INSTITUTO
UNIVERSITÁRIO
DE LISBOA

Vision-Based Reinforcement Learning: The Role of Transfer Learning and Action Space Design in Symmetric Reward Financial Markets

Miguel Gonçalves Teodoro

Master in Computer Engineering

Supervisor:
PhD Luís Miguel Martins Nunes, Associate Professor,
Iscte – Instituto Universitário de Lisboa

Supervisor:
PhD Ana Maria Carvalho de Almeida, Associate Professor (with Aggregation),
Iscte – Instituto Universitário de Lisboa

September, 2025

TECNOLOGIAS
E ARQUITETURA

Vision-Based Reinforcement Learning: The Role of Transfer Learning and Action Space Design in Symmetric Reward Financial Markets

Miguel Gonçalves Teodoro

Master in Computer Engineering

Supervisor:
PhD Luís Miguel Martins Nunes, Associate Professor,
Iscte – Instituto Universitário de Lisboa

Supervisor:
PhD Ana Maria Carvalho de Almeida, Associate Professor (with Aggregation),
Iscte – Instituto Universitário de Lisboa

September, 2025

This work was partially supported by Fundação para a Ciência e a Tecnologia, I.P. (FCT) [ISTAR Projects: UIDB/04466/2020 and UIDP/04466/2020].

Acknowledgment

I would like to express my deepest gratitude to my supervisors, Professor Luís Nunes and Professor Ana Maria de Almeida, for their support, guidance and feedback.

I would like to thank my family and friends. Their support continuously helps me achieve my goals.

Finally, a special thanks goes to Beatriz, for your patience and kindness.

Resumo

Esta dissertação investiga se um sistema de *reinforcement learning* (RL) baseado em informação visual pode negociar de forma lucrativa num mercado sem *drift*. A *pipeline* proposta converte janelas de preços EUR/USD em gráficos, extrai variáveis com uma *convolutional neural network* (CNN) e fornece-as a um agente *soft actor-critic* que gera alocações de portefólio num espaço de ações contínuo. O modelo de RL é treinado com cerca de 20 anos de dados. Comparam-se diferentes variações do sistema proposto: pré-treino da CNN, congelamento do extrator de variáveis durante o treino do RL e equilibrio de classes no conjunto de treino do RL. Os resultados demonstram que o *transfer learning* facilita a convergência do RL. A maioria dos agentes aprende políticas com valor esperado positivo durante o treino. A variante com o extrator de variáveis congelado apresenta fraco desempenho em teste, enquanto o agente com equilibrio de classes terminou o período de teste com um ganho de 4%, evidenciando o potencial deste sistema. Contudo, este agente revela limitações no aproveitamento do espaço de ações contínuo, com alocações negativas de maior dimensão a apresentarem melhor desempenho nas caudas da distribuição dos retornos, mas sem estender este padrão a alocações positivas de maior dimensão. Este trabalho contribui para a literatura ao demonstrar como a representação visual, os espaços de ações contínuos e o *transfer learning* influenciam o desempenho de sistemas de negociação baseados em RL em ambientes de recompensas simétricas.

Palavras-chaves: aprendizagem de reforço, redes neuronais convolucionais, aprendizagem por tranferência, ambiente de recompensas simétricas

Abstract

This dissertation investigates whether a vision-driven reinforcement learning (RL) system can trade a non-drift market profitably. The proposed pipeline converts EUR/USD price windows into chart images, extracts features with a convolutional neural network (CNN), and feeds them to a soft actor-critic agent that outputs portfolio allocations in a continuous action space. The RL model is trained on roughly 20 years of data. We compare different variations of the proposed system: pre-training the CNN, freezing the feature extractor during RL training and class-balancing the RL training set. Results demonstrate that transfer learning aids RL convergence. Most agents learn positive expected value policies during training. The frozen encoder variant performs poorly in testing, while the class-balanced agent ended the test period with a 4% gain, showing the system's potential. However, this agent demonstrates limited upside from its continuous action space, with larger negative bets performing better in the returns' distribution tails, but not extending this pattern to large positive bets. This work contributes to the literature by demonstrating how visual representation, continuous actions spaces and transfer learning impact RL-based trading systems performance in symmetric reward environments.

Keywords: reinforcement learning, convolutional neural networks, transfer learning, symmetric reward environment

Contents

Acknowledgment i

Resumo iii

Abstract v

List of Figures ix

List of Tables xi

List of Acronyms xiii

Chapter 1. Introduction 1
1.1. Problem Description 1
1.2. Contributions 2
1.3. Research Questions 3

Chapter 2. State of the Art 5

Chapter 3. Context 9
3.1. Financial Markets, Foreign Exchange and the Trading Game Setting 9
3.1.1. Foreign Exchange Market 9
3.1.2. Trading Game Setting 10
3.2. Transfer Learning 12
3.3. Computer Vision and Reinforcement Learning 12
3.3.1. Machine Learning and Artificial Intelligence 12
3.3.2. Computer Vision and Convolutional Neural Networks 13
3.3.3. Reinforcement Learning 14
3.3.4. Soft Actor-Critic Algorithm 15
3.3.5. Integration of Computer Vision and Reinforcement Learning 16

Chapter 4. Methodology 19
4.1. Game Setup 19
4.2. Game Mechanics 19
4.2.1. Action Space (A) 20
4.2.2. State Space (S) 20
4.2.3. Reward Function (R) 20
4.3. System Setup 20
4.4. Data Collection and Preparation 22

4.4.1. Data Requirements 22
4.4.2. Reliability Considerations 23
4.4.3. Data Transformation Process 23
4.5. Computer Vision Module 25
4.6. Convolutional Neural Network Training 27
4.7. Reinforcement Learning Agent 28

Chapter 5. Results 31
5.1. Reinforcement Learning Training Metrics 31
5.1.1. CNN Pre-training + RL Fine-Tuning (Baseline) 32
5.1.2. RL Training (no Pre-training) 33
5.1.3. CNN Pre-training + RL with Frozen Encoder 34
5.1.4. CNN Pre-training + RL with Balanced Training Dataset 34
5.2. System Performance 35
5.2.1. RL Training (no Pre-training) 37
5.2.2. CNN Pre-training + RL with Frozen Encoder 37
5.2.3. CNN Pre-training + RL Fine-Tuning (Baseline) 37
5.2.4. CNN Pre-training + RL with Balanced Training Dataset 39

Chapter 6. Discussion 45

Chapter 7. Conclusions 49

Bibliography 51

List of Figures

2.1 Cui et al. [6] trading agent system. 7

3.1 EUR/USD price since inception in 1999. 11
3.2 S&P 500 price since 1999. 11

4.1 System setup diagram. 21
4.2 Example chart image generated. 24
4.3 CNN architecture. 26
4.4 CNN training metrics over 50 epochs. 28

5.1 Episode reward mean evolution in training - CNN Pre-training + RL Fine-Tuning (Baseline). 32
5.2 Critic loss evolution in training - CNN Pre-training + RL Fine-Tuning (Baseline). 33
5.3 Actor loss evolution in training - CNN Pre-training + RL Fine-Tuning (Baseline). 33
5.4 Entropy coefficient evolution in training - CNN Pre-training + RL Fine-Tuning (Baseline). 34
5.5 Entropy coefficient loss evolution in training - CNN Pre-training + RL Fine-Tuning (Baseline). 34
5.6 Episode reward mean evolution in training - RL Training (no Pre-training). 35
5.7 Episode reward mean evolution in training - CNN Pre-training + RL with Frozen Encoder. 35
5.8 Episode reward mean evolution in training - CNN Pre-training + RL with Balanced Training Dataset. 36
5.9 EUR/USD price evolution in test set. 36
5.10 Distribution of actions taken - CNN Pre-training + RL with Frozen Encoder. 37
5.11 Cumulative portfolio return - CNN Pre-training + RL with Frozen Encoder. 38
5.12 Distribution of actions taken - CNN Pre-training + RL Fine-Tuning (Baseline). 38
5.13 Cumulative portfolio return - CNN Pre-training + RL Fine-Tuning (Baseline). 39
5.14 Confusion Matrix of actions and returns - CNN Pre-training + RL Fine-Tuning (Baseline). The label '>0' represents the positive class and '<0' the negative class. 39
5.15 Distribution of actions taken - CNN Pre-training + RL with Balanced Training Dataset. 40

5.16 Cumulative portfolio return - CNN Pre-training + RL with Balanced Training Dataset. The label '>0' represents the positive class and '<0' the negative class. 40
5.17 Confusion Matrix of actions and returns - CNN Pre-training + RL with Balanced Training Dataset. 41
5.18 Detailed Confusion Matrix of actions and returns - CNN Pre-training + RL with Balanced Training Dataset. 43
5.19 Bar Plot and Kernel Density Estimate Plot - Negative Allocations - CNN Pre-training + RL with Balanced Training Dataset. 44
5.20 Bar Plot and Kernel Density Estimate Plot - Positive Allocations - CNN Pre-training + RL with Balanced Training Dataset. 44

List of Tables

1 Detailed distribution of returns - CNN Pre-training + RL with Balanced Training Dataset. 42

2 Detailed distribution of actions - CNN Pre-training + RL with Balanced Training Dataset. 42

List of Acronyms

AI	Artificial Intelligence
CNN	Convolutional Neural Network
CV	Computer Vision
DDPG	Deep Deterministic Policy Gradient
DDQN	Double Deep Q-Network
DQN	Deep Q-Network
FX	Foreign Exchange
GAN	Generative Adversarial Network
KDE	Kernel Density Estimate
LSTM	Long Short-Term Memory
MDP	Markov Decision Process
ML	Machine Learning
OTC	Over-the-Counter
PnL	Profit and Loss
PPO	Proximal Policy Optimization
RL	Reinforcement Learning
RNN	Recurrent Neural Network
SAC	Soft Actor-Critic

CHAPTER 1

Introduction

1.1. Problem Description

The problem of trading in any market, where decisions revolve around buying and selling a given asset, is equivalent to a complex, high-stakes decision-making game. In this game, each choice influences the potential for future gains or losses, creating a dynamic and sequential challenge that is well-suited to the principles of Reinforcement Learning (RL). RL frameworks are specifically designed for learning optimal decision-making strategies through interaction with an environment. An agent improves over time by receiving feedback in the form of rewards or penalties for each action, gradually refining its strategy to maximize a cumulative reward [1]. This work extends the current body of research applying RL techniques to financial markets environments.

The FX market, the global marketplace for trading national currencies, is a highly dynamic and non-stationary environment [2]. Its inherent unpredictability makes it difficult for any agent to generalize effectively. Patterns and strategies learned in one set of conditions may not apply in others, requiring continuous adaptation. Furthermore, the noisy nature of financial data [3] makes price movements reflect both meaningful trends and random fluctuations, which can mislead agents into finding patterns where none exist. By contrast, many RL settings deal with relatively stable dynamics which make credit assignment and evaluation more straightforward [4]. This presents a challenge for applying RL to financial markets.

Applying Computer Vision (CV) techniques to financial charts makes it possible to extract features from visual representations used by many practitioners [5]. Processing such charts with CV can systematize feature extraction from graphical representations of prices.

RL's feedback mechanism directs agents to develop strategies that maximize long-term rewards, therefore it is intrinsically suited to sequential decision-making. Trading is a chain of interdependent actions whose effects accumulate, so optimizing for the long-run is essential for performance.

Successfully automating decisions in the FX market promises clear benefits. Automating pattern recognition can provide a consistent, data-driven tool that reduces reliance on subjective human interpretation. Once effectively trained, an RL agent can operate continuously and objectively without the cognitive biases, fatigue or emotional responses that affect humans, offering the prospect of more stable and scalable profitability over time.

The recent expansion of RL research in finance has shown that automated profitable trading agents are feasible. Cui et al. [6] work processes price-chart images, extracts features and passes them to an RL agent that issues discrete buy or sell actions. This research relies on value-based RL methods, which are suited to discrete action spaces, and it is evaluated in the equity market, as most of ML research is, which provides an accessible but narrow test basis.

Important gaps remain in the current literature. First, most studies restrict the agent's action space to a binary buy/sell decision, whereas a realistic scenario should reflect principles of bankroll management and position sizing, essential to reflect the complete action space of a trader. This omission blurs the difference between choosing a direction and choosing how much risk to take. Second, current research on similar systems is restricted to value-based RL methods, which are suited to discrete action spaces, leaving gaps in the exploration of different approaches and algorithms. Third, applications are concentrated in the stock market, which has a persistent positive drift. Meaning that staying long (buying stocks) often yields frequent positive rewards. This can inflate perceived performance and encourage one-sided policies. In a more neutral market, such feedback is weaker or misleading, raising difficulties for reward signaling in RL setups, with gains possibly not reflecting genuine learning.

In this work, we propose a system that addresses the three gaps identified in current research. The system couples feature extraction from price-chart images with a RL agent that controls continuous position sizes. Our work extends the current literature with the use of an Actor-Critic RL method, applying the Soft Actor-Critic (SAC) algorithm. Finally, we base our evaluation of the developed system on the Foreign Exchange (FX) market, which does not present a long-term drift, and therefore, presents a more challenging environment for consistent reward feedback. Furthermore, this work explores the use of transfer learning in the initialization of feature extractors for RL.

1.2. Contributions

This work designs, implements and evaluates a system that combines the decision-making capabilities of RL with the pattern-recognition strengths of CV and applies it to financial markets. The objective is to move current research beyond the limitations found in the current body of knowledge. By integrating these techniques, the research aims to contribute to academic understanding and to practical advances in trading technology, extending the boundaries of ML in finance.

Integrating CV with RL in trading addresses gaps in both academic research and industry practice. The broader impact reaches beyond finance: demonstrating how ML supports adaptive behavior in non-stationary settings is relevant to other fields such as robotics, autonomous systems and healthcare, where adaptability is crucial.

1.3. Research Questions

This section outlines the core questions that this dissertation seeks to answer, situating them within the broader context of the research objectives. By articulating these questions explicitly, the section serves as a roadmap for the subsequent analysis and discussion.

(1) *Can a CV-RL combined system unlock positive performance in a non-drift market?*

Existing financial research predominantly focuses on stock markets. These markets have a positive drift [7], attributable to factors such as inflation, corporate earnings growth and efficiency improvements. Models applied to stock markets often reflect this positive drift by frequently generating buy signals, as stocks generally tend to appreciate over time. In contrast, FX markets inherently differ, as currency pairs consist of two active currencies without either exhibiting a systematic positive drift. Consequently, model outcomes in FX markets necessitate more balanced, two-sided trading strategies. This research aims to explore whether CV-RL models can still deliver positive performance within such a symmetric, non-trending market context.

(2) *Can the developed agent learn about the quality of its own actions?*

This study implements an RL model with a continuous action space. The agent is trained to output an allocation of its current bankroll, forcing it to predict not only where price is going to move but also, how much should be allocated to capture this movement. This naturally forces the agent to learn about the quality of its own actions, allocating a smaller part of its bankroll when it is less confident of its own decision. This research aims to understand whether the agent developed can learn about the quality of its own actions, and if so, how it affects its performance.

(3) *Is transfer learning an effective strategy in improving the CV-RL system?*

The approach employed in this research explores the combination of a Convolutional Neural Network (CNN) as a feature extractor, with an RL model. The model uses the outputs from the CNN as its states. We compare the performance of the model when the feature extractor is pre-trained and when it is not. This serves the purpose of evaluating whether transfer learning is an effective strategy in improving the CV-RL system.

By focusing on these specific questions, the dissertation aims to provide a comprehensive and structured investigation into the use of RL and CV in trading. These questions ensure that the research remains targeted and relevant and also provide a framework for evaluating the success of the proposed system and identifying areas for further exploration.

CHAPTER 2

State of the Art

The application of RL and CV techniques in financial trading has gained increasing attention from researchers [8], leading to a proliferation of studies that attempt to optimize decision-making processes in trading environments. However, most studies focus on either RL or CV in isolation, rather than combining both approaches.

Considering only the studies that focus on CV, Sezer et al. [9] generates images from a variety of technical indicators' time series (not price time series data), and demonstrate the efficacy of signal construction based on the images generated. Jiang et al. [10] turn stock price/volume time series into images and use CNNs to autonomously learn visual patterns that are able to forecast future returns, instead of testing pre-specified rules like momentum or reversal. They show that the learned patterns are distinct from standard technical signals, deliver higher out-of-sample accuracy and strong portfolio performance. Finally, they demonstrate that the patterns learned in the US stock market transfer well across international stock markets.

On the other hand, studies focused on RL tend to emphasize the development of optimal trading policies without necessarily leveraging the power of visual representations. Jiang et al. [11] developed a deep RL framework for portfolio management in the cryptocurrency market, leveraging an Ensemble of Identical Independent Evaluators (EIIE) along with deterministic policy gradient methods. Their approach integrates CNNs, RNNs and LSTMs to extract features from historical price and volume data (data is fed as numerical, and not images), outperforming traditional strategies in back-tests. Similarly, Deng et al. [12] introduced a deep recurrent neural network (DRNN) for financial trading, combining RL and deep learning for real-time trading optimization. Their framework used actor-based direct RL optimized via Backpropagation Through Time (BPTT), successfully extracting features and optimizing performance. These studies highlight the adaptability of RL models in responding to different markets and optimizing strategies. However, they rely on numerical inputs and lack the ability to process the spatial relationships inherent to data visualization.

Additionally, Mortaji et al. [13] conducted a comprehensive review of RL applications in portfolio optimization, analyzing various RL algorithms and their performance across different environments. The study identified several key RL methods frequently used in financial markets. Q-learning, as a model-free value-based approach, has been widely applied to optimize portfolio allocation. SARSA, an on-policy RL method, allows for continuous state updates during training. Deep Q-Networks (DQN) have been employed to handle large action spaces by integrating deep learning techniques. Deep Deterministic

Policy Gradient (DDPG) combines Q-learning and policy gradient methods for continuous control, making it particularly useful in portfolio optimization. Finally, Proximal Policy Optimization (PPO) has demonstrated advantages in policy stability and convergence speed.

A smaller subset of studies attempts to integrate RL and CV, aiming to leverage the strengths of both methodologies. For example, Polamuri et al. [14] experimented with Generative Adversarial Networks (GANs) in combination with RL to optimize hyperparameters for price prediction. Similarly, Vullam et al. [15] introduced a hybrid deep learning approach for stock market prediction, named Stock-GAN, which combines GANs with RL techniques. Their work improves upon traditional deep learning methods by integrating Bayesian optimization for hyperparameter tuning and leveraging a hybrid prediction algorithm (MMGAN-HPA) that combines multiple models for better accuracy.

