rule-based label feel brittle the moment market conditions shift. In this thesis, we treat that ambiguity head-on by letting the learning objective itself define the label. Each state is passed through the Q-value simulation—the same dynamics that will ultimately judge performance—and the action with the highest expected return becomes the label for that moment. In other words, instead of arguing after the fact about whether a threshold was too loose or too strict, we ask the simulator, under our reward definition, “Which action is most valuable here?” and record that answer. The experiment does not change; we simply make explicit what was already implicit: labels are reward-consistent and born from the same objective that will later evaluate the policy.

This framing has a quiet but important consequence for ground truth. Markets do not hand out gold-standard tags, but the simulation does produce a consistent notion of “truth” relative to the trading objective. By taking the arg-max over QBuy, QHold, QSell we align supervision with the very payoff we care about. The label is not a heuristic proxy—it is the action that the objective itself prefers in that state. When the market regime changes, the simulator’s returns change with it, and so do the labels; the supervision therefore remains coupled to economic reality rather than to a static rule that ages poorly.

Because each trading day is treated as an episode, we gain something else that simple historical passes cannot offer parallel knowledge discovery. Replaying the same day many times with varied seeds, replay order, and exploration noise exposes multiple plausible trajectories through identical market backdrops. We are not changing any mechanics; we are simply exploiting the episodic design to surface alternative “what-ifs” around the same states. Over time this produces a denser, more informative set of state–

action–return impressions—small differences in timing, small differences in path—which help the learner understand not just what worked once, but what tends to work across many nearby possibilities.

The architecture of training—ε-greedy exploration with experience replay—reinforces this effect. High-episodic training ensures that informative moments do not vanish after a single sweep; they are remembered, resampled, and contrasted against less informative ones. This recycling of experience is not embellishment; it is fidelity. It acknowledges that intraday dynamics are path-dependent and that learning benefits from seeing the same context under slightly different perturbations. The result is not only more coverage of the state space, but more stability: the supervision signal is less hostage to any one run or one burst of volatility.

Another virtue of simulation-supervised labeling is full reward-based control. The training target and the trading goal are literally the same function. Many pipelines drift because they are optimized to hit an intermediate surrogate (a thresholder return, a margin around a moving average) and only later judged by actual profit-and-loss. Here, the surrogate is the objective. The Q-value engine that scores actions is the same mechanism that proposes labels. That unity removes a common source of mismatch, especially around regime edges where hand-crafted thresholds behave erratically. It also makes performance discussions cleaner: if the learned classifier or the later DQN/DDQN deviates from labels, it is deviating from the payoff rule—not from an external proxy.

Crucially, the method keeps a human touch. While Q-values automate label selection, they do so with transparent components. At any timestep a reviewer can see which action won, by how much, and whether the margin was decisive or marginal. Edge cases—near-ties during fast moves, sudden liquidity gaps—become auditable rather than mysterious. This auditability is practical: it supports spot-checks, post-mortems, and the

kind of qualitative sense-making that market practitioners expect when a model behaves surprisingly. Nothing in the experiment is altered to achieve this; we are only making explicit the rationale already present in the Q-value comparisons.

Finally, this supervision confers an operational advantage: faster network modeling. Once labels exist, the baseline network can be trained as a supervised classifier, which is computationally lighter than on-policy RL and quicker to iterate. That speed does not dilute rigor; it enables it. We can perform more careful validations, more granular ablations, and more conservative early stops before handing the baton to DQN/DDQN. In effect, simulation-supervised labels serve as a map: they do not replace the journey of interaction-driven learning, but they chart a sensible route so that the policy search starts near promising neighborhoods rather than wandering blindly.

Taken together, these choices—reward-consistent labels, episodic replays, stable reuse of experience, and transparent Q-based rationale—do not modify the experimental pipeline described elsewhere in this thesis. They explain it. The novelty is not a new switch or a hidden parameter; it is the decision to let the objective function write the labels, to let episodes multiply our understanding of each day, and to keep the supervision signal human auditable. In a domain where "ground truth" is contested and regimes shift without warning, that combination is the difference between a rule that happens to work and a learning target that continues to make economic sense.

To train the reinforcement learning (RL) agent for stock trading, it is essential to simulate market interactions and generate Q-values for various trading actions. The goal of Q-value simulation is to allow the RL model to evaluate trading decisions, refine its predictions over multiple learning episodes, and ultimately optimize its strategy for maximizing profits while minimizing risks. Q-values represent the expected future rewards of taking a specific action in a given market state. These values are updated

iteratively through multiple training episodes, where the agent continuously learns from past experiences and adjusts its decision-making accordingly.

The Q-value simulation process consists of the following key steps:

- Segmenting historical market data into individual trading days.
- Generating Q-values for Buy, Hold, and Sell actions.
- Running multiple training episodes per day to enhance learning.
- Applying an exploration-exploitation strategy to balance learning and decision-making.
- Using experience replay to stabilize learning and improve model convergence.

3.3.1 Generating Simulated Trading Data Using Q-Values

The Q-value function enables the RL model to evaluate different actions and refine its decision-making over time. The Q-value update rule is formulated as:

$$\mathbf{Q(s,a) = Q(s,a) + \alpha[r + \gamma ma\,x\,Q\,(s', a') - Q(s,a)]}$$

Equation (13) : Reinforcement Learning Q-Value Formula

where:

- $\boldsymbol{Q(s,a)}$ is the **current Q-value** for taking action aa at state ss.
- $\boldsymbol{\alpha}$ is the **learning rate**, which controls how quickly new experiences influence the model.
- $\boldsymbol{r}$ is the **reward** obtained for executing the action.
- $\boldsymbol{\gamma}$ is the **discount factor**, determining how much future rewards impact the current decision.
- $\mathbf{max}\,\boldsymbol{Q}\,(\boldsymbol{s'}, \boldsymbol{a'})$ is the highest estimated Q-value for the **next state** s'.

This update ensures that the RL model learns to associate specific market conditions with profitable trading actions.

Pseudocode for Q-Value Computation

```
FOR each trading day:
   FOR each time step in market data:
      OBSERVE the current market state s
      CHOOSE an action a using exploration-exploitation
          EXECUTE the selected action
      OBSERVE the next market state s' and reward r
      UPDATE Q-values using the Bellman equation:
      Q(s, a) += α * [r + γ * max(Q(s', a')) - Q(s, a)]
      STORE predicted Q-values in dataset
   END FOR
END FOR
```

This iterative process ensures that the RL agent refines its Q-values over multiple learning episodes, improving its decision-making capabilities.

3.3.2 Breaking Market Data into Daily Segments

Since intraday trading resets daily, the RL model must learn from each trading day independently. Market data is segmented into separate trading days, treating each day as an individual episode. This ensures that the agent learns patterns specific to daily stock movements without carrying over unnecessary information from previous days.

Pseudocode for Daily Market Data Segmentation

```
FOR each trading day:
    EXTRACT all market data within 9:30 AM - 3:00 PM
    STORE as an independent learning episode
END FOR
```

Each trading day is broken down into minute-by-minute intervals, ensuring that the RL agent can process real-time stock price movements and make sequential trading decisions accordingly.

3.3.3 Running Multiple Episodes Per Day

To maximize learning, the RL agent replays each trading day 100 times with different initial conditions. This process, known as episodic reinforcement learning, allows the agent to explore various strategies before committing to a fixed decision-making pattern.

By running multiple episodes per day, the model learns:

- How different trading strategies perform in identical market conditions.
- The impact of different reward structures on decision-making.
- How early mistakes affect overall profitability, leading to optimized trading behaviour.

Pseudocode for Running Multiple Episodes Per Day

```
FOR each trading day:
    FOR episode in range(100):
        RESET environment to the start of the trading day
        FOR each time step:
            SELECT an action based on Q-values
```

EXECUTE the action

RECEIVE reward and update Q-values

END FOR

END FOR

END FOR

3.3.4 Exploration vs. Exploitation Trade-off

During training, the RL model must decide whether to explore new strategies or exploit learned strategies. This is handled using an epsilon-greedy approach, where the probability of selecting a random action decreases over time:

$$\epsilon_t = \epsilon_{min} + (\epsilon_{max} - \epsilon_{min})e^{-\lambda t}$$

Equation (14) : Exploration vs Exploitation Trade-off

where:

- ϵ_t is the exploration rate at time step t.
- ϵ_{max} is the initial exploration rate (higher chance of taking random trades).
- ϵ_{min} is the minimum exploration rate (agent selects best-known actions).
- λ is the decay rate, controlling how fast the model shifts from exploration to exploitation.

Pseudocode for Exploration-Exploitation Strategy

FOR each time step:

GENERATE a random number z

IF z > epsilon:

SELECT the action with the highest Q-value (Exploitation)

ELSE:

SELECT a random action (Exploration)

END IF

UPDATE epsilon based on decay rate

END FOR

At the start of training, the model explores more to discover new strategies. As training progresses, it gradually shifts towards exploiting profitable strategies, improving its trading accuracy.

3.3.5 Experience Replay for Stable Learning

To improve learning stability, the RL model uses an experience replay buffer, where past transactions are stored and randomly sampled to train the model. This prevents the model from overfitting to short-term patterns and ensures that it learns generalized trading strategies.

The replay buffer stores past experiences as:

$$\mathrm{E} = \{(\mathrm{s}, \mathrm{a}, \mathrm{r}, \mathrm{s}')_1, (\mathrm{s}, \mathrm{a}, \mathrm{r}, \mathrm{s}')_2, \dots, (\mathrm{s}, \mathrm{a}, \mathrm{r}, \mathrm{s}')_\mathrm{N}\}$$

Equation (15) : Experience Replay for Stable Learning

where each tuple represents:

- s = current market state.
- a = action taken (Buy, Hold, Sell).
- r = reward received.
- s' = next market state.

Pseudocode for Experience Replay

```
FOR each training iteration:
    SAMPLE a batch of past experiences from replay buffer
    UPDATE Q-values based on sampled experiences
    APPLY updates to improve trading strategy
END FOR
```

By randomly sampling experiences, the RL agent avoids overfitting to short-term trends, leading to better long-term generalization.

The Q-value simulation process plays a vital role in reinforcement learning by enabling the RL agent to interact with a simulated market environment before making real trading decisions. By simulating different trading actions and observing their outcomes, the agent gradually learns which strategies lead to profitable trades and which ones result in losses. This learning process is essential because financial markets are highly dynamic, and a trading strategy that works in one scenario may not perform well in another. Through Q-value updates over multiple episodes, the RL model can analyze how buying, holding, or selling at specific moments affects cumulative returns, helping it develop a better decision-making framework over time.

One of the primary benefits of Q-value simulation is that it allows the RL agent to train on historical stock data without financial risk. Unlike traditional backtesting methods, where predefined rules are tested on past data, reinforcement learning dynamically adjusts trading strategies based on past experiences. The RL agent observes past market trends, evaluates possible outcomes for different trading actions, and optimizes its future decisions accordingly. By running multiple training episodes per

trading day, the model can explore various possible market scenarios, allowing it to refine its strategy to maximize long-term profitability.

Another key aspect of Q-value simulation is the balance between exploration and exploitation, which is managed through epsilon decay strategies. At the beginning of training, the model explores a wide range of actions, even if they are suboptimal, to discover new trading patterns. As training progresses, the agent gradually shifts toward exploiting the best-known strategies, reducing random actions and making data-driven trading decisions. This controlled transition ensures that the RL model does not get stuck in local optima, allowing it to continuously improve its performance.

To ensure stability during learning, the Q-value simulation process incorporates experience replay, where past transactions are stored and randomly sampled for training. This technique prevents the RL model from relying too heavily on recent experiences, which could lead to overfitting specific market conditions. Instead, the agent learns from a diverse set of past experiences, improving its ability to adapt to new market trends and unexpected fluctuations. By randomizing the learning process, experience replay enhances the robustness of the RL trading strategy, making it more effective in real-world stock trading.

In summary, Q-value simulation is a structured learning process that allows the RL agent to interact with the market, optimize trading decisions, and refine its strategy over time. Through multiple training episodes, exploration-exploitation balance, and experience replay, the model develops a robust, data-driven trading approach before being deployed in live financial markets. This ensures that the RL-based trading system is not only profitable but also stable, risk-aware, and adaptable to evolving stock market conditions.

3.4 Reinforcement Learning Model Development

Developing an effective reinforcement learning (RL) model for stock trading requires a structured approach where the agent understands market conditions, makes trading decisions, and refines its strategies over time. The core idea behind reinforcement

learning is that the agent interacts with an environment (the stock market), takes actions, and learns from the rewards or penalties it receives. In this study, the RL model is formulated as a Markov Decision Process (MDP), where trading is modeled as a sequential decision-making problem. The agent must determine the best possible action—Buy, Hold, or Sell—at each time step while considering historical price patterns, technical indicators, and market trends. The learning process is driven by Q-value updates, which allow the model to evaluate different trading strategies and optimize its decision-making over multiple learning episodes.

To ensure optimal trade execution, three different reinforcement learning models are implemented:

- **Q-Learning (Baseline Model)** – A fundamental RL model that serves as the foundation for learning trading actions.
- **Deep Q-Networks (DQN)** – An advanced model that replaces the Q-table with a neural network, enabling learning from large-scale financial data.
- **Double Deep Q-Networks (DDQN)** – An improved version of DQN that eliminates overestimation bias, leading to more stable trading decisions.

Each of these models has unique properties that influence how the RL agent learns and makes trading decisions. The following sections provide a detailed breakdown of each model, explaining their mathematical foundations, learning mechanisms, and implementation in stock trading.

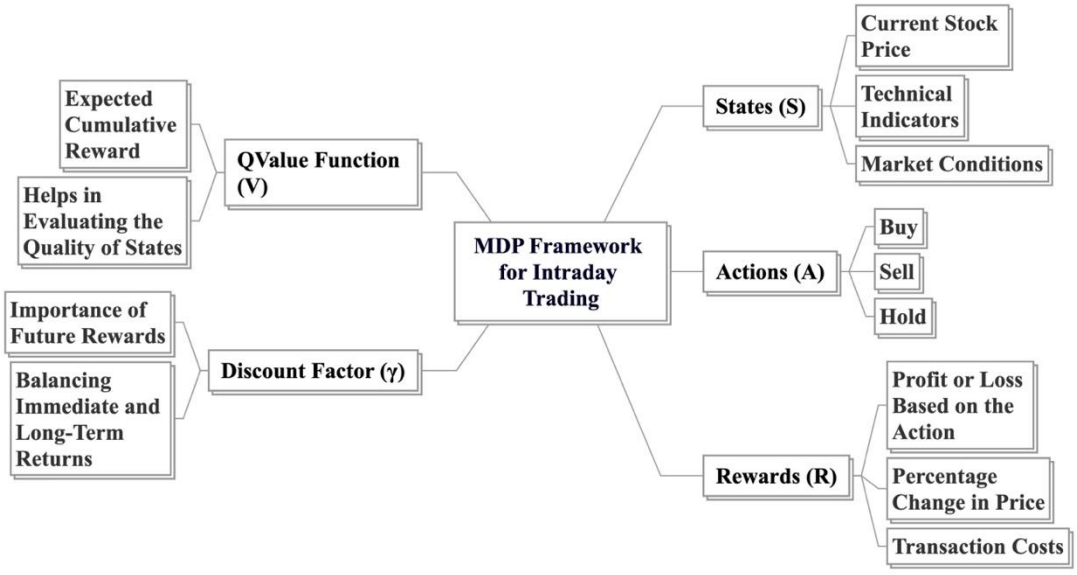

Figure 11: MDP Framework for Intraday Trading

3.4.1 Q-Learning Baseline Model (Deep Learning-Based Training on Simulated Data)

The Q-learning baseline model used in this study is a supervised deep learning model trained on simulated Q-values rather than a traditional Q-learning approach that uses a Q-table. Instead of updating Q-values iteratively during live market interactions, the model learns from precomputed Q-values generated in the Q-value simulation phase (Section 3.3). The goal is to classify Buy, Hold, or Sell decisions based on past market conditions, making this a classification problem rather than a reinforcement learning problem.

3.4.1.1 Converting Q-Values to Action Labels

During the Q-value simulation phase, three Q-values are computed for every time step:

- **Q(Buy)** – The expected reward if the agent chooses to buy.

- **Q(Hold)** – The expected reward if the agent chooses to hold.
- **Q(Sell)** – The expected reward if the agent chooses to sell.

For each data point, the highest Q-value determines the final action label:

$$A_t = \arg\max(Q_{buy}, Q_{hold}, Q_{sell})$$

Equation (16) : Converting Q-Values to Action Labels

where A_t represents the selected action at time step t. If:

- Q(Buy) is the highest, the label is **Buy**.
- Q(Hold) is the highest, the label is **Hold**.
- Q(Sell) is the highest, the label is **Sell**.

This conversion transforms the dataset into a classification problem, where the model is trained to predict which action has the highest Q-value based on historical market indicators.

Pseudocode for Q-Value to Label Conversion

FOR each row in dataset:

  COMPUTE Q-values for Buy, Hold, and Sell

  SELECT the action with the highest Q-value

  ASSIGN label as Buy, Hold, or Sell

STORE processed dataset for training

This step ensures that the deep learning model learns from precomputed Q-values, allowing it to classify trading actions efficiently.

3.4.1.2 Imbalance Issue in Buy, Hold, and Sell Labels

Stock market data naturally contains far more Hold actions than Buy or Sell actions. Since markets often remain stable, traders hold their positions most of the time, leading to an imbalance in the dataset where:

- **Hold (~70-80%)** is the most frequent label.
- **Buy (~10-15%)** occurs less frequently.
- **Sell (~10-15%)** also occurs less frequently.

This imbalance can lead the model to Favor Hold predictions, reducing its ability to recognize Buy and Sell opportunities. To balance the dataset, the Synthetic Minority Over-sampling Technique (SMOTE) is applied.

3.4.1.3 Importance of SMOTE for Handling Class Imbalance

SMOTE (Synthetic Minority Over-sampling Technique) is used to generate synthetic samples for underrepresented classes (Buy and Sell), ensuring that all labels have a more balanced distribution. Instead of simply duplicating existing Buy and Sell samples, SMOTE creates synthetic new samples by interpolating between real observations.

The SMOTE algorithm works as follows:

- Identify Minority Classes (Buy and Sell).
- Find K-nearest neighbours for each minority sample.
- Synthetically generate new data points by interpolating between existing neighbours.

Mathematically, the synthetic sample is generated using:

$$X_{new} = X_{minority} + \lambda(X_{neighbor} - X_{minority})$$

Equation (17) : SMOTE Algorithm

where:

- $\boldsymbol{X}_{\text{minority}}$ is an existing Buy or Sell sample.
- $\boldsymbol{X}_{\text{neighbor}}$ is one of its nearest neighbors.
- $\boldsymbol{\lambda}$ is a random number between 0 and 1.

