POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Latent Variable-based Reinforcement Learning for FX Trading

Tesi di Laurea Magistrale in
Computer Science Engineering - Ingegneria Informatica

Adriano Mundo, 944684

Advisor:
Prof. Marcello Restelli

Co-advisors:
Riccardo Poiani

Academic year:
2021-2022

Abstract: The Foreign Exchange, also known as Forex, is a highly dynamic and active market for trading currencies, offering participants an exciting and challenging possibility to develop algorithmic trading strategies that can outperform human traders. However, learning and exploiting trading opportunities is difficult due to the high volatility and non-stationarity of the exchange rates, with the signal-to-noise ratio playing a significant role in effective learning. Trading opportunities are present at multiple time scales, but not all are easy to learn. Traditional methods tackle the problem by focusing on inner non-stationarity and using advanced statistical methods, but the potential of Reinforcement Learning remains largely untapped. This thesis introduces an efficient method to handle the complexity of this environment by combining the unsupervised training of deep Variational Auto-encoders and a state-of-the-art batch Reinforcement learning algorithm, Fitted Q-Iteration, without relying on any assumption about market dynamics. The key idea is to extract a latent representation of the time series and incorporate it as a feature space into the optimization procedure to exploit the structure of temporal sequences. Empirical evidence has shown that this method is effective in learning policies and, consequently, trading strategies on synthetic data and trading cash pairs, such as EUR/USD and USD/JPY. Additionally, it has demonstrated noticeable performance improvement compared to existing algorithms and strategies based on technical indicators. It can accumulate profit with good Sharpe ratios and minimal drawdowns, indicating low risk across various market conditions. To our knowledge, this research is the first to apply a two-step approach of latent models and Reinforcement Learning to Forex markets, showing great promise in overcoming the challenges of currency pairs intraday trading.

Key-words: Reinforcement Learning, FX Trading, FQI, Persistence, XGBoost, Latent Variables, VAE

Ai miei genitori,
che con il loro sostegno,
amore e dedizione
mi hanno permesso di diventare
l'uomo che sono oggi.

Abstract in lingua italiana

Il Foreign Exchange, anche noto come Forex, è un mercato altamente dinamico e attivo per il trading di valute, che offre ai partecipanti la possibilità stimolante ed eccitante di sviluppare strategie di trading algoritmico in grado di superare le prestazioni dei trader umani. Tuttavia, imparare a sfruttare le opportunità di trading è un'attività difficile a causa dell'alta volatilità e non stazionarietà dei tassi di cambio, con il rapporto segnale-rumore che gioca un ruolo significativo nell'apprendimento. Le opportunità di trading sono presenti a diversi intervalli temporali, ma non tutte sono facili da apprendere. I metodi tradizionali affrontano il problema concentrando l'attenzione sulla non stazionarietà intrinseca utilizzando metodi statistici avanzati, ma il potenziale del Reinforcement Learning (apprendimento con rinforzo) rimane in gran parte inutilizzato. Questa tesi introduce un metodo efficiente per gestire la complessità di questo ambiente combinando l'addestramento non supervisionato di deep (profondi) Variational Auto-encoders e un algoritmo di batch Reinforcement Learning allo stato dell'arte, Fitted Q-Iteration, senza fare affidamento su alcuna assunzione relativa alle dinamiche di mercato. L'idea chiave è quella di estrarre una rappresentazione latente della serie temporale, e incorporarla nello spazio dei vettori delle features, per la fase di ottimizzazione e in grado di sfruttare la struttura delle sequenze temporali. L' evidenza empirica ha dimostrato che questo metodo è efficace nell'apprendimento di politiche e, di conseguenza, strategie di trading su dati sintetici e nel trading di coppie di valute, come EUR/USD e USD/JPY. Inoltre, ha dimostrato un notevole miglioramento delle prestazioni rispetto agli algoritmi esistenti e alle strategie basate su indicatori tecnici. E in grado di accumulare profitti con buoni valori di Sharpe ratio e drawdowns minimi, indicando il suo basso rischio in diverse condizioni di mercato. Limitatamente alla nostra conoscenza, questa ricerca è la prima a combinare l'approccio a due fasi di modelli latenti e Reinforcement Learning nei mercati Forex, mostrando grandi potenzialità nel superare le sfide del trading giornaliero di coppie di valute.

Parole chiave: Reinforcement Learning, FX Trading, FQI, Persistenza, XGBoost, Variabili Latenti, VAE

Contents

1. Introduction

The *Foreign Exchange*, also known as Forex or FX, dominates as the world's largest financial market. It is a decentralized global marketplace where currencies are bought and sold simultaneously 24 hours daily from Sunday EST to Friday EST. According to the Triennial Central Bank Survey of Foreign Exchange and Over-the-Counter (OTC) derivatives markets in 2022 conducted by the Bank of International Settlements (BIS), the FX market has a volume of traded instruments up to \$7.5 trillion per day [43], surpassing even the stock market in size and inheriting a complexity that stems from high volatility, non-linearity, and irregularity. These distinguishing traits give traders potential opportunities and advantages to make profitable trades. Following years of consolidation, the market is currently recognized for its low transaction cost, high liquidity, immediate execution, transparency and absence of strict regulation, as each trade is conducted bilaterally via an agreement between two parties: a market participant such as a bank or asset manager with the intention to buy/sell and a market maker, or so-called liquidity provider who is responsible for offering quotes for various currency pairs. Indeed, FX trading happens through a network of brokers, given that there is no central exchange where instruments are listed. A transaction that occurs when one currency is bought (foreign currency) and one currency is sold (base currency) at an agreed price is also called Spot, distinguishing it from derivatives transactions such as Forward and Swaps. The Spot transactions account for most of the volume in the market [43]. The motivations behind why FX trading is commonly pursued are the facilitation of international trade and commerce, hedging against risks and speculation.

In the past, currency exchange was conducted in open outcry markets where members of financial institutions would use various forms of communication, such as hand signals and shouting, to agree on a transaction. However, with the advancement of commercial computer technology in the '90s, communication and quoting have become much easier, leading to a significant increase in the number of participants entering the market [28]. Furthermore, the technology supporting FX has completely evolved in the last two decades. The advancement in the computational power of machines, coupled with the availability of high-frequency data, has facilitated the development of new advanced strategies. For example, the continuously growing popularity of algorithmic trading has been a trend from major investment banks that replaced human traders with electronic desks. Recently, the research and development of algorithmic strategy and Machine Learning (ML) techniques have been extensively studied in financial markets with application to equities, FX and commodities. In these studies, the goal was to construct automated systems able to generate profits by outperforming human traders, and they often showed significant results [54, 96].

The trading problem, independently from the asset class, can be tackled from a mathematical and computational perspective framed as a Sequential Decision Problem (SDP) where an agent must continuously decide what action to perform to maximize returns. It is possible to model such a sequential decision problem in a discrete-time setting as a Markov Decision Process (MDP), where the agent collects information from the environment and selects the position to hold at each time step. A change in position involves a transaction cost with a profit and loss (P&L) that depends on the position and the market's movement. If the underlying model is known, the MDP can be solved through Dynamic Programming (DP), but since the dynamics of the FX market are uncertain, Reinforcement Learning (RL) has to be used to solve the MDP [86]. It is clear that the two areas are perfectly correlated: the core idea of RL is to find an optimal policy by interacting with the environment and aiming to maximize the reward, and the objective of trading is to have a successful (algorithmic) strategy. As a result, numerous studies have focused on designing and implementing RL algorithms for solving trading problems [42, 76, 85]. Despite these similarities, many challenges exist to overcome since financial markets are renowned for their unpredictable nature due to a low signal-to-noise ratio. They are primarily driven by non-stationary dynamics and complex feedback mechanisms that can lead to non-linear effects [31]. In this context, it is not easy to understand how machines can learn to trade automatically and what data they need to reach this goal. Indeed, the characteristics mentioned above are even more evident at a lower level of granularity, the so-called microstructure level [73], because of the diversity of market participants and their strategies. It is reasonable to consider that the market is built upon different time-scale with scaled trading opportunities [67], but learning and predicting them is difficult. Indeed, most of the agents that operate on the market in the high-frequency-trading (HFT) space have a hard-coded behaviour to gain profits based on short-lived opportunities; on the other hand, the players who operate at low-frequency such as days and months, cannot exclude exogenous (macro) economic factors. Therefore, the ideal scenario for ML techniques is to execute at a frequency of minutes or hours.

The **scope** of the thesis was to investigate the effectiveness of RL techniques enhanced by latent variable models in a non-stationary setting. The intuition behind the analysis was that the latent models can extract relevant information for learning from the so-called latent variables. Those variables are not directly observable but are inferred from other observable variables and used to represent underlying constructs or factors that cannot be measured. Then, the model was applied to solve the FX Trading task. The study had two primary objectives. Firstly, it aimed at conducting experiments using multiple models to extract latent variables since it is complex to tease out a useful compact representation. Secondly, it evaluated how to integrate the learnt feature space into the RL algorithm to extract a trading policy capable of executing profitable trades. The thesis is positioned in the research area that lives at the boundary between RL and Trading, for which the classification is shown in Figure 1.

Therefore, the work sought to answer the two following **research questions**:

(*i*) *What is the impact of incorporating latent variables on the performance of a Batch-RL algorithm in financial markets?*

(*ii*) *Can an RL agent be trained to learn trading strategies that outperform human traders and algorithmic strategies?*

The study experimented with the execution of the learned strategies for the most liquid currency pairs: EUR/USD and USD/JPY, to trade intraday without any market impact. The focus was on the application of a Batch-RL algorithm called Fitted-Q-Iteration (FQI) [40], which is extremely efficient in an offline setting with historical data, and its extensions with action persistence (PFQI) [69]. It is a model-free and off-policy algorithm that learns the optimal policy without interacting with the environment. Indeed, directly employing model-based planning approaches can lead to significant errors when the underlying model is imperfect. At the same time, the proposed approach in the study is robust to such imperfections offering a more reliable solution for trading. In the literature, the FQI algorithm has already been successfully applied to such scenarios [13, 78, 79]. In the thesis, the FQI algorithm has been extended with latent variables, namely Latent Variable (Persistent) Fitted-Q-Iteration (LV-FQI, LV-PFQI), where the latent representation was extracted through Variational Auto-Encoders (VAE) models [58]. VAE helps obtain a set of compressed latent features representing the time series in a more suitable and relevant form where only the most influential and significant features are kept. The purpose was to investigate whether incorporating the latent space into the RL agent could enhance the ability to learn trading strategies and achieve superior performance compared to existing algorithms. The resulting model demonstrates the ability to leverage the structure of future temporal sequences effectively. This implies the algorithm can capture and utilise the inherent patterns and dynamics of the non-stationary time series without relying on explicit model-based assumptions. The simulation of a real-world setting is crucial, so transaction costs are incorporated into the experiments, which are factored in by the agent while devising trading strategies. But, to keep the calculation simple, it is assumed a relatively minor trading size that did not result in any slippage, the difference between the expected price of a trade and the price at which the trade is executed. Lately, an analysis of the feature's importance and the agent's actions was carried out. The algorithm's performance was compared with the existing RL baselines and with classical trading strategies executed by human traders or algorithmic trading software. The idea validation around a latent Batch-RL algorithm has been confirmed with experiments on synthetic data generated via Vasicek and Geometric Brownian Motion models.

The **contribution** of the study is two-fold and mainly empirical. On one side, it provides a performance comparison of LV-FQI, and its persistent version, augmented with latent variables, in a realistic trading scenario based on FX historical data and transaction fees, against existing RL algorithms and trading baselines. On the other side, it presented a two-step approach for combining the training of deep variational auto-encoders for learning compact feature spaces with state-of-the-art batch-RL algorithms to learn trading strategies (or policies) that can capture the structure of future temporal sequences of exchange rates. Consequently, the model is not exposed to the flaws of model-based approaches, and the performance is remarkable for a non-stationary setting. This is the first work that leverages latent features as part of the RL framework. Finally, the results clearly demonstrated how traders' capabilities could be augmented with quantitative ML models and a data-driven approach.

The rest of the thesis is organized as follows. Section 2 introduces the theoretical background of trading, the fundamental RL concepts, and their applications; lastly, the non-stationarity and VAE are explained as a chosen model to extract latent variables in the experimental phase. Then, Section 3 illustrates the main related work in the space of RL applied to FX Trading and their classification. In the following Section 4, the methodology is described in detail with an RL pipeline that goes from the problem formulation to the agent design and evaluation. Section 5 describes the experiments and discusses the results on synthetic and FX data. Finally, Section 6 concludes the work, highlighting potential future research directions.

2. Background

The purpose of this chapter is to provide a comprehensive overview of the theoretical concepts that form the foundation of the present work. Specifically, the chapter outlines the fundamental principles and dynamics of the financial markets, focusing on the FX trading business. The aim is to provide the necessary context to understand the essential concepts of Reinforcement Learning (RL) and its applications in trading. To achieve this objective, the chapter introduces the mathematical formulation of the trading problem, helping readers develop an understanding of the problem abstraction. The chapter also explores the Variational Auto-Encoders (VAE) and latent variables models used in the experimental phase to optimize the learned strategies. Overall, the chapter is a compass for readers to understand the rest of the thesis.

2.1. Financial Markets & Trading

Financial Markets are generally known as the *markets* and represent where the exchange of financial instruments occurs. Their principal function is enabling government entities and businesses to raise capital. They can be divided into two main types: namely, the *primary market*, where the assets are issued and the *secondary market*, where the same assets are exchanged. They can also be categorized depending on the location and the asset, but they can be considered a unique place where everything is interconnected. For example, governments can issue or sell debt securities (bonds) to fund infrastructure projects or provide services to their citizens. In contrast, firms can issue debt securities or sell equity (stock) to gain funds for investments and business expansion. Lastly, the markets facilitate the transfer of existing debt and equity securities from current to new investors. There are different types of traded instruments or assets: stocks, bonds, currencies, commodities and derivatives. *Stocks* represent partial ownership in a company, while *bonds* are debt contracts that can be traded in small units. *Commodities* are goods utilized in commerce such as gold, oil, gas or corn and many others while *currencies* are the instrument around which the thesis was elaborated. Finally, *derivatives* are complex financial instruments whose value is based on the price of an underlying instrument. All the asset classes mentioned before have their own derivatives counterparts that are irrelevant in this context; the interested reader can refer to the most famous derivatives book [55].

2.1.1 Foreign Exchange Markets

In most countries, unique currencies are used, such as the *Dollar* in the United States and the *Euro* in the Euro in the Eurozone. Furthermore, today companies operate globally, and the international trade of goods and services involves the exchange of different currencies, typically in the form of bank deposits. For instance, when an Italian company purchases a service abroad, the equivalent value of Euros must be exchanged for the respective foreign currency. All these transactions occur in the Foreign Exchange (FX) market, where *exchange rates* are determined. The rates, in turn, impact the cost of acquiring foreign goods, services, and financial assets. Therefore, the FX market is critical for facilitating international trade and transactions. In addition, exchange rates are quite flexible and vary day by day, with central banks sometimes intervening in the markets by altering interest rates. The sensitivity to rate changes imposes the necessity of techniques to manage and hedge against this risk.