The study by Kai Cui et al. [6] stands out as the closest to this dissertation's objectives. It introduces a Multi-Scale Convolutional Neural Network (MS-CNN) combined with Double Deep Q-Networks (DDQN) for stock trading. The MS-DDQN framework uses feature extraction from financial time-series data, outperforming traditional methods in terms of profit and risk-adjusted returns. The model employs historical price data from the U.S. stock market, using daily OHLC (open, high, low, close) prices and trading volume over a fifteen-year period (2007-2020). The study's experimental setup includes a training phase from 2007 to 2017, followed by a testing period from 2018 to 2020, and evaluates performance using the short-term Sharpe ratio [16] as a reward function. Benchmark comparisons were made against baseline models such as Buy and Hold, Sell and Hold and other RL models like DQN-vanilla and DQN-pattern.

The MS-CNN in Cui et al.'s [6] work is designed to extract stock trend features at different time scales, mimicking the way human traders analyze various moving averages (days, weeks, months). The network treats historical stock prices and trading volume as a two-dimensional image, applying convolutional layers with different kernel sizes to extract local and general patterns. These extracted features are then fed into the DDQN agent, which refines trading strategies by reducing Q-value overestimation, leading to more stable decision-making. A visual representation of the developed system can be observed in Figure 2.1.

While Cui et al.'s [6] work presents a well-structured RL-CV integration, this dissertation aims to extend the approach and fill in gaps left by the current literature. The discrete action space in the presented model, limited to buy, hold and sell, restricts the agent's flexibility. Our study will implement a continuous action space, allowing for dynamic position sizing. Additionally, the absence of bankroll management assumes an unlimited trading capacity, which does not reflect real-world constraints. Our approach will include a limited bankroll that the agent must manage, ensuring that decisions are constrained by available capital, leading to more realistic simulations of real-world conditions.

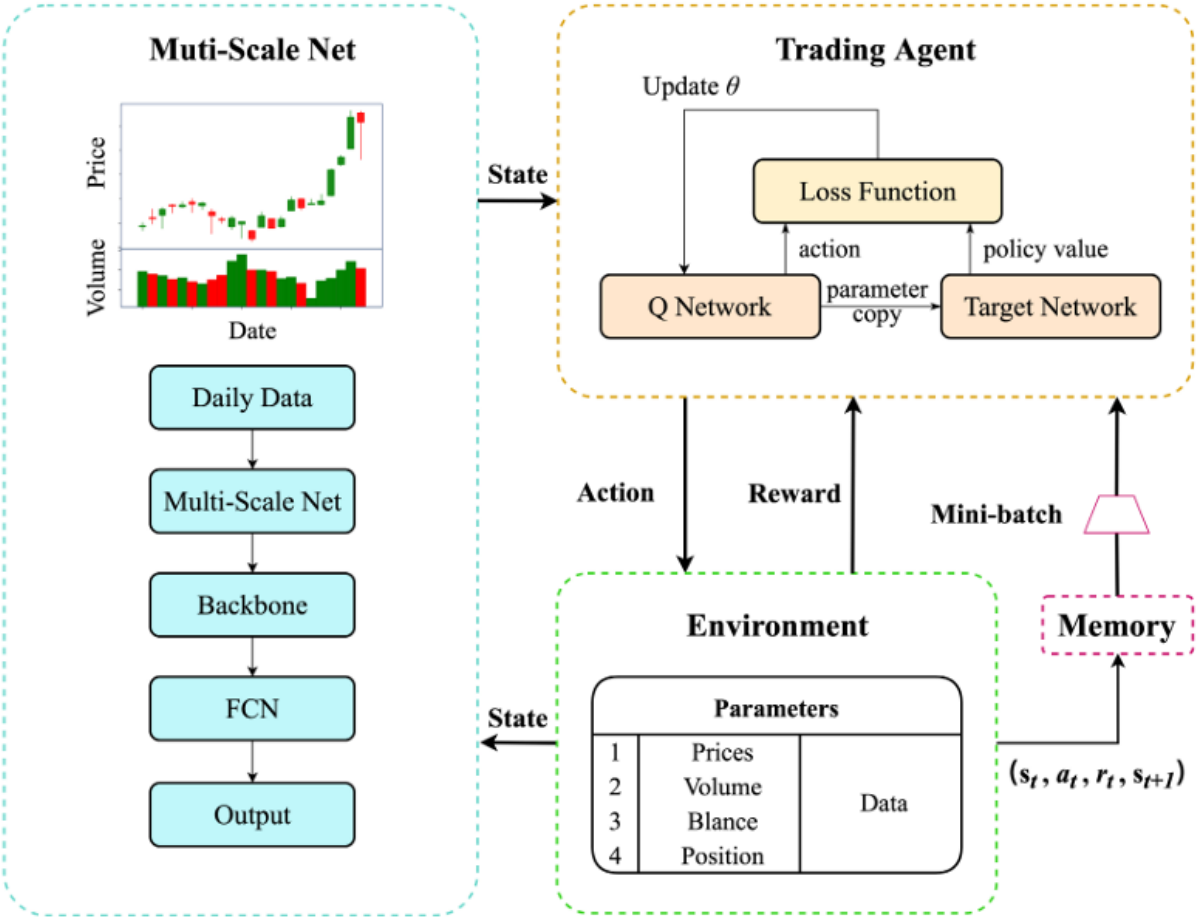


FIGURE 2.1. Cui et al. [6] trading agent system.

To further extend on the current body of knowledge, we choose as an environment the FX market, instead of positive drift markets such as the stock and cryptocurrency markets used in the mentioned research. This work also explores the use of transfer learning in the initialization of feature extractors for RL, taking inspiration from the Jiang et al. [10] study.

Furthermore, while Cui et al.'s [6] system relies on DDQN, this dissertation will explore the use of SAC. This algorithm provides tools for improved policy stability, sample efficiency, and the ability to handle continuous action spaces.

CHAPTER 3

Context

3.1. Financial Markets, Foreign Exchange and the Trading Game Setting

Financial markets are platforms where individuals, institutions and governments trade financial instruments. These markets play a vital role in the global economy by facilitating capital allocation, enabling risk management and providing liquidity (ease to buy and sell) to participants [17]. Broadly, financial markets can be divided into [18]:

- Stock markets: where companies list their shares.
- Over-the-Counter (OTC) markets: where participants trade directly with each other (without a broker).
- Bond markets: where companies and governments issue debt instruments.
- Money markets: for products with short-term maturities (e.g. certificates of deposit).
- Derivatives markets: for products whose value is based on an underlying asset.
- Foreign Exchange (FX) Markets: for currencies.
- Commodities markets: for physical goods (e.g. gold, oil, corn).
- Cryptocurrency markets: for digital currencies.

Participants in financial markets include retail investors, institutional investors (such as hedge funds, mutual funds and pension funds), corporations and governments. Each group has distinct motivations for participating, ranging from investment and speculation to hedging against financial risks and financing operations [17]. These participants collectively contribute to the valuations of assets, market efficiency and capital flows across global economies.

3.1.1. Foreign Exchange Market

The FX market is the largest and most liquid financial market globally [19], with a daily trading volume exceeding $7 trillion [20]. It is a decentralized market where currencies are bought and sold, enabling international trade and investment. Unlike centralized exchanges for stocks, the FX market operates over-the-counter (bilaterally between two counterparties) through a network of financial institutions, brokers and electronic trading platforms [19].

In the FX market, the primary products traded are currency pairs, representing the exchange rate between two currencies (e.g., EUR/USD or USD/JPY). These pairs are classified into: Major Pairs, the most widely traded pairs, such as EUR/USD and USD/JPY; Cross Pairs, currency pairs that do not involve the US dollar, such as EUR/GBP; Exotic Pairs, which involve emerging market currencies, such as USD/ZAR [21].

A diverse range of actors participate in this market, including [22]:

- Central banks: manage monetary policy and stabilize currency values.
- Commercial banks: facilitate currency trading for clients and speculate on currency values.
- Corporations: hedge against currency risks associated with international operations.
- Institutional investors: speculate or hedge using large-scale trades.
- Retail traders: individual investors trading smaller volumes.
- Market makers and brokers: provide liquidity and match buy/sell orders.

The FX market is known for its high liquidity (in major pairs) [20] ensuring tight bid-ask spreads (the difference between buy and sell prices at a given point in time) and the ability to execute large trades without significant price impact. Its decentralized structure allows for continuous trading 24 hours a day, five days a week, starting in the Asia-Pacific region on Monday morning and closing in New York on Friday evening [22].

Despite its high liquidity, the FX market carries some associated inherent risks:

- Gap risk [23]: arises during weekends or holidays when unexpected geopolitical or economic events can cause significant price movements before the market reopens.
- Liquidity shortages [24]: occur around major economic announcements or geopolitical events, when market participants exit the market (refuse to take risk), and the lack of participants makes it hard to execute trades at desired prices (bid-ask spread increases).
- Leverage risks [25]: leverage (borrowing money to invest) commonly used in FX trading amplifies both potential gains and losses, increasing overall risk exposure.

As mentioned in Section 1.3, stock markets have a positive drift. In contrast, FX markets differ, as currency pairs do not tend to exhibit any systematic drift. This can be observed in Figure 3.1 and Figure 3.2, which compare the EUR/USD price since the introduction of the Euro in 1999 and the S&P 500 (an index composed by the largest 500 companies in the United States) price in the same time period. This representation clearly demonstrates the difference between the two, with the EUR/USD, after a 25 year period, being close to its initial value, whereas the S&P 500 grew about four times in value.

3.1.2. Trading Game Setting

This section examines the historical evolution of the "trading game", providing essential context for understanding the current market landscape and the factors that have shaped its development.

Before the integration of computer systems, financial trading was a highly human-centric activity, conducted on physical trading floors or via telephones. Traders relied on personal expertise, intuition and direct communication to execute trades. The most



FIGURE 3.1. EUR/USD price since inception in 1999.



FIGURE 3.2. S&P 500 price since 1999.

common market participants were commercial banks, investment banks and individual speculators, who engaged in face-to-face negotiations or placed orders through brokers.

Human traders often used technical analysis, fundamental analysis and market sentiment to inform their decisions. Information flow was slower and traders with faster access to news or deeper social networks often held a competitive edge.

The advent of computers and electronic trading platforms transformed the trading game, automating many of the processes that were once manual. Algorithms and high-frequency trading systems now dominate the market, capable of analyzing vast amounts of data and executing trades within milliseconds. This shift has leveled the playing field for many participants while also increasing competition.

Market participants, from retail traders to institutional investors, now aim to "beat" the market using advanced strategies. These include algorithmic trading, quantitative analysis and machine learning techniques that exploit inefficiencies in pricing or identify patterns that human traders may overlook. Trading systems are designed to process

market data in real-time, enabling participants to react to news and events faster than ever before.

The integration of AI and ML into trading has further elevated the complexity of the market. RL and predictive analytics are increasingly used to develop adaptive systems that evolve with market conditions. The "game" has become a competition not only between human traders but also between increasingly sophisticated algorithms, each seeking to beat the other.

3.2. Transfer Learning

Transfer learning [26] is a machine learning approach in which knowledge gained from solving one task is leveraged to enhance a model's performance on a different, but related, task. This is valuable when the new task suffers from either limited data or limited computational resources. Unlike traditional models, which are built and trained from scratch for each separate problem, transfer learning reuses parts or all of a previously trained model, reducing the required data and training time.

In practice, transfer learning is commonly applied in deep learning, where models are first trained on large benchmark datasets (e.g., ImageNet [27]) and later fine-tuned for more specialized applications. For example, a model initially built for general image recognition can be adapted to medical imaging by retraining with new data. Pre-trained CNNs like VGG [28] and ResNet [29] models are often repurposed to address new problems. Initializing from a pre-trained CNN lets practitioners build high-performing models in scenarios where labeled training data is scarce.

3.3. Computer Vision and Reinforcement Learning

3.3.1. Machine Learning and Artificial Intelligence

Machine Learning (ML) and Artificial Intelligence (AI) have revolutionized numerous domains by enabling systems to learn from data, adapt to changing conditions and make informed decisions. Tasks that traditionally require human intelligence, such as problem-solving, reasoning and perception, are now being performed by intelligent systems, what we call AI. Within AI, ML focuses on developing algorithms that allow machines to learn patterns and relationships from data without being explicitly programmed.

ML is typically categorized into three main paradigms: supervised learning, where models learn from labelled data; unsupervised learning, which involves discovering hidden patterns in unlabelled data; and RL, where an agent learns to make decisions by interacting with an environment and receiving feedback [30]. These paradigms form the foundation for many applications across industries, including healthcare [31], robotics [32] and finance [33].

As AI and ML continue to evolve, new subfields, such as CV and RL, have branched from it and created powerful tools for solving complex problems. This chapter explores these fields individually and then examines their interplay, highlighting how they complement each other to create intelligent systems capable of addressing real-world challenges.

3.3.2. Computer Vision and Convolutional Neural Networks

CV is a branch of AI focused on enabling machines to interpret and process visual data in a manner similar to human vision. CV systems extract information from images or videos, such as identifying objects, detecting patterns and recognizing scenes [34]. This capability allows CV to power a wide range of applications, including facial recognition [35], medical imaging [36], autonomous vehicles [37] and augmented reality [38].

The underlying process of CV involves several stages: preprocessing raw visual data, extracting features and interpreting these features to draw conclusions or make decisions. Techniques such as edge detection, object segmentation and feature matching are commonly employed when analyzing visual data. Recent advancements in deep learning, particularly CNNs, have improved the accuracy and efficiency of CV systems. CNNs are particularly effective at learning hierarchical patterns in visual data, making them a cornerstone of modern CV and a common tool for tasks such as image classification, object detection [39] and image segmentation [40].

CNNs are composed by multiple layers [41], each serving a distinct purpose in feature extraction and transformation. The primary layers include:

(1) Convolutional layer: the convolutional layer applies a set of learnable filters (kernels) to the input image. Each filter slides over the input, computing dot products to produce feature maps which detect patterns such as edges and shapes. The output feature map is computed as: $\text{Feature Map} = \text{Input} * \text{Kernel} + \text{Bias}$.
(2) Pooling layer: pooling reduces spatial dimensions, decreasing complexity and controlling overfitting, whilst retaining the most important information. The two most common types are Max Pooling (selects the maximum value in a window) and Average Pooling (computes the average of values in a window).
(3) Activation function: non-linearity is introduced using activation functions, enabling the network to model complex relationships.
(4) Fully Connected (FC) layer: after convolutional and pooling layers, high-level features are flattened and fed into fully connected layers for classification or regression.

CNNs are trained using supervised learning and labeled data. The weights of kernels/filters and fully connected layers are optimized by minimizing a loss function through backpropagation. Backpropagation [42] consists of two main phases. It starts with a Forward Pass, where the input data is passed through the network. In this phase, each layer of the network computes an output, and the final output is compared to the true label using the loss function. This is followed by a Backward Pass, where the error is propagated backward from the output layer to the input layer. The gradient of the loss with respect to each weight is computed, and weights are updated using an optimizer (e.g. Adam).

3.3.3. Reinforcement Learning

RL is a subfield of ML in which an agent learns to make decisions by interacting with its own environment [1]. The agent's goal is to maximize cumulative rewards over time by selecting actions that lead to favourable outcomes. Unlike supervised learning, RL does not rely on labelled data, instead, it uses feedback signals in the form of rewards or penalties to guide learning.

The RL process is typically modelled as a Markov Decision Process (MDP), which consists of the following components:

- State (S): the representation of the environment at a given time.
- Action (A): the set of possible decisions the agent can take.
- Reward (R): the feedback signal received after performing an action.
- Policy (π): a mapping from states to actions that guides the agent's behaviour.
- Value Function (V): a measure of the expected cumulative rewards for a given state or action.

An MDP is defined by the tuple: (S, A, P, R, γ), where:

- S: set of possible states.
- A: set of possible actions.
- $P(s'|s, a)$: transition probability from state s to s' given action a.
- $R(s, a)$: expected reward from taking action a in state s.
- γ: discount factor that balances immediate and future rewards.

RL algorithms are categorized into model-free and model-based approaches [43]. Model-free methods (such as Q-learning and Deep Q-Networks), rely solely on interactions with the environment. One example of this, is their use in building agents to play video games, where the agent learns how to maximize game score through direct trial and error [44]. On the other hand, model-based methods attempt to use an explicit model of the environment to improve learning efficiency. A common example of this is the training of factory robots to do common tasks such as stacking objects, where the enviroment is known/predictable and can be modeled to guide learning [45]. Most common approaches in modern RL are Model-free methods due to their flexibility.

Besides the model-free and model-based distinction, RL algorithms can also be categorized into Value-Based, Policy-Based and Actor-Critic methods [46]:

- Value-Based methods [47] learn a value function (most commonly the action-value function $Q(s, a)$) that estimates future rewards for taking action a in state s. The optimal policy is implicit and derived by selecting actions that maximize the estimated values. These models are efficient in discrete action spaces where enumeration of all possible actions for a given state is feasible. Exploration is typically handled through strategies such as ϵ-greedy, ensuring balance between exploiting high-value actions and discovering potentially better ones.

- Policy-Based methods [48] directly learn a policy (π) that maps states to actions, without estimating value functions. These methods optimize policy parameters to maximize expected cumulative rewards (often done using gradient ascent). They are effective in environments with high dimensionality and/or continuous actions spaces and are mostly model-free since policy is optimized based on interaction with environment, without modelling its dynamics.
- Actor-Critic methods are an hybrid approach that combines Value-Based and Policy-Based learning, where the Actor learns the policy (π) and the Critic evaluates actions using a value function. In these methods, the Actor chooses actions while the Critic provides feedback on the actions chosen to guide policy improvement. This hybrid approach reduces the high variance associated with the gradient estimation of Policy-Based methods while mantaining the flexibility it offers.

3.3.4. Soft Actor-Critic Algorithm

Soft Actor-Critic (SAC) is a model-free and actor-critic RL algorithm designed for continuous action spaces [49]. While in conventional RL, the goal is to find a policy (π) that maximises the expected cumulative discounted return, the SAC algorithm extends this objective and encourages exploration through an entropy term ($H(\pi(\cdot \mid s_t))$). SAC's objective function ($J(\pi)$) is expressed mathematically as:

$$J(\pi) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)\right] \tag{3.1}$$

Where:

- $r(s_t, a_t)$ is the reward at time t,
- $\gamma \in [0, 1]$ is the discount factor.

$$J(\pi) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t \left(r(s_t, a_t) + \alpha\, \mathcal{H}(\pi(\cdot \mid s_t))\right)\right] \tag{3.2}$$

Where:

- $\mathcal{H}(\pi(\cdot|s_t)) = -\mathbb{E}_{a_t \sim \pi}\left[\log \pi(a_t|s_t)\right]$ is the entropy of the policy as state s_t,
- $\alpha > 0$ is the temperature parameter, that controls the trade-off between reward maximisation and exploration.

By maximizing both rewards and policy entropy, SAC achieves more robust exploration and prevents premature convergence to suboptimal policies.