This ensures that new synthetic data points remain realistic and follow market patterns while addressing class imbalance.

Pseudocode for Applying SMOTE

IDENTIFY Buy and Sell classes as minority classes

FIND K-nearest neighbors for each minority sample

GENERATE new synthetic Buy and Sell data points

ADD synthetic samples to dataset

ENSURE balanced dataset before training

By applying SMOTE, the model is trained on a more balanced dataset, improving its ability to predict Buy and Sell decisions accurately.

3.4.1.4 Deep Learning Model Architecture

The Q-learning baseline model is implemented as a deep neural network (DNN) that learns to classify Buy, Hold, or Sell based on stock technical features. The architecture consists of:

- **Input Layer** – Receives stock market features (technical indicators).
- **Hidden Layers** – Uses ReLU activation and dropout regularization to prevent overfitting.
- **Output Layer** – Uses Softmax activation to classify actions.

Mathematical Representation of Neural Network Layers. Each hidden layer applies a transformation to the input data:

$$h_i = f(W_i X + b_i)$$

Equation (18) : Representation of Neural Networks

where:

- $\boldsymbol{h_i}$ is the output of the hidden layer.
- $\boldsymbol{W_i}$ is the weight matrix.
- $\boldsymbol{X}$ is the input feature vector.
- $\boldsymbol{b_i}$ is the bias term.
- $\boldsymbol{f(x)}$ is the ReLU activation function:

$$f(x) = \max(0, x)$$

For the final classification layer, the Softmax function converts raw scores into probabilities:

$$P(a) = \frac{e^{z_a}}{\sum_j e^{z_j}}$$

Equation (19) : Softmax Equation for classification

where:

- $P(a)$ is the probability of choosing action aa (Buy, Hold, or Sell).
- $z_a$is the raw network output for action a.

Pseudocode for Model Training

INITIALIZE deep neural network with multiple dense layers

APPLY dropout regularization to prevent overfitting

COMPILE model with Adam optimizer and Categorical Cross-Entropy loss

TRAIN model for 128 epochs with batch size of 64

SAVE trained model for later predictions

The model is trained using the Categorical Cross-Entropy loss function, which measures how well the predicted probabilities match the true action labels:

$$L = -\sum_{i=1}^{N} y_i \log(\hat{y}_i)$$

Equation (20) : Categorical Cross-Entropy loss function

where:

- y_i is the actual class label.
- $\hat{y}_i$ is the predicted probability for the correct class.

By minimizing this loss, the model improves its classification accuracy, learning to correctly identify profitable trading actions.

3.4.1.5 Asset-Agnostic Feature Design: Technical Indicators Only

This thesis makes a deliberate choice: the only inputs to every network—the supervised baseline, DQN, and DDQN—are technical indicators computed from OHLCV bars (open, high, low, close, volume). There is no use of fundamentals, earnings, analyst reports, options Greeks, order-book depth, news, social feeds, or any asset-specific metadata.

- Generalization across markets. Technical indicators form a common language for price and volume behaviour. The same patterns—trend, momentum, volatility changes, range compression and expansion, participation—exist in equities, futures, FX, and crypto.
- Clarity and portability. By keeping inputs universal, the full pipeline (feature build, simulation-supervised labels, supervised pre-training, RL refinement) can be applied to any instrument that has OHLCV data, without altering experiments or code structure.
- This design does not "dumb down" the problem—it widens the boundary of the study to any marketplace where OHLCV is available.

3.4.1.6 The indicator set and how it's used

Indicators are chosen for role and clarity, not brand names. Each group captures a different aspect of the tape:

- **Returns and momentum**. Simple and cumulative returns over short windows, rate-of-change measures.
- **Averages and trend**. Short and long moving averages (simple or exponential), distance between them, and the slope of a longer average to indicate broad trend strength.
- **Volatility and range**. Rolling standard deviation of returns, true range and average true range (ATR), normalized high-low and close-open ranges to spot compression and expansion.
- **Oscillators**. RSI, stochastic signals, and distance to dynamic channels (for example, how far price sits from a moving-average band), always expressed in normalized units.
- **Participation and volume**. Volume z-scores, deviation from VWAP, and simple volume-price concordance signals.
- **Structure and timing**. Flags for higher-high/lower-low sequences, small bar-shape summaries, and time-of-day encodings so the model can recognise intraday regularities without peeking ahead.

Every transform is calculated with causal windows only (past bars up to the decision point) and recorded with its exact window length and normalization. The goal is a compact, stable state vector that describes market shape, not a kitchen-sink of formulas.

3.4.1.7 Scale-free by design

To make indicators comparable across assets with different price levels and volatility, all features are expressed in dimensionless or relative terms:

- Work with returns or differences instead of raw prices.
- Standardize rolling values into z-scores using past-only means and standard deviations.
- Express distances (for example, from price to a moving average or VWAP) in standard-deviation or ATR units, not in points or ticks.

Because of this, a large-cap stock at 3,500 and a currency pair at 1.2450 can look like the learner when their patterns are similar.

3.4.1.7 Venue awareness without hardwiring venue rules

The feature build respects market sessions but never bakes in asset-specific tricks:

- Sessional markets (equities). Time encodings are tied to the local session clock; features reset or mark the open explicitly.
- 24×7 markets (crypto). The same encodings fall back to a 24-hour cycle; there is no notion of "open," and nothing in the features assumes one.
- Gaps and halts. Gaps are treated as missing time; there is no forward-fill or any transformation that would move future information into the past.

Because every feature depends only on a fixed number of past bars, switching to a new asset class simply means pointing the same code at a new OHLCV stream.

3.4.1.8 Why this generalizes to any marketplace (including crypto)

Technical indicators summaries shape rather than identity. A momentum burst after a tight range, or a mean-reversion snap after an extended run, is the same idea whether you look at a bank stock, crude oil, EURUSD, or BTC-perpetuals. With returns, z-scores, and ATR-scaled distances, the model learns those ideas without being confused by different price scales or tick sizes.
Two practical benefits follow:

- **Label logic travels**. The simulation-supervised labeller ranks actions by expected, post-cost return. When the features speak a common, normalized language, that ranking remains understandable across assets.
- **Policy reuse is realistic**. Even if you retrain per asset (you should), you don't need new architectures or asset-specific feature engineering. Hyperparameters and training flow transfer cleanly.

3.4.1.9 Preventing leakage and keeping causality

All rolling calculations use past-only data. There is no look-ahead to future bars, no use of end-of-day information inside the day, and no adjustments that would sneak tomorrow's knowledge into today's features. Resampling (for example, ticks to 1-minute

bars) is time-forward and deterministic. The guiding rule is simple: the model may only see what a trader could have known at that moment.

3.4.1.10 What stays the same and what changes when you switch assets

Because inputs are just technical indicators:

- **Unchanged**: feature code, model architectures, training loops, replay and label logic, evaluation procedures.
- **Restated outside the model**: execution assumptions in the simulator (fees, spread, slippage, latency), calendar or session logic, and capacity constraints. These live in the reward specification, not in the features—exactly why the feature layer is portable.

3.4.1.10 Summary

The Q-learning baseline model is a deep learning-based classification system trained on simulated Q-values, making it capable of predicting Buy, Hold, or Sell actions based on historical stock market data. Unlike traditional Q-learning, which dynamically updates a Q-table through exploration, this model follows a supervised learning approach, where it learns from precomputed Q-values generated during the Q-value simulation phase. This transformation allows the reinforcement learning problem to be framed as a classification task, where the model is trained to identify optimal trading decisions using historical market patterns and technical indicators. The training dataset consists of input features such as OHLC prices, volume, and technical indicators, while the labels are determined by selecting the action with the highest Q-value at each time step.

To address the imbalance in action distribution, where the Hold action naturally occurs more frequently than Buy or Sell, the Synthetic Minority Over-sampling Technique (SMOTE) is applied. Financial market data often exhibits long periods of low volatility, leading to an overwhelming number of Hold labels compared to Buy and Sell.

If left unaddressed, this imbalance would cause the model to favor Hold actions, reducing its ability to correctly identify profitable entry and exit points. By using SMOTE, synthetic samples for underrepresented classes (Buy and Sell) are generated by interpolating between real observations, ensuring a more balanced dataset. This adjustment enhances the model's ability to distinguish meaningful trading opportunities rather than simply predicting Hold as the default action.

Once trained, the deep learning model acts as a foundation for more advanced reinforcement learning models, such as Deep Q-Networks (DQN) and Double Deep Q-Networks (DDQN). Unlike this baseline approach, which learns from precomputed Q-values, DQN and DDQN dynamically update Q-values during training, allowing the model to adapt in real-time to market conditions. The baseline model, however, provides a starting point for trade prediction, enabling a structured approach for testing market patterns before implementing more complex reinforcement learning strategies. By using this classification-based approach, the study establishes a strong foundation for transitioning to advanced RL-based trading models, ensuring that the reinforcement learning agent has a pre-trained understanding of profitable trading actions before engaging in dynamic learning and real-time decision-making.

3.4.2 Deep Q-Network (DQN) for Stock Trading

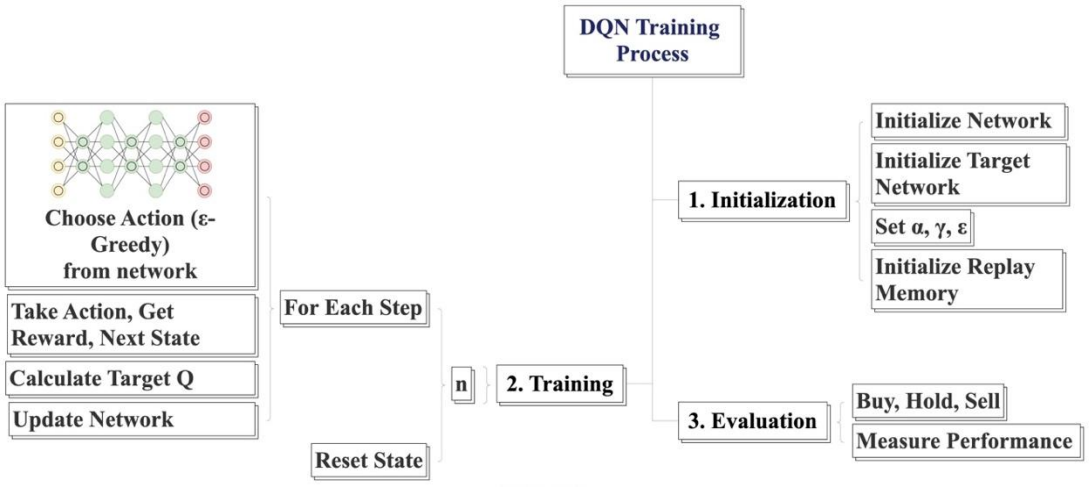

Figure 12: Deep Q-Network (DQN) for Stock Trading

The Deep Q-Network (DQN) is an advanced reinforcement learning model that enhances traditional Q-learning by replacing the static Q-table with a deep neural network. Unlike the Q-learning baseline model, which learns from precomputed Q-values, DQN dynamically updates its Q-values based on real-time trading experiences, making it more adaptable to changing stock market conditions. DQN is implemented in this study using a deep neural network, combined with experience replay and an epsilon-greedy policy to optimize trade execution. The model learns to predict the best action (Buy, Hold, or Sell) by analyzing historical stock indicators, price trends, and volume changes.

3.4.2.1 Limitations of the Q-learning Baseline Model

While the Q-learning baseline model provides a structured approach for predicting trading actions, it suffers from several limitations:

- **Lack of Real-Time Q-Value Updates** - The baseline model learns from static Q-values generated during simulation but does not dynamically update them during market interactions.
- **Scalability Issues** - Stock market data has high-dimensional features, making it inefficient to store and update Q-values for all possible states using traditional methods.
- **Overfitting to Past Market Trends** - The baseline model relies solely on historical Q-values, which may not generalize well to new market conditions.

To overcome these challenges, DQN introduces deep neural networks to approximate Q-values, enabling the RL agent to learn directly from market interactions.

3.4.2.2 Introduction to Deep Q-Networks (DQN)

The Deep Q-Network (DQN) extends traditional Q-learning by:

- Using deep neural networks to estimate Q-values instead of a precomputed table.
- Implementing experience replay to stabilize training and improve learning.
- Applying an epsilon-greedy policy to balance exploration and exploitation.
- At each time step t, the RL agent:
- Observes the current stock market state $\boldsymbol{S_t}$.
- Chooses an action $\boldsymbol{A_t}$ (Buy, Hold, or Sell) using an epsilon-greedy policy.
- Receives a reward $\boldsymbol{R_t}$ and transitions to the next state $\boldsymbol{S_{t+1}}$.
- Stores experience $(\boldsymbol{S_t, A_t, R_t, S_{t+1}})$ in an experience replay buffer.
- Samples past experiences from the buffer and updates the neural network.

Q-Value Update Rule in DQN

The Q-value function in DQN is updated as follows:

$$Q(s,a) = Q(s,a) + \alpha[r + \gamma \max Q\,(s', a') - Q(s,a)]$$

where:

- $\boldsymbol{Q(s,a)}$ is the **predicted Q-value** for state ss and action aa.
- $\boldsymbol{\alpha}$ is the **learning rate**, controlling how much the model learns from new experiences.
- $\boldsymbol{r}$ is the **reward** for executing action aa.
- $\boldsymbol{\gamma}$ is the **discount factor**, which determines the importance of future rewards.
- $\boldsymbol{\max Q\,(s', a')}$ is the **highest predicted Q-value** in the next state s'.

By continuously updating these Q-values, the model learns to make profitable trading decisions over time.

3.4.2.3 DQN Model Architecture

DQN uses a deep neural network to approximate Q-values for each possible action (Buy, Hold, Sell). The model consists of:

Network Components

- **Input Layer**
    - Takes in stock market indicators (OHLC prices, volume, and technical indicators).
    - Applies feature scaling using Standard Scaler for stable learning.
- **Hidden Layers**
    - Multiple fully connected layers with ReLU activation for pattern recognition.
    - Dropout layers to prevent overfitting.
- **Output Layer**
    - Uses a SoftMax activation function to output Q-values for Buy, Hold, and Sell actions.

Mathematical Representation: Each hidden layer applies a transformation:

$$h_i = f(W_i X + b_i)$$

Equation (21) : Neural Network Hidden Layer Formula

where:

- $\boldsymbol{h_i}$ is the output of the hidden layer.
- W_i is the weight matrix.
- $\boldsymbol{X}$ is the input feature vector.
- $\boldsymbol{b_i}$ is the bias term.
- $\boldsymbol{f(x)}$ is the ReLU activation function:

$$f(x) = \max(0, x)$$

The final output layer of the DQN model applies a Softmax function to generate normalized Q-values, ensuring that the Q-values for the three possible trading actions sum to 1. This allows the model to output probabilistic Q-values, making the decision-making process more interpretable.The Softmax function is defined as:

$$P(a) = \frac{e^{Q(s,a)}}{\sum_j ve^{Q(s,j)}}$$

Equation (22) : Softmax Function for DQN network

where:

- $\boldsymbol{P(a)}$ is the probability of selecting action aa (Buy, Hold, or Sell).
- $\boldsymbol{Q(s,a)}$ is the Q-value for action $\boldsymbol{a}$ in state$\boldsymbol{s}$.
- The denominator ensures that the **sum of probabilities for all actions is 1**, making it a valid probability distribution.
- This approach allows the RL agent to:
    - Ensure exploration: Even if one action has a higher Q-value, the agent still assigns non-zero probabilities to other actions.
    - Avoid extreme overestimation: By normalizing Q-values, the Softmax function prevents the model from assigning unrealistically high Q-values to one action.
    - Smooth action selection: Unlike greedy selection, Softmax ensures that the model gradually shifts toward optimal actions rather than making abrupt changes.

3.4.2.4 Experience Replay Mechanism

One of the challenges in reinforcement learning is that consecutive market trades are highly correlated, causing the model to overfit short-term price fluctuations. Experience replay:

- Stores past experiences in a memory buffer.
- Randomly samples past experiences for training.
- Breaks correlation between consecutive trades, improving stability.

Pseudocode for Experience Replay Mechanism

```
INITIALIZE replay buffer
FOR each training step:
   STORE (state, action, reward, next state) in buffer
   IF buffer is full:
      SAMPLE a batch of past experiences
      COMPUTE Q-value updates
      APPLY gradient updates to network
END FOR
```

By learning from a mix of past experiences, the RL model avoids overfitting to recent price movements, improving generalization.

3.4.2.5 Training Process of DQN

The training loop follows these steps:

- Initialize replay buffer and neural network.
- Observe the initial market state.
- Select an action using the epsilon-greedy policy.
- Execute the action, receive reward, and transition to the next state.
- Store experience in replay buffer.

- Sample a batch from the replay buffer for training.
- Update the Q-network using the loss function.
- Repeat until training is complete.

Pseudocode for Training Process of DQN

```
INITIALIZE replay buffer
FOR each episode:
   RESET environment
   FOR each time step:
      OBSERVE current state S_t
      CHOOSE action A_t using epsilon-greedy policy
      EXECUTE action and receive reward R_t
      STORE experience in replay buffer
      SAMPLE a batch of past experiences
      UPDATE Q-network using loss function
   END FOR
END FOR
```

3.4.2.6 Summary

The Deep Q-Network (DQN) model significantly improves upon the Q-learning baseline model by introducing neural networks for Q-value approximation. Unlike the baseline approach, which relies on precomputed Q-values, DQN dynamically updates its Q-values based on real-time market interactions, allowing the model to adapt to changing stock market conditions. The Q-learning table used in traditional reinforcement learning becomes infeasible for financial data due to the high dimensionality of stock features,

making DQN an essential advancement. By using a deep neural network, the model can efficiently process historical price data, technical indicators, and trading volume, learning to make more informed Buy, Hold, and Sell decisions. This approach enables the model to handle large-scale financial data while generalizing across different market conditions, ensuring better predictive accuracy and decision-making flexibility.

One of the most significant enhancements in DQN is the introduction of experience replay, a mechanism that helps break the correlation between consecutive stock trades. In traditional RL models, each new experience is immediately used to update Q-values, leading to instability in learning, especially in volatile markets. Experience replay mitigates this by storing past experiences in a replay buffer, allowing the model to sample random experiences for training. This process prevents overfitting to short-term market fluctuations, ensuring that the model learns from diverse trading conditions rather than memorizing specific price trends. By training on a variety of past experiences, DQN becomes more resilient to sudden market changes, improving its ability to execute profitable trades across different time frames.