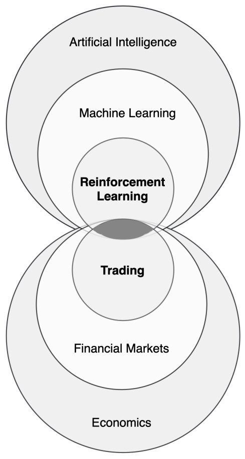


Figure 1: Interdisciplinarity between Trading and Reinforcement Learning

Trading Venues & Participants
In the FX market, differently from the regulated stock exchange such as Borsa Italiana, currencies are traded Over-The-Counter (OTC) or off-exchange. This means that the financial instrument is exchanged on dealer markets, in other words, through a network of brokers. This network is no longer physically represented by trading floors but exists electronically. The OTC trading involves two main participants: *dealers*, also known as *liquidity providers* in the FX world, and *institutional clients*. Instead, retail clients usually exchange cash via their bank or online platform, where this process is invisible. Liquidity providers (LPs) continuously provide bid and ask prices, with some offering firm quotes, which are final prices that can be executed. In contrast, others offer indicative quotes after receiving a Request For Quote (RFQ). The clients execute their orders through multi-dealer platforms directly connected to LPs. Among the most famous platforms, there is Bloomberg [1], but there are also other Execution Management Systems (EMS) available. The most traded product in FX is *spot*. Still, there also exist others like *forward* and *futures*, an agreement to buy/sell a currency at an agreed price on a future date, and*swaps*, which involve the exchange of two currencies on a specific date at a predetermined exchange rate and then reversing the exchange on a future date.

Order Book Functioning
An *order book* is an electronic list of all buy and sell orders for a specific financial instrument at different price levels. In the case of the FX market, the instrument is a *currency pair*. A matching engine uses the order book to determine which orders can be fully or partially executed based on price and time priority. The only lit order books for FX are EBS [2] and Refinitiv [4] platforms. Instead, the book of the OTC trading shows the traders an aggregation of all the quotes from the LPs at different sizes and price buckets. To measure the distance between the bid (buyers) and the ask (sellers) side of an exchange rate are used the *Pips* (Percentage in points), and typically for the main pairs, a pip is 0.0001, the 4th decimal digit. Another definition is the *bid-ask spread* calculated as the difference between the best ask, or lowest sell price, and best bid, or highest buy price. An example can be found in Figure 2. When trading on an order book, it is possible to employ different types of orders: *limit orders* are the most used and entail specifying the price of the execution for the trade so it can be executed only at this price or a better one, and *market orders* that consist in a request to perform the order immediately at the best available price on the market [63].

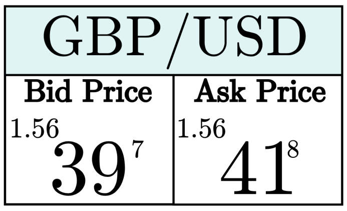


Figure 2: This figure shows an example of the bid-ask spread for the GBP/USD currency pair.

2.1.2 Recent Evolution of Trading

Electronification
The electronification of trading has disrupted the financial environment transforming it from intense human work into a system facilitated by computers and statistics. Electronic trading is the process of transacting orders over a computer system or network rather than via a phone call when you need to state your order, intentions, and any special instruction. It could be as simple as entering a buy order into a retail system or via a mobile app. Trading floors are quiet and no longer filled with paper orders and confirmations. In 2019, electronic trading comprised approximately 99.9% of all equity volume, and algorithmic trading comprised approximately 92% of all equity volume. It grew from 15% in 2000 to today's numbers [57]. This represents the state-of-the-art of financial markets today.

Algorithmic Trading
The term *algorithmic trading* describes a collection of methods for automating trading strategies. The main task is to take care of the trading once a decision to buy/sell has been taken. They emulate the role of a broker trying to achieve *optimal execution* or *optimal trading* to minimize the market impact and the transaction fees. The field has continuously evolved in the last 20 years, and it now includes *market making* and *high-frequency-trading* under its umbrella. The market makers invest in technology and quantitative models to analyze the market microstructure dynamics to provide quotes/liquidity for specific financial instruments gaining profits from the bid-ask spread. Instead, the HFT firms are implementing strategies to capitalize on short opportunities and get an edge at the lowest level of granularity (i.e. nanoseconds) based on market inefficiencies [8].

2.2. Sequential Decision Problems

A problem where decisions are taken in several steps can be represented in a mathematical form as a *sequential decision model* where, at each stage, an agent interacts with an environment and a specific cost or reward has to be paid or received, the observation. Moreover, a decision influences the future and, consequently, future costs/rewards. Thus, is it better to act greedily to get an immediate reward or sacrifice it to maximize an expected future reward? Financial trading is a typical instance of this problem where traders want to achieve long-term profits alongside immediate returns.

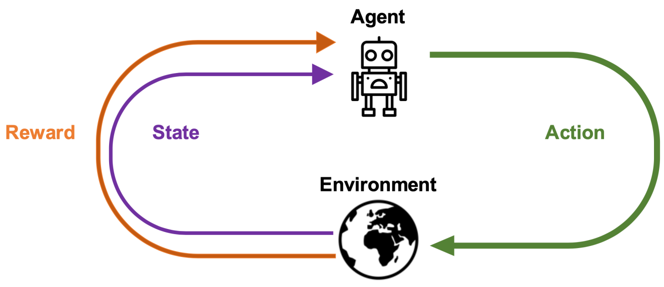


Figure 3: Representation of the interaction between agent and environment with state, action, reward.

2.2.1 Markov Decision Process

A sequential decision-making problem is often modelled as a *Markov Decision Process* (MDP) [77], derived from previously existing work [11]. The key assumption behind MDP is that the state is completely observable and Markovian. If the first assumption does not hold, then a Partially-Observable MDP (PO-MDP) needs to be used [26]. Instead, the Markov property implies that the future is independent of the past given the present, or in a formal way that the dynamic depends only on the current state and action and not on the previous history. An example of a visual representation of an MDP can be found in Figure 3 A discrete-time MDP is defined as a 6-tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma, \mu \rangle$ where $\mathcal{S}$ is a measurable state-space containing all the states, $\mathcal{A}$ is a measurable action-space containing all the actions, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ is the transition model, or kernel, that assigns to each state-action pair (s, a) a probability measure $\mathcal{P}(\cdot|s, a)$ of the next state, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, which assigns to every (s, a) a probability measure $\mathcal{R}(\cdot|s, a)$, $\gamma \in [0, 1)$ is the discount factor to weight future reward and $\mu : \mathcal{S} \to \Delta(\mathcal{S})$ is the initial state distribution from which the starting state is sampled. Here $\Delta(\mathcal{S})$ denotes the set of probability measures over $\mathcal{S}$. The sequence of states, actions, and rewards up to a certain time-step t is called the *history* and can be defined as $H_t := (s_0, a_0, r_1, s_1, a_1, r_1, \dots, s_t, a_t, r_t)$. The interaction between the agent and the environment can be finite or infinite, so the *time-horizon*. This thesis focuses on the finite horizon or *episodic* MDP. The agent aims to maximize the discounted sum of rewards, also called return. Formally, defined a trajectory for a certain horizon T as $\tau = H_T$, the return is $G_\tau = \sum_{t=0}^{T} \gamma^t r_{t+1}$. To take actions, the agent follows a *policy* π that associates to each state a probability over the actions; hence it is defined as $\pi : S \to \pi(\cdot|s) \in \mathcal{P}(A)$. Instead, to evaluate the utility of a state is used the *state-value function* of a given policy π, formally defined as $V^\pi = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s, \pi]$ where r_t is the immediate observed reward at time step t. The expectation is over all the possible trajectories starting in state s and by following the policy π. A more useful quantity used in practice is the *action-value function*, or $Q-$function, which allows stating the first action explicitly, and it is defined as $Q^\pi = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s, a_0 = a, \pi]$.
The two functions can be further decomposed to obtain the Bellman *expectation equations* so that the state-value function becomes $V^\pi(s) = \sum_{a \in A} \pi(a|s)(\mathcal{R}(s, a) + \gamma \sum_{s' \in S} \mathcal{P}(s'|s, a) V^\pi(s'))$ and the action-value function $Q^\pi(s, a) = \mathcal{R}(s, a) + \gamma \sum_{s' \in S} \mathcal{P}(s'|s, a) \sum_{a \in A} \pi(a'|s') Q^\pi(s', a')$. In general, to solve an MDP is important to find the *optimal* value-function of the state under all available policies, defined as the maximum over all policies. Therefore, $V^*(s) = \max_\pi V^\pi(s) = \max_a \left\{ \mathcal{R}(s, a) + \gamma \sum_{s' \in S} \mathcal{P}(s'|s, a) V^*(s') \right\}$ while the action-vale function becomes $Q^*(s, a) = \max_\pi Q^\pi(s, a) = \mathcal{R}(s, a) + \gamma \sum_{s' \in S} \mathcal{P}(s'|s, a) \max_{a'} Q^*(s', a')$ obtaining the Bellman *optimality equations*. A family of methods that can be used to solve MDP when the model is available (model-based) is called *Dynamic Programming* (DP). This technique is called *planning*, where we can plan ahead and solve the equations, thus finding an optimal policy, and the main two are *policy iteration* and *value iteration*. The first approach implies learning the value function of a given policy and then improving the policy with a new one acting greedily with respect to the learned value function, while the latter tries to find the optimal policy directly.

2.3. Reinforcement Learning

Machine Learning (ML) is the subfield of Artificial Intelligence (AI) aiming to design algorithms able to learn tasks from data while exploiting statistical techniques. In ML, knowledge comes from experience and induction. An attempt at defining an ML program was made by Mitchell (1997): "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E" [70]. *Reinforcement Learning* (RL) is one of the main paradigms of ML alongside Supervised and Unsupervised Learning. It focuses on developing artificial agents able to learn from their interaction with the environment. A sequential decision process models the interaction, explained in the previous chapter, which comprises certain steps. The agent receives an observation from the environment and chooses an action. Then, the environment, influenced by the action, transitions to a new state according to its dynamics and produces a reward. This loop repeats for the length of time the agent is in the condition of interacting with the environment. A central subject for RL is the *exploration-exploitation* dilemma, which refers to the trade-off between selecting actions that are already known to produce rewards (exploitation) versus trying out new actions to discover potentially better ones (exploration). Finding the right balance between these two approaches is critical for achieving optimal performance in the learning process. We invite the reader to refer to these sources for a complete review of RL [86, 87]. Furthermore, recent years have seen a surge of RL incorporated with neural networks, the so-called Deep-RL, obtaining superior performance in various domains. However, most methods use games or simulated environments as benchmarks, and often real life is different from games [12, 21, 82].

2.3.1 Taxonomies in RL

Dynamic Programming (DP) can be used to find an exact solution for a finite MDP with complete knowledge of the underlying dynamics. However, this assumption does not hold in real-world applications where the state-space may be large and/or continuous. Therefore, when the model is *unknown*, there exist several approaches to solve MDP via RL since it gives way for either learning value functions, also called prediction, or control in the best way decision problems that cannot be solved exactly. The following dichotomies can explain a taxonomy of the main RL settings.

In the more general case, the three components are *policy*, *model* and *value function*.

- *Prediction vs Control* $\rightarrow$ are the tasks an RL agent has to solve where prediction is a synonym for policy evaluation in the RL context to highlight the estimation nature; instead, policy or value function optimization is known as control.
- *Model-free vs Model-based* $\rightarrow$ model-based approaches aim at estimating either the transition kernel or the reward and then solve the MDP via exact or approximate methods; model-free techniques work without a model to learn the policy or the value function.
- *Value-based, Policy-based, Actor-Critic* $\rightarrow$ critic-only (or value-based) methods learn the optimal value function as in the case of value iteration, instead policy-based (or actor-only) aims at directly finding the optimal policies in the policy parameter space; finally, the combination of the two approaches is called actor-critic and tries to combine the best of both worlds.

Instead, based on the characteristics of the RL algorithm, they can be classified as:

- *Online vs Offline* $\rightarrow$ online approaches consider a scenario where the agent learns while interacting with the environment, so at each time step, it receives new samples; in the offline setting, all the samples are available beforehand, and no further interaction with the environment is possible. Also, a hybrid setting exists and it is called *semi-batch*.
- *On-policy vs Off-policy* $\rightarrow$ the learning task does not need to target the current policy. Indeed, the interaction policy may be used exploratory for gathering information. If the target and behavioural policy coincide, the setting is called on-policy; otherwise is off-policy.
- *Tabular vs Function approximation* $\rightarrow$ when the MDP is finite, the setting is also called tabular since value functions and policies can be considered as tables or matrices instead if the state-action space is infinite, it is necessary to resort to function approximation for estimating the value function.

The positioning of this thesis in the RL taxonomy can be summarized as an attempt to solve a control task, FX trading, with a value-based, model-free, and off-policy algorithm, FQI, in an offline setting via function approximation. The following section will explain the theory of FQI and offline RL in detail.

2.3.2 Batch-RL

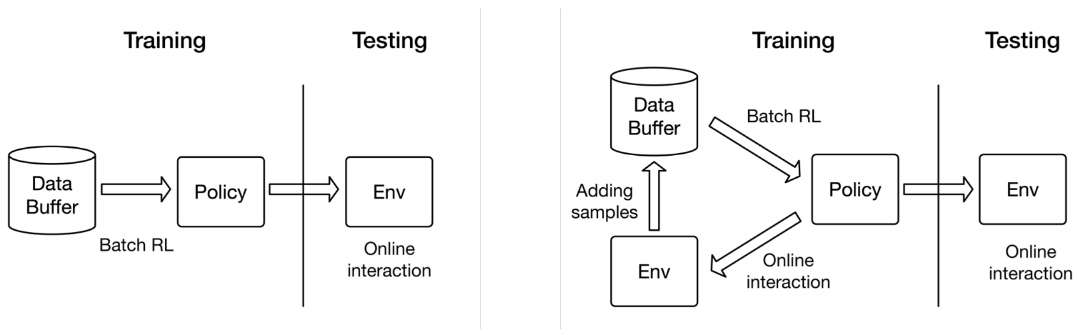
RL often requires having a suitable simulator for the agent to interact with. This simulator is not always available because it is either nonexistent or hard to build due to the complex unknown systems dynamics [38]. Indeed, training the agent by directly interacting with the real environment may lead to disastrous outcomes or be unsuitable for the task at hand.