The SAC architecture consists of three neural networks: one Actor and two Critics. The Actor $\pi_\phi(a \mid s)$ policy network produces a probability distribution over possible actions for each state. For continuous actions, as in this work, it is often parameterized as a Gaussian distribution. The two Critics, $Q_{\theta_1}(s, a)$, $Q_{\theta_2}(s, a)$, estimate the expected return for state-action pairs. SAC uses two Q-value critics to reduce overestimation bias

(a common issue where Q-values are systematically higher than true expected returns), by using the minimum of the two estimated Q-values for target updates.

In on-policy algorithms, each training batch must be collected using the current policy, and experiences are discarded after one update to keep policy consistency. In SAC (an off-policy algorithm), experience replay allows it to store and sample past environment interactions from a buffer, even if they were generated by older policy versions. This improves sample efficiency and training stability.

The SAC algorithm consists of the following steps:

ALGORITHM 3.1.

1. Initialize Actor and Critics networks with random weights.

2. Interact with the environment and store transitions (state, action, reward and next state) in a replay buffer.

3. Sample random batches from the buffer for parameter updates.

4. Update Critic networks. Minimize the Bellman residual for the soft Q-function, taking into account both rewards and policy entropy.

5. Update Actor network. Adjust policy to maximize expected Q-value plus entropy, balancing exploitation and exploratory diversity.

6. Optionally update the entropy temperature α to automatically tune the exploration-exploitation balance.

7. Repeat steps 2-6 until convergence.

SAC represents a major advancement in RL by combining maximum entropy RL, actor-critic methods and off-policy learning.

3.3.5. Integration of Computer Vision and Reinforcement Learning

While RL focuses on decision-making through interactions with an environment, CV can enhance this process by providing high-dimensional visual data that informs the agent's understanding of its environment. The CV module serves as a perception system that extracts features from visual data, which are then fed into the RL agent, which in turn uses them to make decisions.

Several real-world applications illustrate the power of integrating CV and RL:

- Autonomous vehicles: CV systems process camera feeds to identify objects such as pedestrians, vehicles and road signs. RL agents use this visual information to make driving decisions, such as accelerating, braking, or turning [50].
- Robotics: in robotic manipulation, CV systems recognize objects and their spatial orientation, enabling robots to interact with their environment. RL then determines the optimal sequence of actions to complete tasks such as assembling components or picking and placing objects. Figure AI's different robots are examples of this integration [51] [52] [53].

- Gaming and simulations: CV is used to interpret complex game environments [54], while RL agents develop strategies to achieve high scores or complete objectives [55].

CHAPTER 4

Methodology

The following chapter presents the methodology used to design and implement a trading game system. This methodology is inspired by the one presented in Cui et al. [6], as laid out in Chapter 2.

4.1. Game Setup

The objective of the trading game is to maximize long-term profit through a series of consecutive investments (actions). Unlike traditional supervised learning models, which focus on predicting immediate outcomes from labelled data, this setup employs RL with an agent to make a sequence of decisions over time, each influenced by past outcomes (a continuously changing bankroll). The agent must learn to optimize cumulative returns, making the problem well-suited to the RL framework due to its inherently sequential and feedback-driven nature [1].

The agent begins with an initial bankroll, representing the capital available for trading. At each time step, defined as a one-hour interval, the agent must allocate a portion of this bankroll toward a position on the EUR/USD pair, either long (buying Euro) or short (selling Euro). The goal is to grow this bankroll through successive decisions and quantifying success as the compounded return over the course of the trading episode. Profitability is thus directly measurable: the agent's final bankroll reflects its performance in achieving the game's objective.

The setup for this work was built with the EUR/USD exchange rate, as it is the most traded pair of currencies on the planet [20], and therefore, also the most liquid market. Data was gathered from the introduction of the Euro in 1999 (although the set used only includes data from 2003 onwards, as explained in Section 4.4), until the 28 of July 2025, the last day when data was collected.

4.2. Game Mechanics

The game unfolds in a discrete-time simulation, where each step corresponds to one hour. At every time step, the agent is presented with a chart image representing the previous nine hours of asset price movement. For reference, major global financial centers operate within comparable time frames. As mentioned in Section 3.1, the FX market operates 24 hours a day, but is often divided into major overlapping "sessions" based on regional activity: Tokyo (Asian session): 00:00-09:00 GMT, London (European session): 07:00-16:00 GMT and New York (North American session): 12:00-21:00 GMT. The overlaps between sessions create the most active periods (e.g., London-New York overlap). Therefore, the

nine-hour period was selected as it aligns with the standard duration of a typical day trading session.

4.2.1. Action Space (A)

Using the provided visual input, the agent selects an action, which is the proportion of its bankroll to allocate to a long or short position. The environment then transitions to the next time step, revealing the actual price movement over the next hour and calculating the reward based on the accuracy and magnitude of the position taken.

The agent's action space is continuous, ranging from -1.0 to +1.0, where:

- +1.0 denotes a fully long position (buying the asset and therefore 100% of the bankroll invested in expectation of price increase),
- -1.0 denotes a fully short position (buying the asset and therefore 100% of the bankroll invested in expectation of price decrease),
- 0.0 denotes a neutral stance (no capital allocated to a position).

4.2.2. State Space (S)

Each state presented to the agent is a grayscale image representing the asset's price chart over the last nine hours. The image is the sole input for decision-making, making the learning task one of visual interpretation. To extract meaningful features from these images, the agent is equipped with a CV module, in this case, a CNN, which translates raw pixels into latent representations. These representations inform the RL policy's decisions.

4.2.3. Reward Function (R)

The reward r_t at step t is computed as:

$$r_t = a_t \times return_t \tag{4.1}$$

where:

- $a_t \in [-1, 1]$ is the action taken (proportion of bankroll allocated),
- $return_t$ is the asset's return over the next time interval, defined as $\frac{P_{t+1}-P_t}{P_t}$.

This structure ensures that the reward is proportional to the agent's exposure: large, correct bets yield high rewards, while incorrect bets result in negative rewards. No reward is given for staying out of the market (i.e., $a_t = 0$).

4.3. System Setup

This chapter presents the design and structure of the full system developed to address the trading game described earlier. The objective of the system is to construct an end-to-end pipeline that learns to make decisions based solely on historical chart images of the EUR/USD exchange rate.

The system begins by transforming raw price data at a minute frequency into a large set of rolling 9-hour price chart images, each labelled with the return in the subsequent hour. These image-return pairs form the basis for two core learning tasks:

(1) Pre-training a CNN to predict the direction (up/down) of the next-hour return based on the past 9-hour chart.
(2) Training a RL agent that uses the previous CNN to guide dynamic capital allocation decisions with the objective of maximizing long-term profit.

The system's ultimate goal is to simulate a self-adaptive, vision-driven agent that can operate under realistic conditions and continually improve its performance over time.

The system is composed of three interconnected and interdependent modules. The Data Preparation subsection is responsible for the collection, transformation, and formatting of financial time series into visual and structured datasets. The CV module contains the CNN model, which is pre-trained to interpret images and recognize whether a price increase or decrease is likely in the hour ahead. Finally, the RL module houses the agent, which integrates visual inputs from the CNN and outputs actions.

Each of the three subsections is tightly coupled with the others in a modular but sequential pipeline. The data preparation module outputs the essential inputs (images and returns) that feed both the CNN and RL training stages. The CV module produces a CNN pre-trained for feature extraction, speeding up RL training, as the model does not need to be trained from scratch [26]. The RL module uses both the data and the CNN. It continues training the CNN end-to-end to align feature extraction more closely with the agent's goals.

In this setup, represented in Figure 4.1, the CNN acts as a bridge between perception (chart interpretation) and action (capital allocation), learning features that are optimized not just for classification but ultimately for a long-term goal.

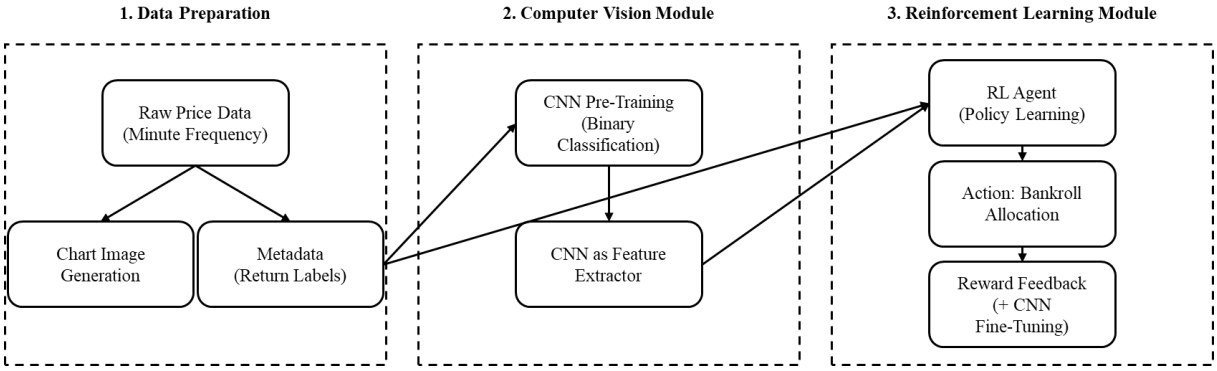


FIGURE 4.1. System setup diagram.

The first step in assembling the system involves the acquisition and preprocessing of historical price data. The following operations define the Data Preparation module:

- Data collection: minute-level price data for the EUR/USD currency pair is collected, covering a period of approximately 25 years, starting from the year 2000.
- Image generation: from this time series, a rolling window of 9 hours is applied to generate a set of line chart images (e.g. we collect prices from 00:00 to 09:00, with this data we generate an image, after this we advance one hour and collect prices between 01:00 and 10:00. The process is repeated until the dataset is exhausted).

- Metadata and labelling: alongside image generation, a metadata dataset is collected, where each entry corresponds to one image and includes timestamp information, the return of the next hour, the number of observations within both the 9-hour window and the following hour window. The return serves as the label for supervised learning and the reward signal for RL.

This dataset forms the foundation for both the pre-training of the CNN and the training of the RL agent.

The Computer Vision component of the system focuses on equipping the agent with the ability to extract features from chart images. This is achieved through a CNN. The CNN is trained in a binary classification task where it predicts whether the return following a given image is positive or negative. It passes images through convolutional and fully connected layers that extract features and generate predictions. The training uses the binary cross-entropy loss with logits, adjusted for class imbalance, and updates model parameters with the Adam optimizer through backpropagation. Performance is tracked with accuracy and F1 score on training and validation sets, and the best model is saved when validation results improve. The primary goal of this module is not to produce a final decision-making model, but rather to pre-train a CNN that can serve as a feature extractor for RL.

The Reinforcement Learning module implements the full decision-making agent that learns to allocate a bankroll over time. Its structure includes:

- Image processing: each time step provides the agent with an image of the last 9 hours of trading. This image is passed through the pre-trained CNN, which transforms it into a lower-dimensional feature vector.
- Policy learning: the extracted features are fed into an RL agent, which maps these inputs to an action: a real-valued allocation $a_t \in [-1, 1]$, indicating what percentage of its bankroll to go short (-1.0) or long (+1.0) on the asset.
- Reward signal: the agent receives a reward proportional to the return of the asset in the next hour, scaled by its investment allocation. Over time, the agent learns to maximize its total cumulative return.
- CNN fine-tuning: during the training of the RL agent, the CNN is further trained (fine-tuned) through backpropagation. This allows the system to refine the visual features it extracts in a way that directly supports RL performance.

4.4. Data Collection and Preparation

This section outlines the methodology for collecting and preparing the data required to train and evaluate the trading agent.

4.4.1. Data Requirements

The primary objective of this stage was to obtain a 1-minute frequency historical time series for the EUR/USD FX pair. Data was gathered from two distinct sources. The first, Histdata.com [56], is a publicly available dataset that contains long historical series

of FX price data, including open, high, low, and close prices at 1-minute intervals. It was chosen as the main training data source because it offers over 20 years of continuous data, enabling the construction of large datasets necessary for training deep RL agents. The second, is the Bloomberg Terminal. Bloomberg [57] is a widely renowned source of financial data. It is considered more reliable, with rigorous quality controls and consistent data handling. However, Bloomberg imposes constraints that limit the extraction of 1-minute frequency data to less than a year at a time, making it impractical for constructing large training datasets. Therefore, Bloomberg data was used for testing and validation, ensuring that model evaluation is carried out on a high-quality dataset.

4.4.2. Reliability Considerations

The free data from Histdata does come with potential limitations. In some rare cases, gaps in data (e.g., missing minutes) are observed, due to data collection limitations. However, the dataset is still realistic and does not contain fabricated or anomalous price data. These small and rare imperfections are acceptable for training purposes because the sheer volume of data mitigates the impact of occasional gaps.

The test data, sourced from Bloomberg, provides the reliability needed for robust evaluation. Testing exclusively on high-quality data ensures that final performance metrics accurately reflect the agent's ability to operate under real conditions.

4.4.3. Data Transformation Process

The raw 1-minute EUR/USD time series data was transformed into a dataset usable by both the CV module and the RL agent. This transformation consisted of the following steps:

(1) **Construction of rolling 9-hour windows**
The input to the agent was defined as rolling windows of 9 hours, corresponding to 540 consecutive 1-minute observations. These windows were designed to give the model sufficient context about recent market dynamics, as they represent what is usually considered a "trading session" as explained in 4.2. The window then advances by one hour, creating overlapping sequences and significantly increasing the number of samples available for training.

(2) **Criteria for inclusion and exclusion**
Not all 9-hour windows were retained. To be considered valid, a window must contain at least 8 and a half hours of data, which corresponds to 510 observations. Windows spanning market closures (e.g., weekends or holidays) were discarded. This avoids exposing the agent to gap risk (3.1), where prices jump sharply between the close and re-opening of markets. Finally, any window that did not have the full subsequent hour of data available was excluded because the next-hour return could not be computed.

(3) **Metadata file generation**

A metadata file was generated as the central catalogue linking the chart images generated to their labels and associated timestamps. This metadata file contained the following main features (among others):

- *image_path*: Path to the chart image generated for the 9-hour window.
- *start_time*: Timestamp at the beginning of the 9-hour period.
- *end_time*: Timestamp marking the end of the 9-hour period.
- *return_next_hour*: The price return of the EUR/USD pair in the hour immediately following the 9-hour window.

This metadata file ensures traceability and allows the same dataset to be used consistently across the different stages of the pipeline.

(4) **Image generation**

Each 9-hour window was visualized as a chart image (Figure 4.2), closely resembling the types of charts human traders typically use for technical analysis. These images capture the temporal evolution of price movements in a high-dimensional visual format. Each image is uniquely named and stored in a designated folder, with its location indexed in the metadata file.

(5) **Collection of the following hour return**

For each 9-hour window, the return of the following hour was computed. This value serves two purposes: as the label for supervised training of the CNN, which predicts whether the next-hour return will be positive or negative; as the reward signal for the RL agent, which is rewarded proportionally to the return generated by its actions. The final dataset consists of a large set of chart images representing rolling 9-hour periods and a metadata file linking each image to its timestamps and the next-hour return label.

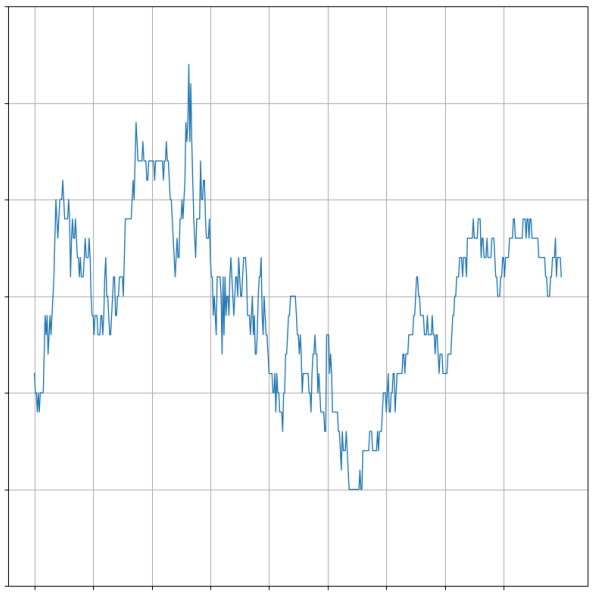


FIGURE 4.2. Example chart image generated.

It is relevant to mention that despite data being collected from 1999, the lack of liquidity in the early years of the pair, led to gaps in the data, which made us exclude the first four years from the dataset, keeping only data from 2003 onwards.

The structured dataset described forms the basis for the CV Module and RL Module that follow.

4.5. Computer Vision Module

The CV module leverages a CNN to extract features from chart images, which serve as inputs to the RL agent. CNNs are particularly effective at identifying hierarchical patterns in images and can autonomously learn the most relevant features, including in financial data charts [10]. By integrating CV into the pipeline, we allow the agent to provide a compact, high-dimensional feature representation for RL.

The CNN is pre-trained before being integrated into the RL agent for two principal reasons. First, pre-training enhances learning efficiency. Training the CNN from scratch in parallel with the RL agent requires simultaneous optimisation of both the perception component and the decision-making component, which can slow convergence. Second, it provides a strong foundation for the RL process. By initiating with a CNN that already possesses an understanding of chart structures, the RL agent is initiated with a bias towards correct actions, avoiding the need to learn from scratch and possibly avoiding local optima.

Once integrated, the CNN is further fine-tuned during RL training, enabling the model to adapt its feature extraction in a way that maximizes performance.

The architecture implemented is referred to as the Nature CNN [58], and is used as the base feature extractor by the Stable Baselines library [59]. We chose this network for three main reasons. First, due to its proven effectiveness in RL, as it has been shown to perform well in various applications. Second, using a recognized architecture facilitates reproductibility and allows for an easier comparison with other systems. Finally, as it is used as the base implementation of the Stable Baselines library, it facilitates integration.

The network's architecture is represented in Figure 4.3. The following is a high-level overview of the architecture:

- Input: grayscale chart image of fixed dimensions corresponding to 9-hour window.
- Encoder: consists of three convolutional layers. The first convolutional layer uses 32 filters with 8 x 8 kernel size and stride 4, followed by ReLU activation. The second convolutional layer uses 64 filters with 4 x 4 kernel size and stride 2, followed by ReLU activation. The third convolutional layer uses 64 filters with 3 x 3 kernel size and stride 1, followed by ReLU activation. The output from the last convolution is flattened and then passed through a fully connected layer to yield a final feature vector.
- Classification head: during pre-training, a separate classifier head processes the encoder's output. The encoder's feature vector is passed through a single linear output layer (feature dimension $\rightarrow 1$), producing the logit for binary classification.