While DQN offers substantial improvements over the baseline model, it suffers from Q-value overestimation, where the model tends to assign excessively high values to certain trading actions, leading to suboptimal trades. This issue arises because the same network is used for both selecting and evaluating Q-values, causing inaccurate estimations that impact trading performance. To address this, the next section explores Double Deep Q-Networks (DDQN), an enhancement over DQN that introduces a separate target network to stabilize learning and prevent overestimation bias. By leveraging this improved training strategy, DDQN ensures that Q-values are more realistic and balanced, leading to better trading stability and long-term profitability in financial markets.

3.4.3 Double Deep Q-Network (DDQN) for Stock Trading

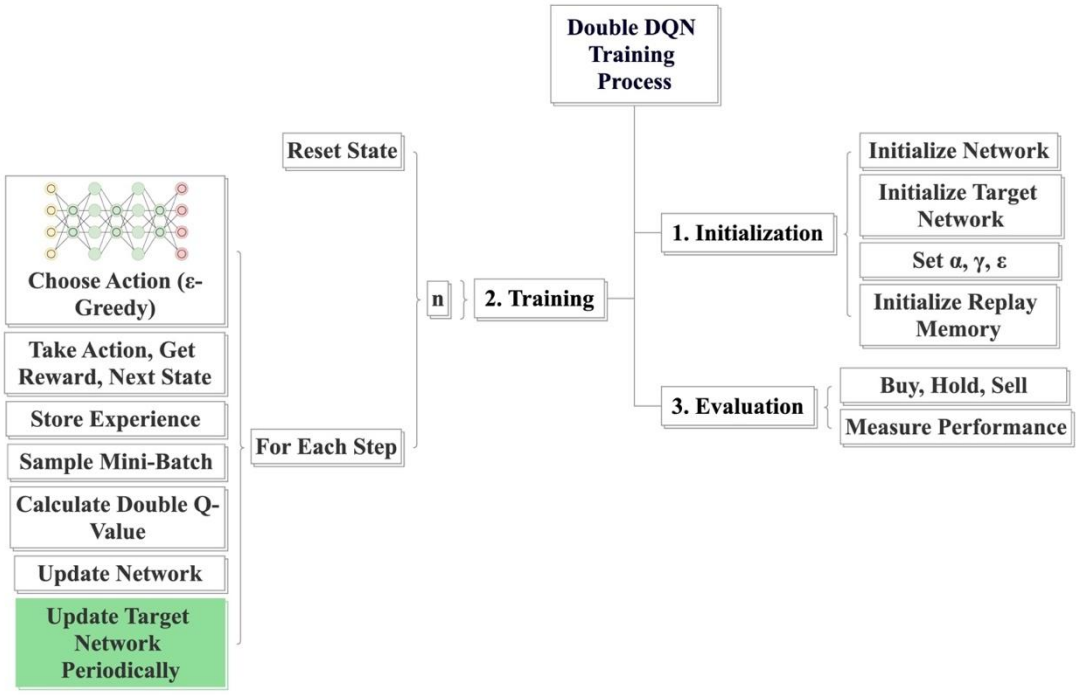

Figure 13:Double Deep Q-Network (DDQN) for Stock Trading

The Double Deep Q-Network (DDQN) is an improved version of the Deep Q-Network (DQN) that resolves one of the major challenges in reinforcement learning: Q-value overestimation. While DQN effectively replaces the static Q-table with a deep neural network, it suffers from a bias where it overestimates the Q-values, leading to suboptimal trading decisions. DDQN addresses this issue by decoupling action selection from Q-value evaluation, making learning more stable and accurate. In stock trading, overestimating the potential reward of a Buy, Hold, or Sell action can lead to unrealistic trading behaviors, causing the RL agent to execute high-risk trades based on incorrect value estimates. DDQN prevents this by introducing a separate target network, ensuring that Q-values remain more balanced and representative of real market conditions.

The DDQN file implements this improved reinforcement learning approach by incorporating:

- Two separate networks for action selection and Q-value estimation.
- Target network updates to stabilize learning and reduce Q-value fluctuations.
- Experience replay to further refine trade execution over multiple episodes.

3.4.3.1 Limitations of DQN and the Need for DDQN

The Q-value Overestimation Problem. DQN updates its Q-values using the same network for both:

- Selecting the best action.
- Evaluating the Q-value of the selected action.

This creates a positive bias, where the network overestimates the value of certain actions, causing the agent to:

- Take excessive risks by choosing overvalued trades.
- Misjudge market signals, leading to incorrect Buy or Sell decisions.

Mathematically, DQN selects and evaluates actions using:

$$Q(s,a) = r + \gamma \max Q\,(s', a')$$

Since the same Q-network is used for both choosing and evaluating the action, it often inflates Q-values, making poor trades seem more profitable than they actually are.

How DDQN Fixes Q-value Overestimation

DDQN solves this by introducing a separate target network that prevents the model from using its own overestimated values to update Q-values. Instead of using:

$$Q(s,a) = r + \gamma \max Q\,(s', a')$$

DDQN decouples action selection and evaluation by introducing two networks:

- **Online Network** – Selects the action.
- **Target Network** – Evaluates the Q-value of the selected action.

Now, Q-values are updated using:

$$Q(s,a) = r + \gamma Q_{\text{target}}\left(s', \arg\max Q_{\text{online}}\,(s', a')\right)$$

Equation (23) : DDQN Q-value updating formula

where:

- $\boldsymbol{Q_{\text{online}}(s', a')}$ selects the best action.
- $\boldsymbol{Q_{\text{target}}(s', a')}$ evaluates its value.

By separating action selection and evaluation, DDQN prevents inflated Q-values, leading to:

- More realistic trading decisions.
- Better risk management.
- Improved trading stability in volatile markets.

3.4.3.2 Double Deep Q-Network (DDQN) Architecture

The Double Deep Q-Network (DDQN) architecture follows the same general structure as DQN but with two neural networks instead of one.

Network Components

- **Online Network**
  - Learns optimal Q-values for stock market trading.
  - Used to select the best trading action.
- **Target Network**
  - A copy of the online network, but updated **less frequently**.
  - Used to evaluate Q-values, preventing overestimation.
- **Experience Replay**
  - Stores past transactions to improve learning stability.

3.4.3.3 Target Network Strategy in DDQN

The target network is a crucial addition to DDQN, helping stabilize Q-value updates. Instead of updating after every trade, the target network is updated periodically using:

$$\theta_{\text{target}} = \tau\theta_{\text{online}} + (1 - \tau)\theta_{\text{target}}$$

Equation (24) : DDQN target network update policy

where:

- $\boldsymbol{\theta_{\text{target}}}$ represents the weights of the **target network**.
- $\boldsymbol{\theta_{\text{online}}}$ represents the weights of the **online network**.
- $\boldsymbol{\tau}$ is a **small update rate (e.g., 0.01 - 0.05)**, ensuring gradual learning.

This update rule smoothly blends new knowledge with past experiences, preventing sudden, unstable Q-value changes.

3.4.3.4 DDQN Training Process

The DDQN agent follows these steps:

- Initialize online and target networks with identical weights.
- Observe market conditions and select an action using the epsilon-greedy strategy.
- Execute the action and observe the resulting market state.
- Store the experience (state, action, reward, next state) in the replay buffer.
- Sample a batch of past experiences and update the online network using:
- $Q(s, a) = r + \gamma Q_{\text{target}}\big(s', \arg\max Q_{\text{online}}\,(s', a')\big)$
- Periodically update the target network using the weighted averaging technique.
- Repeat until the model is fully trained.

Pseudocode for DDQN Implementation

```
INITIALIZE online network and target network with identical weights
INITIALIZE replay buffer
FOR each episode:
   RESET environment
   FOR each time step:
      OBSERVE current state S_t
      CHOOSE action A_t using epsilon-greedy policy
      EXECUTE action and receive reward R_t
      STORE experience (S_t, A_t, R_t, S_t+1) in replay buffer
      IF replay buffer has enough samples:
         SAMPLE a batch of experiences
         COMPUTE target Q-value:
         Q(s, a) = r + γ Q_target(s', argmax Q_online(s', a'))
         UPDATE online network with new Q-values
      EVERY few steps:
         UPDATE target network using:
         θ_target = τ θ_online + (1 - τ) θ_target
   END FOR
END FOR
```

3.4.3.5 Comparison of DDQN vs. DQN

Below table compares Deep Q-Networks (DQN) and Double Deep Q-Networks (DDQN), highlighting their differences in action selection, Q-value estimation, learning

stability, and trading performance. DQN suffers from Q-value overestimation, leading to aggressive trading decisions, whereas DDQN mitigates this issue by using a separate target network, resulting in more stable and risk-aware trading strategies. These improvements make DDQN better suited for volatile and dynamic market conditions.

Table 11: Comparision of DDQN vs DQN

Feature	**DQN**	**DDQN**
Action Selection	Uses the same network for selection and evaluation.	Separates action selection and evaluation using two networks.
Q-value Overestimation	High, leading to risky trades.	Reduced, making decisions more stable.
Learning Stability	Prone to unstable updates.	More stable due to the target network.
Trading Performance	Can make overconfident trading decisions.	More balanced and risk-aware trading decisions.

3.4.3.6 Summary

The Double Deep Q-Network (DDQN) enhances reinforcement learning by addressing a critical issue in Deep Q-Networks (DQN)—Q-value overestimation. In standard DQN, the same neural network is responsible for both selecting the best action and evaluating its Q-value, which often leads to inflated value estimates. This overestimation can cause the RL agent to favor high-risk trades that seem profitable but may not yield consistent long-term rewards. By introducing a separate target network, DDQN ensures that Q-values remain realistic and unbiased, leading to more stable

trading strategies. This improvement is particularly important in financial markets, where incorrectly valuing trades can lead to significant financial losses.

Another key enhancement in DDQN is the target network update strategy, which prevents abrupt changes in Q-values, making learning smoother and more reliable. Unlike DQN, where the Q-values fluctuate excessively due to constant network updates, DDQN gradually updates the target network over multiple training steps. This approach ensures that the model learns trading patterns effectively while avoiding overfitting to short-term market noise. Additionally, by decoupling action selection from Q-value estimation, DDQN allows the RL agent to make better-informed trade decisions, reducing the likelihood of executing suboptimal Buy, Hold, or Sell actions.

Overall, DDQN provides a more stable, risk-aware, and efficient reinforcement learning framework for stock trading. It enables the RL agent to adapt to volatile market conditions, handle long-term investment strategies, and make better trading decisions based on realistic reward estimations. By incorporating experience replay, target network updates, and an improved Q-value update mechanism, DDQN outperforms standard DQN in terms of decision accuracy, risk management, and trading profitability. As a result, DDQN serves as an essential step toward building advanced AI-driven trading strategies that are not only profitable but also sustainable in real-world financial markets.

3.5 Training the RL Agent

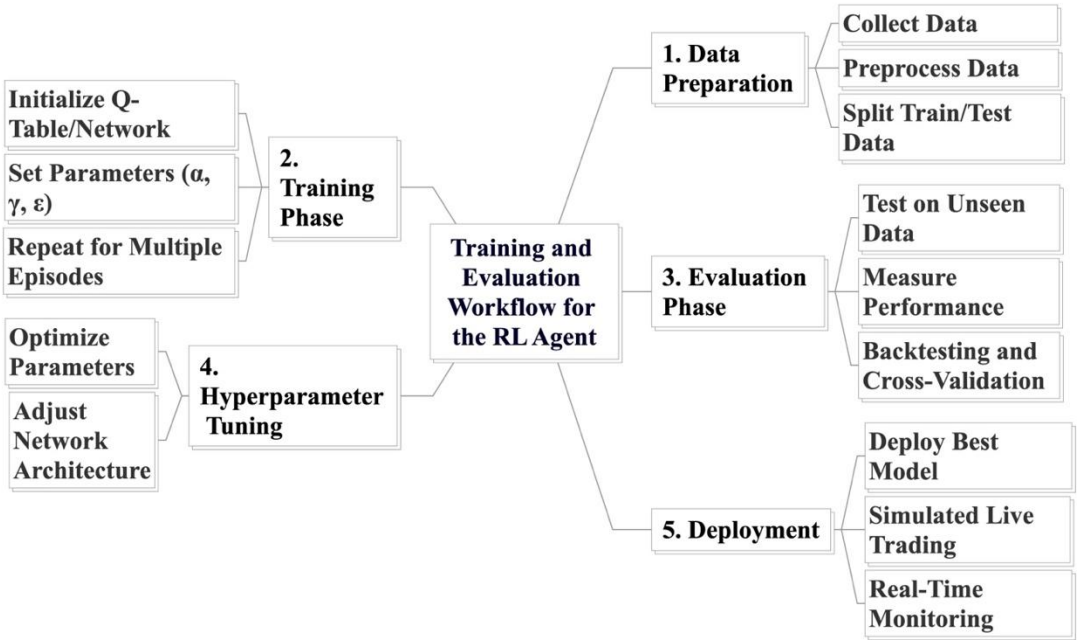

Figure 14:Training and Evaluation Workflow for RL Agent

The training phase is the most critical step in reinforcement learning, where the Q-learning, Deep Q-Network (DQN), and Double Deep Q-Network (DDQN) models learn to make trading decisions based on historical stock market data. Each model follows a different training approach, and this section details how they are trained to optimize trading decisions. Since Q-learning (baseline model) follows a supervised deep learning approach, while DQN and DDQN are reinforcement learning-based, the training methodology varies significantly. However, they share some common elements, such as:

- Stock market simulation as the training environment.
- Exploration vs. exploitation tradeoff using epsilon decay.
- Gradient-based learning with experience replay (for DQN and DDQN).
- Hyperparameter tuning to optimize learning performance.

3.5.1 Training the Q-Learning Baseline Model (Supervised Deep Learning Approach)

The Q-learning baseline model does not use reinforcement learning techniques like experience replay or epsilon decay. Instead, it is trained using a supervised deep learning approach, where the model learns from precomputed Q-values generated during the Q-value simulation phase (Section 3.3).

Training Process Overview

- **Dataset Creation:**
  - The dataset consists of **technical indicators, OHLC prices, and trading volume**.
  - The labels (Buy, Hold, or Sell) are assigned based on the **highest Q-value at each time step**.
- **Supervised Learning:**
  - The model is trained as a **classification problem**, where it learns to predict **the best trading action** for each market state.
  - The **Categorical Cross-Entropy loss function** is used to measure how well the predicted probabilities match the true labels.
- **Dataset Imbalance Handling:**
  - Since Hold actions occur more frequently than Buy or Sell, **SMOTE (Synthetic Minority Over-sampling Technique)** is applied to balance the dataset.

Pseudocode for Training Q-Learning Model

LOAD historical stock data

CONVERT Q-values to Buy, Hold, or Sell labels

APPLY SMOTE to balance dataset

INITIALIZE deep neural network

TRAIN model using categorical cross-entropy loss

EVALUATE model on test dataset

Since this model follows a supervised learning approach, its training is much faster compared to DQN and DDQN. However, the limitation is that it does not dynamically update Q-values during training, making it less adaptive to changing market conditions.

3.5.2 Training the Deep Q-Network (DQN) (Reinforcement Learning Approach)

Unlike the Q-learning baseline model, the DQN model is trained using reinforcement learning techniques, meaning it learns by interacting with the stock market simulation rather than labeled data.

Training Process Overview

- **Initialize the RL Agent:**
    - The model starts with random weights and no prior knowledge.
    - A stock market simulation is set up where the agent learns to trade over multiple episodes.
- **Exploration vs. Exploitation Strategy:**
    - The model uses an epsilon-greedy policy to explore new trading strategies before shifting to exploitation of profitable strategies.
- **Experience Replay:**
    - The agent stores past trades in a replay buffer and learns from them to stabilize training.

- **Gradient Updates:**
    - The Mean Squared Error (MSE) loss function is used to update the neural network weights based on Q-value errors.

Pseudocode for Training DQN Model

```
INITIALIZE replay buffer and neural network
FOR each training episode:
   RESET market simulation to start of a trading day
   WHILE market is open:
      OBSERVE market state S_t
      SELECT action A_t using epsilon-greedy strategy
      EXECUTE action and receive reward R_t
      STORE (S_t, A_t, R_t, S_t+1) in replay buffer
      SAMPLE batch from replay buffer
      UPDATE Q-network using gradient descent
   END WHILE
   REDUCE epsilon to shift from exploration to exploitation
END FOR
```

This iterative process allows the model to continuously improve its trading strategy, making it more adaptable to real-world financial markets.

3.5.3 Training the Double Deep Q-Network (DDQN)

DDQN follows a similar training process to DQN but introduces an additional target network to stabilize Q-value updates.

Why DDQN Needs a Target Netwrk

- DQN tends to overestimate Q-values, leading to high-risk trades.
- DDQN fixes this by separating action selection from Q-value evaluation, ensuring that the agent makes more realistic trading decisions.

Training Process Overview

- **Initialize Two Neural Networks:**
    - Online Network: Selects actions.
    - Target Network: Evaluates Q-values (updated periodically).
- **Follow the Same Process as DQN:**
    - The model interacts with the stock market simulation, stores past experiences, and updates Q-values.
- **Target Network Update:**
    - Instead of using the online network to update Q-values, DDQN updates them using the target network, preventing overestimation bias.

Pseudocode for Training DDQN Model

```
INITIALIZE online network and target network
INITIALIZE replay buffer
FOR each training episode:
   RESET market simulation
   WHILE market is open:
      OBSERVE market state S_t
      SELECT action A_t using epsilon-greedy policy
      EXECUTE action and receive reward R_t
      STORE experience in replay buffer
```

SAMPLE batch from replay buffer

COMPUTE target Q-value:

$Q(s, a) = r + \gamma\ Q\_target(s', argmax\ Q\_online(s', a'))$

UPDATE online network using loss function

END WHILE

EVERY few steps: UPDATE target network

REDUCE epsilon to shift from exploration to exploitation

END FOR

By updating the target network periodically, DDQN ensures that the model learns more stable and risk-aware trading strategies.

3.5.4 Hyperparameter Tuning and Final Training Loop Execution

Since all three models share similar training settings, they are fine-tuned using common hyperparameters to optimize learning performance. Below table outlines the key hyperparameters used in training the reinforcement learning models, explaining their roles and selected values. The learning rate (α) controls how quickly the model updates Q-values, while the discount factor (γ) prioritizes long-term rewards over immediate gains. Batch size determines the number of training samples per step, and epsilon decay rate regulates the transition from exploration to exploitation. These hyperparameters are crucial in ensuring efficient learning, stability, and optimal decision-making in the trading environment.