Batch-RL, also known as *offline-RL*, aims at filling this gap by providing algorithms that can learn near-optimal control policy from a fixed dataset without additional interaction with the environment [60, 65]. Batch-RL is one of the most promising methods for real-world RL because it allows leveraging existing datasets that can be costly and hard to simulate and has shown better sample efficiency [23]. Furthermore, it can be used with advanced supervised methods [75], and many algorithms are model-free, so they do not require a knowledge of the system dynamics [17].

There exist two different settings in Batch-RL, also illustrated in Figure 4:

(i) *Pure Batch* $\rightarrow$ the agent learns a policy from the pre-collected dataset without interacting with the environment during training.
(ii) *Growing Batch* $\rightarrow$ the agent can occasionally interact with the environment with the latest policy to build the batch incrementally.

For a complete overview of existing batch algorithms and real-world applications, the reader can look at the following surveys [44, 60].



(a) pure batch. (b) growing batch.

Figure 4: Illustration of different Batch-RL settings.

Episodic MDP are often used to model naturally episodic tasks, such as playing a game of chess or completing a navigation task. The agent learns to optimize its behaviour within the context of each episode without considering an indefinite horizon of future states and actions, as is the case in *infinite-horizon* MDP. They can also be used in Batch-RL, where a fixed dataset of episodes is collected and used. Thus, in practice, it is convenient to define a problem with a distinct beginning and end via *episodic* MDP, where the agent interacts with the environment over a series of discrete episodes, and each episode consists of a finite number of time steps, or horizon T. At the beginning of each episode, the agent starts in a particular state, takes actions based on its policy, and receives rewards based on its actions. The goal is to learn an optimal policy π^* is the one that maximizes the expected sum of rewards over all possible episodes:

$$\pi^* = \arg\max_{\pi} J(\pi) = \arg\max_{\pi} \mathbb{E}\left[\sum_{t=1}^{T} r_t \mid \pi\right] \tag{1}$$

where $J(\pi)$ is the sum of rewards from time step t to the end of the episode.

To summarize, *episodic* MDP are used in a variety of Batch-RL algorithms, such as the Fitted Q-Iteration algorithm [40] and Direct Policy Search algorithm [64], which learns a policy directly from the dataset of episodes. The former is used and explained later in this chapter in this thesis.

2.3.3 Fitted Q-Iteration

In this thesis, we leverage an existing Batch-RL algorithm to train the agent to learn policies for FX trading. *Fitted-Q-Iteration* (FQI) [40] is a model-free, off-policy, and offline algorithm that learns an approximation of the optimal action-value function Q^* by extending the optimization horizon starting from a finite set of experience samples collected in the dataset $\mathcal{D} = \left\{(s_t^k, a_t^k, s_{t+1}^k, r_{t+1}^k)|k = 1, 2, ..., |\mathcal{D}|\right\}$ where s_{t+1} is the state that the agent reaches after applying action a_t in state s_t while collecting reward r_{t+1} for this transition. FQI is derived from the fitted value iteration approach [49] and reformulates the original problem as a sequence of supervised regression problems (Algorithm 1). Specifically, at the N-th iteration, given the action-value Q-function approximated at the previous iteration $Q_{N-1}(s, a)\forall(s, a), Q_N$ is trained on the following training set: $TS_{FQI} = \left\{(i^k, o^k)|k = 1, 2, ..., |\mathcal{D}|\right\}$ where the input is the state-action pair $i^k = (s_t^k, a_t^k)$ and the output is $o^k = r_{t+1}^k + \gamma \max_{a\in\mathcal{A}} Q_{N-1}(s_{t+1}^k, a)$. In other words, at each iteration, FQI computes the error to update the Q-function with all samples in the dataset. The batch approach allows us to use regression algorithms like tree-based, kernel-based regressors, or neural networks. FQI can be understood intuitively as expanding the optimization horizon at each iteration, but it is important to highlight the observed phenomena surrounding it; thus, an increasing number of FQI iterations leads to a larger planning horizon and propagation of regression errors, and the sequence of Q-function may not converge to Q^*.

Algorithm 1 Fitted-Q-Iteration

```
1: Input: transition samples $\mathcal{D}$, a regression algorithm $f_\theta$
2: Initialize $Q$-function: $Q_0(s, a) = 0 \ \forall s \in \mathcal{S}, \forall a \in \mathcal{A}$
3: for step $i$ in $\{1, ..., N\}$ do
4:     Evaluate: $Q_{i-1}(s^k, a^k)$ for all $\mathcal{D} = \left\{(s^k, a^k)|k = 1, 2, ..., |\mathcal{D}|)\right\}$
5:     Compute: $o^k = r_{t+1}^k + \gamma \max_{a\in\mathcal{A}} Q_{i-1}(s_{t+1}^k, a)$ for each sample
6:     Learn $Q_i(s, a)$ via regression $f_\theta$ on $TS = \left\{(s_t^k, a_t^k), o^k\right\}$
7: end for
8: Return: $Q_N(s, a)$
```

Action Persistence

An extension of FQI is *Persistent Fitted-Q-Iteration* (PFQI) [69], where the original algorithm takes into account the possibility of persisting actions. RL transforms a continuous time problem into a discrete-time MDP by introducing a time discretization and a control frequency. *Persistence*, by repeating each action for a certain number of consecutive steps ($k > 1$), allows tuning the control frequency. A higher frequency gives the agent more control opportunities but decreases the signal-to-noise ratio and negatively impacts sample complexity. Thus, higher frequency agents could potentially earn greater returns but may have more difficulties during learning. Lastly, persistence influences the optimization horizon since a higher persistence agent requires fewer iterations than a less persistent agent to reach the same horizon. Therefore, persistence is another helpful tool to deal with the trade-off between control capacity and learning ability. In a mathematical form, *persistence* can be seen as an environmental parameter k that can be configured to generate a family of related decision processes $\mathcal{M}_k = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}_k, \mathcal{R}_k, \gamma^k, \mu\rangle$ that whenever an action is issued, the resulting transition lasts for k steps, with all the one-step rewards collected with discount in the new distribution $\mathcal{R}_k$.

2.4. Non-Stationary Reinforcement Learning

In this section, the *non-stationarity* problem will be described from a time-series perspective, and a brief overview of the sources of non-stationarity in RL will be presented. The rest of the section will be devoted to the Latent Variables modelling through Variational Auto-encoders (VAE), the model used in this work to capture the time series and market dynamics to overcome non-stationarity.

2.4.1 Time Series Modelling

The usual meaning of *non-stationarity* is linked to time series and processes. By definition, a stochastic process is a collection of random variables $\{X_t\}_{t\in T}$ defined on a probability space and assuming values in $\mathbb{R}^N$. Thus, it is possible to see a stochastic process as a continuous or discrete set of random variables that assume a particular value in a certain time index. If the time is discrete, the index of the stochastic process is based on integer values of t, defining it as a time series. When the process depends on time, hence time steps are not drawn from the same distribution, and the process is non-stationary.

A non-stationary process is subjected to a radical change in its evolution. The scope of most active approaches is identifying the source of non-stationarity in the underlying process and removing its effects. A possible classification of the phenomenon may be provided based on how the distribution drift occurs. A non-stationary process whose changes occur rapidly is *abrupt*, with evident differences in the drifting element such as reward function, transition probability or action space. Instead, a process is defined as *smooth* if its changes occur slower. In this type of setting, the drifting behaviour of environment dynamics/reward function is much more difficult to track but also happens slowly, thus allowing for easier adaptability.

It is possible to use various techniques to assess if a *time series* is *non-stationary.* The straightforward technique assesses how the mean value and the variance of the realisation $\{x_t\}$ is changing through a very simplistic approach, such as dividing the time series realisation into various chunks. Another technique is represented by statistical tests based on enforcing strong assumptions about the data and verifying the underlying hypothesis which should support such assumption. An example is the Augmented Dickey-Fuller (ADF) test, also called a unit root test. Both techniques are used in this thesis for the FX data analysis in Section 5.3.1.

2.4.2 Non-Stationary Modelling

The main assumption of MPDs is that the transition dynamic $\mathcal{P}$ and the reward function $\mathcal{R}$, which define the environment model, are stationary, and this implies that they do not vary over time. In the *non-stationary scenario*, the assumption is invalid since $\mathcal{P}$ an $\mathcal{R}$ may be both affected by non-stationarity, possibly at the same time; hence classical algorithms cannot help for learning optimal policies. In the case of *episodic* MDP, it can be formulated as the existence of an iteration i^* of the RL algorithm after which the trajectories sampled from the environment will be, partially or totally, associated with a new "task".

In a mathematical formulation, non-stationarity in RL can be formalised as that for any $i < i^*$ we have $\mathcal{M}_1 = \{\mathcal{S}, \mathcal{A}, \mathcal{P}_1, \mathcal{R}_1, \gamma\, \mu\}$, whereas for $i \geq i^*$ we have $\mathcal{M}_2 = \{\mathcal{S}, \mathcal{A}, \mathcal{P}_2, \mathcal{R}_2, \gamma, \mu\}$ where $\mathcal{P}_1 \neq \mathcal{P}_2 \vee \mathcal{R}_1 \neq \mathcal{R}_2$. Instead, less common sources of non-stationarity may be linked to the action or observation spaces. In the former, the agent faces different choices from previous time steps; hence it exists a set of action spaces $\hat{\mathcal{A}} = \{\mathcal{A}_1, \ldots, \mathcal{A}_t\}$ which may change over time t. In the latter, the agent receives an observation $x_t = (r_t, s_{t+1})$ different from the real reward, state pair induced by an external phenomenon like an adversary.

2.4.3 Latent Variables Modelling

Deep learning-based generative models have gained mainstream attention in recent years thanks to remarkable improvements in the field [22]. Among these, *Variational Auto-encoders* (VAE) deserve special attention [58]. A VAE is a *latent variable model* which uses an autoencoder architecture trained to ensure that its latent space has general properties able to generate new data, or differently it is a method to learn the probability distribution of large complex datasets. In this section, we will dissect the fundamental notions that pose the basis for VAE and introduce Artificial Neural Networks (ANN) used for function approximation in the context of VAE.

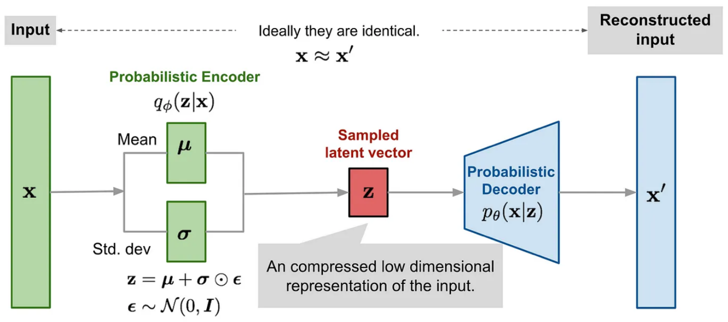

Input
Ideally they are identical.
$\mathbf{x} \approx \mathbf{x}'$
Reconstructed input
$\mathbf{x}$
Probabilistic Encoder
$q_\phi(\mathbf{z}|\mathbf{x})$
Mean
$\boldsymbol{\mu}$
Std. dev
$\boldsymbol{\sigma}$
$\mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}$
$\boldsymbol{\epsilon} \sim \mathcal{N}(0, \boldsymbol{I})$
Sampled latent vector
$\mathbf{z}$
An compressed low dimensional representation of the input.
Probabilistic Decoder
$p_\theta(\mathbf{x}|\mathbf{z})$
$\mathbf{x}'$

Figure 5: Architecture of a VAE, where **X** and **X'** represents respectively the input and the reconstructed input; q_ϕ and p_ϕ the encoder, decoder probabilities, z the reparameterization trick and ϵ the Gaussian noise.

Latent Variables

A *latent variable* is a variable which is not directly observable and is assumed to affect a response variable. They are typically included in *latent variable models* with different aims: representing the effect of unobservable factors or accounting for measurement errors where the latent variables represent the real outcomes and the response variables represent their disturbed versions. The general idea is that certain latent variables parametrize the model, and given a data point, we do not necessarily know which setting generates that point [59].

In a formal setting, we have a dataset X and a vector of *latent variables* z in a high-dimensional space $\mathcal{Z}$ that we can sample according to some probability density function $\mathcal{P}(z)$ over $\mathcal{Z}$. Then, we have a family of deterministic functions $f(z;\theta)$ parameterized by a vector θ in some space Θ, where $f : \mathcal{Z} \times \Theta \to \mathcal{X}$. f is deterministic, but if z is random and θ is fixed, then $f(z;\theta)$ is a random variable in the space $\mathcal{X}$. The wish is to optimize θ such that we can sample z from $\mathcal{P}(z)$ and, with high probability $f(z;\theta)$ will be as close as possible to X. To do so, it is necessary to maximize the probability of each X in the training set according to Equation (2) where $\mathcal{P}(X|z;\theta) = \mathcal{N}(X|f(z;\theta), \sigma^2 \cdot I)$ and $f(z;\theta)$ is replaced by the distribution $\mathcal{P}(X|z;\theta)$.

$$\mathcal{P}(X) = \int \mathcal{P}(X|z;\theta)\mathcal{P}(z)dz \tag{2}$$

Variational Auto-encoders

The term *Variational Auto-encoders* (VAE) [58] refers to a specific type of generative model that combines elements from different research areas. The term variational stems from the close relationship between the regularization and the *variational inference* method in statistics [16], while *auto-encoders* from the architecture used from the model to reconstruct input data. A complete overview of VAE architecture can be found in Figure (5).

Before jumping into VAE formalities, a concept to understand is *dimensionality reduction*. It is the process of reducing the number of features that describe some data either by selection, only some existing features are preserved, or by extraction, a reduced number of new features are created. It can be useful in many situations that require low-dimensional data. This is related to the *encoder-decoder* architecture, where the encoder process data to produce new features in a low-dimensional space, also called *latent space*, and the decoder does the reverse process by decompressing the space into the original input. The objective is to retain as much information as possible when encoding to have a minimum reconstruction error when decoding.

Starting from Equation (2), VAE take the assumption that there is no simple interpretation of the dimensions of z, and it can be drawn from a simple distribution, so $\mathcal{P}(z) = \mathcal{N}(z|0, I)$ where I is the identity matrix. Then, it can be learned as a function which maps the independent, normally-distributed z values to whatever latent variables might be needed for the model and then map those latent variables to X.
Theoretically, by sampling a large number of z values $\{z_1, z_2, \ldots, z_n\}$, it is possible to approximately compute the probability $\mathcal{P}(X) \approx \frac{1}{n}\sum_{i=1}^{n} \mathcal{P}(X|z_i)$. Instead, in practice, for most z, the value of $\mathcal{P}(X|z)$ will be nearly zero, hence contributing almost nothing to the estimate of $\mathcal{P}(X)$.
The key idea behind the VAE is to attempt sampling values of z that are likely to have produced X and compute $\mathcal{P}(X)$ from those. Thus, it is needed to define a new function $\mathcal{Q}(z|X)$ which can take a value of X and give us a distribution over z values that are likely to produce X, the *latent distribution*. Indeed, this is the main difference compared to standard *auto-encoders* where the input is encoded as a single point. In this way, $\mathbb{E}_{z\sim\mathcal{Q}}\mathcal{P}(X|z)$ can be computed relatively easily.