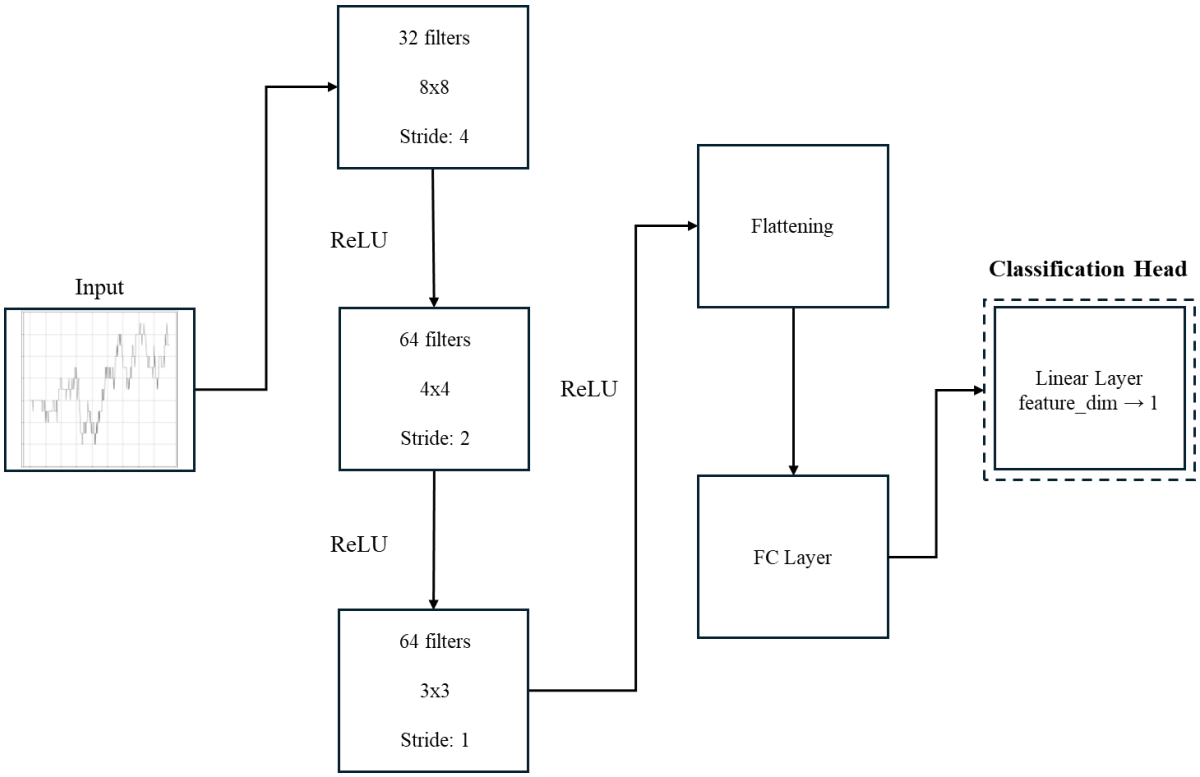


FIGURE 4.3. CNN architecture.

The head is discarded during RL, leaving the encoder as a standalone feature extractor. The architecture is intentionally modular, allowing the feature extractor portion (up to the final dense layers) to be reused within the RL agent.

A classification problem was chosen as the basis for pre-training the CNN. Classification tasks represent the main training paradigm for CNNs, providing a clear and interpretable loss function that supports stable optimisation. In this context, the task will involve predicting whether the next-hour return will be positive or negative, effectively allowing the CNN to learn features associated with upward and downward price movements (similar to Jiang et al. [10]). These learned features are directly relevant to the objectives of the RL module. By contrast, applying an alternative pre-training objective (for example, predicting the volatility of the price depicted in the input image), could still yield meaningful features, but these would not be explicitly aligned with the RL module's goal.

An important design choice was whether to train the CNN using only information already available in the images (e.g. identifying if the price goes up or down within the image window itself) or to use a predictive challenge (predicting the next-hour return). The second approach was selected because it directly aligns the CNN's learning objective with the ultimate aim of the entire setup, producing features more relevant to RL decision-making. It encourages the extraction of forward-looking features and increases the utility of the pre-training stage by embedding a predictive element from the start. This approach is inherently more challenging, since it requires the CNN to generalise beyond the information available in the input window. Additionally, it may introduce noise

when next-hour returns are influenced by factors not visible in the image (which they will most likely be). Nevertheless, the advantages of directly aligning both modules outweigh the limitations associated with increasing task complexity.

After pre-training, the CNN integrates into the RL framework as follows:

- Feature extraction: for each time step, the CNN processes the current 9-hour chart image and outputs a feature vector representing the image/state.
- Policy learning: these features are fed into the RL agent, which maps them to an action.
- Fine-tuning: during RL training, the CNN's parameters continue to be updated, aligning the extracted features more closely with those that maximize cumulative rewards.

This dual-phase training strategy ensures the CNN is both generalized through pre-training and specialized through RL fine-tuning.

4.6. Convolutional Neural Network Training

To train the CNN described in Section 4.5, we employ the PyTorch library [60], a widely used deep learning framework. As there is the possibility of encountering class imbalance (e.g. training set contains a long EUR/USD up/down trend), we mitigate majority-class bias by making use of a weighted binary cross-entropy loss function [61]. This function re-weights positive and negative instances to equalize their effective contribution to the loss. Optimization proceeds by minimizing the defined loss function. Model selection is performed on a held-out validation set using the macro F1-score [62] [63] metric. This ensures that both classes contribute equally to the overall metric.

The model is trained for 50 epochs. The evolution of performance can be seen in Figure 4.4.

Throughout the 50 epochs, training performance increases steadily whilst validation performance displays noisy improvement. Training accuracy increases consistently from 49.70% in epoch 1 to 83.89% in epoch 50. Validation accuracy varies in a range between 46.32% and 53.44%. The validation F1-Score starts at a minimum of 0.3177 and achieves a maximum of 0.5131, at epoch 45. The checkpoint retained as the final model corresponds to epoch 45, since it achieved the highest F1-Score. This epoch has a training accuracy of 78.99% and a validation accuracy of 51.58%. The evolution of training metrics point to the model overfitting to the training data, which is normal given the small subset of data employed (around 5.000 images). This was disregarded as a minor concern, since model achieves the objective of pre-training for RL, and will be subsequently fine-tuned.

These results indicate a modest predictive signal in the chart images for subsequent returns, in line with the results found by Jiang et al. [10], in our case, for the FX market. Consistent with the competitive and stochastic nature of financial markets, the objective is not a high absolute accuracy, but rather a reliable edge over a 50-50 baseline.

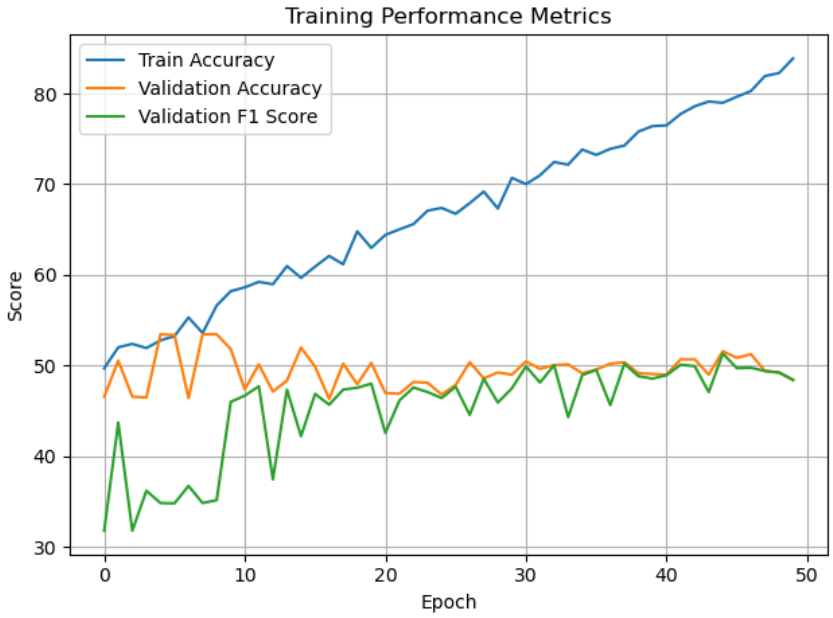


FIGURE 4.4. CNN training metrics over 50 epochs.

4.7. Reinforcement Learning Agent

RL reframes training as an agent interacting with an environment rather than fitting a static input-target mapping. In supervised learning we minimize a loss against ground-truth labels that are assumed i.i.d., and by contrast, our agent receives observations sequentially, chooses actions, and is judged only through returns routed back as rewards. Credit assignment is therefore temporal, not instantaneous, and the data distribution is non-stationary because it depends on the current policy [1].

To train an RL agent, we must first have an environment with which it can interact. There are classic environments built to "plug and play" with RL models. In our case, since we are implementing the algorithm within our "own game", we must build a custom environment with which the agent can interact. This environment was built using the Stable Baselines library [64].

In our environment, each episode corresponds to a trading week, defined as four 24-hour days plus one 21-hour session, starting at midnight in Tokyo and ending at 21:00 in New York, as explained in Section 4.2. This results in a total of 117 timesteps per episode. Observations are single-channel 128×128 images normalized to $[0, 1]$. Actions live in a continuous one-dimensional interval, with action (a_t) space within $[-1, 1]$, which we interpret as proportional exposure from fully short (-1) to fully long (+1). At each step we read the dataset's realized percentage return r_t; the reward is then calculated as $a_t * r_t$, and the portfolio value is compounded via $V_t + 1 = V_t(1 + a_r r_t)$. Episodes reset the bankroll to 100 "points", which makes episode returns directly comparable across runs.

We chose to use the SAC algorithm [65], as it handles continuous actions naturally and includes entropy regularization, which encourages exploration without manual schedules, facilitating the task at hand. SAC's loss function is described in Section 3.3.

Despite scheduling auto-sufficiency, SAC requires other hyperparameters. Gamma (γ) is the reward discount factor, and higher gamma values are associated with less discounts for further out timesteps. The lower the gamma value, the greedier the agent becomes, becoming "myopic". In our case, this would mean a maximisation of shorter term rewards, with less consideration for long-run Profit and Loss (PnL), this can make the model too greedy and cause large drawdowns (declines in value) due to excessive risk exposure (very high portfolio allocations). On the other hand, for the model to be able to learn good actions, we must guide it using short-term rewards, meaning there is a balance to be found between exploration and exploitation. We set γ (gamma) $= 0.995$, which discounts slowly within a 117-step episode: $\gamma^{117} = 0.556$. This preserves a strong incentive to optimize later-in-week outcomes while still attempting to guide the model with nearer rewards.

Target entropy sets the desired randomness of the policy. SAC's temperature (α) is tuned to hit the desired target, therefore this hyperparameter directly controls the exploration-exploitation trade-off. Higher (less negative) target entropy raises α, resulting in broader exploration but potentially slower and noisier convergence. Lower (more negative) target entropy lowers α, producing greedier actions that can speed up learning but risk premature convergence. The target entropy was set to -0.5 (default is -1), encouraging a slightly higher exploration.

The learning rate controls how big each parameter update is. Higher values speed up learning but risk unstable critics/actors, especially with noisy returns, as in our case. Lower values make training steadier and reduce overshooting, but can slow convergence or get stuck in suboptimal policies. We chose a learning rate of $1e^{-4}$, which is a conservative choice but necessary due to the highly random nature of our rewards, as observed in the CNN training (slightly above 50% accuracy).

The batch size is the number of samples (state-action-reward transitions) used in a single gradient update during training. Larger batches reduce gradient noise (better signal), while smaller batches add noise that can act as regularization. We chose a batch size of 64, typically considered small, which is paired with a conservative learning rate.

Rewards are normalized and clipped (limited to $[-5, 5]$), and this helps reduce the impact of outliers and makes training more stable. This is particularly important for our environment since financial market returns are known for their Student-t distributions [66], which are often largely heavy-tailed. Clipping prevents extreme returns from dominating critic targets and destabilizing the policy. Observations are left unnormalized since images are already scaled $[0, 1]$.

Reproducibility is addressed on multiple levels. We fix a global seed, seed NumPy [67], define Python's random [68], PyTorch [69] and Stable Baselines' [64] utilities, and configure cuDNN [70] for determinism. We also seed the environment creation so that episode start indices are replicable under identical conditions.

The total training runs are constrained by how many full weeks of training we can extract from the data and by computational restraints. With 97.556 (image, return) pairs

in the training dataset and episodes formed as strictly consecutive sequences of length 117, the number of possible starting points is 97.439. If we were to advance the start by a single observation each time, the entire training would take $97.439 * 117 = 11.400.363$ timesteps, which is infeasible. To reduce this, while keeping the coverage across the dataset, we divide the possible starting indexes by a fifth: $117/5 = 23.4$. Advancing one episode every 23 observations reduces the total number of timesteps by a factor of 23, leaving us with $11.400.363/23 \approx 495.668$ timesteps. We round to a clean target of 500.000 timesteps for training.

Checkpointing and normalization statistics are saved alongside the model so that evaluation uses the same normalization state as training.

CHAPTER 5

Results

The following chapter presents the training logs and evaluation of the proposed system. The discussion of the results presented can be found in Chapter 6. The system described in Chapter 4 is tested under four different setups to compare how their differences affect performance. The four setups are:

(1) CNN Pre-training + RL Fine-Tuning (Baseline): this setup matches the system described so far, where the CNN is pre-trained before being integrated with the RL model and fine-tuned as a feature extractor. From this point onwards it is considered as the baseline for comparison with other setups.

(2) RL Training (no Pre-training): the second setup does not apply the pre-trained CNN weights and instead aims to train the feature extractor alongside the RL networks from the start. This corresponds to the original SAC implementation in the Stable Baselines library [65]. It serves as a comparison to measure the impact of pre-training.

(3) CNN Pre-training + RL with Frozen Encoder: this setup employs the pre-trained CNN weights, however it freezes the feature extractor and only updates the SAC networks during training (does not backpropagate loss to the CNN). It measures the impact of fine-tuning the CNN within the RL model.

(4) CNN Pre-training + RL Fine-Tuning with balanced training dataset: in the final setup, the training dataset is balanced through undersampling, with the number of positive and negative return samples being equal (similar categories to the CNN training). Since a bias might arise from the training set, balancing it might help the model output a more even performance across the action space.

5.1. Reinforcement Learning Training Metrics

As mentioned in Section 4.7, the model is trained for a total of 500.000 timesteps. Throughout training, we collect metrics to assess model convergence. The collected metrics include: mean reward per episode, critic loss, actor loss, entropy coefficient and entropy coefficient loss.

The mean reward per episode represents the average cumulative reward obtained per episode. The value is computed internally by the Stable Baselines package, using a rolling window that aggregates returns for the last 100 episodes, thereby providing a direct indicator of learning progress.

The critic loss reflects the difference between the predicted state-action values produced by the critic network and the target values estimated from bootstrapped returns.

A lower loss indicates that the value function is accurately approximating the expected returns, while higher/diverging values suggest instability in training or poor estimates. A healthy training progress will converge toward lower losses, with occasional small spikes reflecting network updates.

The actor loss indicates how well the policy network is optimizing its action selection to maximize expected rewards. A converging loss generally indicates that the network is producing good actions, aligning with the gradients of the objective, while instability in this loss points to difficulties in balancing exploration, reward scaling, or value estimation during training.

The entropy coefficient regulates the trade-off between reward maximization and policy entropy, encouraging exploration by preventing the policy from collapsing too quickly to deterministic actions. The associated loss value reflects how the coefficient is being adapted during training. Higher absolute losses indicate stronger updates to maintain the target entropy and lower values suggest that the coefficient has stabilized as the policy approaches the desired balance between exploration and exploitation.

5.1.1. CNN Pre-training + RL Fine-Tuning (Baseline)

Figures 5.1 to 5.5 represent the evolution of each collected metric throughout the 500.000 timestep training of the baseline model.

Figure 5.1 demonstrates that the agent's performance stabilizes with positive cumulative rewards from the 100.000 step onwards. Performance converges to rewards between +0.01 and +0.02.

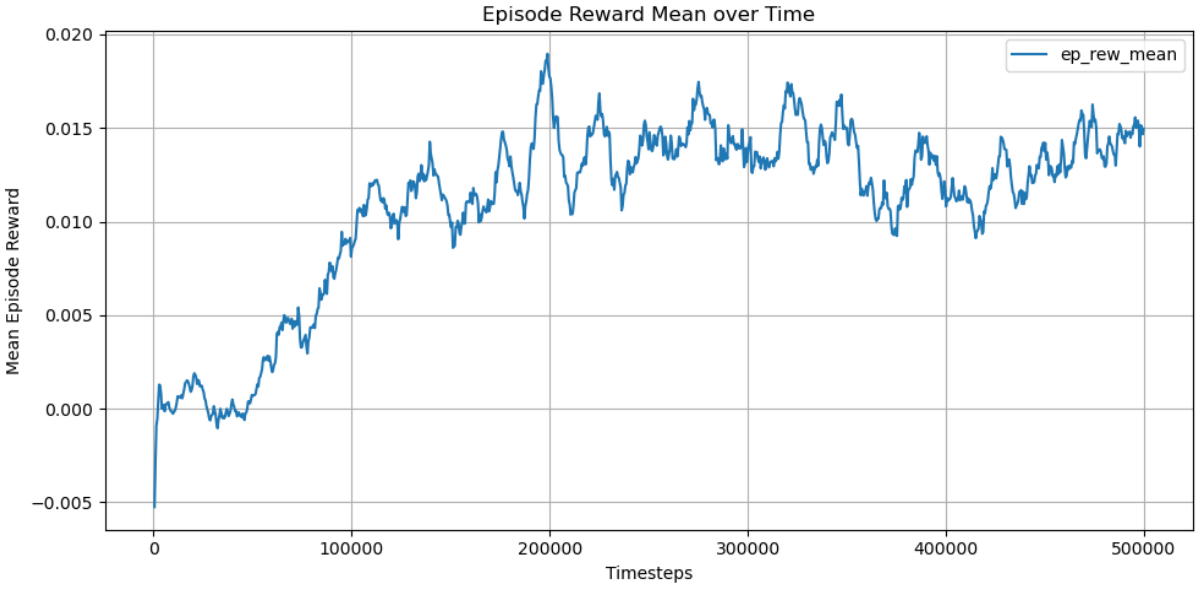


FIGURE 5.1. Episode reward mean evolution in training - CNN Pre-training + RL Fine-Tuning (Baseline).

Figure 5.2 shows the agent's critic loss stabilizing over time. The loss policy takes about 5.000 timesteps to converge towards values close to zero. After this, it shows small

spikes in the loss, which is positive, as it means that the model is still learning and has not stalled.