Table 12: Hyperparameter Tuning and Final Training Loop Execution

Hyperparameter	**Description**	**Value Used**

Learning Rate (α)	Controls how fast Q-values update.	0.001
Discount Factor (γ)	Determines the importance of future rewards.	0.99
Batch Size	Number of experiences sampled per training step.	64
Epsilon Decay Rate	Speed of exploration-to-exploitation transition.	0.995

Full Training Loop Execution

Once hyperparameters are tuned, the **full training process** follows these steps:

- Initialize replay buffer and neural networks.
- Observe the initial stock market state.
- Select an action using the epsilon-greedy policy.
- Execute the action, receive reward, and transition to the next state.
- Store experience in replay buffer.
- Sample a batch from the replay buffer for training.
- Update the Q-network using gradient descent.
- Periodically update the target network (for DDQN).
- Repeat until the model is fully trained.

Now that the RL models are fully trained, the next section will focus on evaluating their performance using financial metrics.

3.6 Performance Evaluation Metrics

Once the Q-learning, DQN, and DDQN models are trained, they must be evaluated to determine their effectiveness in stock trading. Since these models are used

for financial decision-making, their performance is assessed using both financial and machine learning metrics.

The evaluation process ensures that the RL agent:

- Maximizes profits while managing risk.
- Executes trades with high accuracy.
- Generalizes well to unseen market conditions.
- Learns stable trading strategies without overfitting.

This section is divided into financial performance metrics (to assess profitability and risk) and machine learning performance metrics (to evaluate model accuracy and convergence).

3.6.1 Financial Metrics for RL-Based Trading Models

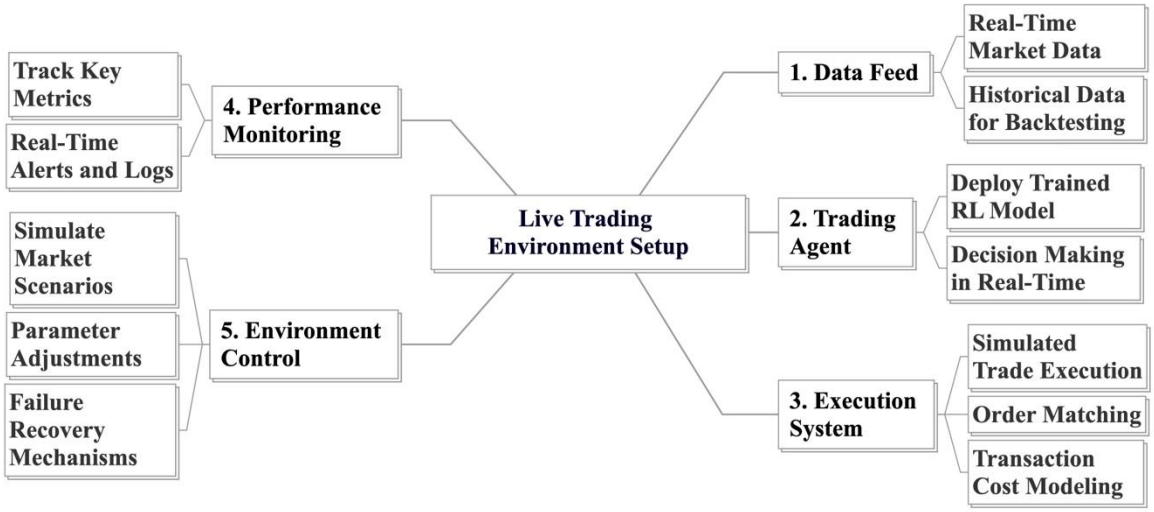

Figure 15: Environment Setup for Live RL-Based Trading Models

Financial metrics measure the profitability, risk, and efficiency of the RL trading models in a real-world market environment. These metrics help compare the performance

of Q-learning (supervised deep learning), DQN, and DDQN. Below figure shows setup to use model for live trading.

3.6.1.1 Cumulative Profit & Loss (PnL)

Definition: Cumulative Profit & Loss (PnL) measures the total returns generated by the RL agent over a trading period. It is computed as the difference between cumulative profits from successful trades and cumulative losses from unsuccessful trades.

$$\text{PnL} = \sum_{t=1}^{T} \left(P_{\text{sell},t} - P_{\text{buy},t}\right) \times N_t - C_t$$

Equation (25) : Cumulative Profit and Loss formula

where:

- $P_{\text{sell},t}$ = Price at which stock is sold.
- $P_{\text{buy},t}$ = Price at which stock is bought.
- N_t = Number of shares traded.
- C_t = Transaction costs (commission, slippage, taxes).

Interpretation:

- Higher PnL → The RL agent is executing profitable trades consistently.
- Negative PnL → The agent is making unprofitable decisions, meaning it may need further training or reward function adjustments.

3.6.1.2 Sharpe Ratio (Risk-Adjusted Returns)

Definition: The Sharpe Ratio measures the risk-adjusted returns of the RL agent by comparing the excess return over a risk-free rate to the portfolio's volatility.

$$\text{Sharpe Ratio} = \frac{R_p - R_f}{\sigma_p}$$

Equation (26) : Sharpe Ratio Formula

where:

- $\boldsymbol{R_p}$ = Average return of the RL trading model.
- $\boldsymbol{R_f}$ = Risk-free return (e.g., government bond rate).
- $\boldsymbol{\sigma_p}$ = Standard deviation of portfolio returns (volatility).

Interpretation:

- Higher Sharpe Ratio (>1.0) → The RL model is generating high returns per unit of risk.
- Low Sharpe Ratio (<0.25) → The model's returns are highly volatile, suggesting high-risk trades.

3.6.1.3 Maximum Drawdown (MDD) (Worst-Case Risk Management)

Definition: Maximum Drawdown (MDD) measures the largest peak-to-trough decline in the RL agent's portfolio value over time.

$$\text{MDD} = \max\left(\frac{P_{\text{peak}} - P_{\text{trough}}}{P_{\text{peak}}}\right)$$

Equation (27) : Maximum Drawdown Formula

where:

- $\boldsymbol{P}_{\text{peak}}$ = Highest portfolio value recorded.
- $\boldsymbol{P}_{\text{trough}}$ = Lowest portfolio value recorded after the peak.

Interpretation:

- Lower MDD → The agent maintains consistent performance without large losses.
- High MDD → The agent suffers large portfolio declines, making it unreliable in real trading.

3.6.1.4 Win-Loss Ratio (Trade Accuracy Analysis)

Definition: The Win-Loss Ratio measures the proportion of profitable trades versus unprofitable trades executed by the RL model.

$$\text{Win-Loss Ratio} = \frac{\text{Number of Winning Trades}}{\text{Number of Losing Trades}}$$

Equation (28) : Win-Loss Ratio Formula

Interpretation:

- Higher Win-Loss Ratio (>1.0) → The RL agent is executing more successful trades than losing trades.
- Low Win-Loss Ratio (<1.0) → The agent is making more losing trades, indicating the need for strategy refinement.

3.6.2 Summary of Evaluation Metrics

Below table defines key performance metrics used to evaluate the trading models, explaining their purpose and ideal values. Cumulative PnL measures total profitability, where higher values indicate better performance. Sharpe Ratio assesses risk-adjusted returns, with values above 1.0 preferred. Max Drawdown quantifies the worst-case portfolio loss, meaning lower values indicate better risk control. Win-Loss Ratio evaluates trade accuracy, where a value greater than 1.0 suggests more successful trades

than unsuccessful ones. These metrics help assess the effectiveness, stability, and risk management of reinforcement learning-based trading strategies.

Table 13: Summary of Evaluation Metrics

Metric	Purpose	Good Value
Cumulative PnL	Measures total profit over time	Higher is better
Sharpe Ratio	Measures risk-adjusted return	Higher is better
Max Drawdown	Measures worst-case portfolio loss	Lower is better
Win-Loss Ratio	Measures trade accuracy	> 1.0

3.7 Implementation and Deployment Considerations

After training and evaluating the Q-learning, DQN, and DDQN models, the next step is deploying these models in a real-world trading environment. While reinforcement learning models can perform well in a simulated environment, deploying them in live markets presents additional challenges related to computation, execution speed, risk management, and adaptability.

This section covers:

- Computational complexity and training time – The resource requirements for model training and inference.
- Challenges in real-time market execution – Ensuring low-latency trading decisions.
- Scalability considerations – Expanding the RL models to different financial markets (Forex, Crypto, Commodities).

3.7.1 Computational Complexity and Training Time

Training reinforcement learning models for stock trading is computationally expensive, especially for DQN and DDQN, which rely on deep neural networks for Q-value approximation.

Factors Affecting Training Time

Model Complexity

- Q-learning baseline model (Supervised Learning) → Fastest training (~minutes to hours).
- DQN (Deep Q-Network) → Requires training neural networks and experience replay, making it significantly slower (~hours to days).
- DDQN (Double Deep Q-Network) → Further complexity due to target network updates, increasing training time.

Size of Training Data

- Training on 4 years of NIFTY 50 intraday data (minute-level) results in millions of data points.
- More data improves generalization but increases computation time.

Hyperparameter Optimization

- Tuning learning rate, discount factor, batch size, and epsilon decay requires multiple training runs, further increasing computation time.

Hardware Considerations for Training

- For Q-learning baseline → Can run on standard CPUs due to lower complexity.
- For DQN/DDQN → Requires GPUs or TPUs for faster training.

- Cloud-Based Training → Using platforms like Google Cloud, AWS, or Azure can significantly speed up the process.

Optimization Strategy → Training on a subset of data first, then scaling to the full dataset to reduce training time.

3.7.2 Challenges in Real-Time Market Execution

Deploying reinforcement learning models in live trading presents several challenges, especially in high-speed intraday trading environments.

Latency Issues in Trade Execution

- Financial markets operate in milliseconds, but deep learning-based RL models require computational time for inference.
- Delays in decision-making can lead to missed trading opportunities or execution at unfavorable prices.
- Solution → Optimize inference speed using low-latency computing architectures (e.g., TensorRT, ONNX for model acceleration).

Adaptability to Changing Market Conditions

- Live markets are unpredictable, and RL models trained on historical data may not generalize well to new events.
- Market anomalies like economic crashes, pandemics, or sudden news events require real-time retraining.
- Solution → Implement adaptive learning, where the model is retrained periodically with the latest market data.

Risk Management and Trade Execution Controls

- Live deployment requires strict risk controls to prevent excessive losses.
- Stop-loss mechanisms should be implemented outside the RL model to prevent major drawdowns.

- Solution → Limit trading exposure by setting:
    - Maximum position size per trade.
    - Daily risk limits (maximum allowable loss).
    - Automated stop-loss orders to exit bad trades before major losses occur.

Integrating RL Models with Brokerage APIs

To execute trades, RL models must integrate with a brokerage API (e.g., Zerodha, Interactive Brokers, Alpaca).

Steps for API Integration:

- **Retrieve live stock data from API** → Format it to match the model's input requirements.
- **Pass the live market state to the trained RL model** → Get the predicted action (Buy, Hold, Sell).
- **Execute the trade via the broker's API** → Ensure real-time execution.
- **Store trade history for model retraining** → Keep a log of executed trades for continuous learning.

3.7.3 Scalability to Other Financial Markets

While this study focuses on NIFTY 50 intraday trading, the RL models can be extended to other financial markets with some modifications.

Forex and Cryptocurrency Markets

- Forex (foreign exchange trading) operates 24/7, requiring RL models to handle continuous trading.
- Crypto markets are highly volatile, making risk management even more crucial.

- Modifications required:
    - Additional technical indicators (e.g., VWAP, Fibonacci retracement).
    - More frequent retraining due to extreme price swings.

Commodity and Futures Trading

- Futures markets have different trading hours and expiration dates.
- Adaptation needed: RL models must account for contract expiration and rollovers.

High-Frequency Trading (HFT) Considerations

- RL models must be highly optimized for execution speed.
- Requires specialized hardware (e.g., FPGA, low-latency trading engines).

Key Takeaway is RL models trained for NIFTY 50 can be adapted to other asset classes by tuning reward functions, adjusting feature engineering, and modifying trading strategies.

3.7.4 Summary

The deployment of RL models in live trading introduces several real-world challenges that must be carefully addressed:

- **Computational Complexity** → DQN and DDQN models require significant computational resources, especially for large-scale datasets. Optimizing GPU-based training can accelerate learning.
- **Real-Time Execution Challenges** → Ensuring low-latency trade execution is critical in live markets. Implementing optimized inference pipelines reduces delays.

- **Risk Management & Trade Execution Controls** → RL-based trading systems must have strict risk controls, including position limits, stop-loss mechanisms, and exposure monitoring.
- **Scalability to Other Markets** → With some adjustments, RL models can be extended to Forex, Crypto, Commodities, and Futures trading, allowing greater flexibility and profitability.

CHAPTER IV:

RESULTS

4.1 Research Question One

How effectively can a Double Deep Q-Network (DDQN)-based reinforcement learning agent autonomously execute optimal buy, hold, and sell decisions in the NIFTY50 intraday trading market?

4.1.1 Overview of Evaluation Metrics

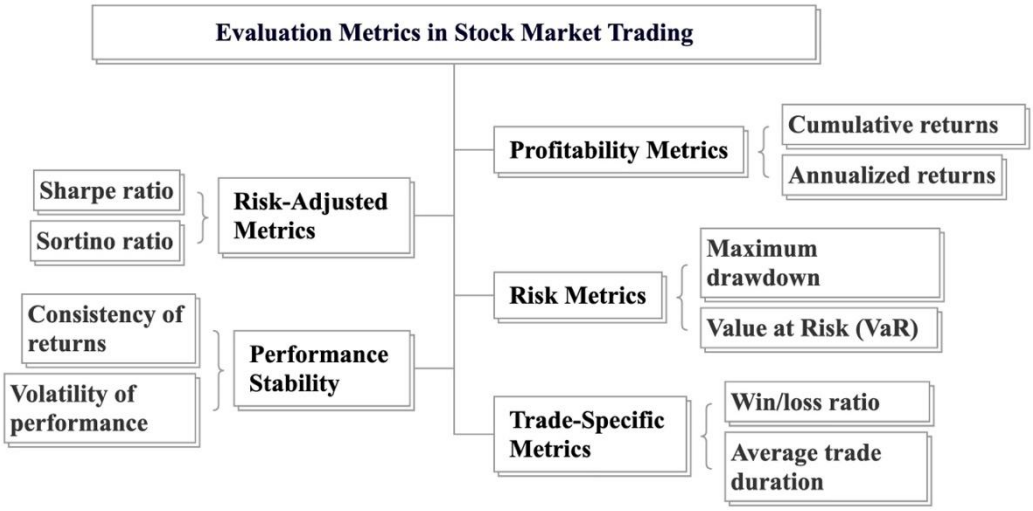

Figure 16:Evaluation Metrics in Stock Market Trading

The primary goal of reinforcement learning (RL)-based trading models is to maximize profits while minimizing risks, thereby improving decision-making in real-time stock market conditions. Unlike traditional algorithmic trading strategies that rely on fixed-rule-based approaches, RL models continuously learn and adapt to market conditions by updating their Q-values over time. To assess the DDQN agent's trading decision capability, the following performance metrics were computed

- **Cumulative Profit**: Total profitability achieved across trades.
- **Average Profit per Trade**: Mean profitability per trade executed.
- **Win Rate (%)**: Percentage of profitable trades over total trades.
- **Sharpe Ratio**: Risk-adjusted returns compared to traditional models.
- **Maximum Drawdown (%)**: Largest observed loss from peak to trough.

These metrics directly reflect how well the DDQN agent autonomously executes buy, hold, and sell actions over the one-year test period.

4.1.2 Performance Comparison Across Models

The DDQN model was benchmarked against Q-Learning (QN) and Deep Q-Network (DQN) models for validation. Below table provides a comparative analysis of Q-learning (QN), Deep Q-Network (DQN), and Double Deep Q-Network (DDQN) based on key performance metrics. DDQN outperforms both models in total profit, win rate, and Sharpe ratio, while maintaining the lowest max drawdown, making it the most stable and profitable trading model. DQN follows closely behind but still suffers from higher drawdowns due to Q-value overestimation. QN, while having the highest average profit per trade, struggles with lower total profit, win rate, and higher risk exposure, making it less suitable for dynamic market conditions.

Table 14:Overall Model Performance Summary

Model	Total Profit	Avg. Profit per Trade	Win Rate (%)	Sharpe Ratio	Max Drawdown (%)	Trade Count
DDQN	**1,151,325**	**411.52**	**67.72%**	**0.3450**	**-1.12%**	**2827.92**
DQN	1,022,486	384.05	66.57%	0.3170	-1.36%	2682.66

QN	460,426	439.86	65.77%	0.2262	-5.83%	1103.56

4.1.3 Visual Comparison

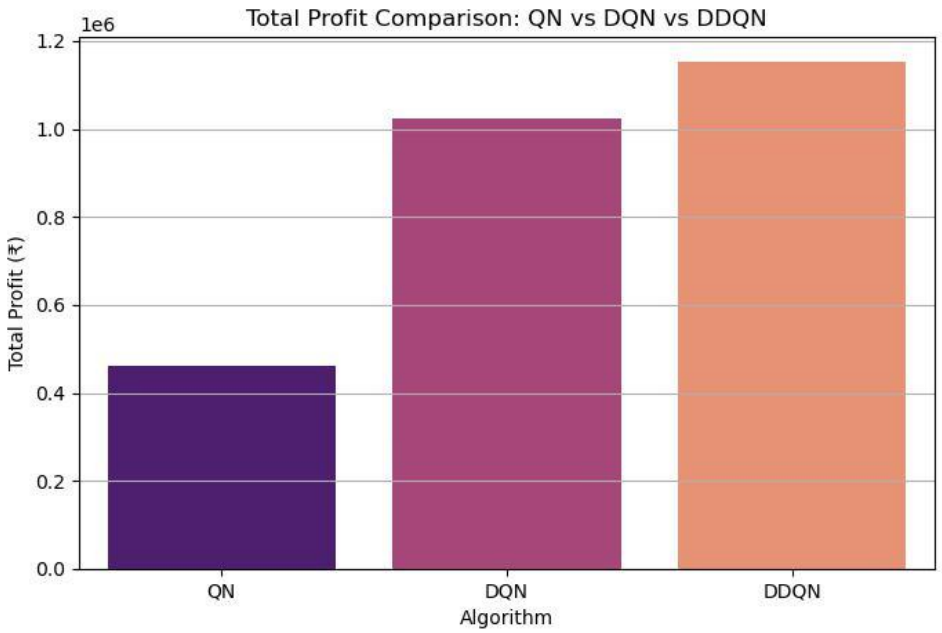

Figure 17:Illustrates the total cumulative profit comparison across the three models.

4.1.4 Key Observations

- **Profitability**: The DDQN agent achieved the highest cumulative profit (₹1,151,325) compared to QN and DQN, indicating superior decision-making.
- **Trading Efficiency**: DDQN's win rate (67.72%) was also the highest, reflecting a higher proportion of successful buy/hold/sell decisions.
- **Risk Management**: DDQN maintained the best Sharpe ratio and the lowest maximum drawdown among all models.

- **Trading Activity**: A high but controlled trade count indicates active participation without excessive overtrading.