The relationship between $\mathbb{E}_{z\sim\mathcal{Q}}\mathcal{P}(X|z)$ and $\mathcal{P}(X)$ is fundamental for variational Bayesian methods and it represent the *reconstruction loss* in the form of *Kullback-Leibler* (KL) divergence between $\mathcal{P}(z|X)$ and $\mathcal{Q}(z|X)$:

$$\mathcal{KL}[\mathcal{Q}(z|X)\|\mathcal{P}(z|X)] = \mathbb{E}_{z\sim\mathcal{Q}}[\log\mathcal{Q}(z|X) - \log\mathcal{P}(z|X)] \tag{3}$$

In this way, $\mathcal{Q}$ is "encoding" X into z, and $\mathcal{P}$ is "decoding" z to reconstruct X. Therefore, the objective is two-fold: maximizing $\log\mathcal{P}(X)$ while simultaneously minimizing $\mathcal{KL}[\mathcal{Q}(z|X)\|\mathcal{P}(z|X)]$. Assuming an arbitrarily high-capacity model for $\mathcal{Q}(z|X)$, then $\mathcal{Q}(z|X)$ will match $\mathcal{P}(z|X)$, in which case the $\mathcal{KL}$-divergence term will be zero and the result is directly optimizing log $\mathcal{P}(X)$. To compute the optimization can be sampled a single value of X and a single value of z from the distribution $\mathcal{Q}(z|X)$, and consequently computed the gradient of:

$$\log\mathcal{P}(X|z) - \mathcal{KL}[\mathcal{Q}(z|X)\|\mathcal{P}(z)] \tag{4}$$

Then, the result will converge by averaging the gradient of the function over arbitrarily many samples of X and z. For a mathematical introduction to all VAE components, the reader can refer to [59]. To better understand the optimization process and how to overcome the bottlenecks of variational inference, refer to [56].

Artificial Neural Networks

Artificial Neural Networks (ANN) are a type of ML model inspired by the brain's biological structure. ANN consists of nodes, or neurons, connected in layers, and each layer performs a specific type of computation on the input data. The nodes take one or more inputs and produce an output based on a set of learnt parameters called weights. Each input is multiplied by its corresponding weight, and the resulting products are summed up. The sum is then passed through an activation function, which determines the output of the last node [62]. ANN are usually used to model the $f(z;\theta)$ in Equation (2) using an iterative optimization process feeding the auto-encoder with data, comparing output with the initial data and backpropagate the error to update the weights of the networks.

In this thesis, we concentrate on a specific subset of architectures and layers, given that numerous variations are available. They comprise *Long Short-Term Memory* (LSTM) networks, designed to handle sequential data such as time series with a memory cell that can store information for long periods of time and gates to control the flow of information into and out of the cell [52]. Instead, *Attention Mechanisms* (AT) are another type of ANN that can selectively focus on parts of the input data, allowing the network to learn how to give more weight to important features. Attention layers are often used in natural language processing tasks [91].

2.5. Regression Methods in Reinforcement Learning

In Fitted-Q-Iteration (FQI), *regression* estimates the action-value function from a set of state-action pairs. The original paper studied the convergence with different tree-based regressors [40], showing that *totally randomized trees* guarantee good approximate performance and convergence. Another tree-based algorithm, named *extremely randomized trees*, performs consistently better even though it does not ensure the convergence of the approximation [47]. Finally, another tree-based model is *XGBoost* [29], an implementation of gradient-boosted decision trees. In the literature, only a few examples combining FQI and XGBoost are available, but the latter beats most of the algorithms available in terms of performance and computational efficiency for supervised learning tasks. All the methods above are an evolution of decision trees [19] and random forests [51]. This thesis focuses on the FQI with XGBoost as a tree-regressor model for the estimation of the Q-function.

2.5.1 Decision Trees

A *decision tree* [19] is a supervised learning method based on a tree model that, in regression problems, predicts the value of the target y_i, given a datum $x_i \in \mathbb{R}^d$. Considering X as the data space, a decision tree iteratively produces hyperplanes to recursively split the data space partitioning it until the points inside each set of the partition are relatively homogeneous in terms of the target y_i. After the tree is trained, X is partitioned, collecting train data into subgroups. Then, it is possible to assign each test point to one subgroup, a leaf of the tree, and its target can be predicted as the mean value of the targets of the train data belonging to the same set. Summing up, a decision tree consists of internal nodes that split the data depending on the value of a feature selected for that node and leaf nodes that represent a set of the partition of X. They are labelled with the predicted value of the target of data in that set, computed in regression as the mean of train targets.

2.5.2 Random Forest

A *random forest* [19] is an ensemble model ensemble consisting of many decision trees with a random selection of features at each split. The idea is to use many weak learners as uncorrelated as possible to form a strong learner. The correlation between decision trees is reduced in two ways. Firstly, each decision tree has a training set composed of the same number of data N as the original training set, extracted using the bootstrap technique, which consists of sampling with replacement N data from the original set. Secondly, the feature on which the split is based at each node split can be selected only between m variables, randomly selected from the d available features. All decision trees are trained this way: the training set is extracted with Bootstrap, and at each split, only some features can be selected, and the trees are unpruned. Each one of them is a weak regressor since its training is not optimized, and it will probably overfit, but all together, they become a strong learner since their learning procedure, which singularly is not optimal, reduces the correlation among trees, decreasing the variance. A possible variant is called *extremely randomized trees* [47], where randomness is even more exploited. In particular, a third random procedure is performed to decrease the variance: as in random forests, a random subset of m features is available at each node to perform the split, but instead of looking for the best threshold, some values are randomly chosen for each available feature, and the best of these randomly generated thresholds is used to perform the split.

2.5.3 XGBoost

XGBoost stands for Extreme Gradient Boosting, and it is a decision-tree-based ensemble algorithm that uses a gradient-boosting framework, where *boosting* is an ensemble technique that consists in adding new models to correct the errors made by the old ones. Models are added sequentially until no further improvements can be made. Instead, *gradient boosting* is a boosting approach where new models are created to predict the residuals, or errors, of prior models and then added together to make the final prediction. The name stems from the fact that it uses a gradient descent algorithm to minimize the loss when adding new models. XGBoost has been successfully applied to solve a wide range of problems, from classification to regression tasks, obtaining optimal performances and high computational efficiency.

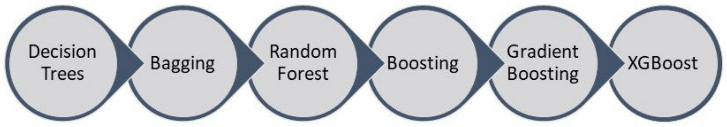


Figure 6: Evolution of tree-methods from decision trees to XGBoost

2.6. Reinforcement Learning in Trading

The previous chapters highlighted how market participants are sequentially taking decisions based on the currently available information and then observing the performance of their trades, called *profit and loss* (P&L). Thus, the goal of *quantitative trading* is to maximize long-term profits under a certain risk tolerance by executing an appropriate strategy.

The mathematical approach to many financial decision-making problems has traditionally been through modelling with stochastic processes. Still, recently the financial ML field has grown in popularity following advancements in both theory and computational methods, even though a significant portion of the existing trading algorithms are *expert systems* [68]. Usually, computer scientists write hard-coded rules to exploit opportunities or implement knowledge from experienced traders from high to low frequencies. Either way, they exhibit poor generalization ability and only perform well in certain market conditions.

Indeed, the advantages of RL for trading are numerous:

- (*i*) provides a framework to solve sequential decision-making problems;
- (*ii*) allows training agents that take as input the market information and outputs actions directly, also without making assumptions;
- (*iii*) bypasses the extremely hard task of future price prediction given the noisy nature of the markets;
- (*iv*) task-specific constraints can be imported into RL objectives easily;
- (*v*) allows solving the problem by learning optimal policy for short-term gain and capable of handling future conditions.

However, creating autonomous agents that can operate effectively in the real-world demands sophisticated simulators used for training and testing, particularly in cases where potential opportunities are brief and offer only limited gains. This theory finds support in the economic field, as explained by the seminal article on the *adaptive market hypothesis* [67]. This hypothesis posits that some markets may not yet be mature enough to be fully efficient. Therefore inefficiencies can be exploited to make a profit, in contrast to the *efficient market hypothesis* [41]. To summarize, quantitative traders try to find patterns, trends and inefficiencies of a large number of instruments leveraging statistical and ML tools that do not necessarily correlate with the general market movements. Various successes have been achieved in the finance industry by implementing RL algorithms for diverse trading problems, most known under the name of *algorithmic trading.*

2.6.1 Asset Trading

The most renowned is the *single asset trading*, which refers to the process where traders consistently buy/sell one financial instrument to make a profit. It can be applied to stocks and commodities other than FX. In the beginning, traders allocate some cash, and at each time step t, they buy/hold/sell some amount for changing positions to maximize the final net value at the end of the period. Pushed to the extreme, it becomes *scalping trading* that aims at discovering micro-level opportunities and making a profit in a few minutes or *high-frequency trading*, characterized by high turnover rates and order-to-trade. These are some attempts of applying RL to this scenario [20, 66, 88, 99].

2.6.2 Optimal Execution

Another RL application is to solve the problem of *optimal execution* [6], essential in financial modelling, where a trader aims to buy/sell a specific amount of a single asset within a designated time frame. The trader's objective is to find strategies that minimize the execution cost of the transaction and the market impact. The most popular types of RL methods used for this use case are Q-learning and double DQN algorithms [50, 72].

2.6.3 Market Making

Differently, the objective of *market making* is not to profit from identifying the correct price movement direction but profit from earning the bid-ask spread without accumulating significant positions. Indeed, a market maker in a financial instrument is an individual trader or an institution that provides liquidity to the market by placing buy and sell limit orders in the order book while earning the spread. Also, most of the RL approaches are value-based, either around Q-learning algorithm [5] or SARSA [84].

2.6.4 Options Pricing

A cornerstone of computational finance is understanding how to price and hedge derivatives, named *options pricing*. Options are contracts that give the holder the right but not the obligation to buy/sell an underlying asset at a specified price (strike) on a specified date (expiration date). The most applied mathematical model to price options is the Black-Scholes model [15], but recently RL approaches, including deep Q-learning [74], PPO [37] and DDPG [24] have been applied for hedging and derivatives.

2.6.5 Further Developments

The application of RL in finance may discern from the trading landscape. Thus it has been applied with success also in asset and wealth management to enhance asset allocation via *portfolio optimization* [97] and *robo-advising* [7] to suggest investment opportunities in an automated fashion. Finally, a list of future directions for RL in finance may be interesting for the in-depth interested reader, and this includes Multi-Objective-RL [100], Dark Pools optimal execution [45] and Sample Efficiency for non-stationarity [95].

3. Related Works

As explained in Section 1 and the following chapter, FX trading changed entirely in the last years due to the increasing amount of data available and the adoption of advanced technologies such as ML and RL. The following section reviews the relevant literature on RL in FX trading. While most of the existing literature focuses on other asset classes, several contributions explore the application to the currency market. An overview of the state-of-the-art and classification of the existing based on the RL paradigms and algorithms is presented in Table (1). To apply RL in the real markets, it is necessary to build simulations that emulate the highly complex and dynamic trading system. Therefore we aggregated the assumptions and characteristics used by the authors in Table (2). It is essential to highlight that while algorithmic trading and quantitative models are extensively used in the finance industry, limited information is publicly available due to their proprietary nature. Highly sophisticated players rely on advanced statistics, signal processing, and ML techniques to extract trading signals and generate automated decisions, but these profitable strategies are rarely disclosed.

3.1. Evolutionary Reinforcement Learning

The core idea behind *Evolutionary RL* (ERL) is that *genetic algorithms* (GAs) can improve the performance of RL trading systems by finding a suitable state representation; hence the works try to combine value-based methods with such algorithms. All the works share the Watkins Q-Learning for PO-MDP [93]. In ERL with the GA formulation, a population of possible solutions is encoded as a set known as parameter strings, each of the same fixed length. The fitness of each string in the population is estimated, and the GA operators (crossover, selection and mutation) are applied. The process provides a second-generation population whose average fitness is greater than the initial population's. Then, the process is repeated until some stopping criterion is met.

In their seminal work, *Dempster et al.* [35] used eight technical indicators as state variables and a GA to identify the subset that yielded the best trading performance [33]. They divided the training into two parts: in the first, a separate RL trading agent is trained for each combination of technical indicators generated by the GA; in the second, the trading performance of each of these agents is evaluated on the training data. Then, the results were fed back to the GA to evaluate the fitness of the individuals and produce a superior generation. They claimed the hybrid system outperforms an RL-only system on various currency pairs, suggesting that GAs can be successfully employed to optimize the state representation.

Similarly, *Hryshko et al.* [53] employed a GA in-sample trading strategy search to select the optimal trading strategy composed of entry and exit rules. Then, the financial indicators extracted from the data were passed to the RL, in this case, Q-Learning. The algorithm could then be used online to search for the optimal strategy through repetitive experience continually. To estimate the profitability of trading systems, the authors consider the Stirling Ratio, defined as the profit divided by the maximum drawdown, the maximum loss of capital.

Instead, *Bates et al.* [10] used computational intelligence techniques on some major currency pairs (GBP/USD, EUR/USD and USD/JPY) to analyze the order flow and order book data. They extracted indicators from the order book, and, coupled with GA; they created an ERL system that mapped the indicators representing the market state to a direct position.

3.2. Recurrent Reinforcement Learning

The *Recurrent RL* (RRL) approach represents the first attempt to overcome the limitation of supervised learning methods by creating a system that combines recurrent neural networks and RL via a policy-based mechanism to outperform classical value-based strategy and maximize the profit of a trading strategy.

The pioneers of RRL are *Moody et al.* [71] that in 2001 introduced a direct RL algorithm which outperformed Q-Learning in an algorithmic trading task and forecasting methods on artificial data. Their implementation is a one-layer neural network taking as input the past eight returns and its previous output aiming to directly maximize a risk-adjusted profit metric, the differential Sharpe Ratio. The strategy was tested on eight months of half-hourly data achieving an annualized profit of 15%.

A further test of the RRL approach has been done by *Gold et al.* [48] by increasing the complexity with multi-layer neural networks to perform a comprehensive empirical study of more complex decision functions on several currency pairs with a one-year half-hourly data. The final profit level varied considerably across currency pairs, from -82.1% to 49.3%, with an average of 4.2% over ten pairs, but no substantial improvement is observed.