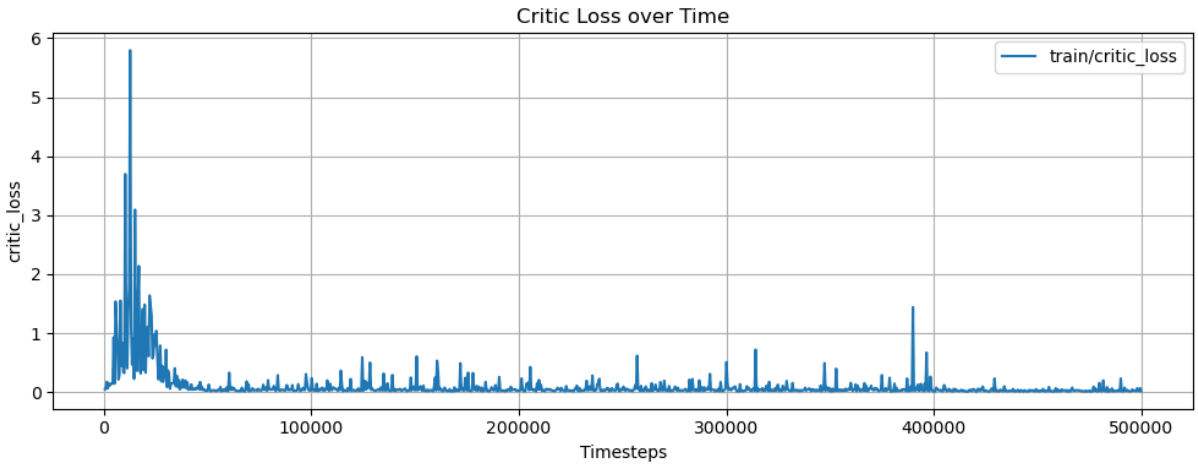


FIGURE 5.2. Critic loss evolution in training - CNN Pre-training + RL Fine-Tuning (Baseline).

Figure 5.3 displays unstable losses in the actor network for the first 100.000 timesteps, but stabilizes onwards.

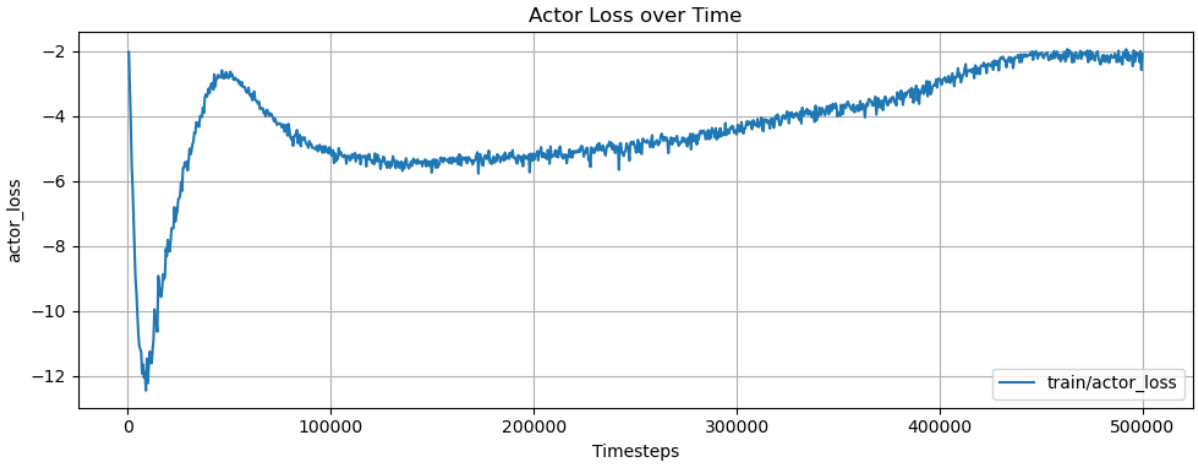


FIGURE 5.3. Actor loss evolution in training - CNN Pre-training + RL Fine-Tuning (Baseline).

Figure 5.4 and Figure 5.5 demonstrate that the entropy coefficient converges to values close to zero and losses stabilize within a short range $[-2, 2]$.

5.1.2. RL Training (no Pre-training)

In the second setup the agent is initialized without the pre-trained weights. Figure 5.6 demonstrates that the rewards for this setup fail to stabilize. Rewards range between -0.001 and +0.001 throughout most of the training period. This is the only setup where the agent's performance does not converge towards positive rewards.

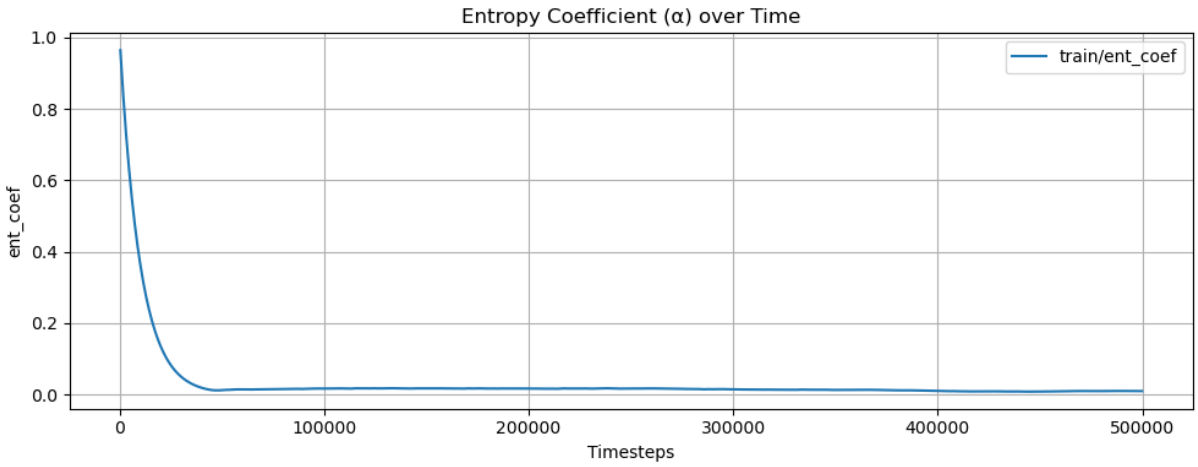


FIGURE 5.4. Entropy coefficient evolution in training - CNN Pre-training + RL Fine-Tuning (Baseline).

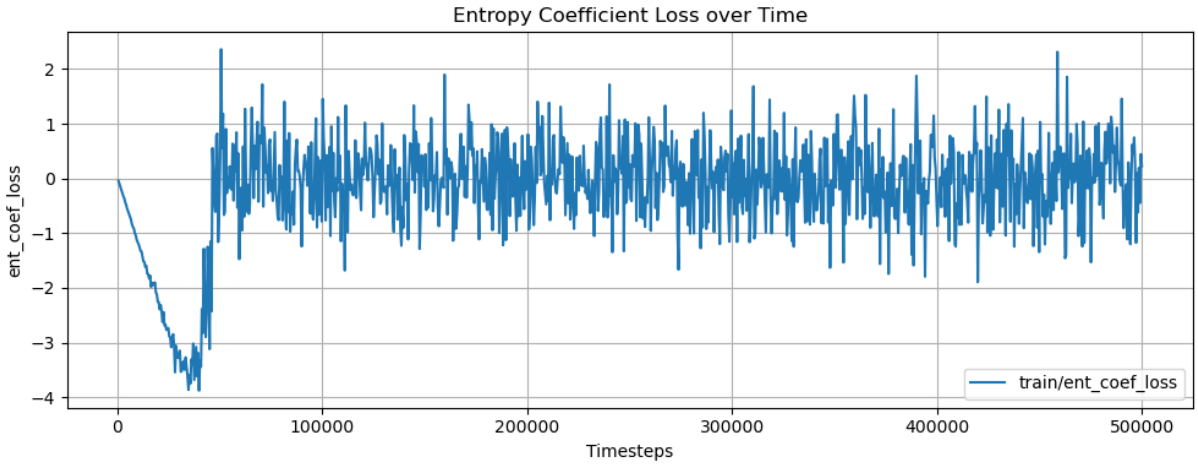


FIGURE 5.5. Entropy coefficient loss evolution in training - CNN Pre-training + RL Fine-Tuning (Baseline).

5.1.3. CNN Pre-training + RL with Frozen Encoder

For the third setup, Figure 5.7 shows the agent's performance improving throughout training, terminating at the +0.015 reward mark. This setup takes longer to converge towards the +0.01 reward mark than the baseline. Figure 5.1 demonstrates the first setup achieving this level from the 100.000 timestep onwards, whereas this setup only achieves it at the 400.000 timestep.

5.1.4. CNN Pre-training + RL with Balanced Training Dataset

Finally, for the last setup, Figure 5.8 demonstrates the agent's performance improving throughout training, consistently within the range +0.015 to +0.02 from the 300.000 timestep onwards. This setup converges before the first 100.000 timesteps, being the fastest to do so. It also maintains a range between +0.015 and +0.02 from the 350.000 timestep onwards, being the only one that is able to achieve this level. Therefore, this model becomes the most promising one for evaluation.

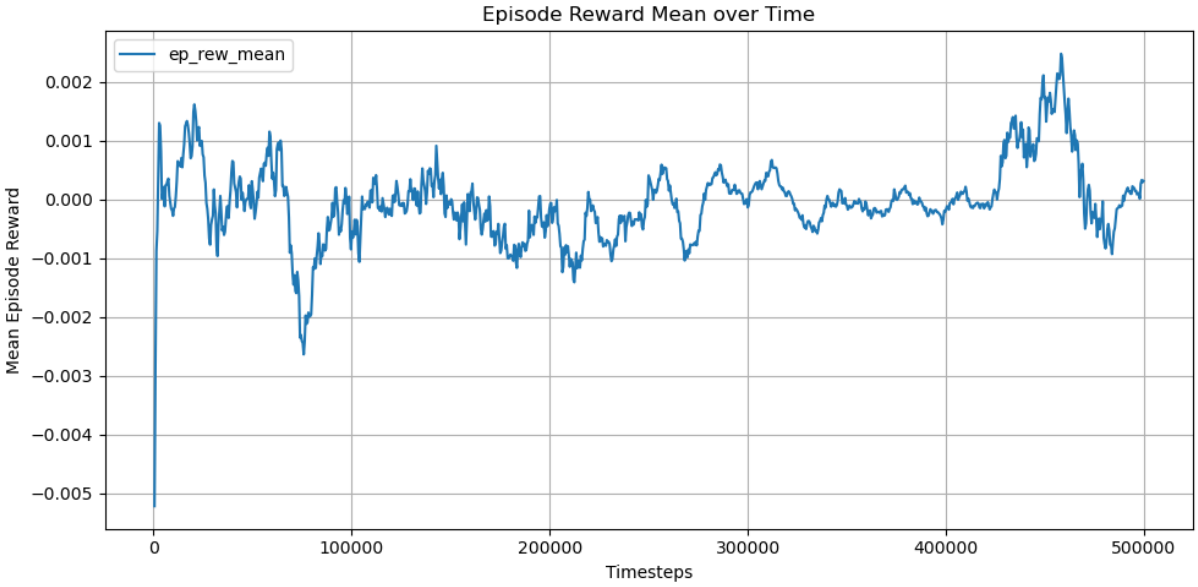


FIGURE 5.6. Episode reward mean evolution in training - RL Training (no Pre-training).

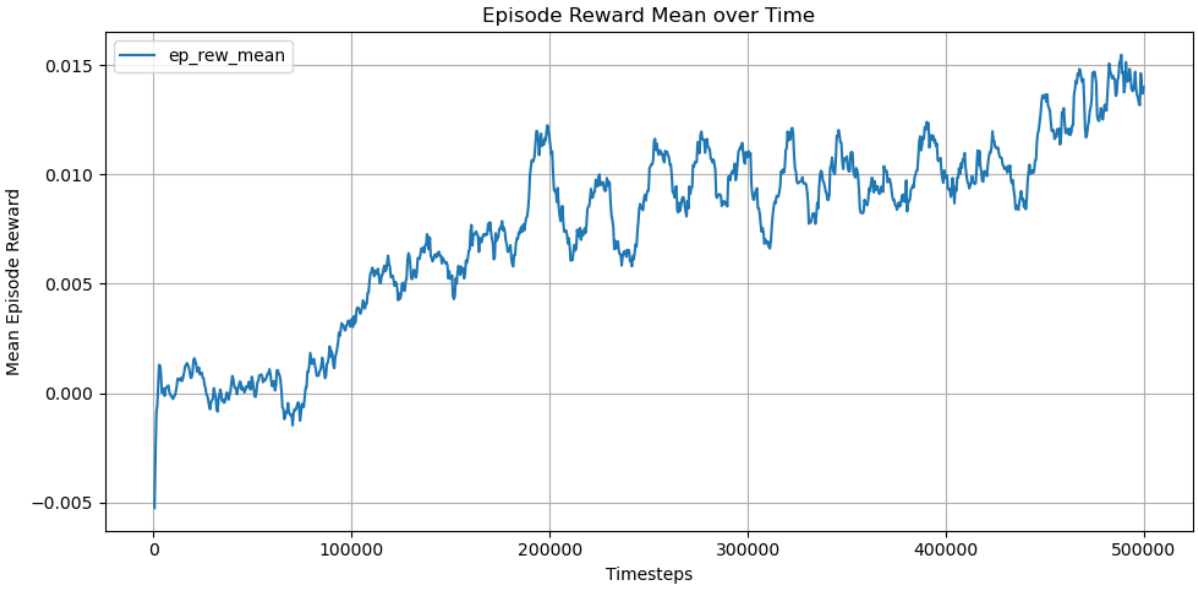


FIGURE 5.7. Episode reward mean evolution in training - CNN Pre-training + RL with Frozen Encoder.

5.2. System Performance

From the training logs gathered in Section 5.1, we can move on to the evaluation of the models developed on the test set, and assess their real-world performance.

Our test set contains data from 16 July 2024 until 28 July 2025, with a total of 6.415 image-return pairs. The model and normalization statistics saved after training (as mentioned in Section 5.2) are employed to generate predictions. These are generated in a sequential order to maintain the correct timeline and sequence of events. During

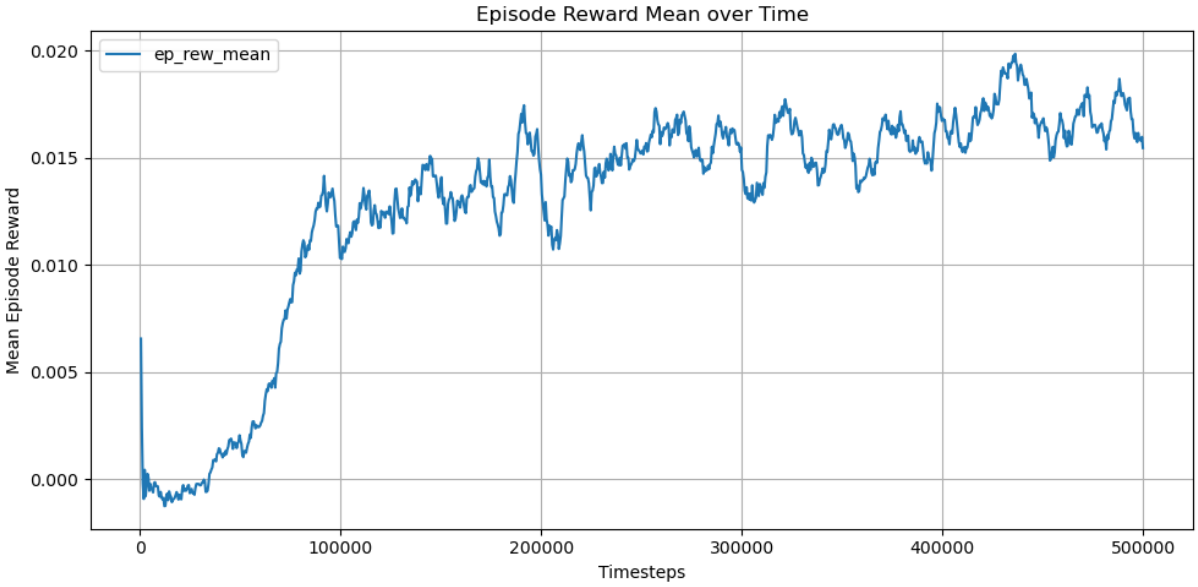


FIGURE 5.8. Episode reward mean evolution in training - CNN Pre-training + RL with Balanced Training Dataset.

this process, we collect the actions taken by each agent, the rewards associated and the cumulative value of the portfolio, considering a start at 100 "points".

The price evolution of the EUR/USD pair in the test set is displayed in Figure 5.9. The pair presents three main trends within the test period. The first, observed between July 2024 and October 2024, sees the pair rise from 1.09 to 1.12, a modest 2.7% gain. Following this period, the pair enters a downtrend from 1.12 to a bottom of 1.02 in January 2025. This corresponds to a 9% loss. The third trend, is observed from February 2025 to July 2025, when the pair rises from 1.02 to a peak of 1.18, a 15.5% gain.

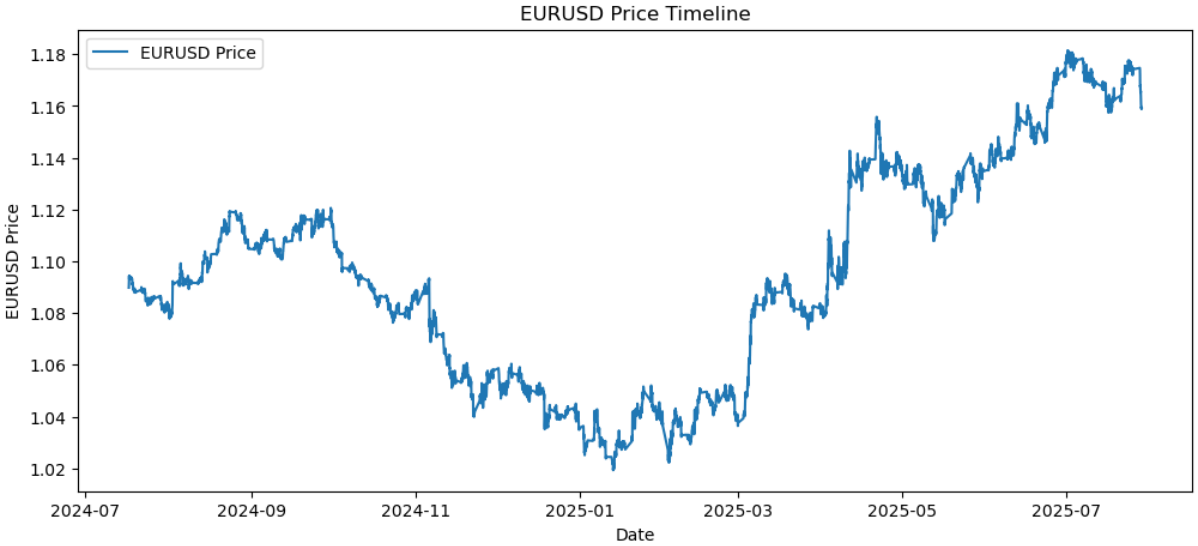


FIGURE 5.9. EUR/USD price evolution in test set.