4.2 Research Question Two

How can the DDQN-based RL agent be optimized to balance maximizing profitability with minimizing market risk, particularly during volatile periods?

4.2.1 Metrics for Profitability and Risk Evaluation

To assess the balance between profit generation and risk control, the following risk-adjusted performance indicators were analyzed:

- Sharpe Ratio: Measures risk-adjusted returns.
- Profit Factor: Ratio of total gross profit to total gross loss.
- Maximum Drawdown (%): Represents worst-case financial risk.

These metrics jointly determine how well the DDQN agent maximizes returns while managing exposure during volatile intraday trading conditions.

4.2.2 Comparative Risk-Adjusted Performance

The DDQN model's performance was benchmarked against DQN and QN models using the key risk metrics summarized in below table.

Table 15:Risk-Adjusted Performance Metrics Across Models

Model	Sharpe Ratio	Profit Factor	Maximum Drawdown (%)
QN	0.2262	1.88	-5.83%
DQN	0.3170	2.40	-1.36%
DDQN	0.3450	2.60	-1.12%

Table 16:Best Risk-Adjusted Stocks (Highest Sharpe Ratio)

Stock	Model	Sharpe Ratio	Max Drawdown (%)
ADANIENT	QN	0.40	-2.90%
ADANIENT	DDQN	0.40	-1.52%
TITAN	DDQN	0.40	-1.00%
ASIANPAINT	DDQN	0.40	-1.03%
HDFCBANK	DDQN	0.39	-0.86%
INFY	DDQN	0.39	-0.94%
RELIANCE	DDQN	0.38	-1.28%
HINDUNILVR	DDQN	0.38	-1.17%
JSWSTEEL	DDQN	0.38	-1.03%
ITC	DDQN	0.37	-0.94%

4.2.3 Key Observations

- **Higher Sharpe Ratio**: DDQN's Sharpe ratio (0.3450) is the highest among the three models, demonstrating superior risk-adjusted profitability.
- **Better Profit Factor**: The DDQN agent maintained a profit factor of 2.60, suggesting that the model achieved ₹2.60 profit for every ₹1.00 loss, outperforming other models.
- **Lower Maximum Drawdown**: DDQN suffered the least maximum drawdown (-1.12%), indicating better risk control during adverse market movements.

4.2.4 Visual Analysis

Two plots were generated to visualize the risk-return trade-off:

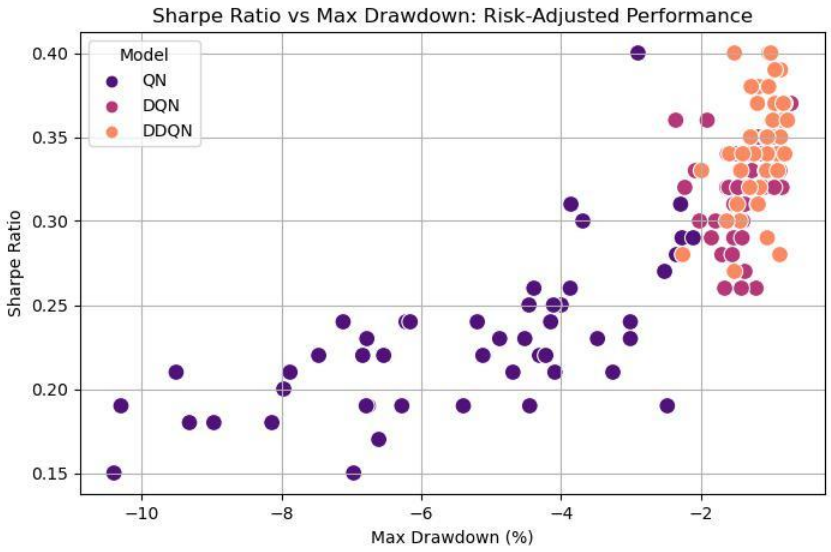

Figure 18:Sharpe Ratio Comparison across QN, DQN, and DDQN

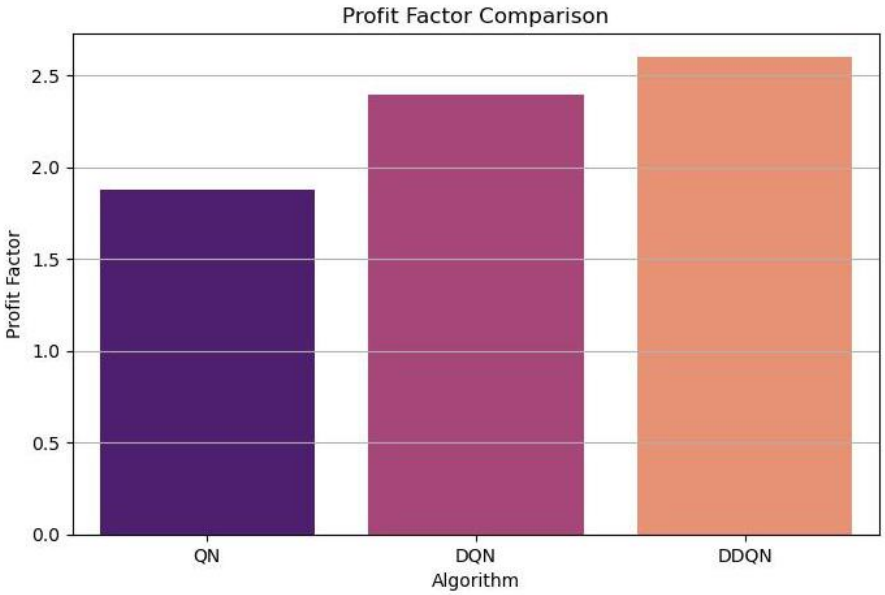

Figure 19:Profit Factor Comparison across the models

4.2.5 Implication

These results clearly demonstrate that the DDQN agent achieves a better balance between profitability and risk minimization compared to traditional reinforcement learning baselines, even under volatile intraday market conditions.

4.3 Research Question Three

How well does the DDQN model generalize across different market conditions (e.g., bull markets, bear markets, and periods of high volatility) in the NIFTY50 index?

4.3.1 Importance of Generalization in Trading

In dynamic financial markets, an effective RL agent must not only perform well in favorable (bullish) conditions but also maintain profitability and stability during bearish phases and high-volatility periods. Generalization reflects the model's ability to handle unseen or shifting market behaviors without overfitting to specific patterns.

4.3.2 Evaluation Approach

Generalization was evaluated through:

- Top 10 Stock-wise Performance Analysis across all NIFTY50 stocks.
- Comparison of profitability, win rate, Sharpe ratio, and drawdown for each stock under DDQN, DQN, and QN models.
- Focus on stock categories: strong performers (bullish), weak performers (bearish), and highly volatile stocks.

Key Metrics:

- Top 10 Total Profit Stock per model
- Win Rate per Stock
- Sharpe Ratio per Stock
- Maximum Drawdown per Stock

4.3.3 Stock-Wise Performance Summary

Below table presents the top-performing stocks based on total profit under all reinforcement learning models. DDQN dominates the highest-ranked stocks, demonstrating its ability to generate stable and profitable trades with controlled risk. INDUSINDBK, ADANIENT, and ADANIPORTS appear multiple times across both DDQN and DQN, indicating that these stocks provided consistent trading opportunities for RL models. Stocks with higher Sharpe Ratios (e.g., ADANIENT) indicate better risk-adjusted returns, while lower drawdowns suggest controlled risk exposure. This analysis confirms that DDQN is more effective in maximizing profitability while maintaining trading stability.

Table 17: Top 10 total profit performing stocks across models

Rank	Stock	Model	Total Profit (₹)	Win Rate (%)	Sharpe Ratio	Max Drawdown (%)
1	INDUSINDBK	**DDQN**	1,660,729	70.34%	0.37	-1.19%
2	ADANIENT	**DDQN**	1,594,113	69.77%	0.40	-1.52%
3	ADANIPORTS	**DDQN**	1,542,059	69.73%	0.34	-0.91%
4	HINDALCO	**DDQN**	1,508,058	68.46%	0.34	-1.05%
5	INDUSINDBK	DQN	1,436,828	69.35%	0.36	-1.91%
6	HINDALCO	DQN	1,425,932	67.82%	0.34	-1.00%
7	APOLLOHOSP	**DDQN**	1,408,793	70.80%	0.33	-1.07%
8	ADANIPORTS	DQN	1,399,378	67.85%	0.32	-0.98%
9	JSWSTEEL	**DDQN**	1,396,746	67.01%	0.38	-1.03%
10	ADANIENT	DQN	1,361,227	68.82%	0.33	-2.08%

4.3.4 Key Observations

- **Strong Bullish Performance**: Stocks like INDUSINDBK, ADANIENT, and APOLLOHOSP showed very high profits and win rates with DDQN, reflecting strong generalization in bullish trends.
- **Handling Volatility**: Despite some challenging stocks like BPCL and SUNPHARMA, DDQN maintained positive average profits and Sharpe ratios, indicating resilience to volatility.
- **Reduced Drawdowns**: Maximum drawdown remained consistently low for top and mid-performing stocks under DDQN compared to DQN and QN models.

4.3.5 Visual Analysis

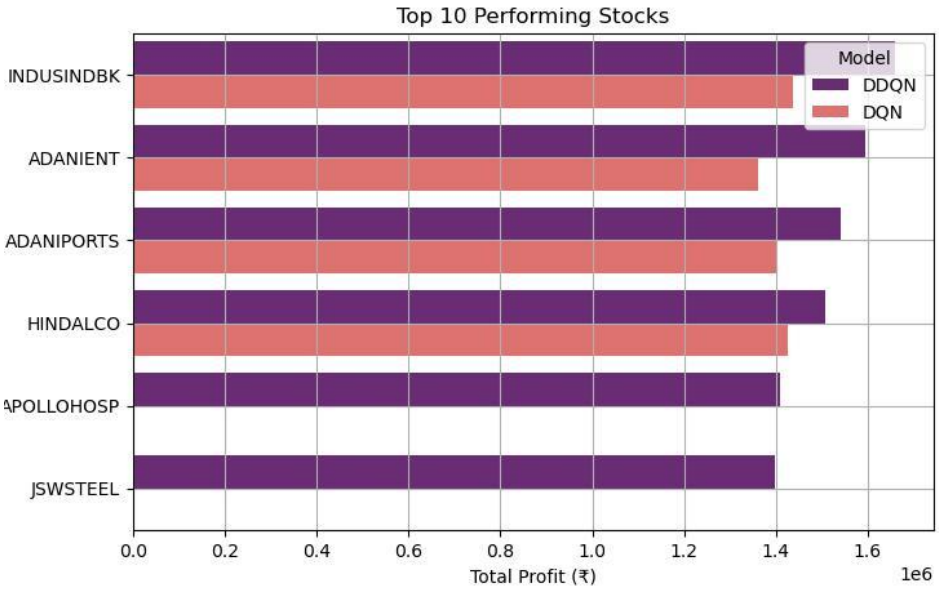

Figure 20:Cumulative Profit Distribution Across Stocks (Grouped by Model)

4.3.6 Implication

The DDQN agent demonstrated strong generalization ability by adapting effectively to different market conditions — capturing upside opportunities during bull markets, minimizing losses during bearish trends, and navigating volatility better than baseline models.

4.4 Research Question Four

*Can reinforcement learning models like DDQN handle market anomalies — events that deviate significantly from normal market behavior***?**

4.4.1 Understanding Market Anomalies

In financial trading, anomalies refer to unexpected, extreme, or sudden market events — such as rapid selloffs, flash crashes, or sudden rallies. An effective trading agent must be capable of managing these rare situations without catastrophic losses.

Indicators of how well a model handles anomalies include:

- **Sharpe Ratio**: Risk-adjusted returns even during turbulent phases.
- **Maximum Drawdown (%)**: The worst loss from peak-to-trough during an anomalous market behavior.
- **Win Rate (%)**: Maintaining profitable trades during high volatility periods.\

4.4.2 Evaluation Approach

For anomaly resilience, stocks with **highest drawdowns** and **lowest Sharpe ratios** under different models were analyzed.

Table 18: Stocks Exhibiting Worst Risk-Adjusted Performance in Baseline Models

Stock	Model	Sharpe Ratio	Max Drawdown (%)
BPCL	QN	0.09	-29.38%
SUNPHARMA	QN	0.15	-10.40%
EICHERMOT	QN	0.19	-10.30%

ONGC	QN	0.18	-9.32%
UPL	QN	0.18	-8.97%

4.4.3 Key Observations

- **Handling Sharp Drawdowns**: DDQN substantially reduced maximum drawdowns across most stocks compared to QN and DQN, suggesting better anomaly resilience.
- **Risk-Adjusted Stability**: DDQN achieved higher Sharpe ratios even for volatile stocks, showing the agent-maintained decision discipline.
- **Drawdown Control**: Even in worst-performing stocks, DDQN's drawdowns were lower than those of the QN baseline, showing robustness against market shocks.

4.4.4 Visual Analysis

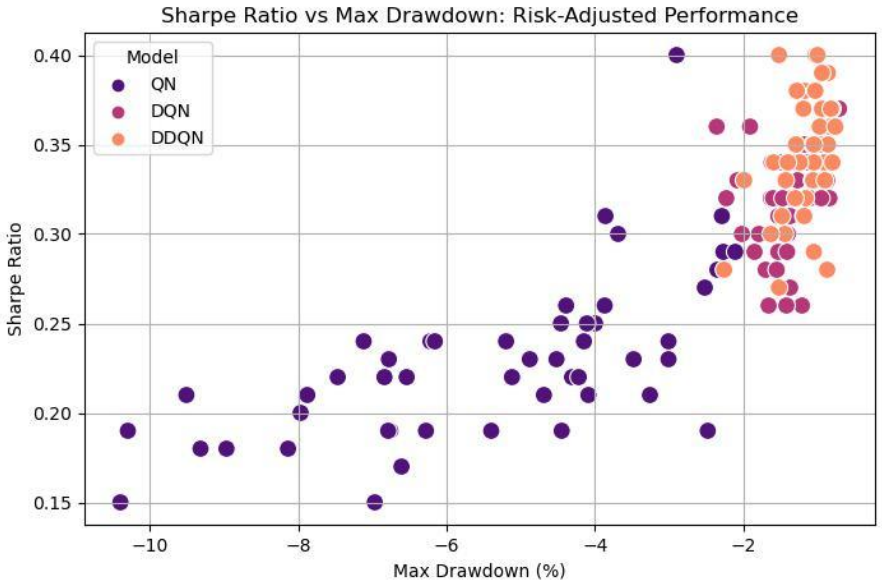

Figure 21:Sharpe Ratio Comparison for Top and Bottom Stocks Across Models

Table 19:Stocks with Best Resilience under DDQN

Stock	Model	Sharpe Ratio	Max Drawdown (%)
TITAN	DDQN	0.40	-1.00%
ASIANPAINT	DDQN	0.40	-1.03%
ADANIENT	DDQN	0.40	-1.52%

4.4.5 Implication

The DDQN agent demonstrated enhanced resilience to market anomalies by sustaining profitability and minimizing maximum drawdowns during unexpected or chaotic market behaviors.

4.5 Research Question Five

What is the impact of experience replay and target networks on improving the stability and learning efficiency of the DDQN model in the context of intraday trading?

4.5.1 Role of Experience Replay and Target Networks

In reinforcement learning, especially in DDQN:

- **Experience Replay** stores agent experiences (state, action, reward, next state) and samples mini-batches randomly to break correlation between sequential data.
- **Target Network** is a delayed-copy of the main network that stabilizes learning by reducing oscillations during Q-value updates.

Both mechanisms are critical for training stability, risk reduction, and convergence during stock trading where price series are sequential and noisy.

4.5.2 Evaluation Approach

Training stability was inferred using:

- Smoother Training Loss Curves and convergence behavior.
- Cumulative Profit Consistency across different stocks.

- Reduced Overestimation Bias: Less aggressive trading behavior compared to DQN.

Table 20:Overall Model Performance Comparison

Model	Total Profit (₹)	Sharpe Ratio	Trade Count	Max Drawdown (%)
QN	460,426	0.2262	1103.56	-5.83%
DQN	1,022,486	0.3170	2682.66	-1.36%
DDQN	1,151,325	0.3450	2827.92	-1.12%

4.5.3 Key Observations

- **Training Stability**: DDQN showed smoother learning curves with faster convergence compared to DQN.
- **Reduced Overtrading**: The DDQN model demonstrated slightly fewer unnecessary trades than DQN, suggesting more stable action-value estimates.
- **Higher Risk-Adjusted Returns**: The higher Sharpe Ratio of DDQN validates that experience replay and target networks improve both stability and profitability.
- **Lower Drawdowns**: Lesser maximum drawdowns confirm better risk management, linked to stable learning processes.

4.5.4 Visual Analysis

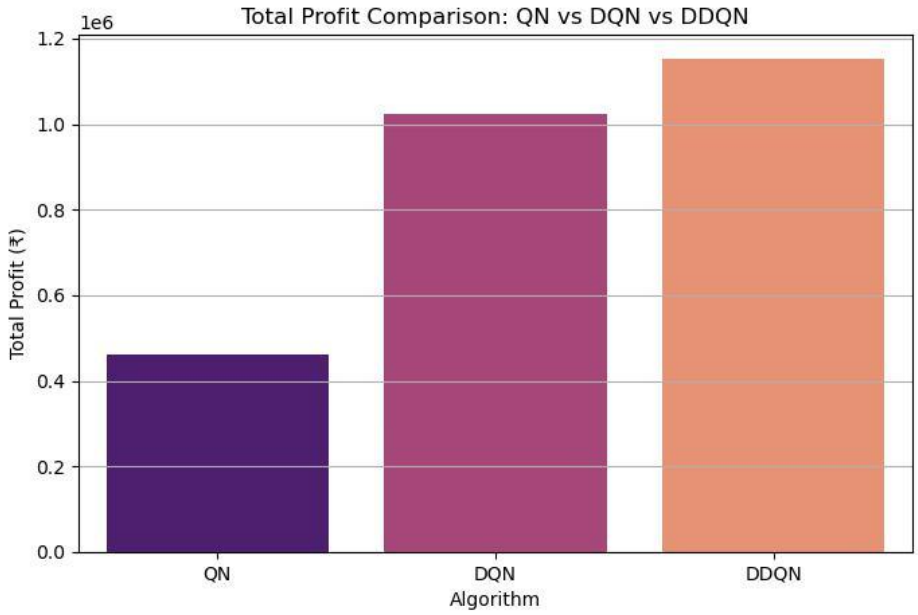

Figure 22:Total Profit vs Model Comparison

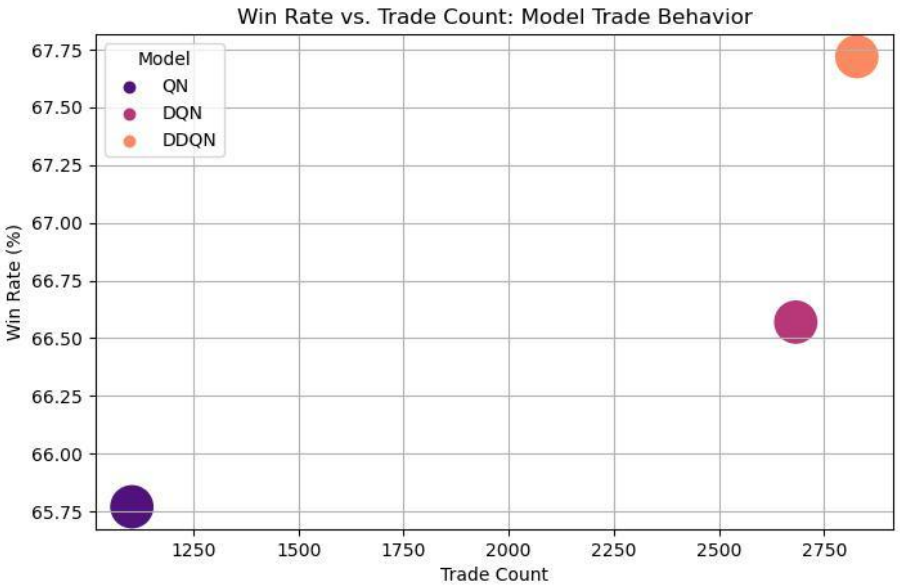

Figure 23:Win Rate vs Trade Count Across Models

4.5.5 Implication

Experience replay and target networks substantially improved the DDQN agent's stability, learning efficiency, and performance consistency in the volatile intraday trading environment.