Reinforcement Learning in FX Trading				
Article	**Paradigm**	**Classification**	**Algorithm**	**Time**
Riva et al. [79]	Batch RL	Value-based	Persistent-FQI, OAMP	1-min
Riva et al. [78]	Batch RL	Value-based	Persistent-FQI	1-min
Borragiero et al [18]	Recurrent RL	Policy-based	Policy Gradient	1-day
Bisi et al. [13]	Batch RL	Value-based	FQI, MO-FQI	1-min
Zarkias et al. [98]	Deep RL	Value-based	DDQN	4-hours
Sornmayura et al. [83]	Deep RL	Value-based	DQN	1-day
Carapuco et al. [25]	Deep RL	Value-based	DQN	1-tick
Huang et al. [54]	Deep RL	Value-based	DRQN	15-mins
Cumming et al. [32]	-	Value-based	LSTD	1-min
Dempster et al. [34]	Recurrent RL	Policy-based	Policy Gradient	1-min
Hryshko et al. [53]	Evolutionary RL	Value-based	GA, Q-Learning	5-mins
Bates et al. [10]	Evolutionary RL	Value-based	GA, Q-Learning	10-mins
Gold et al. [48]	Recurrent RL	Policy-based	Policy Gradient ANN	30-mins
Dempster et al. [35]	Evolutionary RL	Value-based	GA, Q-Learning	1-min
Moody et al. [71]	Recurrent RL	Policy-based	Policy Gradient	30-mins
Gao et al. [46]	-	Value-based	Q-Learning, QSR	-
Vittori et al. [92]	-	Online Planning	MCTS	1-min
Bisi et al. [14]	-	Policy-based	TRVO	1-min
D'Eramo et al. [39]	-	Value-based	D-QL, W-QL	-

Table 1: Classification of RL in FX Trading papers in chronological order starting from the most recent, except from the last three which are using the setting only as an experiment of their own proposed algorithm.

From a practical standpoint, it may be beneficial to determine currency pair-specific parameters. The authors can be considered as early precursors of Deep RL.

In 2006, *Dempster et al.* [34] moved forward the RRL methodology by proposing the *adaptive RL* (ARL) three-layers system built upon [71]. The ARL system has an added risk management layer and dynamic hyper-parameter optimization layer on top of the RRL algorithm. It allows users to input their risk preferences and parameters to adjust the agent's behaviour to market conditions. The ARL system was tested on two years of EUR/USD historical data with a one-minute granularity, achieving an average 26% annual return.

The most recent approach is from *Borragiero et al.* [18], who investigated transfer learning and RRL by transferring feature representation from a radial basis function network to an agent trained to target a position directly using a quadratic Sharpe Ratio as a reward function. The agent is applied in an FX trading scenario accounting for transaction and funding costs. The results showed an annualized information ratio of 0.52 with a compound return of 9.3% over a seven years test set.

3.3. Deep Reinforcement Learning

The emergence of *Deep Reinforcement Learning* (DRL) has led to promising new approaches to trading, where agents powered by ANN learn how to optimize trading strategies directly. Building over existing approaches that relied on simpler networks [48, 71], DRL uses complex networks that can handle the diversity of market dynamics, and it has shown the potential to outperform traditional trading strategies with better returns. The availability of computational power has been a key factor in the recent development of DRL.

Carapuco et al. were the first to propose a combination of artificial neural networks (ANN) and RL for forex trading, as described in their paper [25]. The authors developed a three-layer Deep Q-Network (DQN), with ReLU as the activation function, that made profitable trades on out-of-sample data. The algorithm was tested on the EUR/USD pair over eight years using different initial conditions with an overall yearly average profit of over 15%. A weak point is more sophisticated measures than profitability could have measured that model.

Similarly, *Sornmayura et al.* [83] applied DQN to forex trading using 15 years of data. The experimental results showed that the DQN guarantees higher returns than the baseline strategies, buy&hold and expert trader. However, it significantly outperformed the expert trader only on EUR/USD with a 43.88% annualized return and not on USD/JPY with a 26.73% annualized return. The strength of their work is that the algorithm was compared with the performance of an expert trader, which was rarely done.

Huang et al. [54] tested a variant of the Deep Q-Network (DQN), called Deep Recurrent Q-Network (DRQN) leveraging recurrent layers to process temporal sequences. They also considered a second target network to stabilize the process further, eliminating the need for random exploration. They used OHLC, volume and indicators as state composition plus a small replay memory during training. The algorithm was 12 currency pairs within a 15-minute time frame with realistic costs. The algorithm yielded a 26.3% return assuming transaction costs equal to 0.1% over all the traded currency pairs. Interestingly, a larger transaction spread generated better returns, suggesting the algorithm sought more creative ways to trade profitably under certain constraints.

Recently, *Zarkias et al.* [98] proposed an extension of the DQN algorithm, named Double Deep Q-Learning (DDQN) by reformulating FX trading as a control problem. They introduced a novel price trailing mechanism that goes beyond traditional price forecasting. The trading agent closely follows the asset's price, within a certain margin, instead of accurately predicting the future price direction or deciding whether a trade has to be performed. The control-based approach, an alternative formulation to traditional trading RL methods, increases the ability of the agent to withstand large amounts of noise while still capturing the price trends and allowing it to take profitable decisions.

3.4. Further Reinforcement Learning approaches

In addition to the approaches described above, other methods in the literature apply RL algorithms to FX trading. These methods may not fit neatly into the previous paradigms, but they are still significant contributions to the field.

Gao et al. [46] proposed a trading framework for the FX market that combines Q-Learning and Sharpe Ratio maximization algorithms. The RL system was trained to optimize the absolute profit via reward and used supervised learning for a second system that maximizes Sharpe Ratio (SR) return. The RL algorithm estimates the Q-function of long/short positions on the selected currency pair, while the SR maximization optimizes target portfolio weights. The final investment decision is made by combining the results of the two algorithms.

More recently, *Cumming et al.* [32] introduced an RL trading mechanism based on *Least-Squares Temporal Difference* (LSTD), a technique that estimates the state value function. The state comprises a window of OHLC data of the last 8 minutes, while the reward is purely the profit from each transaction. The idea was tested on several currency pairs such as EUR/GBP, USD/CAD, USD/CHF and USD/JPY with limited success yielding only 0.839% annualised return overall currency pairs. Furthermore, the authors made the simplifying assumption of not considering transaction costs and still achieved a low level of profitability.

Differently, *Vittori et al.* [92] applied *Monte Carlo Tree Search* (MCTS), an online planning algorithm, in the context of trading to deal with non-stationarity and changing environments. MCTS constructs a search-tree using a one-step model of the environment to evaluate different available actions. The method is an open-loop variant of UCT [61] with a backup procedure that uses Q-Learning Temporal Difference. Furthermore, a novel generative model is proposed, which uses past observations to generate possible future realizations of the market for planning.

The last two works of this chapter do not directly address the FX Trading problem, but it has been considered part of their experiments to validate the performance of the novel proposed method.

Bisi et al. [14] proposed a risk-averse algorithm called *Trust Region Volatility Optimization* (TRVO) for reward-volatility reduction that was tested in the FX trading scenario. TRVO trains a series of agents characterized by different risk-aversion coefficients, and it can span an efficient frontier on the volatility-P&L space. Simulation results demonstrate that it outperforms the classic policy-based model from [71], obtaining good performance.

D'Eramo et al. [39] attempted to use value-based methods with Gaussian approximation to predict the most profitable action to take at any point in time. Different variations of Q-Learning were trained and tested, including transactions cost: Q-Learning, double Q-Learning and *Weighted Q-Learning* (WQL) as well as weighted policy for WQL. All of them showed similar and profitable performance via the proposed approximation method.

FX Trading Environment

Article	State	Action	Reward	T	S	BAS	I
Riva et al.	Window, Time, Position	Buy, Sell, Hold	Profit	✓			
Riva et al. [78]	Window, Time, Position	Buy, Sell, Hold	Profit	✓			
Borragiero et al [18]	Window, EWA, TL	Buy, Sell, Hold	Sharpe Ratio	✓		✓	
Bisi et al. [13]	Window, Time, Position	Buy, Sell, Hold	Profit	✓			
Zarkias et al. [98]	Window	Buy, Sell, Hold	Custom	✓	✓		
Sornmayura et al. [83]	OHLC, Indicators	Buy, Sell, Hold	Profit				
Carapuco et al. [25]	Indicators, Position, Profit	Buy, Sell, Hold	Sortino Ratio			✓	
Huang et al. [54]	OHLC, Position, Time	Buy, Sell, Hold	Log Returns	✓		✓	
Cumming et al. [32]	OHLC, Position, Window	Buy, Sell, Hold	Cum. Profit				
Dempster et al. [34]	Prices, Position	Buy, Sell	Custom	✓		✓	
Hryshko et al. [53]	GA Indicators	Buy, Sell, Hold	Stirling Ratio	✓			
Bates et al. [10]	GA Indicators	Buy, Sell, Hold	Profit	✓	✓		
Gold et al. [48]	Prices, Position	Buy, Sell	Profit, SR	✓		✓	
Dempster et al. [35]	GA Indicators, Position	Buy, Sell, Hold	Profit	✓			
Moody et al. [71]	Prices, Position	Buy, Sell, Hold	Profit, DSR	✓		✓	✓
Gao et al. [46]	Prices, Profit	Buy, Sell	Profit, SR				
Vittori et al. [92]	Window, Time, Position	Buy, Sell, Hold	Profit	✓		✓	
Bisi et al. [14]	Window, Time, Position	Buy, Sell, Hold	Rew. Vol.	✓			
D'Eramo et al. [39]	Indicators, Position	Buy, Sell, Hold	Profit	✓		✓	

Table 2: Comparison of environment characteristics of RL in FX Trading: **T** means transactions, **S** means slippage, **BAS** means bid-ask spread, **I** means market impact; if there is a checkmark in the table, the characteristic has been considered by the author in the environment simulation.

3.5. Batch Reinforcement Learning

In this chapter, we provide a detailed description of the previous studies that are most similar to our work in the *Batch-RL* (BRL) or *offline-RL* framework. In Section 2.3.2, the theoretical background has been extensively explained; hence, we aim to focus on the specific settings and findings of the previous authors. Our goal is to provide a comprehensive understanding of this area's existing research and highlight how our thesis contributes to the field. This way, we provide a foundation for our experimental design and analysis.

3.5.1 Multi-Objective FQI

The first attempt has been proposed by *Bisi et al.* [13], where the authors investigated the usage of FQI for finding a trading strategy. They framed the problem to include risk aversion into the *multi-objective MDP* (MO-MDP) framework to build the Pareto frontier of different financial objectives [80]. They considered as risk measure the *reward volatility*, which evaluates the uncertainty about the step-by-step rewards obtained from the environment. Thus, profit maximization alongside risk minimization creates a multi-objective optimization problem.

To test their ideas, they implemented *Multi-Objective FQI* (MO-FQI) [27], a generalization of FQI, and provided an empirical evaluation on historical datasets from 2014 to 2019. The usual approach is to reduce the problem to a single objective by performing a weighted combination of rewards using a parameter λ. Instead, they enlarged the state by the weight vector λ before applying FQI to approximate the action-value function $Q^*(s, \lambda, a)$. Ideally, if the learning is made over a significant set of weights $(\lambda_i)_{1 \leq i \leq n}$, the resulting policy will successfully generalize to unseen values of λ. As the state space is expanded, MO-FQI allows for a more thorough exploration of the trade-off between multiple objectives, leading to better-informed decision-making in complex environments. Furthermore, MO-FQI can be computationally more efficient than running multiple single-objective FQI when the number of such objectives gets significant. The algorithm has been trained with supervised learning regression using a tree-based model *extra trees*, based on random forests [51]. An important property is that extra trees estimate the features' importance in predicting the target. Thus, they analysed the importance of each feature and fixed all hyper-parameters but the min-split to perform model selection.

The experimental results demonstrated that the agent can identify profitable temporal patterns exploited to maximize returns. Their algorithm is benchmarked against the classical buy&hold, sell&hold strategies plus some technical indicators strategy, namely trend following and mean reverting. They also found that as risk aversion increases, the resulting trading policies tend to be more conservative, resulting in longer-held portfolio positions and smoother policies. Overall, their findings offer insights into using RL for trading strategy optimization and highlight the importance of considering multiple objectives when managing investment risk.

3.5.2 Persistent FQI

A first extension of the previous work was published by *Riva et al.* [78], who introduced a simultaneous multi-currency trading framework via FQI with extra trees, focusing on the two-currencies and three-currencies models. Their contribution is two-fold. On the one hand, they analyzed the performance on a multi-asset scenario; on the other hand, they evaluated performance with different trading frequencies. They included the transaction costs but assumed a small trading size that did not cause any market impact or slippage.

They studied the importance of tuning the control frequency by proposing the idea of *action persistence*, which is crucial to obtain effective policies and exploiting the best opportunities. It consists of the repetition of each individual action for several consecutive steps. Thus, the transition lasts for k steps whenever the model issues an action. They backtested the developed approach on real FX data from 2017 to 2020, comparing results employing a single pair or both ones simultaneously with two benchmark strategies: buy&hold, sell&hold.

They assessed that *persistence* equal to *5* and *10* outperform the agents trained with *persistence* equal to *1* on both the benchmark strategies. The poor performances of the models with persistence equal to 1 can be explained by the worse signal-to-noise ratio, which deeply affects learning using high frequencies. Looking at the policies learned by the models, another relevant fact is that the higher the persistence, the better the agent exploits temporal patterns. Indeed, it is interesting to mention that the importance of time-related features (i.e. time and weekday) becomes significantly higher as the persistence increases. In contrast, the portfolio feature becomes less relevant for higher persistence values. Furthermore, higher persistence also allows computational advantages since, given the same number of iterations, the optimization horizon becomes shorter as the persistence decreases. Therefore, higher persistence optimizes on longer horizons, using the same number of iterations.

3.5.3 Online Learning with FQI

Building on their previous research, *Riva et al.* [79] proposed a batch RL algorithm that overcomes the financial market regime switches issue by modelling the trading task as a non-stationary RL problem.

They build a novel technique for the dynamic selection of the best RL agent only driven by profit performance. Their two-layer approach allows choosing algorithmically the best strategy among a set of RL models through an online-learning method. They employed an offline algorithm, FQI with XGBoost as a regressor, for the RL layer and *Optimistic Adapt ML-Prod* (OAMP) for online learning [94]. In this way, the model selection procedure becomes dynamic and choosing a strategy based on its performance in past regimes is unnecessary. OAMP is an expert learning method focused on learning the optimal policy in non-stationary environments, including switching and drifting dynamics.

In more detail, the RL algorithm generates a set of trading experts, trained under different conditions and different levels of complexity, while the online learning is in charge of updating, at the beginning of each trading day, the weights associated with these strategies by receiving expert feedback based on the P&L. These weights determine the portion of the total budget assigned to each expert, decoupling the different contributions of each expert.