5.2.1. RL Training (no Pre-training)

Starting with the only model that does not converge towards a positive expected reward. The model's actions collapse towards a fixed value, corresponding to a constant -20% allocation. This is a common occorrence for RL models that do not converge towards a positive reward. Since the model is deterministic, as it fails to find a policy that generates a positive expected reward, it collapses towards a constant action. Due to this occurrance, we consider the test invalid for evaluation since policy is not realistic.

5.2.2. CNN Pre-training + RL with Frozen Encoder

For the setup with the frozen encoder, the one that takes longer to converge towards positive reward values (Figure 5.7), we find that the agent adopts large actions on both sides of the sprectrum (Figure 5.10). We can also see that the model takes more extreme negative actions than positive ones.

As for performance, it can be seen in Figure 5.11 that the evaluation presents a negative final return in the test set, corresponding to about 92% of the initial portfolio value. The model presents a positive performance up until April 2025, this period coincides with the EUR/USD down trend seen in Figure 5.9.

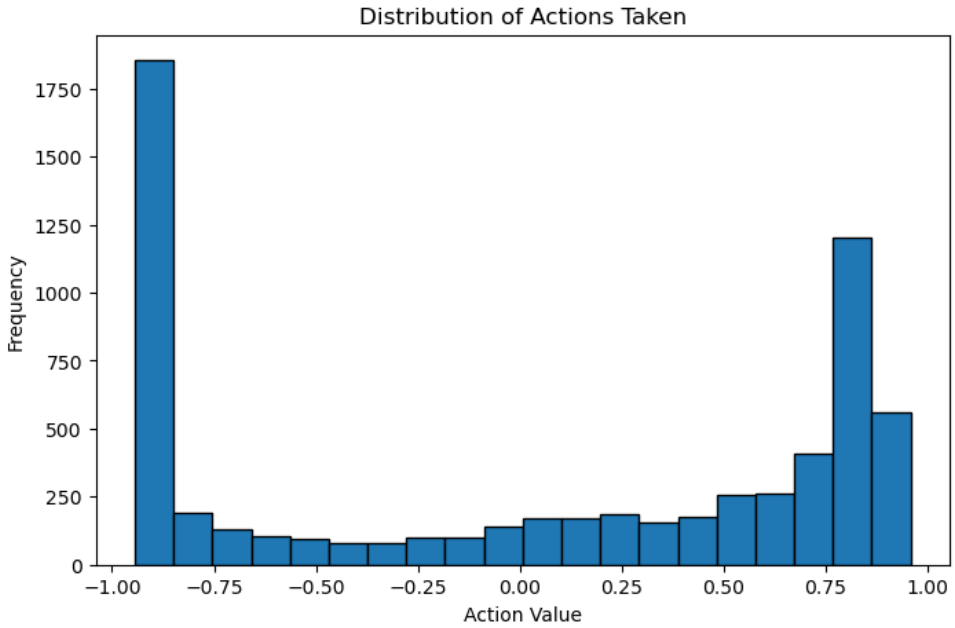


FIGURE 5.10. Distribution of actions taken - CNN Pre-training + RL with Frozen Encoder.

5.2.3. CNN Pre-training + RL Fine-Tuning (Baseline)

For the baseline model, Figure 5.12 demonstrates a more even distribution of actions, with a slight tilt towards negative allocations (the average action is -9%). Similarly to the setup with the frozen encoder, the model's performance is negative in the test set (Figure 5.13), with the final portfolio value being approximately 97% of the initial value. The model also displays a positive performance up until April 2025 and deteriorates afterwards.

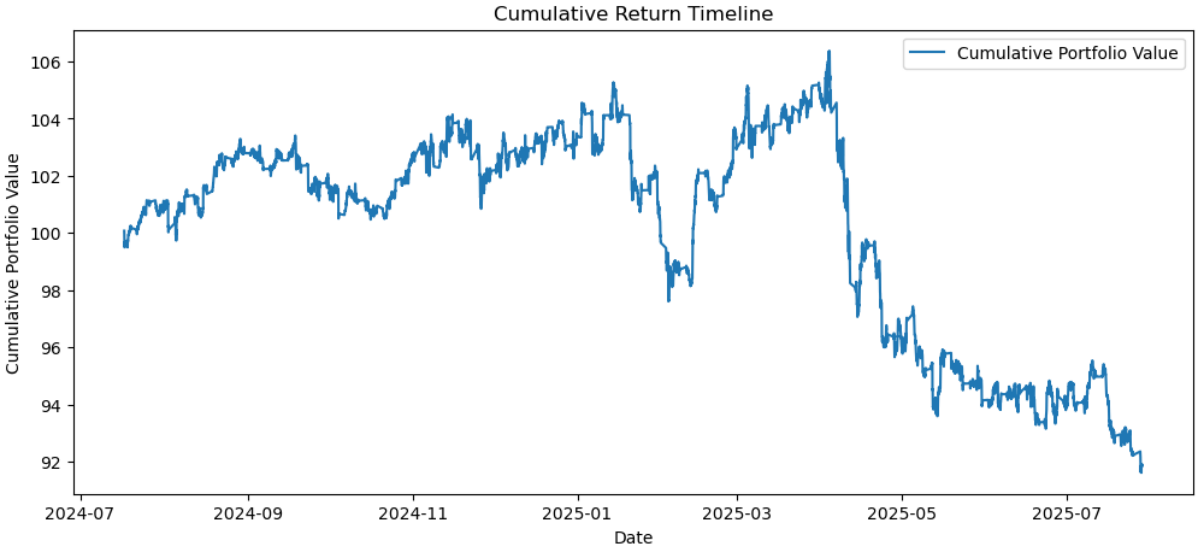


FIGURE 5.11. Cumulative portfolio return - CNN Pre-training + RL with Frozen Encoder.

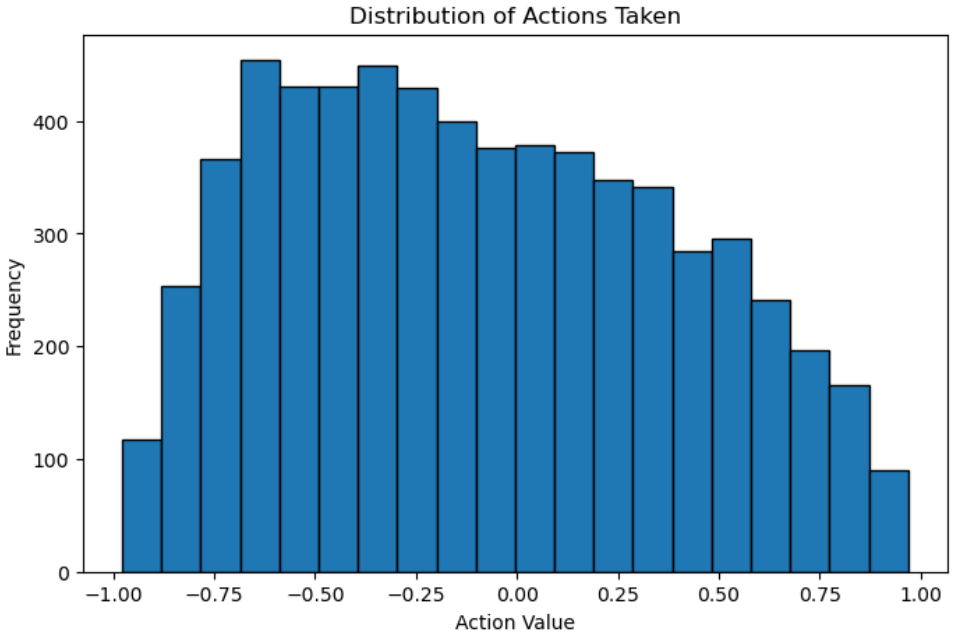


FIGURE 5.12. Distribution of actions taken - CNN Pre-training + RL Fine-Tuning (Baseline).

Examining further the association between the actions taken and the return at each timestep, we construct the confusion matrix presented in Figure 5.14. We find that the model mostly outputs negative actions (58%), whilst most test set returns are actually positive (54%). This makes the precision high (55%), whilst recall remains low (43%), using the positive class as target.

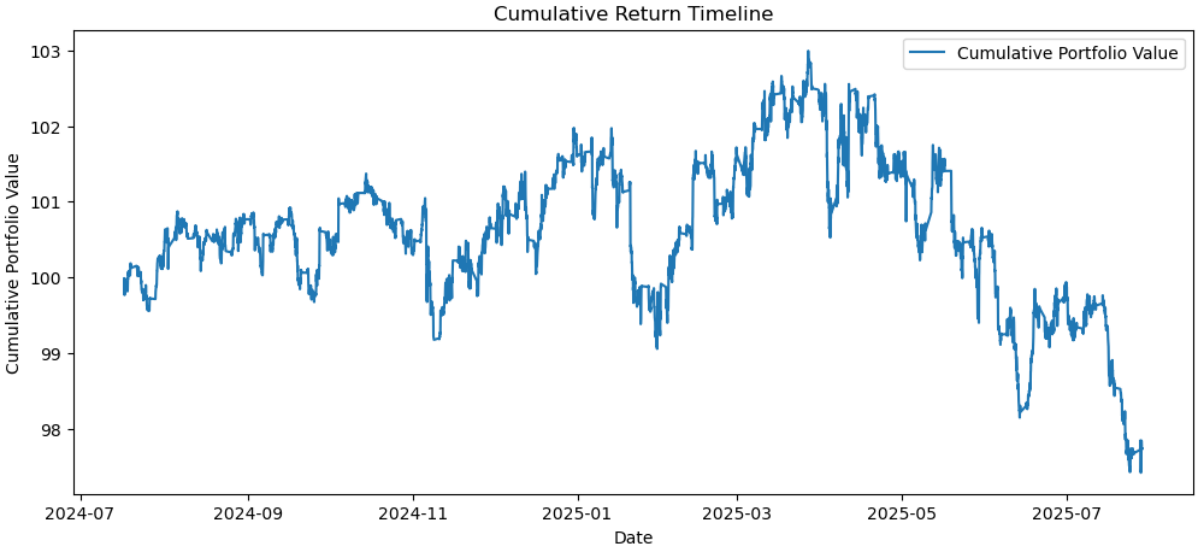


FIGURE 5.13. Cumulative portfolio return - CNN Pre-training + RL Fine-Tuning (Baseline).

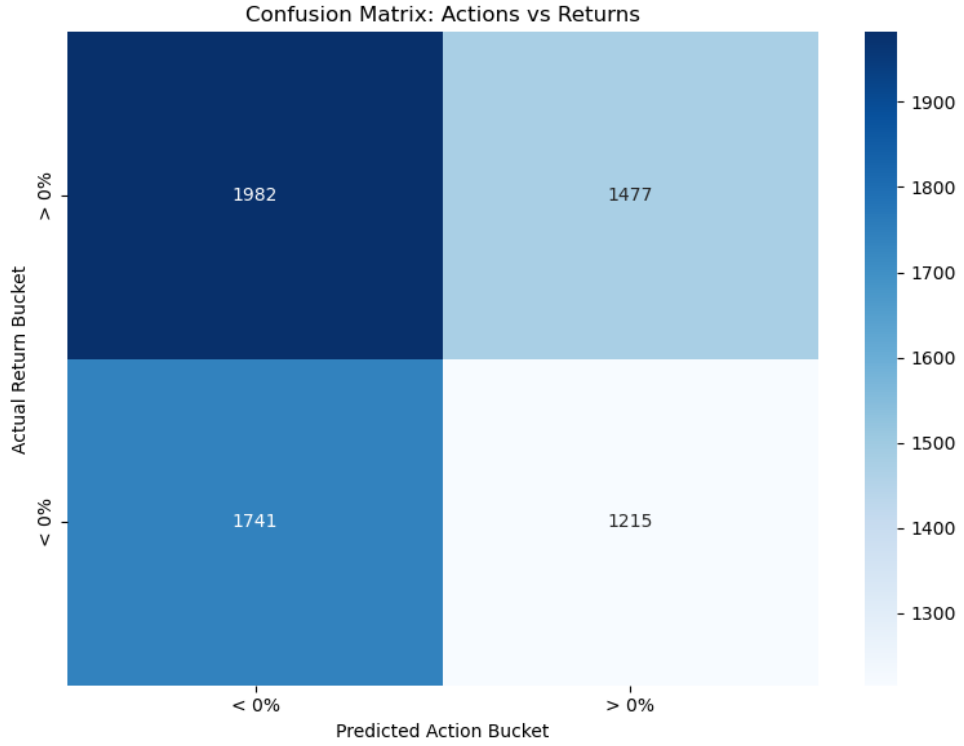


FIGURE 5.14. Confusion Matrix of actions and returns - CNN Pre-training + RL Fine-Tuning (Baseline). The label '>0' represents the positive class and '<0' the negative class.

5.2.4. CNN Pre-training + RL with Balanced Training Dataset

Lastly, we analyse the performance of the model that is trained on a balanced dataset. This model is the one that converges faster and to the highest reward range in the training set.

Although the model is trained on a balanced set, the distribution of actions taken (presented in Figure 5.15) in the test set is slightly more biased than the one found for the basline model, with the average allocation standing at -12%.

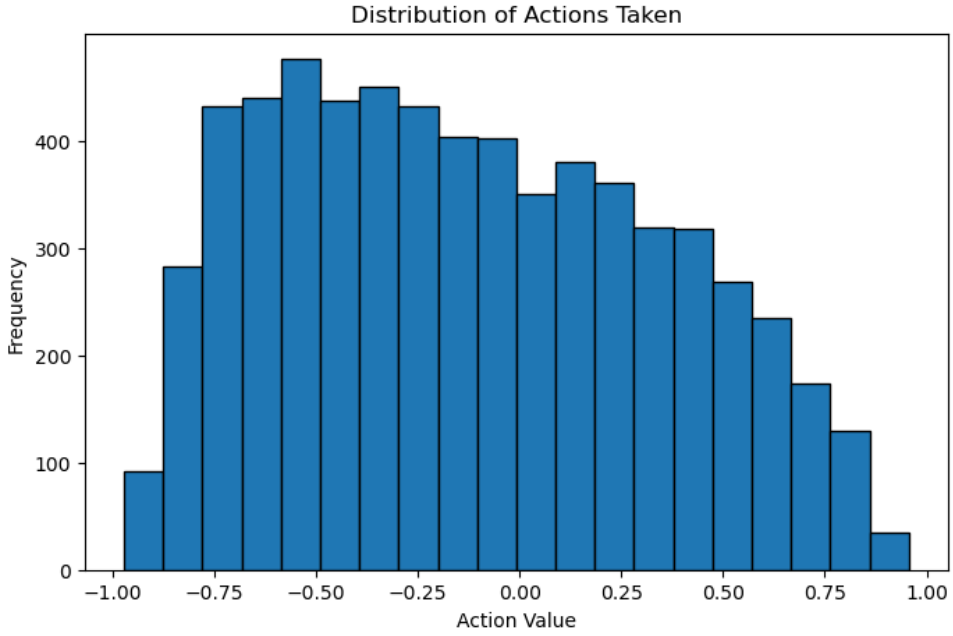


FIGURE 5.15. Distribution of actions taken - CNN Pre-training + RL with Balanced Training Dataset.

Test set performance is the only positive one (shown in Figure 5.16) among the four agents developed. It presents a constant positive drift with the only large drawdown appearing in April 2025. The final portfolio value represents a 4% gain in the space of nearly one year.

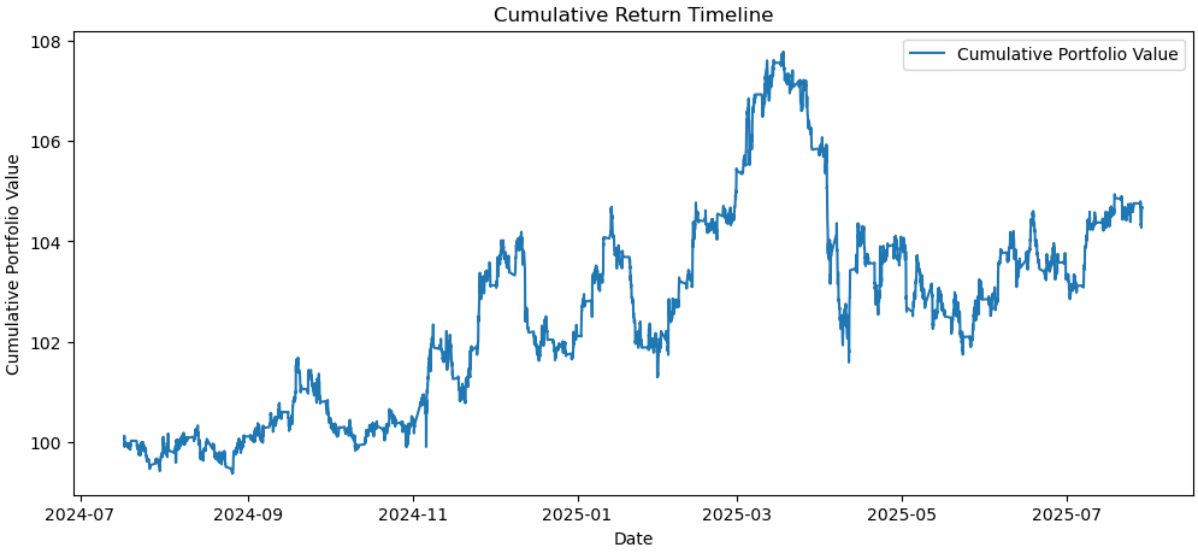


FIGURE 5.16. Cumulative portfolio return - CNN Pre-training + RL with Balanced Training Dataset. The label '>0' represents the positive class and '<0' the negative class.

Constructing a similar confusion matrix as for the baseline model (Figure 5.17), we can observe the agent's bias to output negative actions (61%) in a set where most returns are positive (54%). Despite predicting the smaller class more often than the larger class, the agent's predictions are good enough to have an accuracy that sits just slightly above 50%, at 50.09%.