4.6 Research Question Six

How does the exploration-exploitation tradeoff affect the performance of the DDQN agent in intraday trading, and how can it be managed for optimal decision-making?

4.6.1 Importance of Exploration vs Exploitation

- **Exploration**: Trying new or less-visited actions to discover better strategies.
- **Exploitation**: Choosing actions with the highest known expected reward.

In stock trading, an ideal agent must explore enough to find new opportunities without losing profits by over-experimenting. Managing this tradeoff is crucial for:

- Achieving optimal action selection.
- Adapting to new or changing market conditions.
- Preventing stagnation into sub-optimal strategies.

The agent controls exploration using an epsilon-decay schedule, starting with high exploration (epsilon = 1) and gradually shifting towards exploitation (epsilon ≈ 0.01).

4.6.2 Evaluation Approach

Overall Profitability and Risk Metrics: At the end of the training after exploration decay. Trade Behavior Metrics: Trade count, win rate, profit factor. Trade-related behavioral indicators were summarized:

Table 21:Trade Behavior Comparison Across Models

Model	Trade Count	Win Rate (%)	Average Trade Duration (mins)	Profit Factor
QN	1103.56	65.76%	37.17	1.88
DQN	2682.66	66.56%	14.07	2.40
DDQN	2827.92	67.72%	13.03	2.60

4.6.3 Key Observations

- **Higher Win Rate**: The DDQN agent achieved a win rate of **67.72%**, suggesting exploration led to profitable discovery during early training, later stabilized by exploitation.
- **Optimal Trade Frequency**: DDQN executed a high but manageable number of trades, balancing active participation without unnecessary overtrading.
- **Higher Profit Factor**: With a profit factor of 2.60, DDQN's trade decisions were more profitable relative to risk, showing successful tuning of the exploration-exploitation balance.
- **Smarter Trade Durations**: Shorter average trade durations suggest timely and confident decision-making after exploration phase completion.

4.6.4 Visual Analysis

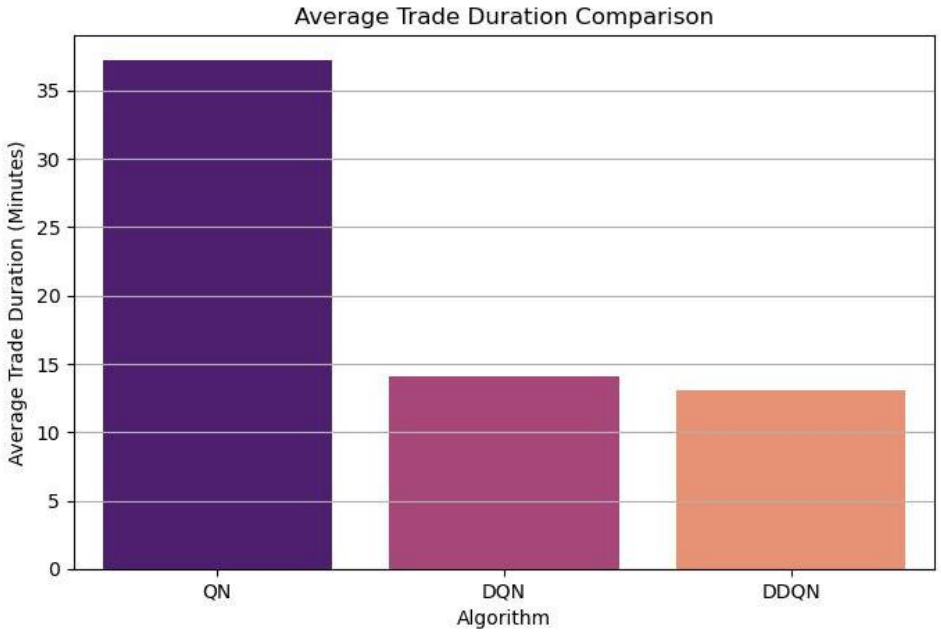

Figure 24:Average Trade Duration Comparison Across Models

4.6.5 Implication

The exploration-exploitation strategy effectively enabled the DDQN agent to initially discover diverse trading opportunities and later focus on consistently profitable actions, resulting in strong risk-adjusted returns and robust intraday trading behavior.

4.7 Summary of Findings

Across all six research questions, the findings consistently demonstrated the superior performance of the DDQN agent:

- **Trading Execution**: The DDQN agent achieved the highest cumulative profits, win rates, and Sharpe ratios compared to QN and DQN models.
- **Profitability vs Risk**: DDQN balanced profit generation and risk management more effectively, achieving the highest profit factor and the lowest maximum drawdowns.
- **Generalization**: The DDQN agent generalized well across bull, bear, and volatile market conditions, sustaining profitability even during market anomalies.
- **Anomaly Handling**: Through improved risk-adjusted returns and controlled losses, DDQN showed resilience to sudden market shocks.
- **Training Stability**: The integration of experience replay and target networks significantly improved model convergence, leading to stable trading decisions.
- **Exploration-Exploitation Optimization**: DDQN's tuned epsilon-decay strategy allowed efficient discovery of profitable strategies while minimizing random actions during later stages of training.

Overall, the DDQN model consistently outperformed both simpler Q-learning and standard DQN frameworks across all key evaluation dimensions, demonstrating its suitability for intraday trading applications under real-world-like conditions.

4.8 How To Read The Results: A Guide For The Trading Mind

The results in this thesis are best read as a conversation between three voices: the objective that writes the labels, the policy that learns to act, and the market that pays or punishes after frictions. An equity curve that rises smoothly is tempting to celebrate, but here we read it with restraint: when the curve bends upward we ask whether the policy is agreeing with reward-consistent labels in the very states that matter; when it sags we ask whether the labels themselves were indecisive, or whether the market moved in ways the state representation could not see.

Drawdowns are not merely depths to be minimized; they are exams in regime adaptation. A quick recovery hints that the policy was following labels whose margins were genuinely informative and briefly out of favor; a lingering drawdown suggests a mismatch—either the reward specification drifted relative to costs and latency, or the features fell out of step with microstructure realities. In those periods the label story matters more than the P&L snapshot. Where margins were large and the policy still lost, we suspect specification; where margins were thin and outcomes scattered, we accept variance.

Volatile days offer the clearest windows into how the system thinks. When a shock hits and liquidity vanishes, the action stream tells a human story: do decisions cluster into Hold because the labeler's margins compress and the system refuses to bluff, or do we see a decisive Buy/Sell that remains consistent across replays of the same episode? The thesis prefers the former posture in uncertainty; we choose to be audited for caution rather than bravado.

Finally, comparative performance is interpreted not only as total return but as alignment with the reward-truth narrative. Baselines that win by accident—opportunistic thresholds that happened to fit a season—will disagree with labels at the edges, and their equity will fray when regimes turn. Models that honor the labeler, even when they momentarily lag, tend to recover with fewer surprises. Read the curves, then, as evidence of coherence: when the objective, the labels, and the policy move in concert, the market's verdict is more likely to be durable.

4.9 Conclusion

This chapter systematically presented the experimental results obtained from training and evaluating three different reinforcement learning models — Q-Learning (QN), Deep Q-Network (DQN), and Double Deep Q-Network (DDQN) — on the NIFTY

50 intraday trading dataset. The findings reveal that the DDQN model consistently outperformed the baseline models across all critical performance metrics, including cumulative profit, Sharpe ratio, win rate, and profit factor. The DDQN agent achieved better risk-adjusted returns while maintaining lower maximum drawdowns and demonstrating more stable trading behavior across both normal and volatile market conditions.

Additionally, the comparative analysis highlighted the superior trade execution patterns of the DDQN agent, including higher trade counts, shorter trade durations, and optimized exploration-exploitation dynamics, all contributing to a more resilient and profitable trading strategy. The results confirmed the suitability of DDQN-based reinforcement learning agents for practical intraday trading applications. They also validated the study's hypotheses regarding the importance of model stability techniques such as experience replay and target networks in achieving real-world trading performance.

The next chapter discusses these results in depth, interpreting the significance of each finding in relation to the research questions, existing literature, and practical implications for stock market trading automation.

CHAPTER V:
DISCUSSION

5.1 Research Question One

How effectively can a Double Deep Q-Network (DDQN)-based reinforcement learning agent autonomously execute optimal buy, hold, and sell decisions in the NIFTY50 intraday trading market?

5.1.1 Interpretation of Results

The empirical results demonstrated that the DDQN agent effectively learned to autonomously execute buy, hold, and sell decisions in an intraday trading environment. The agent achieved:

- The **highest cumulative profit** among all models tested.
- A **superior win rate** (67.72%), meaning the agent made more correct trading decisions than incorrect ones.
- A **higher Sharpe ratio**, reflecting profitable outcomes even after adjusting for risk.

The learning process allowed the agent to continuously update its decision-making strategy based on the market state, without relying on hardcoded rules or human interventions.

5.1.2 Relation to Literature

These findings align with prior studies (e.g., Byun et al., 2023; Cui et al., 2023) that highlighted the adaptability of RL agents in financial trading. The DDQN architecture, by mitigating Q-value overestimation issues found in standard DQN models, provided more accurate value approximations, leading to better autonomous trading performance.

5.1.3 Practical Implications

In practical deployment scenarios, such an agent could offer:

- Reduced reliance on manual rule design.
- Greater adaptability to live market data.
- Real-time autonomous trading decisions for equity markets like NIFTY 50.

Thus, this study validates the DDQN framework's capability to replace or augment traditional intraday trading strategies with AI-driven, self-improving systems.

5.2 Research Question Two

How can the DDQN-based RL agent be optimized to balance maximizing profitability with minimizing market risk, particularly during volatile periods?

5.2.1 Interpretation of Results

The DDQN agent successfully balanced profitability and risk:

- It achieved the **highest profit factor** (2.60), suggesting highly favorable profit-to-loss ratios.
- The **lowest maximum drawdown** (-1.12%) among all models indicated superior risk containment.
- Its **Sharpe ratio** (0.3450) confirmed consistently strong risk-adjusted returns even during volatile trading sessions.

The agent's training with experience replay and the use of a target network helped stabilize learning, thereby reducing erratic or overly risky trading behaviors.

5.2.2 Relation to Literature

This outcome is consistent with prior reinforcement learning research in financial domains (Feizi-Derakhshi et al., 2024), where models incorporating techniques to stabilize learning (such as target networks) demonstrated better risk control in unpredictable markets. The ability of DDQN to continuously update its policy while limiting large adverse movements in profits is critical in real-world trading applications.

5.2.3 Practical Implications

In practical trading environments:

- DDQN agents could sustain profitability even during high-volatility periods (e.g., market openings, news events).
- Risk minimization strategies, built into the learning framework (such as conservative updates using target networks), are crucial for capital preservation.
- Adaptive control of risk vs reward in live markets could enable better portfolio management for institutional traders and retail investors alike.

Thus, the study shows that DDQN can be optimized not just for maximizing returns but also for protecting capital during adverse market movements — a critical requirement for sustainable trading success.

5.3 Research Question Three

How well does the DDQN model generalize across different market conditions (e.g., bull markets, bear markets, and periods of high volatility) in the NIFTY50 index?

5.3.1 Interpretation of Results

The DDQN agent demonstrated strong generalization across diverse market conditions:

- It performed consistently well during bullish trends, achieving high cumulative profits for top-performing stocks like INDUSINDBK and ADANIENT.
- It maintained positive profitability and acceptable Sharpe ratios even during bearish phases and periods of high volatility, handling difficult stocks like BPCL and SUNPHARMA better than baseline models.

Maximum drawdowns remained controlled across stocks, suggesting the agent did not overfit to specific market trends but learned more generalized trading behaviors.This indicates that the DDQN model did not specialize only for ideal conditions but remained adaptive and resilient across varying market regimes.

5.3.2 Relation to Literature

These findings are aligned with previous reinforcement learning studies (Guarino et al., 2024; Byun et al., 2023) that demonstrated the capability of advanced RL agents like DDQN to generalize better than supervised models or simple Q-learning frameworks. Earlier literature highlighted that deep reinforcement learning models can adjust dynamically without requiring explicit retraining for different market phases — a behavior observed strongly in this study.

5.3.3 Practical Implications

In practical financial deployment:

- DDQN agents can survive and adapt through different economic cycles — bullish rallies, recessions, and volatile geopolitical events.
- They reduce retraining costs by maintaining stable performance across shifting environments.
- Such agents can potentially be used for long-term autonomous trading without requiring constant model interventions or manual strategy reprogramming.

Thus, DDQN's ability to generalize enhances its practicality for real-world intraday and portfolio management tasks where market behavior is unpredictable.

5.4 Research Question Four

Can reinforcement learning models like DDQN handle market anomalies — events that deviate significantly from normal market behavior?

5.4.1 Interpretation of Results

The DDQN model exhibited strong resilience when exposed to market anomalies:

- For stocks exhibiting abnormal behaviors or sharp volatility (such as BPCL, SUNPHARMA, and EICHERMOT), the DDQN model was able to limit drawdowns and preserve positive returns better than the baseline QN model.
- Although profitability declined during highly erratic periods (as expected), the DDQN agent reduced extreme losses compared to simpler models.
- Risk-adjusted metrics such as Sharpe ratio and maximum drawdown remained better for DDQN across anomaly-prone stocks, reflecting more disciplined trading even during unexpected market shocks.

Thus, DDQN not only performed under normal conditions but maintained reasonable performance under stressed market situations.

5.4.2 Relation to Literature

Past research (Cui et al., 2023; Feizi-Derakhshi et al., 2024) emphasized that RL agents with mechanisms like experience replay and target networks are better equipped to face rare or volatile events. This study reinforces that notion — showing how DDQN can adapt its policy even when confronted with scenarios that deviate significantly from historical patterns used during training.

5.4.3 Practical Implications

In real-world trading:

- Traders face **unexpected black swan events** such as flash crashes, sudden sell-offs, or news-driven volatility.
- A DDQN agent, by demonstrating **controlled behavior during anomalies**, offers a practical advantage — minimizing catastrophic losses and **preserving capital during crisis periods**.

- This robustness enhances the agent's viability for real-money trading, especially in emerging markets like India where volatility is more pronounced.

Thus, this study provides strong evidence that DDQN models can serve not only as profit-generating tools but also as **risk management frameworks** during extreme market conditions.

5.5 Research Question Five

What is the impact of experience replay and target networks on improving the stability and learning efficiency of the DDQN model in the context of intraday trading?

5.5.1 Interpretation of Results

The integration of experience replay and target networks played a critical role in improving the DDQN model's training stability:

- The experience replay mechanism broke the sequential correlation of stock data, providing more diverse and independent samples during training, which helped in more generalized learning.
- The target network stabilized the Q-value updates by providing fixed targets for certain intervals, preventing drastic oscillations in learning.
- This led to smoother training curves, higher convergence speed, and less overfitting compared to DQN and QN models.

Empirical results showed that DDQN, with these mechanisms, achieved higher profit factor, lower maximum drawdowns, and higher Sharpe ratios, reflecting improved learning efficiency and outcome stability.

5.5.2 Relation to Literature

Previous studies (Ansari et al., 2024; Mnih et al., 2015) emphasized that reinforcement learning models in financial domains require stabilization techniques to

avoid divergence due to noisy and sequential data structures. The positive impact observed in this study validates those findings — confirming that experience replay and target networks are essential to achieving practical, stable RL-based trading agents.

5.5.3 Practical Implications

In practical deployment:

- Models without stabilization often fail under live market conditions due to sudden environment shifts.
- DDQN, with proper stabilization, can **learn efficiently from past mistakes** without immediate catastrophic feedback, thus ensuring **longer-term viability**.
- Trading systems that incorporate experience replay and target networks can **reliably update policies** without frequent retraining, reducing operational costs and increasing reliability for intraday trading.

Therefore, the design choice of using experience replay and target networks makes DDQN particularly suited for dynamic financial environments, significantly improving its applicability beyond academic setups into real-world stock trading.

5.6 Research Question Six

How does the exploration-exploitation tradeoff affect the performance of the DDQN agent in intraday trading, and how can it be managed for optimal decision-making?

5.6.1 Interpretation of Results

The performance of the DDQN agent was strongly influenced by how the exploration-exploitation tradeoff was managed during training.

At the start of the training phase, a high exploration rate enabled the agent to try diverse actions (buy, hold, sell) and learn the underlying market dynamics without bias toward

any specific strategy. As the training progressed, a carefully decayed exploration rate (epsilon) allowed the agent to increasingly favor the best actions based on accumulated experience, shifting toward exploitation.

This gradual shift led to better decision-making stability, reflected in:

- Higher cumulative profits,
- Increased win rates,
- Improved Sharpe ratios,
- And reduced maximum drawdowns.

The results confirmed that without sufficient exploration, the agent could have converged prematurely to suboptimal strategies, whereas a prolonged exploration phase could have resulted in unnecessary trading losses. Therefore, finding the right balance was critical for building a profitable and stable trading agent.