The system was trained and tested on FX data for the AUS/USD and GBP/USD currency pairs and compared with two benchmark strategies: the Buy&Hold and the Sell&Hold. The results showed the two-step approach outperformed the baselines considered. More significantly, it ensured greater returns than those obtained by the experts that would have been chosen through the offline model selection procedure.

4. Research Method

There exist three main cultures around data-centric applications in quantitative finance [9]:

(i) the *data modelling culture* is characterized by the belief that financial markets can be described as a black box with a simple model that generates the observational data, and the argument is that simple models are prone to failure;

(ii) the *machine learning culture* makes use of complex and sometimes opaque functions to model the observations without always claiming the nature of the underlying process;

(iii) the *algorithmic decision-making culture* that bypasses the stage of learning and focuses on distinguishing good decisions from bad decisions to train electronic agents directly.

In this thesis, we approach the Forex trading problem from a *machine learning* perspective, in particular, using RL value-based techniques. To provide a clear overview of the research methodology for building the RL system, we present a workflow pipeline in Figure (7). This pipeline consists of five sequential steps that must be followed to ensure proper execution and accurate performance assessment. The rest of the section will guide the reader throughout the pipeline, starting from the problem formulation.

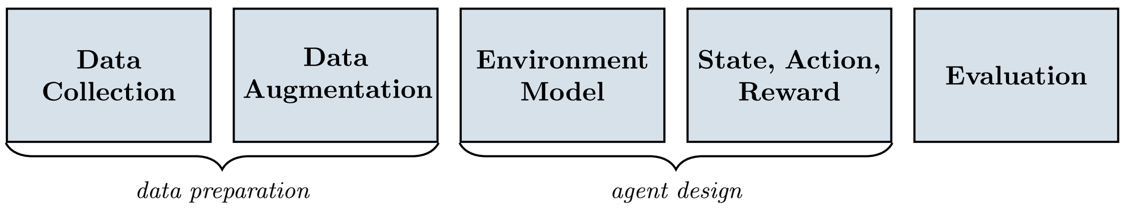


Figure 7: RL pipeline workflow for single asset trading problem.

4.1. Problem Formulation

This section aims to define the *trading* task for a single Forex currency pair and explain how it can be mapped into a mathematical framework. We begin with a formal definition of the task, followed by a detailed explanation of how we translate the problem into a *Markov Decision Process* throughout the paragraphs.

Definition 4.1. *(Trading). Given an asset to trade, trading can be defined as a sequential decision process in which at each (discrete) round $t \in \{1, \dots, T\}$ over a trading horizon $T \in \mathbb{N}$, a trader decides whether to go long, short or stay flat with respect to the asset to maximize his wealth. The trader's position is represented by the action $a_t \in \{-1, 0, 1\}$.*

In our problem, the asset is a *currency pair.* It refers to the exchange rate between two different currencies in the foreign exchange (FX) market. For example, the EUR/USD currency pair represents the exchange rate between the Euro and the US Dollar. In this case, the EUR is the *domestic, or base* currency, and the USD is the *quote, or foreign* currency. An exchange rate indicates how much foreign currency is needed to buy one unit of the base currency. So, for instance, if the exchange rate is 1.20, it means that one Euro can be exchanged for 1.20 US Dollars.

Our *problem formulation* aims to maximize the profits obtained in the domestic currency. Therefore, there are two viable options: trading a fixed quantity of the foreign currency for a variable amount of the domestic or trading a variable amount of foreign currency for some fixed amount of the base one. In this work, it is considered the second case since, assuming USD as the base currency, it is possible to see the base currency as an asset, where we have the instantaneous exchange rate in place of the price. As a result, rewards are expressed in the foreign currency. However, returns must be expressed in the same currency to adequately evaluate performance. Thus, the collected rewards are converted into the base currency at the end of a trading day. To make a realistic assumption about the problem, we are considering transaction costs, but it is assumed that there is no market impact or slippage because operations are made on highly liquid instruments and of small size; hence no decision around the size of the allocation has to be made.

4.2. Data Collection

This section describes the data collection process pursued for the experiments of this thesis. This work experiments on one-minute bar FX real data of EUR/USD and USD/JPY currency pairs from 2018 to 2022, downloaded from HistData.com [3]. Once the data was downloaded and stored on an offline database, it had to be extracted, parsed to create a CSV and cleaned to create the input for RL algorithms.

The dataset used in the thesis consists of the EUR/USD and USD/JPY exchange rate per minute covering the period from Monday to Friday since the markets are closed on Saturday and Sunday. The *original* dataset spans 2018 to 2022, and for each day, it includes observations collected between 00:00 and 23:59, based on the Eastern Standard Time Zone (EST). A small sample of EUR/USD exchange rates from 2022 is provided in Table (3), where it is shown the data includes date and time information that allows for temporal analysis of the exchange rate, while the minute bars provide price information on the opening, closing, highest, and lowest quotes for each minute (OHLC bar), which are crucial components for developing and testing trading strategies. Together, these pieces of information form a comprehensive dataset that can be used to gain insights into the dynamics of the two currency pairs.

Date	Time	Open	High	Low	Close
2022.01.04	02:50	1.12864	1.12888	1.12863	1.12888
2022.01.04	02:51	1.12887	1.12893	1.12879	1.12889
2022.01.04	02:52	1.12889	1.12927	1.12889	1.12920
2022.01.04	02:53	1.12917	1.12923	1.12908	1.12910
2022.01.04	02:54	1.12910	1.12910	1.12889	1.12893
2022.01.04	02:55	1.12892	1.12892	1.12873	1.12876

Table 3: Sample of EUR/USD exchange rates for a 5-minute interval in 2022.

4.3. Data Augmentation with Latent Variables

This chapter presents the novel methodology used to extract a latent state representation to augment the Batch-RL formulation of the trading problem. The proposed framework involves evaluating various *Variational Auto-encoders*(VAE) [58] architectures, namely *TimeVAE* [36] and *LSTM-VAE* [30], to identify highly accurate models for reconstructing time-series data, and then extracting the latent feature representation from the encoder output.

The *latent variables* are used as features for the FQI algorithm to improve the learning of trading strategies in a non-stationary setting. The goal was to demonstrate that such latent variables can capture the underlying patterns and dynamics of the data in a compressed form, allowing better to exploit the temporal structure during the regression mechanism.

The VAE models are trained using a specific loss function that consists of two parts: the *reconstruction* loss and the $\mathcal{KL}$*-divergence* loss.

The *reconstruction* loss measures how well the VAE can reconstruct the input data from the latent variables generated by the encoder network and it is the mean squared error between the original input data and the output data. The reconstruction loss encourages the VAE to generate outputs that are similar to the input data.

The $\mathcal{KL}$*-divergence* loss measures the difference between the distribution of the latent variables and a predefined prior distribution. The goal of this loss is to encourage the latent variables to follow a particular distribution, in our case a Gaussian distribution, that allows for easy sampling and interpolation while preventing the VAE from overfitting the training data.

In a mathematical formulation, it is:

$$L_{total} = L_{recon} + L_{\mathcal{KL}} \tag{5}$$

where L_{recon} is the *reconstruction* loss and $L_{\mathcal{KL}}$ is the $\mathcal{KL}$*-divergence* loss.

By breaking down the equation, we have:

$$L_{recon} = \frac{1}{N}\sum_{i=1}^{N}||x_i - \hat{x_i}||^2 \tag{6}$$

where N is the number of data points in the training set, x_i is the input data point, and $\hat{x_i}$ is the output of the decoder network when it receives the corresponding latent variable z_i as input.

$$L_{\mathcal{KL}} = -\frac{1}{2}\sum_{i=1}^{I}(1 + \log(\sigma_i^2) - \mu_i^2 - \sigma_i^2) \tag{7}$$

where I is the dimensionality of the latent space, μ_j and σ_j are the mean and standard deviation of the i-th element of the latent variable z, and the sum is taken over all I elements of z.

The two models used to extract latent variables are explained in detail in the rest of the chapter.

4.3.1 TimeVAE

The *TimeVAE* is a model proposed for multivariate time series generation, and it has several distinct properties: interpretability, ability to encode domain knowledge, and reduced training times. The architecture uses a combination of traditional deep learning layers such as dense and convolutional layers and custom layers to model time-series specific components such as level, multi-polynomial trend, and seasonal patterns [36].

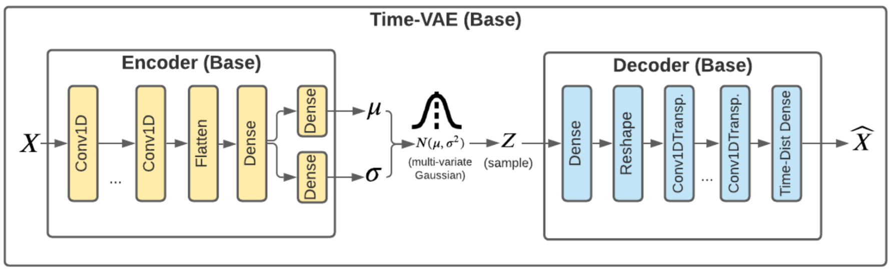


Figure 8: Block diagram of the main components in TimeVAE.

Their model showed that the VAE could accurately represent the temporal attributes of the original data with an architecture that can incorporate domain-specific time patterns such as polynomial trends and seasonalities to provide interpretable output that can accurately model the temporal components of real-world data. Still, it was not tested with financial data. The authors claimed their model meets the performance of top generative models when measuring similarity with original data and outperforming available methods on the next-step prediction task.

4.3.2 LSTM-VAE

The *LSTM-VAE* (Long Short-Term Memory Variational Autoencoder) is a deep learning model that is used to extract meaningful features from time series data by using a combination of two powerful techniques: LSTM and VAE [30]. In this architecture, LSTM is a type of recurrent neural network responsible for learning and representing long-term dependencies in sequential data, while VAE learns a compressed representation of the data.

The main idea behind LSTM-VAE is to use the LSTM component to encode the time series data into a latent space representation and then use the VAE component to decode this representation back into the original time series. It is trained to optimize the reconstruction error between the input time series data and the output generated by the VAE and maximise the effectiveness of the latent space representation. The resulting model can be used for various tasks such as anomaly detection, data imputation, and time series forecasting and has shown promising results in a wide range of applications. In our context, we are interested in extracting the output of the encoder architecture to retain the latent variables for a further prediction step.

4.4. Agent Design

In this section on *agent design*, we focus on the process of designing an RL agent as an *episodic Markov Decision Process* (MDP). To do that, we need to consider how the environment is described as an MDP and the actions, states, and rewards. The action space specifies the set of actions the agent can take, while the state space specifies the set of states the environment can be in. The reward function defines the reward that the agent receives at each time step based on its actions and the state of the environment. Formulating these three components is critical to designing an RL agent since it significantly impacts its ability to explore and learn.

4.4.1 Environment

As mentioned above, the single currency pair (or asset) trading problem can be modelled as an *episodic* MDP, refer to 2.3.2, where each episode corresponds to a trading day.

There are two reasons for the choice of the fixed-length episode:

(*i*) it allows for the undiscounted setting ($\gamma = 1$).

(*ii*) closing all positions before the end of the day is more practical from a financial standpoint.

Furthermore, it reduces the overnight risk and fees charged by FX brokers. To ensure consistency and robustness in our approach, we focused on the trading hours able to cover US, EU and Asian trading hours given the chosen currency pairs; hence we restricted the daily window from 00.00 EST to 8.30 PM EST (Eastern Standard Time), covering the necessary time frame from 4:00 AM to 7:00 PM with an *episode* length of 1230 time steps.

4.4.2 State

The agent *state* design depends on the specific problem being tackled. While modelling the Forex trading environment as an MDP, a critical aspect is the *Markov property.* This means that the transition to the next state should be independent of previous states, given the latest observation. To ensure this property, the state should contain all relevant information related to past market observations that may have affected the transition to the next state.
Therefore, we included:

(*i*) the last 60 exchange rate variations between consecutive minutes computed as the differences between the price at a certain time-step and the previous one, normalized by the value of the former one:

$$d_{k,t} = \frac{p_{t-k+1} - p_{t-k}}{p_{t-k}}$$

where t is the time, p the reference price and $\{k = 1, 2, \ldots, 60\}$

(*ii*) the portfolio position x_t with respect to the currency pair of the previous time-step.

(*iii*) the current time denoting the fraction of time remaining until the end of the trading day, or episode.

(*iv*) the encoding day of the week as an ordinal number between 1 (Monday) and 5 (Friday).

(*v*) the vector $\lambda_z = \{\lambda_1, \lambda_2, \ldots, \lambda_n\}$ containing n estimated latent variables from the underlying process.

It is essential to highlight that normalizing the exchange rate variations between consecutive minutes enable the agent to generalize market observations that it has never encountered before while the current time is normalized in $[0, 1]$. The portfolio position is the only feature of the state which is not retrieved from market data. Still, it is directly controlled and determined by the agent's behaviour. It is expressed as a value in the set $\{-1, 0, 1\}$, whose elements correspond respectively to the three allocations defined in the single asset setting: *buy, sell*, and *hold.* Last but not least, the introduction of the vector λ_z is the latent representation of the time series that is included in the state space and represents a novelty compared to existing previous work.

4.4.3 Action

The *action* consists of the allocation the agent wants to keep for the next minute or the next ρ minutes if we consider an action persistence $\rho > 1$. We consider a discrete action space where the action at time t, namely a_t is the portfolio position. Indeed, with the portfolio position included in the set of features that define the state, the set of available actions can be defined as:

$$\mathcal{A}(s) = \begin{cases} \{0\} & \text{if } s \in \mathcal{S}^{\mathcal{T}} \\ \{\text{-}1, 0, 1\} & \text{otherwise} \end{cases} \tag{8}$$

where $\mathcal{S}^{\mathcal{T}}$ is defined as the set of terminal states, hence the states corresponding to the last minute of each trading day. The values correspond to *sell*, *hold*, and *buy* position.

4.4.4 Transition Probability

The *transition probability* refers to the probability of moving to a new state s' given the current state-action pair (s, a). In our case, the action affects only the portfolio feature, so we have $x_{t+1} = a_t$, while all the other features are exogenous, and their value is not affected by the action.
The transition of the current time step and the already observed prices have a deterministic transition, with only one new stochastic addition from market data. Therefore, the transition probability $\mathcal{P}(s'|s, a)$ represents the probability of observing the next set of prices given the current set of prices and the action taken.