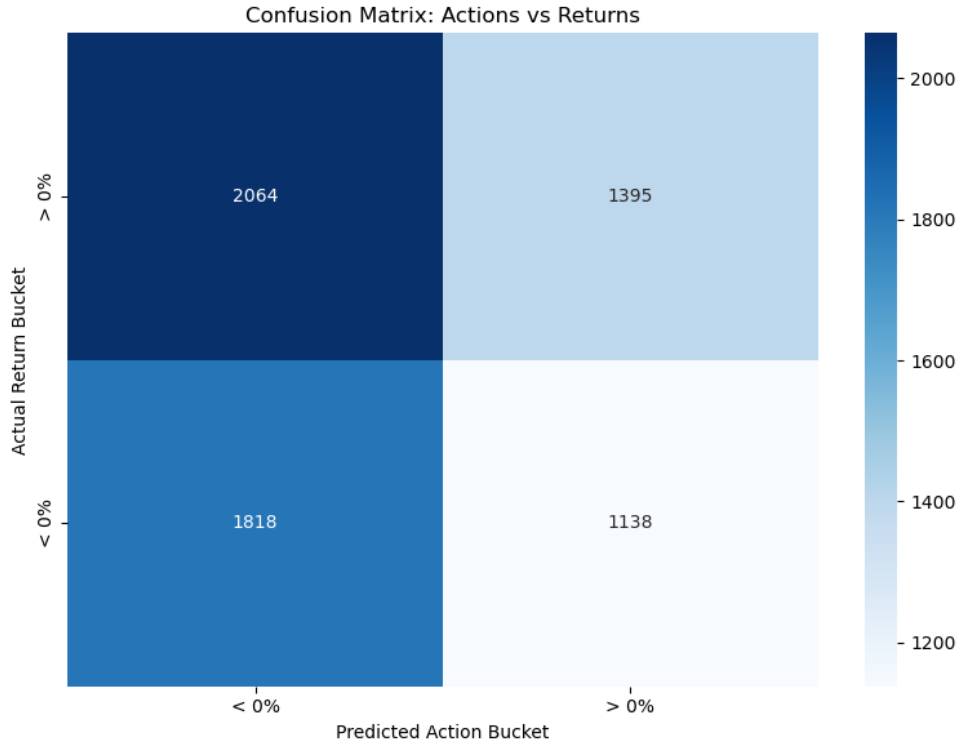


FIGURE 5.17. Confusion Matrix of actions and returns - CNN Pre-training + RL with Balanced Training Dataset.

Since the model is able to achieve a positive performance despite having a strong bias towards the least represented class, it is possible to hypothesize that it is effective in learning about the quality of its own predictions, outputting smaller allocations in lower conviction states.

To dive deeper into the model's performance, we construct a detailed confusion matrix, presented in Figure 5.18. This confusion matrix now separates actions and returns into four categories. Actions are separated into: under -50%, between -50% and 0%, between 0% and 50%, and over 50%. Returns are separated into: under -0.1%, between -0.1% and 0%, between 0% and 0.1%, and over 0.1%.

Table 1 and Table 2 demonstrate the distributions of actions and returns, according to our newly constructed categories. Starting with the returns in Table 1, positive returns have a greater presence of large (over 0.1%) returns (43%), than negative returns (34%). This implies that positive returns should have a larger impact on the final portfolio value. For actions in Table 2, the agent tends to ouput larger allocations for negative actions (43% are over -50% allocations) than for positive ones (31% are over +50% allocations).

TABLE 1. Detailed distribution of returns - CNN Pre-training + RL with Balanced Training Dataset.

Returns	**Count**	**% of Class**	**% of Total**
<-0.1%	1015	34.3%	15.8%
<0%	1941	65.7%	30.3%
>0%	1959	56.6%	30.5%
>+0.1%	1500	43.4%	23.4%

<0%	2956	-	46.1%
>0%	3459	-	53.9%

Total	6415

TABLE 2. Detailed distribution of actions - CNN Pre-training + RL with Balanced Training Dataset.

Actions	**Count**	**% of Class**	**% of Total**
<-50%	1679	43.3%	26.2%
<0%	2203	56.7%	34.3%
>0%	1751	69.1%	27.3%
>+50%	782	30.9%	12.2%

<0%	3882	-	60.5%
>0%	2533	-	39.5%

Total	6415

This suggests that the agent is often more confident in negative allocations than in positive ones.

To dissect the detailed confusion matrix (presented in Figure 5.18), we will go through each action category separately. The objective of the continuous action space is to allow the agent to discern high expected value states, allocating higher proportions of its bankroll to these states. To verify whether the agent is learning this, we should identify better comparative performance for larger allocations in the results.

Starting with negative allocation actions. As mentioned before, these actions represent roughly 60% of the actions taken. Computing the precision (using negative actions as the target category), the rate of true positives within the total of positive predictions, we get a value of 47% (Figure 5.17), which demonstrates that performance is below target for negative allocations.

Negative allocation buckets and their corresponding return distributions are displayed in Figure 5.19. Larger negative allocations (dark red) correspond to a higher percentage of large negative returns and a smaller percentage of large positive returns (represented in the tail buckets). The distribution of returns for these allocations are balanced in

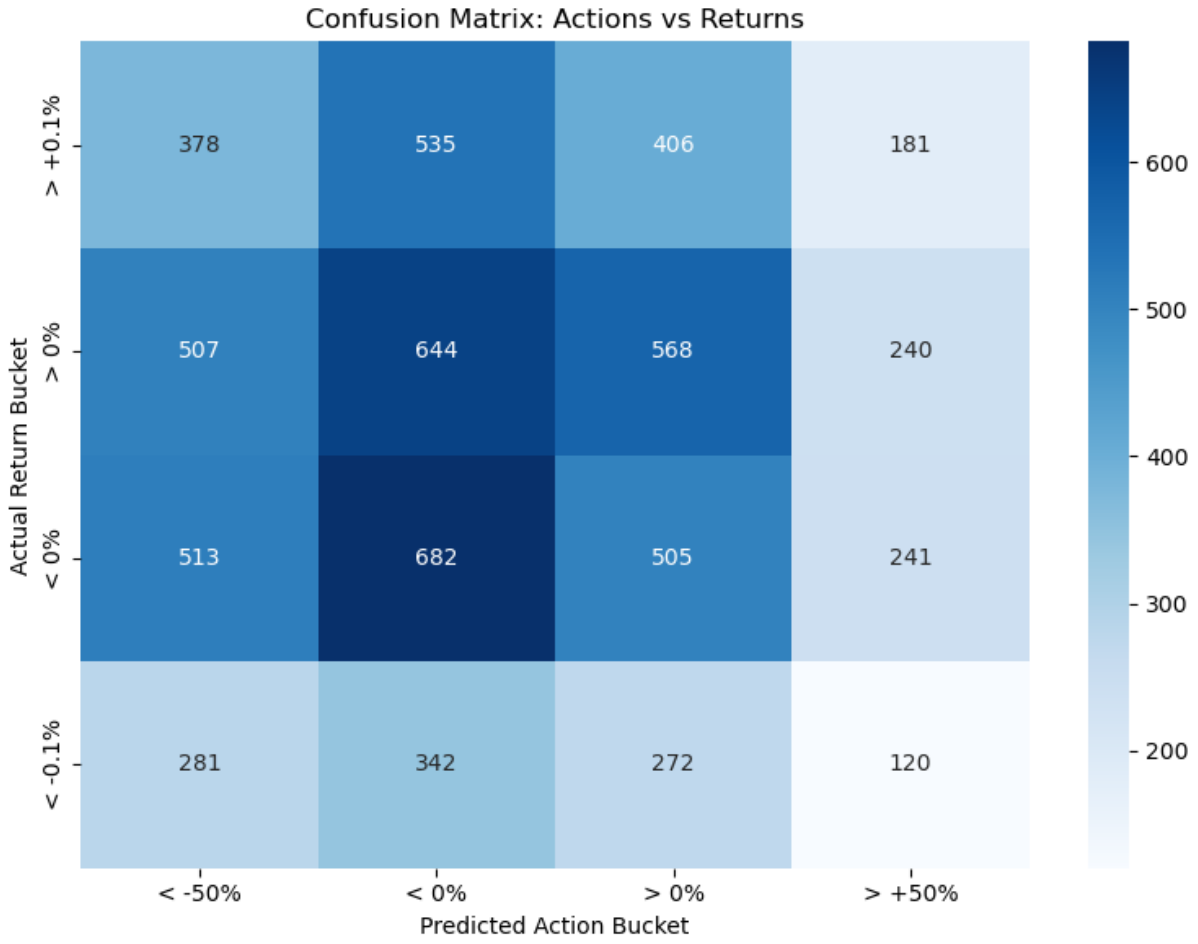


FIGURE 5.18. Detailed Confusion Matrix of actions and returns - CNN Pre-training + RL with Balanced Training Dataset.

the central buckets presented on the bar chart (whilst smaller allocations, in light red, have a slight negative tilt). The outperformance in positive return territory can also be visualized on the Kernel Density Estimate (KDE) plot, particularly when returns are between +0.001 and +0.003. We can see a smaller density for large allocations, in dark red, compared to smaller allocations in light red.

As for positive allocation actions, they represent 40% of the total actions. Precision, for this class, is 55%, showing that the performance is positive when the model predicts this class (Figure 5.17). Positive allocation buckets and their corresponding return distributions are displayed in Figure 5.20. On the bar chart, we find no significant difference in the tails, with both allocation buckets presenting similar performance. As for the central buckets, there is a slight outperformance by the smaller allocations bucket (in light green). This can also be visualized in the KDE plot, where the distribution of smaller allocations is slightly more skewed to the right.

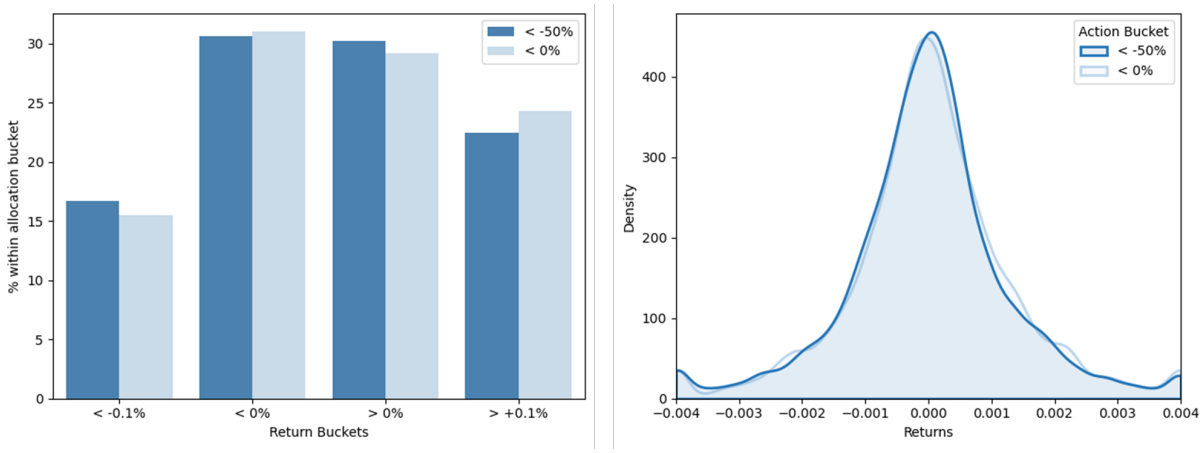


FIGURE 5.19. Bar Plot and Kernel Density Estimate Plot - Negative Allocations - CNN Pre-training + RL with Balanced Training Dataset.

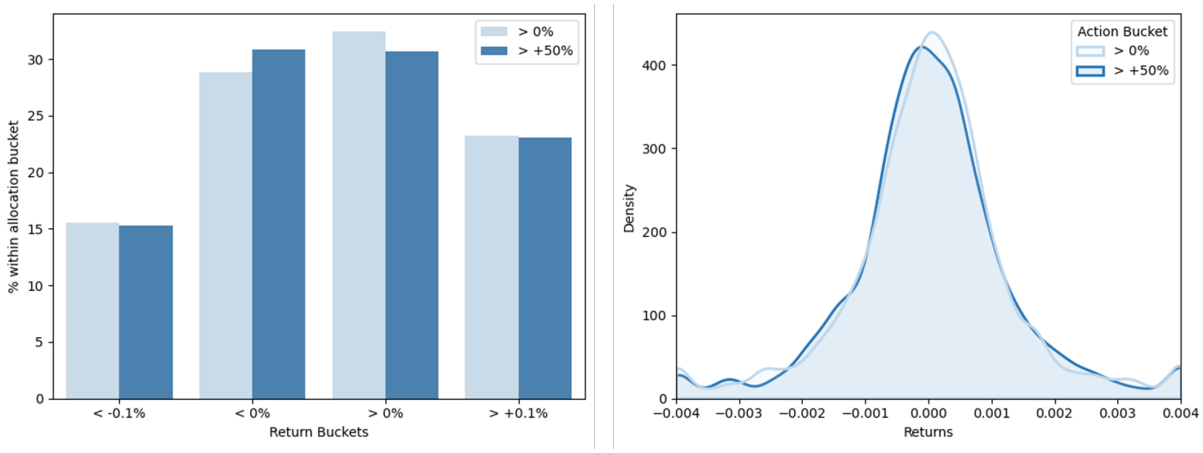


FIGURE 5.20. Bar Plot and Kernel Density Estimate Plot - Positive Allocations - CNN Pre-training + RL with Balanced Training Dataset.

CHAPTER 6

Discussion

This chapter contains a detailed discussion of the results found in Chapter 5. By discussing these results, we seek to answer the research questions posed in Section 1.3.

Regarding the metrics obtained from tracking model convergence in the training set, we find that three out of the four setups presented converge towards positive rewards. This is a positive indicator for the capability of this system in finding positive expected value policies in financial markets, and more specifically the FX market.

The only setup that fails to converge towards a positive reward, is setup number 2. This is the only setup that does not make use of transfer learning. Therefore, it is the only one where the CNN is initiated at the same time as the SAC networks. It presents the challenge of jointly learning perception (feature extraction) and control from scratch, the difficulty of which is elevated by the nature of the environment, since rewards are noisy. The CNN's features drift as the policy explores, which makes both the critic's targets and the actor's gradient unreliable, increasing sample inneficiency and difficulting convergence. As a result, the model is unable to find a policy that maximizes the expected reward, and instead, collapses towards a constant allocation.

This serves as a basis to answer research question number 3: *"Is transfer learning an effective strategy in improving the RL+CV system?"*. Given the gathered evidence, it becomes clear that transfer learning accelerated the convergence of the system. In our case, comparing with the baseline model that converged after 100.000 timesteps, transfer learning accelerated the system's convergence by at least 5 times. There is, however, a drawback that must be taken into consideration. The fact that the system is pre-trained inserts an initial bias into the weights of the CNN that would not be present in a model that does not use transfer learning. There is evidence of this bias in the action distribution of setup number 3 (Figure 5.10), where the feature extractor is not fine-tuned during RL training. The action space presents a bias to more extreme actions, which can be associated with the pre-training supervised learning problem, where the model is challenged to learn a binary label classification.

Overall, the use of transfer learning in this system is a positive indicator in terms of speeding up convergence. However, it should be used carefully, so as to minimize the introduction of bias into the RL model.

All three other setups converge towards positive rewards. There are two main aspects that can be taken into consideration when analysing this convergence: the speeed at which the model converges and the expected reward towards where the model converges. Setup number 3 (CNN Pre-training + RL with Frozen Encoder), presents the slowest

convergence, 400.000 steps to the range [+0.01, +0.015]. The second fastest convergence belongs to setup 1 (baseline), converging after 100.000 steps to the same range. Finally, the fastest convergence belongs to setup 4 (Baseline with Balanced Training Set), converging before the 100.000 steps and finishing in the range [+0.015, +0.02].

These metrics point to the importance of fine-tuning the feature extractor alongside the SAC networks, leading to faster convergence for the two models that do so. This also serves as positive evidence towards the effectiveness of balancing the training set, since it allowed the model to converge faster and to higher rewards.

Moving to the evaluation of the models in the test set. Unsurprisingly, we find that the agent that did not converge in training, failling to find a positive expected value policy, collapses towards a constant action.

The worst test set performance belongs to agent 3 (CNNPre-training + RL with Frozen Encoder), which is the one that converged the slowest. This agent attains a final portfolio value of nearly 92 "points", corresponding to an 8% loss. This occurs despite the average action being a +1.5% allocation (test set has more positive returns than negative returns), and the accuracy of the model being 50,4%. There are two main reasons that can justify this performance. First, allocations are more aggressive on the negative actions side (clearly visible in Figure 5.10), which is the side with the worst performance (it has a precision of 46%, compared to a precision of 54% when using positives as the target class). Second, which is related to the first, the effect of large drawdowns when taking on too much exposure. As the model constantly takes on actions closer to the maximum exposure (100% of portfolio value), drops in value become larger, impairing the ability to recover from them (if we lose 1% and then gain 1%, we lose part of the initial value, this effect is non-linear and is exacerbated by larger exposures [71]). This excessive risk-taking is reflected in the difference between the maximum and minimum portfolio values in test set: $106\% - 92\% = 14\%$ (Figure 5.11). This difference is 6% for the baseline model and 8% for agent 4.

The second best performance in the test set is achieved by the baseline setup. The agent loses nearly 3% of the initial portfolio value. This can be mostly attributed to the bias towards negative actions. The agent performs well in the EUR/USD down trend, but performs poorly in the EUR/USD up trend (Figure 5.9).

Finally, the best performance is found for the agent trained in the balanced dataset. The agent earns a 4% return in the space of one year, having achieved a test set high at 8% in March 2025. It presents a large drawdown (-5/6%) in March/April 2025, but a strong performance in all other time periods. This large drawdown is a common occurance, also observable in the other two agents' performance. It can be justified with what came to be known as "Liberation Day" [72], when the United States (USD) unveiled very high tariff rates on all trade partners, including the EU (EUR). This caused a devaluation of the USD [73], with the EUR/USD pair going from 1.08 to 1.15 in the space of one month (Figure 5.9). This is an exogenous shock to what the model is able to capture (charts do

not capture economic news), and reveals the limitations of an approach solely based on image data.

Despite being trained on a balanced dataset, the agent still presents a bias towards negative actions. However, it displays a higher comparative quality. Using negative as the target class, the agent slightly improves the precision metric (46.83% vs. 46.76%). As for positive class precision, a small edge is also visible (55.07 % vs. 54.87%). These improvements/slight edges are extremelly important in an environment where a near 50/50 outcome is expected, allowing the agent to consistently "beat the game" by a small edge. This outperformance helps answer question 1: *"Can a CV-RL combined system unlock positive performance in a non-drift market?"*, as the developed agent is able to achieve a positive return in the test set.

Examining further the model performance, we compare the quality of the agents' actions between smaller and larger allocations. The objective of differenciating between the two types of allocations is understanding whether the continuous action space allows the agent to learn about the quality of its own predictions, allocating higher shares of its portfolio to predictions where it has more confidence. An agent that learns about the quality of its own actions would have a comparatively better performance in large actions than in small actions. Figure 5.19 demomstrates that this is the case for negative actions, where there is an outperformance in the tails of the returns, for the larger allocation actions. However, for positive allocations, in Figure 5.20 there is no visible outperformance in the tails for the large allocation class. There is even a slight underperformance in the distributions' central buckets.

This is in line with the precision values found for negative/positive classes, where the agent clearly finds an advantage in negative actions. This outperformance is reflected in the tails, whereas the underperformance for positive actions is found in the centre of the return's distribution, having a slighter impact on bankroll growth. This allows us to answer question 2: *"Can the developed agent learn about the quality of its own actions?"*. The results found are inconclusive, as the outperformance found for the negative class is not visible in the positive class. Therefore, the outperformance of large allocations is not consistent across the action space.