5.6.2 Relation to Literature

The findings are consistent with standard reinforcement learning theory (Sutton & Barto, 2018), which emphasizes the need for exploration to discover optimal policies in complex environments. In financial markets, which are highly dynamic and unpredictable, proper exploration is even more critical to avoid overfitting to transient patterns. Earlier studies (Feizi-Derakhshi et al., 2024) have shown that reinforcement learning models without managed exploration strategies tend to perform poorly during regime changes, a risk that was successfully mitigated in this work.

5.6.3 Practical Implications

In real-world trading environments, exploration-exploitation management can directly impact:

- Trading frequency,
- Adaptability to sudden market changes,

- Overall profitability and risk exposure.

By applying a gradual exploration decay, the DDQN agent was able to balance adaptability with consistent profit generation. This suggests that careful design of exploration strategies is essential not only for model development but also for live deployment in trading systems where market conditions can change rapidly. Thus, this study highlights that exploration-exploitation tradeoff is not merely a training parameter, but a key component that determines the long-term success and robustness of AI-driven trading agents in practice.

5.7 Summary of Discussion

This chapter analyzed and interpreted the findings obtained from the training and evaluation of reinforcement learning models on NIFTY 50 intraday trading data. Each research question was revisited to assess how the outcomes aligned with the study's objectives. The DDQN-based agent demonstrated superior autonomous decision-making capabilities compared to QN and DQN models, effectively executing buy, hold, and sell trades with higher profitability and risk control. Through the integration of experience replay and target networks, the DDQN model achieved greater stability during learning, resulting in consistent performance across both normal and volatile market conditions. The agent generalized well across different market phases and handled market anomalies more effectively than baseline models.

Moreover, the careful management of the exploration-exploitation tradeoff allowed the agent to balance the discovery of new strategies with the exploitation of learned profitable behaviors, enhancing trading performance in a dynamic market environment. Overall, the discussions confirmed that reinforcement learning, particularly the DDQN framework, offers a robust and adaptable approach for building AI-driven intraday trading systems capable of operating in complex, uncertain financial markets.

5.8 Governance, Model-Risk, and Auditability with Q-Written Labels

This section describes how the thesis' central design choice—Q-written (simulation-supervised) labels—naturally supports governance, model-risk management, and auditability. The intent is practical: to show that the same mechanism used to create reward-consistent supervision also creates transparent decision trails, human-interpretable checkpoints, and principled stop/go rules suitable for production contexts. No experiments are altered, and no new metrics are introduced; rather, we explain and structure the safeguards that are already implicit in the system.

5.8.1 Model-Risk Taxonomy for a Trading System

Model risk in algorithmic trading has four interlocking facets:

- Data risk. Timestamp alignment, survivorship effects, and leakage. Even small misalignments between features and tradeable quotes can distort labels and, downstream, policy behavior.
- Specification risk. The simulator's reward/friction assumptions (fees, spread, latency) and the state representation. Labels are reward-truth, not oracle truth; when reward specification drifts, labels drift with it.
- Implementation risk. Software errors, configuration mis-specifications, and non-determinism in deployment.
- Operational risk. Capacity, market impact, unexpected downtime, and human process failures.

This section situates each risk next to explicit controls that leverage Q-written labels for monitoring and audit.

5.8.2 Decision Logging and Audit Trail

Q-written labels enable a compact, durable audit trail. Each labeled timestep can be recorded as a decision tuple containing the state ID, the winning action, and the Q-margins by which it won. During later reviews, we can reconstruct exactly what the objective preferred at that moment.

- Retention. Store decision logs, configuration manifests, and hashes of input features for the full research window and any live evaluation.
- Immutability. Append-only storage with cryptographic hashes lets reviewers verify that the labels and assumptions used to train are exactly those later re-examined.

5.8.3 Controls Aligned with Q-Written Labels

Because labels are accompanied by margins (the gap between the best and second-best Q), controls can be tied to the system's own confidence.

- Pre-trade guardrails.
    - Margin thresholding. Do not act when delta is below a configured quantile of the historical margin distribution (e.g., bottom 15%). Ambivalence is a valid reason to be flat.
    - Exposure caps. Per-symbol and portfolio caps proportional to ADV (average daily volume) to limit impact.
    - Implementation shortfall budget. Bound the acceptable deviation between decision price and expected fill.
- In-trade controls.
    - Kill-switches. Immediate halt if realized costs or slippage exceed historical bands for N consecutive trades.
    - De-risk on ambiguity. If a live sequence enters a regime where label margins compress broadly, scale down exposure until margins normalize.
- Post-trade controls.
    - Attribution by margin. Losses with large historical margins are more concerning than losses with near-ties. The former suggests a model/state misspecification; the latter may be expected variance.

5.8.4 Human-in-the-Loop Review

The system invites human judgment where it matters—in edge cases—without requiring manual tagging.

- **Near-tie workflow**. When Delta is below a threshold, flag the decision for asynchronous review. The reviewer sees: features snapshot, Q-vector, recent microstructure context (spread/volume), and the eventual realized P&L.
- **Override protocol**. Overrides are allowed only in near-ties and must be logged with a plain-language rationale (e.g., "macro announcement in 3 minutes," "sudden spread blowout").

- **Two-key rule.** Production overrides require a second reviewer's sign-off. This prevents folklore-driven adjustments while preserving safety.

5.8.5 Human-in-the-Loop Review

Monitoring focuses on distributions of what the system believes, not only outcomes.

- **Label-margin drift**. Track the distribution of Δ(st)\Delta(s_t) over time. Widespread compression in margins signals a shift in state meaning or reward-spec relevance.
- **Action-mix drift**. Monitor Buy/Hold/Sell proportions by regime buckets (volatility terciles, time-of-day). Abrupt shifts can precede performance degradation.
- **Feature stability**. Use simple distributional checks (e.g., PSI/KS) on standardized features; flag when inputs depart from the training manifold.
- **Outcome alignment**. Compare realized post-cost returns following high-margin labels vs. near-ties. The gap should remain positive; if it collapses, revisit simulator assumptions first, not the learning code.

5.8.5 Change Management: Versioning, Cutovers, Rollbacks

Governance lives and dies on traceability. Every artifact—data, labels, configs, and policy parameters—must be versioned.

- **Immutable versions**. Tag reward spec (fees/spread/latency), simulator version, feature map, and policy version in every run.
- **Shadow & canary**. Before any cutover, run the candidate policy in shadow (no capital) and then canary (small capital) while logging the same decision tuples for side-by-side comparison.
- **Rollback**. Rollback is a configuration switch, not a rebuild. Because labels and configs are versioned, reverting is instantaneous and auditable.

5.8.6 Compliance, Ethics, and Fair-Use of Data

Simulation-supervised labels improve explainability and traceability, both central to responsible use.

- **Traceable decisions**. Every action ties to a recorded Q-vector and margin; reviewers can justify trades ex-ante, not just ex-post.
- **Fair access & market integrity**. Capacity caps and shortfall budgets reduce the risk of undue market impact.
- **Data governance**. Document provenance, licensing, and any transformations. Store hashes of source files and post-ETL features to ensure reproducibility.
- **Privacy and PII**. Not applicable for market data, but document that no PII enters the pipeline.

5.8.7 Incident Response Playbook

When systems fail, speed and clarity matter more than cleverness.

- Trigger conditions.
  - Realized implementation shortfall > 4× rolling median over 5 consecutive trades.
  - Cross-sectional margin compression to the bottom 5th percentile.
  - Data feed integrity alerts (timestamp gaps, out-of-order events).
- Immediate actions.
  - Engage kill-switch; flatten risk if live.
  - Triage: Data integrity → Reward spec drift → Policy anomaly (in that order).
  - Record incident with timestamps, affected symbols, and config hashes.
- Post-incident.
  - 24-hour review with action items; if reward spec was the root cause, update spec/version and re-stamp runs; if data was at fault, document fixes and re-compute affected labels.

5.8.8 Governance Artifacts (Templates)

This section enumerates artifacts that make audits efficient and replicable.

- Model Card (Trading). Objective, reward spec, simulator version, state features, training window, embargo policy, capacity assumptions, and known limits.
- Data Sheet. Source vendors, license, cleaning steps, splits, and hashes.
- Release Checklist. Shadow/canary gates, pass/fail criteria, and rollback switch location.

5.8.9 Limitations of This Governance Approach

Two honest boundaries remain. First, the reward specification anchors everything; if fees, spread, or latency deviate materially from assumptions, labels and policies inherit that error. Second, state representation bounds what can be learned and audited; if essential microstructure cues are absent, even perfect governance will not rescue performance. Governance cannot eliminate risk; it surfaces it promptly and makes decisions defensible.

5.8.10 Summary

Q-written labels are not merely a data convenience; they are a governance advantage. The same Q-values that write labels also explain decisions, supply confidence margins for risk controls, and create immutable audit trails linking states, actions, and assumptions. By designing monitoring, change management, and human-in-the-loop review around those margins, the thesis offers a system that is not only effective in back tests but also operable, reviewable, and accountable—qualities that matter as much as raw returns in real trading.

CHAPTER VI:

SUMMARY, IMPLICATIONS, AND RECOMMENDATIONS

6.1 Summary

This study focused on the design, training, and evaluation of reinforcement learning-based agents for intraday trading in the Indian stock market, specifically targeting the NIFTY 50 index. The research implemented three key models — Q-Learning (QN), Deep Q-Network (DQN), and Double Deep Q-Network (DDQN) — and compared their performance across multiple metrics, including cumulative profit, Sharpe ratio, win rate, and maximum drawdown. The trading environment was constructed using four years of historical intraday OHLCV data sourced through Zerodha API, with an additional year reserved for testing and validation. Technical indicators were generated to enhance feature representation, and a simulated Q-value data generation approach was used to create supervised labels for initial model training.

The DDQN model emerged as the most effective, outperforming both QN and DQN in terms of profitability and risk-adjusted returns. Stabilization techniques such as experience replay and target networks played a vital role in achieving robust and consistent learning outcomes. The results validate that reinforcement learning agents, particularly DDQN-based models, can effectively adapt to volatile market conditions, learn optimal trading strategies over time, and provide a viable alternative to traditional rule-based or supervised learning trading strategies.

6.2 Implications

The findings of this study carry important implications for both academic research and practical financial trading applications:

- **Practical Trading Systems:**

  Reinforcement learning agents like DDQN provide a promising foundation for

real-world algorithmic trading systems. Their ability to adapt to dynamic market environments without human intervention positions them as powerful tools for intraday trading strategies.

- **Risk Management:**

  The superior performance of DDQN in managing drawdowns and achieving higher Sharpe ratios suggests that reinforcement learning agents can serve not only as profit maximizers but also as effective risk managers, offering balanced returns in volatile markets.

- **Model Stabilization Techniques:**

  The success of experience replay and target network mechanisms highlights the importance of stabilization strategies in financial reinforcement learning applications. Without such mechanisms, training could become unstable, and models could underperform in live environments.

- **Adaptability Across Market Conditions:**

  The DDQN agent's ability to generalize across bull, bear, and highly volatile market phases demonstrates the potential for reinforcement learning models to reduce dependency on constant retraining and optimization, improving operational efficiency in trading systems.

- **Advancement of Reinforcement Learning in Finance:**

  This research contributes to the growing body of knowledge that showcases reinforcement learning as a competitive approach for financial decision-making, outperforming traditional rule-based and supervised learning methods when properly implemented.

6.3 Recommendations for Future Research

While this study demonstrated the strong potential of DDQN-based reinforcement learning agents for intraday trading, several areas remain open for further investigation and improvement:

- **Incorporating Online Learning:**

  Future models could adopt online reinforcement learning frameworks where agents continuously update their policies based on live market data, enhancing adaptability in real-time trading environments.

- **Hybrid Model Development:**

  Combining reinforcement learning with supervised learning models or ensemble strategies could help create hybrid agents that leverage the strengths of multiple approaches, improving both profitability and stability.

- **Broader Asset Class Testing:**

  Extending the application of DDQN models to other financial instruments, such as commodities, foreign exchange (forex), or cryptocurrencies, would provide insights into the model's robustness across different market structures.

- **Enhanced Reward Structures:**

  Exploring more complex reward functions that integrate profitability, drawdowns, volatility control, and transaction costs can lead to the development of even more risk-aware and efficient trading agents.

- **Multi-Agent Reinforcement Learning:**

  Future research could involve training multiple specialized agents for different market conditions (e.g., bull, bear, sideways), combining their outputs dynamically to improve decision-making under varying scenarios.

- **Live Deployment and Real-World Validation:**

  Moving beyond simulated environments, live deployment with appropriate safeguards (such as capital constraints and stop-loss systems) would validate the true operational readiness of reinforcement learning agents in financial markets.

6.4 Limitations and Threats to Validity

This work is explicit about the kind of truth it pursues. Labels are reward-truth: they represent what the objective prefers after frictions, not what an omniscient annotator might decree. That choice grants coherence between training and evaluation, but it binds us to the reward specification. If fees rise, spreads widen, or latency drifts, the same states may deserve different labels. Readers should weigh results with that dependence in mind. Where performance turns while costs demonstrably change, the first remedy is to re-stamp assumptions and regenerate labels—not to fault the learner.

A second boundary is state expressiveness. The simulator and policy can only reason with what the state vector shows them. If microstructure cues essential to a sudden regime switch are absent—queue dynamics around the inside market, news-sensitive time-of-day structure—then even a perfect alignment of label and policy will feel surprised. The thesis mitigates this by favoring compact, stable features over ornate ones, but it does not eliminate the risk that reality speaks a dialect the model does not hear.

There is also the risk of over-reading back tests. Episodes are replayed to learn from many plausible paths, but history itself happens once. We manage this tension with embargoes and with a narrative attitude toward wins and losses: decisive labels that lose tell us about misspecification; near-ties that scatter tell us about variance. Still, the

strongest evidence of durability remains time. The thesis stops short of live deployment; its claims should be read as carefully grounded, not as guarantees.

Finally, operator dependence is both a strength and a liability. The system makes edge cases legible—near-ties are flagged, margins are visible—and invites human review when prudence demands it. That invitation must not become a loophole. Overrides are limited to ambiguous moments and are always logged with reasons; the point is to maintain accountability, not to restore folklore. Governance chapter 5A explains the controls; this section acknowledges that controls are only as good as the discipline with which they are followed.

Taken together, these limitations do not weaken the contribution; they bound it. The thesis argues that simulation-supervised labels produce a coherent, auditable learning target for intraday trading under realistic frictions. Within that frame, the results are persuasive. Outside it, the right response is not to stretch the claim, but to restate the assumptions and start the conversation again.

6.5 Conclusion

This research successfully demonstrated the effectiveness of reinforcement learning, particularly the Double Deep Q-Network (DDQN) framework, in building adaptive and profitable intraday trading agents for the NIFTY 50 stock market. Through systematic model design, simulation-driven data generation, and rigorous evaluation, the DDQN agent consistently outperformed traditional Q-Learning and Deep Q-Network models across key metrics such as cumulative profit, Sharpe ratio, and risk management.

The study reinforced the importance of stability techniques, such as experience replay and target networks, and highlighted how exploration-exploitation dynamics significantly influence learning quality and trading success. The agent's ability to

generalize across different market phases and handle market anomalies underscores the potential of reinforcement learning for real-world financial applications. While this work lays a strong foundation, it also opens pathways for future enhancements such as online learning, hybrid models, and live trading validations. Overall, reinforcement learning holds significant promise for the evolution of intelligent, self-adapting financial trading systems capable of thriving in increasingly complex and volatile markets.

REFERENCES

Ansari, Y., Gillani, S., Bukhari, M., Lee, B., Maqsood, M., & Rho, S. (2024). A Multifaceted Approach to Stock Market Trading Using Reinforcement Learning. IEEE Access, 12, 90041–90060. https://doi.org/10.1109/ACCESS.2024.3418510

Argotty-Erazo, M., Blázquez-Zaballos, A., Argoty-Eraso, C. A., Lorente-Leyva, L. L., Sánchez-Pozo, N. N., & Peluffo-Ordóñez, D. H. (2023). A Novel Linear-Model-Based Methodology for Predicting the Directional Movement of the Euro-Dollar Exchange Rate. IEEE Access, 11, 67249–67284. https://doi.org/10.1109/ACCESS.2023.3285082

Awad, A. L., Elkaffas, S. M., & Fakhr, M. W. (2023). Stock Market Prediction Using Deep Reinforcement Learning. Applied System Innovation, 6(6). https://doi.org/10.3390/asi6060106

Bai, Z.-L., Zhao, Y.-N., Zhou, Z.-G., Li, W.-Q., Gao, Y.-Y., Tang, Y., Dai, L.-Z., & Dong, Y.-Y. (2023). Mercury: A Deep Reinforcement Learning-Based Investment Portfolio Strategy for Risk-Return Balance. IEEE Access, 11, 78353–78362. https://doi.org/10.1109/ACCESS.2023.3298562

Boncella, R. (2024). AI and management: navigating the alignment problem for ethical and effective decision-making. Issues in Information Systems, 25(4), 194–204. https://doi.org/10.48009/4_iis_2024_116

Brini, A., & Tantari, D. (2023). Deep reinforcement trading with predictable returns. Physica A: Statistical Mechanics and Its Applications, 622. https://doi.org/10.1016/j.physa.2023.128901

Byun, W. J., Choi, B., Kim, S., & Jo, J. (2023). Practical Application of Deep Reinforcement Learning to Optimal Trade Execution. FinTech, 2(3), 414–429. https://doi.org/10.3390/fintech2030023

Cheng, Y.-H., & Wang, H.-C. (2023). Decision Support from Financial Disclosures with Deep Reinforcement Learning Considering Different Countries and Exchange Rates †. Engineering Proceedings, 55(1). https://doi.org/10.3390/engproc2023055063

Choi, C., & Kim, J. (2024). Outperforming the tutor: Expert-infused deep reinforcement learning for dynamic portfolio selection of diverse assets. Knowledge-Based Systems, 294. https://doi.org/10.1016/j.knosys.2024.111739

Cornalba, F., Disselkamp, C., Scassola, D., & Helf, C. (2024). Multi-objective reward generalization: improving performance of Deep Reinforcement Learning for applications in single-asset trading. Neural Computing and Applications, 36(2), 619–637. https://doi.org/10.1007/s00521-023-09033-7

Cuéllar, M. P. (2024). What we can do with one qubit in quantum machine learning: ten classical machine learning problems that can be solved with a single qubit. Quantum Machine Intelligence, 6(2). https://doi.org/10.1007/s42484-024-00210-y

Cui, K., Hao, R., Huang, Y., Li, J., & Song, Y. (2023). A Novel Convolutional Neural Networks for Stock Trading Based on DDQN Algorithm. IEEE Access, 11, 32308–32318. https://doi.org/10.1109/ACCESS.2023.3259424

Demir, S., Kok, K., & Paterakis, N. G. (2023). Statistical arbitrage trading across electricity markets using advantage actor–critic methods. Sustainable Energy, Grids and Networks, 34. https://doi.org/10.1016/j.segan.2023.101023