4.4.5 Reward

Given the current portfolio allocation x_t, the current exchange rate p_t, the action taken a_t, the next exchange rate p_{t+1}, and the fee rate ϕ, the *reward* received by the agent at persistence ρ is defined as:

$$r_{t+1} = \underbrace{a_t(p_{t+\rho} - p_t)}_{\text{gain/loss}} - \underbrace{\phi|a_t - x_t|}_{\text{transaction costs}} \tag{9}$$

In other words, the reward is obtained by two terms: the first consists of the gain (or loss) associated with the exchange rate variation, whereas the second corresponds to the transaction costs that must be paid to change portfolio allocation. We assume that ϕ equals a fixed percentage of the total amount of base currency traded (i.e., $\phi = 2 \cdot 10^{-5}$).
Since we do not have the data with a lower frequency than one minute and no information about the bid/ask spread, the exact buy/sell price has to be calculated as the *instantaneous* buy/sell with the following calculation:

$$p_t = o_t + [(h_t - l_t) \cdot sign(c_t - o_t)] \tag{10}$$

where o_t is the open price, h_t the high price, l_t the low price, c_t the close price.

4.4.6 Latent-Variable FQI

The algorithm *Fitted Q-Iteration* (FQI) is a model-free, off-policy, and offline method that allows an RL agent to learn the optimal policy without interacting with the environment by estimating the Q-function via iteratively performing regression starting from collected past agent-environment interaction, the training set.

Before the first iteration starts, the training set is originally built starting from the experience gathered by the agent in the past, which is represented by the set $\mathcal{D}$ of four-value tuples $(s_t, a_t, s_{t+1}, r_{t+1})$, defined as:

$$\mathcal{D} = \left\{(s_t^k, a_t^k, s_{t+1}^k, r_{t+1}^k) \mid k = 1, 2, ..., |\mathcal{D}|\right\},$$

where s_t corresponds to the state where the agent finds itself at time t, while s_{t+1} and r_{t+1} are, respectively, the next state reached and the immediate reward earned by the agent after taking action a_t. Given the values of x_t and a_t, the reward r_{t+1} is computed accordingly to Equation 9.

To train our FQI models, building the set $\mathcal{D}$ from collected market data is necessary. Since the portfolio allocation is controlled by the agent behaviour ($x_t = a_{t-1}$), we need to simulate all possible (x_t, a_t) combinations. In the single-asset setting, there are three different actions: buy, hold, and sell, so each vector of market observation is repeated nine times. The training set is generated from two years of market observations (2020 and 2021), which is a long enough interval to include a variety of market regimes [13, 78].

The novelty of this work was to include *latent variables* extracted via *variational auto-encoders* methods described in Section 4.3 into the FQI features, and consequently as part of agent state. Hence, it is needed to enlarge the set $\mathcal{D}$ to include a vector λ_z and define $\mathcal{D}_\lambda$ as:

$$\mathcal{D}_\lambda = \left\{ (s_t^k, a_t^k, \lambda_{z,t}^k, s_{t+1}^k, r_{t+1}^k) \mid k = 1, 2, ..., |\mathcal{D}_\lambda| \right\},$$

where $\lambda_{z,t}$ correspond to a latent features vector $\lambda_{z,t} = \{\lambda_1, \lambda_2, \ldots, \lambda_n\}$ at time t with n number of extracted features. The rest of the methodology and steps described above does not change.

4.5. Evaluation

The evaluation of RL algorithms is critical to validate their effectiveness and robustness. In this chapter, we present the *evaluation method* used to assess the performance of our proposed model. The process consisted of two main parts: an initial idea validation phase using synthetic data generated by stochastic differential equation (SDE) models and a following experimental phase using real Forex data.
The SDE models used are respectively the *Vasicek model* and the *Geometric Brownian Motion* (GBM). The experiments conducted with generated data were used to validate the intuition and assumptions on the proposed augmentation with latent variables in a controlled environment. The evaluation metrics used in both phases were carefully selected to analyse the model performance comprehensively.

4.5.1 Vasicek Model

The *Vasicek* model is a stochastic process generally used in finance to model the evolution of interest rates over time. The model was developed by Oldrich Vasicek in 1977 and has become a classic model for interest rate modelling [90]. The model assumes that the short-term interest rate follows a mean-reverting process given by the following stochastic differential equation:

Let S_t be the price of the underlying asset at time t, then:

$$dS_t = a(\mu - S_t)dt + \sigma dW_t \tag{11}$$

where a is the speed of mean reversion, μ the long term mean level, σ is the volatility, dt represents an infinitesimal time interval, and W_t is the Wiener process $W_{t+u} - W_t \sim \mathcal{N}(0, u) = \mathcal{N}(0, 1) \times \sqrt{u}, \mathcal{N}$ being the normal distribution. The first term on the right-hand side represents the mean-reverting process, with $a(\mu - S_t)dt$ driving the interest rate towards its long-run mean. The second term represents the stochastic volatility of the interest rate, with σdW_t representing the random noise in the system.

4.5.2 Geometric Brownian Motion

The *Geometric Brownian Motion* (GBM) is another well-known stochastic process in finance used to model the dynamics of stock prices, commodities and other financial assets [81]. The GBM model assumes that the asset price follows a Brownian motion process with a drift, given by the below equation.

Let S_t be the price of the underlying asset at time t; then a GBM process is defined as:

$$dS_t = \mu S_t dt + \sigma S_t dW_t \tag{12}$$

where W_t is Brownian motion, μ the drift (in general, we assume it to be 0 throughout without loss of generality) and σ the volatility. This process is simple and effective, but constant volatility is a strong financial assumption. The first term represents the drift, which drives the asset's expected return, and the second term represents the random noise, which captures the volatility of the asset return.

4.5.3 Metrics

In this chapter, we present the evaluation *metrics* used to assess the performance of the RL algorithm in both synthetic and real scenarios. This chapter aims to describe and present the metrics from a methodological point of view; indeed, they were chosen to provide a comprehensive performance evaluation from different lenses, such as the ability to generate profits, the involved risk and adaptability to changing market conditions. The application of the metrics is discussed in Section 5.

Therefore, the main financial metrics considered in this thesis are listed below:

- *Profit & Loss*: the *P&L* represents the monetary gains of the strategy and can be calculated in slightly different manners, in our case:

  $$r_{t+1} = a_t(p_{t+\rho} - p_t) - \phi|a_t - x_t| \tag{13}$$

  where x_t is the portfolio allocation, p_t is the current exchange rate, a_t is the action taken, p_{t+1} is the next exchange rate, ϕ and ρ are respectively the fees and the persistence.
- *Cumulative P&L*: when considering a fixed action size and without reinvesting gains/losses, the cumulative P&L becomes a sum:

  $$W_T = \sum_{t=1}^{T} r_t \tag{14}$$

  and the annualized version is:

  $$W_A = W_T \frac{\textit{num-days}}{t_w} \tag{15}$$

  where t_w is the time period in which the profit was generated.
- *P&L variance*: once the daily P&L are calculated, defined as $\{r_t^d\}_{t\in[1,T]}$, the daily variance can be calculated as:

  $$v(r_t^d) = \frac{1}{T}\sum_{t=1}^{T}(r_t^d - \hat{r}^d)^2 \tag{16}$$

  where $\hat{r}^d$ is the average daily P&L and T the number of days. It can be annualized as $v_A = T \cdot v$
- *Volatility*: it is defined as the standard deviation so

  $$vol = \sqrt{v(r_t^d)} \tag{17}$$

  with the annualized volatility $vol_A = \sqrt{T} \cdot vol$ with T the number of days.
- *Sharpe Ratio*: the $\mathcal{SR}$ is usually calculated in an annualized fashion and defined as:

  $$\mathcal{SR} = \frac{W_A - r_f}{vol_A} \tag{18}$$

  where r_t is the risk-free rate, what an investor would gain by leaving his money in a savings account for the investment period.
- *Maximum Drawdown*: the $\mathcal{MDD}$ is a measure of risk defined as the maximum observed loss from peak performance to the trough before a new peak is attained, formally:

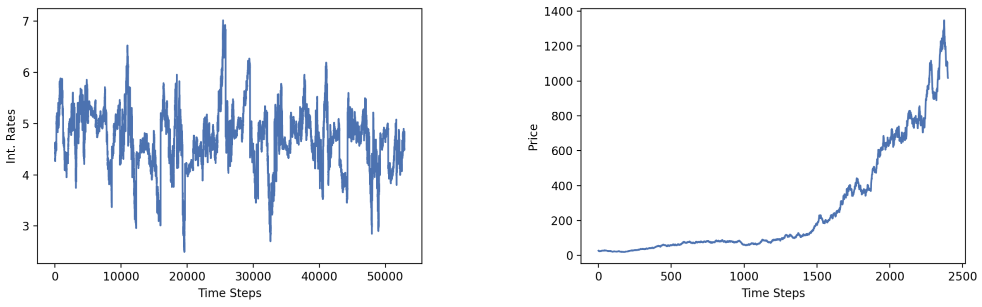
  $$\mathcal{MDD} = \frac{TroughValue - PeakValue}{PeakValue} \tag{19}$$

5. Experimental Results

This chapter presents the empirical results obtained from our Latent-Variable extension of FQI, trained on real FX market data. Firstly, we conduct an exploratory data analysis of the exchange rates, comparing the summary statistics, trends, and non-stationarity of the different currency pairs, namely EUR/USD and USD/JPY, using the Augmented Dickey-Fuller Test (ADF). Next, we explain the procedure to validate our ideas using synthetic and real data, including how we used VAE to extract latent variables to augment FQI. We describe the experimental setup in detail, including the hyperparameters used for the models, the RL algorithm, and the performance metrics used to evaluate the results. Finally, we discuss the findings of our experiments and the implications of using latent variables. Finally, we benchmarked our approach against existing Batch-RL baselines and various passive and active trading strategies.

5.1. Synthetic Data

In this chapter, we present the results of our experiments aimed at testing the effectiveness of our proposed approach of augmenting an offline RL algorithm with latent variables extracted using variational autoencoders (VAE). Specifically, we generate synthetic datasets from two popular models used in financial applications, namely the *Vasicek* and *Geometric Brownian Motion* (GBM) models explained in Section 4.5. These datasets are used to train and validate our approach to improve the performance of the RL agent in a trading scenario. We will explain the experiments and discuss the derivations from the augmentation of the latent variables.



(a) Sample generated from the Vasicek model. (b) Sample generated from the GBM model.

Figure 9: Time-series samples generated synthetic data for the experiments.

In the synthetic experiments, the data have been generated from fixed pre-defined values (Figure 9). In this setup, we hardcoded the latent variables used to generate the time series, which allowed us to compare the results of using these variables against not using them. We assumed a prices environment, where the agents have access to the lagged shift price rate variations normalized at the first price of the day differently from what it is used for the real data, explained in Section 4.4.2. Finally, we trained the LV-FQI algorithm on specific values of the latent variables and validated it on other values, ensuring that the model generalizes well to unseen data. These assumptions allowed us to test the effectiveness of our proposed approach in a controlled setting, where we could evaluate the impact of using latent variables on the performance of the FQI algorithm.

5.1.1 Vasicek

This section presents the experimental results of our proposed Latent-Variable extension of FQI algorithm trained on synthetic data generated from the Vasicek model.

The synthetic dataset was generated using specific model parameter values, including the long-term mean, volatility, and mean reversion rate. Each day lasted 1230 steps, and the variables were changed every day. Here is the list of parameters used:

- *k options:* 0.01, 0.03, 0.1, 0.02 (training), 0.04, 0.05 (testing)
- *theta options:* 0.5, 1.1, 1.8, 1.2 (training), 1, 1.4 (testing)
- *sigma options:* 0.02, 0.04, 0.05, 0.01, 0.015 (training), 0.03 (testing)

	Hyperparameters			Performance
Algorithm	**Persistence**	**MinChildWeight**	**Iterations**	**P&L**
FQI	1	2000	1	27.3 ±8.1
PFQI	5	2000	1	65.7 ±11.1
PFQI	10	2000	1	99.7 ±10.2
LV-FQI	1	1000	1	164.8 ±8.5
LV-PFQI	5	2000	1	98.9 ± 8.8
LV-PFQI	**10**	**2000**	**1**	**189.7 ± 8.5**

Table 4: Summary of the results obtained by different variations of FQI algorithm on Vasicek model with different values of persistence, min child weight and P&L expressed as percentage cumulative return ± standard deviation.

We tested the FQI algorithm with one iteration and different values of the hyperparameters for XGBoost min-child-weight, which regulates the model complexity, as explained in the model selection paragraph 5.3.2. We also evaluated the impact of persistence using different values: 1, 5, and 10. This experiment allowed us to test the algorithm's ability to learn from different datasets with various degrees of persistence and complexity and to evaluate the effectiveness of our proposed approach in generating trading policies on synthetic data. A summary of the results can be found in Table 4.

5.1.2 GBM

Instead, we present the experimental phase on synthetic data generated from the GBM model. Same as before, the synthetic dataset was generated using specific values of the model parameters, and each day lasted 1230 steps, and the variables were changed daily. Here is the list of parameters used:

- *mu options:* 0.8, 0.9, 1, 1.1, 1.2, 1.4, 1.5 (training), 1.6, 2 (testing)
- *sigma options:* 0.5, 0.6, 0.7, 0.8, 1, 1.5 (training), 2 (testing)

Following the same procedure, we tested the FQI algorithm with one iteration and different values of the min-child weight. We evaluated the impact of the persistence with different values: 1, 5, and 10. A summary of the results can be found in Table 5.

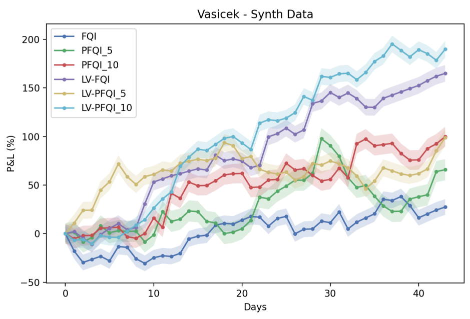
	Hyperparameters			Performance
Algorithm	**Persistence**	**MinChildWeight**	**Iterations**	**P&L**
FQI	1	20.000	1	21.5 ± 5.4
PFQI	5	20.000	1	13.2 ± 4.5
PFQI	10	20.000	1	47.6 ± 2.5
LV-FQI	**1**	**40.000**	**1**	**105.2 ± 5.1**
LV-PFQI	5	40.000	1	84.6 ± 5.2
LV-PFQI	10	40.000	1	63.9 ± 4.2

Table 5: Summary of the results obtained by different variations of FQI algorithm on GBM model with different values of persistence, min child weight and P&L expressed as percentage cumulative return ± standard deviation.

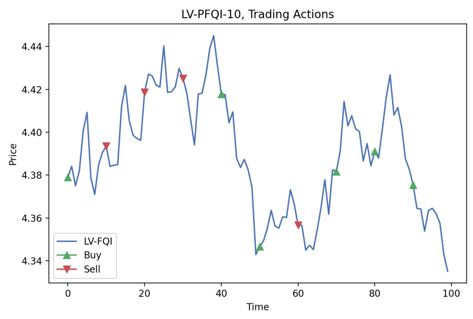
5.1.3 Summary

Figure 10 shows the outperforming performance of LV-PFQI with persistence 10 and LV-FQI compared to the best model without the latent variables. Furthermore, even the worst performer of the model with latent variables is still way above the best without. We can also confirm the contribution of the latent variables towards the prediction of the regressor by looking at the feature importance. This suggests that the latent variables that capture the underlying system are important to the model's performance. As a confirmation, in Figure 11, we experienced the same behaviour and result. Indeed, again the LV version of FQI, either with or without persistence, is beating all the benchmarks, and the variables are among the first towards the prediction.