CHAPTER 7

Conclusions

This dissertation set out to link visual pattern recognition (CNN) with sequential decision-making (RL) in a symetric reward environment. The game developed emulates a trader's decision-making process in the FX market, known for its noisy and non-stationary price dynamics. By transforming price windows into images, extracting features from them with a CNN, and passing those features to a SAC agent that outputs continuous position sizes, this work tested whether a vision-driven RL agent can allocate a bankroll profitably without relying on the positive drift found in stock markets. The constructed pipeline was trained on a nearly 20-year set and assessed out-of-sample, so that conclusions reflect behavior in realistic conditions.

We compare four configurations of the system: pre-train the CNN and fine-tune all layers end-to-end, train both CNN and RL jointly from scratch, use a pre-trained CNN frozen as a fixed feature extractor while training RL and fine-tune both components end-to-end on a class-balanced dataset.

Across setups, the results were clear on two fronts. First, transfer learning mattered: training the encoder and then fine-tuning it inside the RL loop sped convergence and raised reward levels, whereas learning the features from scratch failed to stabilize and collapsed toward a fixed policy. Second, the approach with the balanced dataset generalized best: it converged fastest during training and delivered a steady gain of roughly 4% over the one year test window. This occurred despite a pronounced drawdown in April 2025 during a tariff-driven dollar move.

Freezing the encoder produced over-aggressive exposures and negative performance, raising questions regarding possible bias induced by pre-training. The baseline setup ended slightly down, with a persistent tilt toward negative allocations, whereas the test set is mostly composed by positive returns.

Taken together, the findings address the research questions posed, although not completely. A combined CV-RL system can indeed produce positive performance in a non-drift market, when trained on a balanced set and with a visual encoder both pre-trained and adapted with the policy. The agent showed partial signs of benefitting from a continuous action space: larger negative allocations aligned with better outcomes in the tails, but that pattern did not extend to the positive side. Finally, transfer learning was unambiguously helpful in this setting, accelerating learning and improving stability when compared to training the feature extractor from scratch.

The work's contributions address the gaps found in the current literature with the use of continuous action spaces and an application in a symetric reward environment, whilst

also exploring the use of transfer learning. It extends both academic understanding and practical advances in automated trading systems.

Important limitations also emerge from the research presented. First, relying on images alone leaves the agent blind to macroeconomic events and policy shocks, which explains part of the drawdown during the April 2025 tariff episode. Second, exposure was sometimes excessive, underscoring the need for risk-aware learning objectives. Third, hyperparameterization was not tested, leaving room for further optimizing models and producing better agents. In line with the lack of hyperparameterization, the use of gamma scheduling could have been particularly important for optimizing the setup where both the CNN and the RL model are trained from scratch. Using higher discount rates early in training would incentivize short-term rewards, leaning the model towards learning states that lead to positive/negative returns, similarly to what happens in the CNN pre-training. Later, reducing discount rates would direct the agent towards a more global vision of final portfolio value optimization.

Future work can expand the state with non-visual signals, reducing blindness to exogenous events. Optimizing directly for a risk-adjusted reward or adding penalties could also help improve policy. The use of hyperparameter optimization and gamma scheduling can help improve agent performance. Broader validation is also needed, therefore testing across currency pairs and markets would separate idiosyncratic EUR/USD behavior from a generalizable signal.

Ultimately, this dissertation shows that a vision-driven RL agent can find a small, persistent edge in a symmetric reward environment. The approach is promising but not complete: its full potential lies in moving beyond images toward richer, multi-modal perception and in making risk dynamics explicit in the learning targets.

Bibliography

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning* (Adaptive Computation and Machine Learning series), 2nd ed. Cambridge, MA: Bradford Books, Nov. 2018.

[2] T. A. Schmitt, D. Chetalova, R. Schäfer, and T. Guhr, "Non-stationarity in financial time series: Generic features and tail behavior," *EPL (Europhysics Letters)*, vol. 103, no. 5, p. 58 003, Sep. 2013, ISSN: 1286-4854. DOI: 10.1209/0295-5075/103/58003. [Online]. Available: http://dx.doi.org/10.1209/0295-5075/103/58003.

[3] A. Wehrli and D. Sornette, "The excess volatility puzzle explained by financial noise amplification from endogenous feedbacks," *Scientific Reports*, vol. 12, Nov. 2022. DOI: 10.1038/s41598-022-20879-0.

[4] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, *Deep reinforcement learning that matters*, 2019. arXiv: 1709.06560 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1709.06560.

[5] J. J. Murphy and J. J. Murphy, *Technical analysis of the financial markets*. Fishkill, N.Y.: New York Institute of Finance, 1999, ISBN: 0735200661 9780735200661. [Online]. Available: http://www.worldcat.org/search?qt=worldcat_org_all&q=0735200661.

[6] K. Cui, R. Hao, Y. Huang, J. Li, and Y. Song, "A novel convolutional neural networks for stock trading based on ddqn algorithm," *IEEE Access*, vol. 11, pp. 32 308–32 318, 2023, ISSN: 21693536. DOI: 10.1109/ACCESS.2023.3259424.

[7] D. Chambers, E. Dimson, A. Ilmanen, and P. Rintamäki, "Long-run asset returns," *Annual Review of Financial Economics*, vol. 16, pp. 435–458, Nov. 2024. DOI: 10.1146/annurev-financial-082123-105515.

[8] J. Huang, J. Chai, and S. Cho, "Deep learning in finance and banking: A literature review and classification," *Frontiers of Business Research in China*, vol. 14, no. 1, 2020. DOI: 10.1186/s11782-020-00082-6.

[9] O. B. Sezer and A. M. Ozbayoglu, "Algorithmic financial trading with deep convolutional neural networks: Time series to image conversion approach," *Applied Soft Computing Journal*, vol. 70, pp. 525–538, Sep. 2018, ISSN: 15684946. DOI: 10.1016/j.asoc.2018.04.024.

[10] J. Jiang, B. Kelly, and D. Xiu, "(re-)imag(in)ing price trends," *The Journal of Finance*, vol. 78, no. 6, pp. 3193–3249, 2023. DOI: https://doi.org/10.1111/jofi.13268. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/jofi.13268. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/jofi.13268.

[11] Z. Jiang, D. Xu, and J. Liang, "A deep reinforcement learning framework for the financial portfolio management problem," Jun. 2017. [Online]. Available: http://arxiv.org/abs/1706.10059.

[12] Y. Deng, F. Bao, Y. Kong, Z. Ren, and Q. Dai, "Deep direct reinforcement learning for financial signal representation and trading," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, pp. 653–664, 3 Mar. 2017, ISSN: 21622388. DOI: 10.1109/TNNLS.2016.2522401.

[13] M. Mortaji, A. Khiat, and M. Benhouad, "Reinforcement learning application in portfolio optimization: A comprehensive literature review," in *2024 International Conference on Intelligent Systems and Computer Vision, ISCV 2024*, Institute of Electrical and Electronics Engineers Inc., 2024, ISBN: 9798350350180. DOI: 10.1109/ISCV60512.2024.10620129.

[14] S. R. Polamuri, D. K. Srinivas, and D. A. K. Mohan, "Multi-model generative adversarial network hybrid prediction algorithm (mmgan-hpa) for stock market prices prediction," *Journal of King Saud University - Computer and Information Sciences*, vol. 34, pp. 7433–7444, 9 Oct. 2022, ISSN: 22131248. DOI: 10.1016/j.jksuci.2021.07.001.

[15] N. Vullam, K. Yakubreddy, S. S. Vellela, S. K. K. Basha, B. V. Reddy, and S. S. Priya, "Prediction and analysis using a hybrid model for stock market," in *2023 3rd International Conference on Intelligent Technologies, CONIT 2023*, Institute of Electrical and Electronics Engineers Inc., 2023, ISBN: 9798350338607. DOI: 10.1109/CONIT59222.2023.10205638.

[16] J. Fernando, G. Scott, and K. Munichiello, *Sharpe ratio: Definition, formula, and examples*, May 2025. [Online]. Available: https://www.investopedia.com/terms/s/sharperatio.asp.

[17] L. Harris, *Trading and exchanges* (Financial Management Association Survey and Synthesis Series), en. New York, NY: Oxford University Press, Oct. 2002.

[18] A. Hayes, A. Smith, and K. R. Schmitt, *Financial markets: Role in the economy, importance, types, and examples*, Aug. 2025. [Online]. Available: https://www.investopedia.com/terms/f/financial-market.asp.

[19] R. Kumar, *Strategies of banks and other financial institutions*. San Diego, CA: Academic Press, Jul. 2014.

[20] *Triennial central bank survey of foreign exchange and over-the-counter (otc) derivatives markets in 2022*, 2022. [Online]. Available: https://www.bis.org/statistics/rpfx22.htm.

[21] J. Chen, *What is a currency pair? major, minor, and exotic examples*, 2024. [Online]. Available: https://www.investopedia.com/terms/c/currencypair.asp.

[22] T. Weithers, "Foreign exchange," en, in (Wiley Finance), T. Weithers, Ed., Wiley Finance. Nashville, TN: John Wiley & Sons, Jun. 2006, ch. 4.

[23] J. Chen, G. Scott, and S. Kvilhaug, *Gap risk: What it means, how it works, example*, Dec. 2023. [Online]. Available: https://www.investopedia.com/terms/g/gaprisk.asp.

[24] D. Knipl, *Liquidity risk under stress: Beyond bid-ask spreads*, Aug. 2019. [Online]. Available: https://www.msci.com/research-and-insights/blog-post/liquidity-risk-under-stress-beyond-bid-ask-spreads.

[25] A. Hayes, M. James, and K. R. Schmitt, *What is financial leverage, and why is it important?* Jun. 2025. [Online]. Available: https://www.investopedia.com/terms/l/leverage.asp.

[26] F. Zhuang *et al.*, "A comprehensive survey on transfer learning," *Proceedings of the IEEE*, vol. 109, no. 1, pp. 43–76, 2021. DOI: 10.1109/JPROC.2020.3004555.

[27] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848.

[28] K. Simonyan and A. Zisserman, *Very deep convolutional networks for large-scale image recognition*, 2015. arXiv: 1409.1556 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1409.1556.

[29] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, 2015. arXiv: 1512.03385 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1512.03385.

[30] C. M. Bishop, *Pattern recognition and machine learning*. Springer, 2016.

[31] H. K. Bharadwaj *et al.*, "A review on the role of machine learning in enabling iot based healthcare applications," *IEEE Access*, vol. 9, pp. 38 859–38 890, 2021. DOI: 10.1109/ACCESS.2021.3059858.

[32] M. Soori, B. Arezoo, and R. Dastres, "Artificial intelligence, machine learning and deep learning in advanced robotics, a review," *Cognitive Robotics*, vol. 3, pp. 54–70, 2023, ISSN: 2667-2413. DOI: https://doi.org/10.1016/j.cogr.2023.04.001. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2667241323000113.

[33] S. K. Sahu, A. Mokhade, and N. D. Bokde, "An overview of machine learning, deep learning, and reinforcement learning-based techniques in quantitative finance: Recent progress and challenges," *Applied Sciences*, vol. 13, no. 3, 2023, ISSN: 2076-3417. DOI: 10.3390/app13031956. [Online]. Available: https://www.mdpi.com/2076-3417/13/3/1956.

[34] R. Szeliski, *Computer Vision: Algorithms and Applications*, 1st. Berlin, Heidelberg: Springer-Verlag, 2010, ISBN: 1848829345.

[35] D. Canedo and A. J. R. Neves, "Facial expression recognition using computer vision: A systematic review," *Applied Sciences*, vol. 9, no. 21, 2019, ISSN: 2076-3417. DOI: 10.3390/app9214678. [Online]. Available: https://www.mdpi.com/2076-3417/9/21/4678.

[36] J. Olveres *et al.*, "What is new in computer vision and artificial intelligence in medical image analysis applications," en, *Quant. Imaging Med. Surg.*, vol. 11, no. 8, pp. 3830–3853, Aug. 2021.

[37] X. Dong *et al.*, *Why autonomous vehicles are not ready yet: A multi-disciplinary review of problems, attempted solutions, and future directions*, 2025. arXiv: 2311.09093 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2311.09093.

[38] S. Dargan, S. Bansal, M. Kumar, A. Mittal, and K. Kumar, "Augmented reality: A comprehensive review," en, *Arch. Comput. Methods Eng.*, vol. 30, no. 2, pp. 1057–1080, Mar. 2023.

[39] S. Shah and J. Tembhurne, "Object detection using convolutional neural networks and transformer-based models: A review," en, *J. Electr. Syst. Inf. Technol.*, vol. 10, no. 1, Nov. 2023.

[40] D. Müller and F. Kramer, "MIScnn: A framework for medical image segmentation with convolutional neural networks and deep learning," en, *BMC Med. Imaging*, vol. 21, no. 1, p. 12, Jan. 2021.

[41] M. M. Taye, "Theoretical understanding of convolutional neural network: Concepts, architectures, applications, future directions," *Computation*, vol. 11, no. 3, 2023, ISSN: 2079-3197. DOI: 10.3390/computation11030052. [Online]. Available: https://www.mdpi.com/2079-3197/11/3/52.

[42] S. University, *Optimization 2 - cs231n deep learning for computer vision*, 2025. [Online]. Available: https://cs231n.github.io/optimization-2/.

[43] J. Achiam, "Spinning Up in Deep Reinforcement Learning - Kinds of RL Algorithms," 2018. [Online]. Available: https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html.

[44] V. Mnih *et al.*, *Playing atari with deep reinforcement learning*, 2013. arXiv: 1312.5602 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1312.5602.

[45] C. Plou, A. C. Murillo, and R. Martinez-Cantin, *Active exploration in bayesian model-based reinforcement learning for robot manipulation*, 2024. arXiv: 2404.01867 [cs.RO]. [Online]. Available: https://arxiv.org/abs/2404.01867.

[46] S. Bhagat, H. Banerjee, Z. T. Ho Tse, and H. Ren, "Deep reinforcement learning for soft, flexible robots: Brief review with impending challenges," en, *Robotics*, vol. 8, no. 1, p. 4, Jan. 2019.

[47] M. C. McKenzie and M. D. McDonnell, "Modern value based reinforcement learning: A chronological review," *IEEE Access*, vol. 10, pp. 134 704–134 725, 2022. DOI: 10.1109/ACCESS.2022.3228647.

[48] M. Yu and S. Sun, "Policy-based reinforcement learning for time series anomaly detection," en, *Eng. Appl. Artif. Intell.*, vol. 95, no. 103919, p. 103 919, Oct. 2020.

[49] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, *Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor*, 2018. arXiv: 1801.01290 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1801.01290.

[50] A. R. Fayjie, S. Hossain, D. Oualid, and D.-J. Lee, "Driverless car: Autonomous driving using deep reinforcement learning in urban environment," in *2018 15th International Conference on Ubiquitous Robots (UR)*, 2018, pp. 896–901. DOI: 10.1109/URAI.2018.8441797.

[51] B. group, *Humanoid robots for bmw group plant spartanburg*, 2024. [Online]. Available: https://www.bmwgroup.com/en/news/general/2024/humanoid-robots.html.

[52] FigureAI, *Helix: A vision-language-action model for generalist humanoid control*, 2025. [Online]. Available: https://www.figure.ai/news/helix.

[53] FigureAI, *Natural humanoid walk using reinforcement learning*, 2025. [Online]. Available: https://www.figure.ai/news/reinforcement-learning-walking.

[54] O. Tagobitsky, *How computer vision is transforming the gaming industry in 2025*, 2025. [Online]. Available: https://api4.ai/blog/game-industry-and-ai-how-computer-vision-is-shaping-the-future-of-gaming.

[55] X. Hou, "Exploring the role of reinforcement learning in video game environments," en, in *Advances in Computer Science Research*, Dordrecht: Atlantis Press International BV, 2023, pp. 193–201.

[56] *Eur/usd 2000/may*, 2025. [Online]. Available: https://www.histdata.com/download-free-forex-historical-data/?/excel/1-minute-bar-quotes/EURUSD.

[57] *Currency price for eur/usd 16/07/24 to 28/07/25*, 2025. [Online]. Available: https://www.bloomberg.com.

[58] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–33, Feb. 2015. DOI: 10.1038/nature14236.

[59] A. Hill *et al.*, *Stable baselines - policies*, https://stable-baselines.readthedocs.io/en/master/_modules/stable_baselines/common/policies.html, 2018.

[60] A. Paszke *et al.*, *Pytorch: An imperative style, high-performance deep learning library*, 2019. arXiv: 1912.01703 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1912.01703.

[61] A. Paszke *et al.*, *Bcewithlogitsloss*, 2019. [Online]. Available: https://docs.pytorch.org/docs/stable/generated/torch.nn.BCEWithLogitsLoss.html.

[62] T. Schlosser, M. Friedrich, T. Meyer, and D. Kowerko, "A consolidated overview of evaluation and performance metrics for machine learning and computer vision," *Chemnitz University of Technology*, Aug. 2024. DOI: 10.13140/RG.2.2.14331.69928.

[63] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[64] A. Hill *et al.*, *Stable baselines*, https://github.com/hill-a/stable-baselines, 2018.

[65] A. Hill *et al.*, *Stable baselines - sac*, https://stable-baselines.readthedocs.io/en/master/modules/sac.html, 2018.

[66] E. Platen and R. Sidorowicz, "Empirical evidence on student-t log-returns of diversified world stock indices," *Quantitative Finance Research Center - University of Technology Sydney*, Mar. 2007, ISSN: 1441-8010. [Online]. Available: www.qfrc.uts.edu.au.

[67] C. R. Harris *et al.*, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. DOI: 10.1038/s41586-020-2649-2. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2.

[68] G. Van Rossum, *The Python Library Reference, release 3.8.2*. Python Software Foundation, 2020.

[69] A. Paszke *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[70] S. Chetlur *et al.*, "Cudnn: Efficient primitives for deep learning," *ArXiv*, vol. abs/1410.0759, 2014. [Online]. Available: https://api.semanticscholar.org/CorpusID:12330432.

[71] D. Buckley and W. Berg, *Drawdowns - why they're the worst thing in trading*, 2025. [Online]. Available: https://www.daytrading.com/drawdowns.

[72] B. Mena, *Key takeaways from trump's 'liberation day' tariffs*, 2025. [Online]. Available: https://edition.cnn.com/2025/04/02/economy/key-takeaways-from-trumps-liberation-day-tariffs.

[73] S. L. Giancarlo Corsetti and D. Ostry, *Tariffs and us dollar depreciations: Not so surprising after all*, 2025. [Online]. Available: https://cepr.org/voxeu/columns/tariffs-and-us-dollar-depreciations-not-so-surprising-after-all#:~:text=The%20sharp%20depreciation%20of%20the,currencies%20(see%20Figure%201).