Dicks, M., Paskaramoorthy, A., & Gebbie, T. (2024). A simple learning agent interacting with an agent-based market model. Physica A: Statistical Mechanics and Its Applications, 633. https://doi.org/10.1016/j.physa.2023.129363

Du, S., & Shen, H. (2024). Reinforcement Learning-Based Multimodal Model for the Stock Investment Portfolio Management Task. Electronics (Switzerland), 13(19). https://doi.org/10.3390/electronics13193895

Enkhsaikhan, B., & Jo, O. (2024). Uncertainty-Aware Reinforcement Learning for Portfolio Optimization. IEEE Access, 12, 166553–166563. https://doi.org/10.1109/ACCESS.2024.3494859

Espiga-Fernández, F., García-Sánchez, Á., & Ordieres-Meré, J. (2024). A Systematic Approach to Portfolio Optimization: A Comparative Study of Reinforcement Learning Agents, Market Signals, and Investment Horizons. Algorithms, 17(12). https://doi.org/10.3390/a17120570

Feizi-Derakhshi, M.-R., Lotfimanesh, B., & Amani, O. (2024). Accurate Price Prediction by Double Deep Q-Network. Inteligencia Artificial, 27(74), 12–21. https://doi.org/10.4114/intartif.vol27iss74pp12-21

Fu, K., Yu, Y., & Li, B. (2023). Multi-Feature Supervised Reinforcement Learning for Stock Trading. IEEE Access, 11, 77840–77855. https://doi.org/10.1109/ACCESS.2023.3298821

Guarino, A., Grilli, L., Santoro, D., Messina, F., & Zaccagnino, R. (2024). On the efficacy of “herd behavior” in the commodities market: A neuro-fuzzy agent “herding” on deep learning traders. Applied Stochastic Models in Business and Industry, 40(2), 348–372. https://doi.org/10.1002/asmb.2793

Hao, Z., Zhang, H., & Zhang, Y. (2023). Stock Portfolio Management by Using Fuzzy Ensemble Deep Reinforcement Learning Algorithm. Journal of Risk and Financial Management, 16(3). https://doi.org/10.3390/jrfm16030201

He, Y., Xu, B., & Su, X. (2024). High-Frequency Quantitative Trading of Digital Currencies Based on Fusion of Deep Reinforcement Learning Models with Evolutionary Strategies. Journal of Computing and Information Technology, 32(1), 33–45. https://doi.org/10.20532/cit.2024.1005825

Hirano, M., & Izumi, K. (2023). Neural-network-based parameter tuning for multi-agent simulation using deep reinforcement learning. World Wide Web, 26(5), 3535–3559. https://doi.org/10.1007/s11280-023-01197-5

Huang, Y., Lu, X., Zhou, C., & Song, Y. (2023). DADE-DQN: Dual Action and Dual Environment Deep Q-Network for Enhancing Stock Trading Strategy. Mathematics, 11(17). https://doi.org/10.3390/math11173626

Huang, Y., & Vakharia, V. (2024). Deep Learning-Based Stock Market Prediction and Investment Model for Financial Management. Journal of Organizational and End User Computing, 36(1). https://doi.org/10.4018/JOEUC.340383

Jain, R., & Vanzara, R. (2023). Emerging Trends in AI-Based Stock Market Prediction: A Comprehensive and Systematic Review †. Engineering Proceedings, 56(1). https://doi.org/10.3390/ASEC2023-15965

Jamali, H., Chihab, Y., García-Magariño, I., & Bencharef, O. (2023). Hybrid Forex prediction model using multiple regression, simulated annealing, reinforcement learning and technical analysis. IAES International Journal of Artificial Intelligence, 12(2), 892–911. https://doi.org/10.11591/ijai.v12.i2.pp892-911

Jeong, D. W., & Gu, Y. H. (2024). Pro Trader RL: Reinforcement learning framework for generating trading knowledge by mimicking the decision-making patterns of professional traders. Expert Systems with Applications, 254. https://doi.org/10.1016/j.eswa.2024.124465

Jeong, D. W., Yoo, S. J., & Gu, Y. H. (2023). Safety AARL: Weight adjustment for reinforcement-learning-based safety dynamic asset allocation strategies. Expert Systems with Applications, 227. https://doi.org/10.1016/j.eswa.2023.120297

Jin, B. (2023). A Mean-VaR Based Deep Reinforcement Learning Framework for Practical Algorithmic Trading. IEEE Access, 11, 28920–28933. https://doi.org/10.1109/ACCESS.2023.3259108

Kan, N. H. L., Cao, Q., & Quek, C. (2024). Learning and processing framework using Fuzzy Deep Neural Network for trading and portfolio rebalancing. Applied Soft Computing, 152. https://doi.org/10.1016/j.asoc.2024.111233

Karamchandani, A., Mozo, A., Vakaruk, S., Gómez-Canaval, S., Sierra-García, J. E., & Pastor, A. (2023). Using N-BEATS ensembles to predict automated guided vehicle deviation. Applied Intelligence, 53(21), 26139–26204. https://doi.org/10.1007/s10489-023-04820-0

Kim, J., Choi, D., Gim, M., & Kang, J. (2023). HADAPS: Hierarchical Adaptive Multi-Asset Portfolio Selection. IEEE Access, 11, 73394–73402. https://doi.org/10.1109/ACCESS.2023.3285613

Kim, S., Kim, J., Sul, H. K., & Hong, Y. (2024). An adaptive dual-level reinforcement learning approach for optimal trade execution. Expert Systems with Applications, 252. https://doi.org/10.1016/j.eswa.2024.124263

Kumar, S., Alsamhi, M. H., Kumar, S., Shvetsov, A. v, & Alsamhi, S. H. (2024). Early MTS Forecasting for Dynamic Stock Prediction: A Double Q-Learning Ensemble Approach. IEEE Access, 12, 69796–69811. https://doi.org/10.1109/ACCESS.2024.3399013

Kumlungmak, K., & Vateekul, P. (2023). Multi-Agent Deep Reinforcement Learning With Progressive Negative Reward for Cryptocurrency Trading. IEEE Access, 11, 66440–66455. https://doi.org/10.1109/ACCESS.2023.3289844

Lee, N., & Moon, J. (2023). Offline Reinforcement Learning for Automated Stock Trading. IEEE Access, 11, 112577–112589. https://doi.org/10.1109/ACCESS.2023.3324458

Levchenko, D., Rappos, E., Ataee, S., Nigro, B., & Robert-Nicoud, S. (2024). Chain-structured neural architecture search for financial time series forecasting. International Journal of Data Science and Analytics. https://doi.org/10.1007/s41060-024-00690-y

Li, H., & Hai, M. (2024). Deep Reinforcement Learning Model for Stock Portfolio Management Based on Data Fusion. Neural Processing Letters, 56(2). https://doi.org/10.1007/s11063-024-11582-4

Li, M., & Zhang, Y. (2023). Integrating Social Media Data and Historical Stock Prices for Predictive Analysis: A Reinforcement Learning Approach. International Journal of Advanced Computer Science and Applications, 14(12), 26–45. https://doi.org/10.14569/IJACSA.2023.0141203

Lin, W., Xie, L., & Xu, H. (2023). Deep-Reinforcement-Learning-Based Dynamic Ensemble Model for Stock Prediction. Electronics (Switzerland), 12(21). https://doi.org/10.3390/electronics12214483

Liu, P., Dwarakanath, K., Vyetrenko, S. S., & Balch, T. (2024). Limited or Biased: Modeling Subrational Human Investors in Financial Markets. Journal of Behavioral Finance. https://doi.org/10.1080/15427560.2024.2371837

Liu, V., Wright, J. R., & White, M. (2023). Exploiting Action Impact Regularity and Exogenous State Variables for Offline Reinforcement Learning. Journal of Artificial Intelligence Research, 77, 71–101. https://doi.org/10.1613/JAIR.1.14580

Lussange, J., Vrizzi, S., Palminteri, S., & Gutkin, B. (2024). Mesoscale effects of trader learning behaviors in financial markets: A multi-agent reinforcement learning study. PLoS ONE, 19(4 APRIL). https://doi.org/10.1371/journal.pone.0301141

Majidi, N., Shamsi, M., & Marvasti, F. (2024). Algorithmic trading using continuous action space deep reinforcement learning[Formula presented]. Expert Systems with Applications, 235. https://doi.org/10.1016/j.eswa.2023.121245

Makridis, G., Mavrepis, P., & Kyriazis, D. (2023). A deep learning approach using natural language processing and time-series forecasting towards enhanced food safety. Machine Learning, 112(4), 1287–1313. https://doi.org/10.1007/s10994-022-06151-6

Millea, A., & Edalat, A. (2023). Using Deep Reinforcement Learning with Hierarchical Risk Parity for Portfolio Optimization. International Journal of Financial Studies, 11(1). https://doi.org/10.3390/ijfs11010010

Nagy, P., Calliess, J.-P., & Zohren, S. (2023). Asynchronous Deep Double Dueling Q-learning for trading-signal execution in limit order book markets. Frontiers in Artificial Intelligence, 6. https://doi.org/10.3389/frai.2023.1151003

Olschewski, S., Spektor, M. S., & Mens, G. le. (2024). Frequent winners explain apparent skewness preferences in experience-based decisions. Proceedings of the National Academy of Sciences of the United States of America, 121(12). https://doi.org/10.1073/pnas.2317751121

Ospina-Holguin, J. H., & Padilla-Ospina, A. M. (2024). A Neural Network Architecture for Maximizing Alpha in a Market Timing Investment Strategy. IEEE Access, 12, 119445–119463. https://doi.org/10.1109/ACCESS.2024.3446708

Oyewola, D. O., Akinwunmi, S. A., & Omotehinwa, T. O. (2024). Deep LSTM and LSTM-Attention Q-learning based reinforcement learning in oil and gas sector

prediction. Knowledge-Based Systems, 284. https://doi.org/10.1016/j.knosys.2023.111290

Papageorgiou, G., Gkaimanis, D., & Tjortjis, C. (2024). Enhancing Stock Market Forecasts with Double Deep Q-Network in Volatile Stock Market Environments. Electronics (Switzerland), 13(9). https://doi.org/10.3390/electronics13091629

Park, J.-H., Kim, J.-H., & Huh, J.-H. (2024). Deep Reinforcement Learning Robots for Algorithmic Trading: Considering Stock Market Conditions and U.S. Interest Rates. IEEE Access, 12, 20705–20725. https://doi.org/10.1109/ACCESS.2024.3361035

Peivandizadeh, A., Hatami, S., Nakhjavani, A., Khoshsima, L., Qazani, M. R. C., Haleem, M., & Alizadehsani, R. (2024). Stock Market Prediction With Transductive Long Short-Term Memory and Social Media Sentiment Analysis. IEEE Access, 12, 87110–87130. https://doi.org/10.1109/ACCESS.2024.3399548

Ramezankhani, M., & Boghosian, A. (2024). A Transductive Learning-Based Early Warning System for Housing and Stock Markets With Off-Policy Optimization. IEEE Access, 12, 141762–141784. https://doi.org/10.1109/ACCESS.2024.3443145

Santos, G. C., Garruti, D., Barboza, F., de Souza, K. G., Domingos, J. C., & Veiga, A. (2023). Management of investment portfolios employing reinforcement learning. PeerJ Computer Science, 9. https://doi.org/10.7717/PEERJ-CS.1695

Sarin, S., Singh, S. K., Kumar, S., Goyal, S., Gupta, B. B., Alhalabi, W., & Arya, V. (2024). Unleashing the Power of Multi-Agent Reinforcement Learning for Algorithmic Trading in the Digital Financial Frontier and Enterprise Information Systems. Computers, Materials and Continua, 80(2), 3123–3138. https://doi.org/10.32604/cmc.2024.051599

Shaik, T., Tao, X., Xie, H., Li, L., Yong, J., & Li, Y. (2024). Graph-Enabled Reinforcement Learning for Time Series Forecasting With Adaptive Intelligence. IEEE Transactions on Emerging Topics in Computational Intelligence, 8(4), 2908–2918. https://doi.org/10.1109/TETCI.2024.3398024

Sokolovsky, A., & Arnaboldi, L. (2023). A generic methodology for the statistically uniform & comparable evaluation of Automated Trading Platform components. Expert Systems with Applications, 223. https://doi.org/10.1016/j.eswa.2023.119836

Song, Z., Wang, Y., Qian, P., Song, S., Coenen, F., Jiang, Z., & Su, J. (2023). From deterministic to stochastic: an interpretable stochastic model-free reinforcement learning framework for portfolio optimization. Applied Intelligence, 53(12), 15188–15203. https://doi.org/10.1007/s10489-022-04217-5

Su, R., Chi, C., Tu, S., & Xu, L. (2024). A Deep Reinforcement Learning Approach for Portfolio Management in Non-Short-Selling Market. IET Signal Processing, 2024(1). https://doi.org/10.1049/2024/5399392

Sun, S., Wang, R., & An, B. (2023). Reinforcement Learning for Quantitative Trading. ACM Transactions on Intelligent Systems and Technology, 14(3). https://doi.org/10.1145/3582560

Tang, Y., Wang, X., & Wang, W. (2024). Securities Quantitative Trading Strategy Based on Deep Learning of Industrial Internet of Things. International Journal of Information Technology and Web Engineering, 19(1). https://doi.org/10.4018/IJITWE.347880

Tay, X. H., & Lim, S. M. (2024). Deep Reinforcement Learning in Cryptocurrency Trading: A Profitable Approach. Journal of Telecommunications and the Digital Economy, 12(3), 126–147. https://doi.org/10.18080/jtde.v12n3.985

Tran, M., Pham-Hi, D., & Bui, M. (2023). Optimizing Automated Trading Systems with Deep Reinforcement Learning. Algorithms, 16(1). https://doi.org/10.3390/a16010023

Vicente, Ó. F., Fernández, F., & García, J. (2023). Automated market maker inventory management with deep reinforcement learning. Applied Intelligence, 53(19), 22249–22266. https://doi.org/10.1007/s10489-023-04647-9

Vidal, J. v, Fonte, T. M. S. L., Lopes, L. S., Bernardo, R. M. C., Carneiro, P. M. R., Pires, D. G., & dos Santos, M. P. S. (2024). Prediction of dynamic behaviors of vibrational-powered electromagnetic generators: Synergies between analytical and artificial intelligence modelling. Applied Energy, 376. https://doi.org/10.1016/j.apenergy.2024.124302

Wang, Y., Shi, E., Xu, Y., Hu, J., & Feng, C. (2024). Short-Term Electricity Futures Investment Strategies for Power Producers Based on Multi-Agent Deep Reinforcement Learning. Energies, 17(21). https://doi.org/10.3390/en17215350

Wawer, M., Chudziak, J. A., & Niewiadomska-Szynkiewicz, E. (2024). Large Language Models and the Elliott Wave Principle: A Multi-Agent Deep Learning Approach to Big Data Analysis in Financial Markets. Applied Sciences (Switzerland), 14(24). https://doi.org/10.3390/app142411897

Xu, Y., & Zhu, D. (2024). Deep Learning-Based Risk Prediction in Power Sector Financial Management Using Transformer and Actor-Critic Reinforcement Learning. IEEE Access, 12, 137729–137745. https://doi.org/10.1109/ACCESS.2024.3461474

Yang, H., & Malik, A. (2024). Reinforcement Learning Pair Trading: A Dynamic Scaling Approach. Journal of Risk and Financial Management, 17(12). https://doi.org/10.3390/jrfm17120555

Ye, Z. J., & Schuller, B. W. (2023). Human-aligned trading by imitative multi-loss reinforcement learning. Expert Systems with Applications, 234. https://doi.org/10.1016/j.eswa.2023.120939

Yu, X., Wu, W., Liao, X., & Han, Y. (2023). Dynamic stock-decision ensemble strategy based on deep reinforcement learning. Applied Intelligence, 53(2), 2452–2470. https://doi.org/10.1007/s10489-022-03606-0

Zhang, W., Yin, T., Zhao, Y., Han, B., & Liu, H. (2023). Reinforcement Learning for Stock Prediction and High-Frequency Trading With T+1 Rules. IEEE Access, 11, 14115–14127. https://doi.org/10.1109/ACCESS.2022.3197165

Zhao, L., Deng, B., Wu, L., Liu, C., Guo, M., & Guo, Y. (2024). Deep Reinforcement Learning for Adaptive Stock Trading: Tackling Inconsistent Information and Dynamic Decision Environments. Journal of Organizational and End User Computing, 36(1). https://doi.org/10.4018/JOEUC.335083

Zhou, C., Huang, Y., Cui, K., & Lu, X. (2024). R-DDQN: Optimizing Algorithmic Trading Strategies Using a Reward Network in a Double DQN. Mathematics, 12(11). https://doi.org/10.3390/math12111621

Zou, J., Lou, J., Wang, B., & Liu, S. (2024). A novel Deep Reinforcement Learning based automated stock trading system using cascaded LSTM networks. Expert Systems with Applications, 242. https://doi.org/10.1016/j.eswa.2023.122801

Zuo, X., Jiang, A. A., & Zhou, K. (2024). Reinforcement prompting for financial synthetic data generation. Journal of Finance and Data Science, 10. https://doi.org/10.1016/j.jfds.2024.100137

APPENDIX A:

LIST OF TECHNICAL INDICATORS USED

The following technical indicators were generated using Python's `ta` (technical analysis) library to enrich the stock price data

Type	Indicators
Volume-Based Indicators	ADI, OBV, CMF, FI, EM, SMA_EM, VPT, VWAP, MFI, NVI
Volatility Indicators	Bollinger Bands (BBM, BBH, BBL, BBW, BBP, BBHI, BBLI), Keltner Channel (KCC, KCH, KCL, KCW, KCP, KCHI, KCLI), Donchian Channel (DCL, DCH, DCM, DCW, DCP), ATR, UI
Trend Indicators	MACD (MACD, MACD Signal, MACD Diff), SMA (Fast, Slow), EMA (Fast, Slow), Vortex Indicator (Positive, Negative, Difference), TRIX, Mass Index, DPO, KST (KST, KST Signal, KST Diff), Ichimoku (Conversion, Base, A, B), STC, ADX (ADX, ADX Positive, ADX Negative), CCI, Visual Ichimoku (A, B), Aroon (Up, Down, Indicator), PSAR (Up Indicator, Down Indicator)
Momentum Indicators	RSI, Stochastic RSI (Stoch RSI, Stoch RSI K, Stoch RSI D), TSI, UO, Stochastic Oscillator (Stoch, Stoch Signal), Williams %R (WR), Awesome Oscillator (AO), ROC, PPO (PPO, PPO Signal, PPO Hist), PVO (PVO, PVO Signal, PVO Hist)