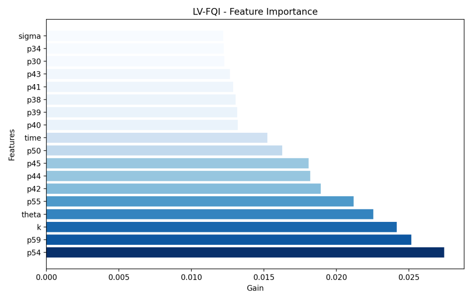
The results using the synthetic dataset with hardcoded latent variables have confirmed our assumption that using latent variables effectively enhances the model performance. The experiments conducted using synthetic data generated from the Vasicek and GBM models indicate that the FQI algorithm can exploit the added knowledge from the latent variables. Therefore, the successful experiment with the synthetic dataset has provided a solid foundation for the subsequent estimation of latent variables from the real dataset.



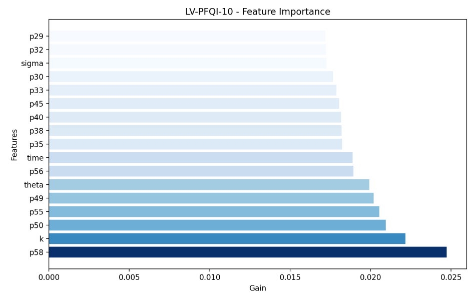
(a) Comparison of the performance in terms of cumulative P&L for FQI variations with data generated from Vasicek model.



(b) Sample of actions performed by the best performer model with Vasicek data, LV-PFQI-10 with action issued every 10 mins.

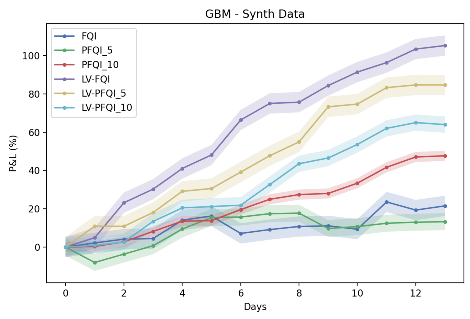


(c) LV-FQI feature importance.

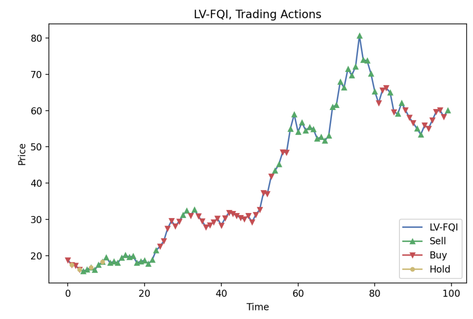


(d) LV-PFQI-10 feature importance.

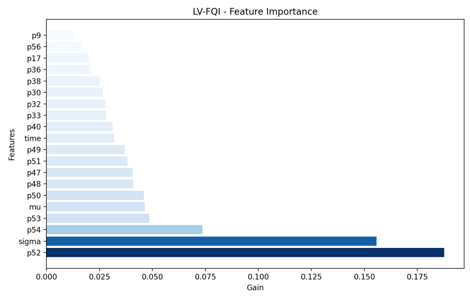
Figure 10



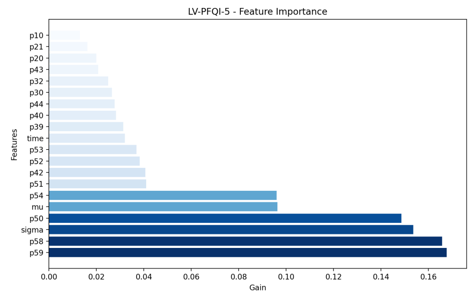
(a) Comparison of the performance in terms of cumulative P&L for FQI variations with data generated from GBM model.



(b) Sample of actions performed by the best performer model with Vasicek data, LV-FQI with action issued every minute.



(c) LV-FQI feature importance.



(d) LV-PFQI-5 feature importance.

Figure 11

5.2. VAE & Latent Variables

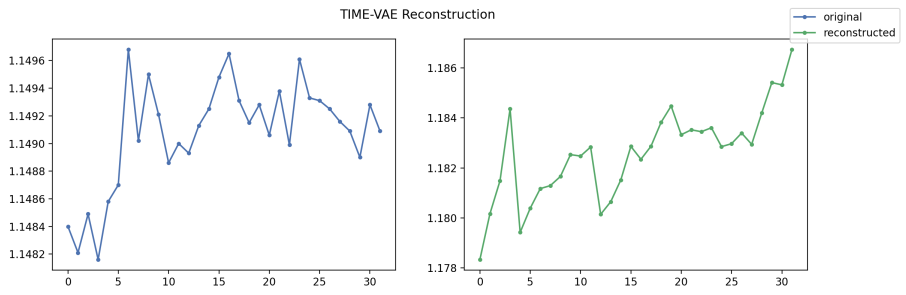
As mentioned in Section 1, one of the primary goals of our work was to investigate the effectiveness of incorporating the latent space into the RL agent for learning trading strategies. We recognized that existing methods often failed to account for the challenges posed by a non-stationary market environment where time series are highly fluctuating and time-varying. Therefore, we leveraged Variational Auto-Encoders (VAE) [58] to filter the fluctuations in exchange rates and derive a compressed time-series representation that contained the essential information for the modelling phase [89].

After validating our idea on synthetic data, we focused on training different VAE models based on TimeVAE [36] and LSTM-VAE [30] architectures to extract the latent representation from real FX time-series data. The primary objective was to evaluate the VAE models' performance in accurately capturing the underlying time-series structure with a compact latent representation that enabled the reconstruction of input data. Since creating a representation of the underlying task can be challenging, we experimented with multiple models. We selected the best-performing one as the "oracle" model for extracting latent variables used as input for the offline RL phase.

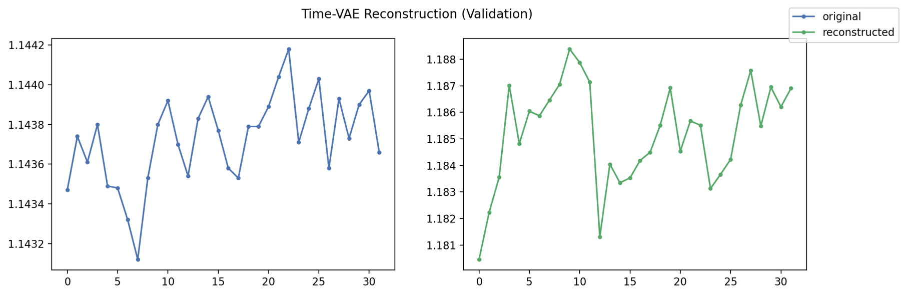
The encoder output provided the latent-space representation that was of interest to us, which we denoted as z. The vector z had a variable number of dimensions and enriched the feature space of the FQI algorithm, serving as the input for the regression mechanism to enable the learning of the optimal trading strategy.

Specifically, we tested two main models, *TimeVAE* and *LSTM-VAE*, to extract the latent representation from real FX data. TimeVAE is a model designed specifically for time-series data, while LSTM-VAE is based on LSTM recurrent neural networks that are particularly effective for sequential data. Moreover, we also experimented with two extensions of the LSTM-VAE model. The first extension used *stacked LSTM* layers to improve the model's ability to capture complex temporal dependencies. The second extension incorporated an *attention mechanism* to enable the model to focus on the most relevant parts of the input sequence, hoping to enhance its ability to extract the latent features.

We performed training and validation phases using FX data from 2020-2021 and used a subset of the data to validate the performance of the models. Then, after the models were trained and validated, we tested each the ability to generalize to new, unseen data by evaluating their performance on the time-series data from 2019. The primary metric for the evaluation was the total loss as the summation of two different loss functions, namely the *Kullback-Leibler* (KL) divergence loss and the *reconstruction* loss.



(a) Reconstruction of the model in Training Years: 2020-2021.



(b) Reconstruction of the model in Test Year: 2019.

Figure 12: Best TimeVAE for EUR/USD time-series reconstruction in train and testing.

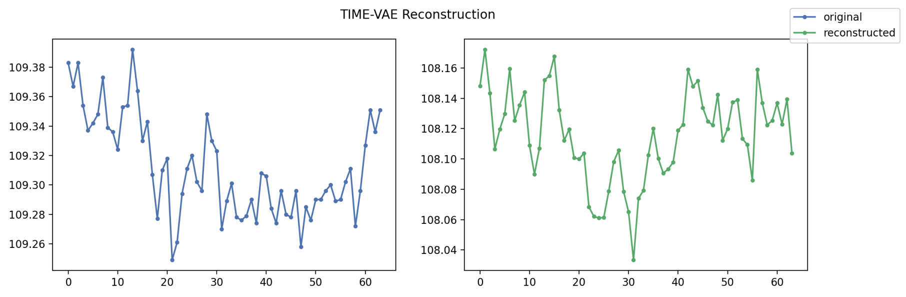
Currency Pair	Model	Total Loss	Total Loss (Validation)
EUR/USD	TimeVAE	0.008	0.024
	LSTM-VAE	0.093	0.104
	stacked-LSTM-VAE	0.256	0.555
	attention-LSTM-VAE	0.354	0.653
USD/JPY	TimeVAE	0.015	0.033
	LSTM-VAE	0.101	0.147
	stacked-LSTM-VAE	0.351	0.550
	attention-LSTM-VAE	0.649	0.948

Table 6: Performance of VAE models for EUR/USD and USD/JPY on train and validation data. TimeVAE has the lowest total loss, while stacked-LSTM-VAE and attention-LSTM-VAE have higher total losses compared to TimeVAE and LSTM-VAE.

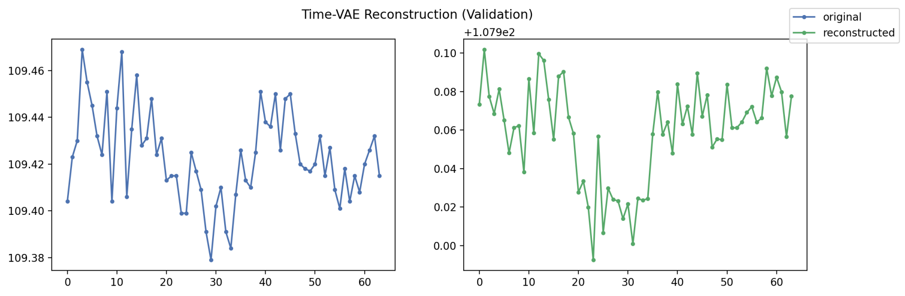
We performed extensive hyperparameter tuning for all VAE architectures, including the number of filters for the convolution layers, the input dimensions for the LSTM and attention layers, dropout, and training parameters such as steps, batch size, learning rate and epochs. However, among all the hyperparameters, the dimensionality of the latent features was the most critical to tune. After conducting numerous experiments, we found that a latent space dimension of 4 was the most effective for both VAE architectures. This means that the compressed latent representation generated by the encoder had four dimensions, providing a sufficiently compact representation of the time-series data while retaining the essential features required for offline RL.

We identified the best model for EUR/USD and USD/JPY data. Specifically, we found that the TimeVAE architecture was the most effective for both currency pairs. For EUR/USD, the best-performing model had a batch size of 32 and a latent space dimension of 4. The model was trained for 30 epochs using an Adam optimizer with a learning rate of 0.001, and early stopping was used to prevent overfitting. For USD/JPY, the best-performing TimeVAE model had the same characteristics but a batch size of 64. A summary can be found in Table 6.

Overall, the TimeVAE models demonstrated superior performance in accurately capturing the underlying structure of the time-series data and extracting a compact latent representation that retained the essential features required used as input for the offline RL algorithm in the second phase.



(a) Reconstruction of the model in Training Years: 2020-2021.



(b) Reconstruction of the model in Test Year: 2019.

Figure 13: Best TimeVAE for EUR/USD time-series reconstruction in train and testing.

5.3. FX Data

After outlining the synthetic experiments and latent variable estimation, this section will go into FX trading. Specifically, it will analyze the non-stationarity of exchange rates via summary statistics and statistical tests and explore LV-FQI experiments by explaining the model selection phase and the algorithm comparison against existing RL benchmarks.

5.3.1 Data Analysis

In Figure 14, the EUR/USD and USD/JPY exchange rates from 2018 to 2022 are reported [3]. From a visual inspection, it is possible to derive a few considerations. First, the data appear non-stationary since it is easy to see upward and downward trends over the years. The concept of stationarity in time series analysis refers to the property whereby the statistical properties of a given process that generates the time series remain constant over time, so determining whether a time series is stationary is important. To confirm our assumption, we calculated the summary statistics, namely mean and variance, available in Table 7, from which it is easy to deduct the values are changing over the years.

Year	$\mathbb{E}$[EUR/USD]	Var[EUR/USD]	$\mathbb{E}$[USD/JPY]	Var[USD/JPY]
2018	1.1812	1.3×10^{-3}	110.45	5.352
2019	1.1195	1.8×10^{-3}	109.01	2.54
2020	1.1414	1.9×10^{-3}	106.75	3.99
2021	1.1826	7.9×10^{-3}	109.86	9.02
2022	1.0533	2.4×10^{-3}	131.50	110.2

Table 7: Yearly Mean and Variance of EUR/USD and USD/JPY

Another way to test the stationarity of a time series is via the *Augmented Dickey-Fuller Test* (ADF), which can be used to inform the degree to which a null hypothesis can be rejected or fail to be rejected. It is a test called unit root, that intuitively determines how strongly a trend defines a time series.
The test's null hypothesis is that a unit root can represent the time series and that it is not stationary and has some time-dependent structure. The alternative hypothesis, or rejecting the null hypothesis, is that the time series is stationary. The result is interpreted using the *p-value* from the test. A p-value below a threshold, in our case 5%, suggests we reject the null hypothesis (stationary); otherwise, a p-value above the threshold suggests we fail to reject the null hypothesis (non-stationary).

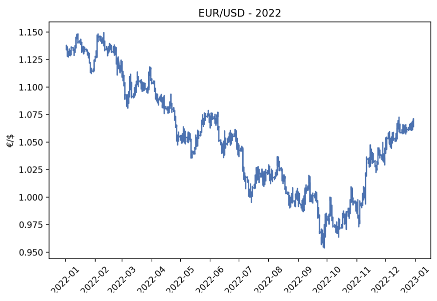
To summarize the hypothesis:

- **Null Hypothesis** (H_0): if failed to be rejected, it suggests the time series has a unit root, meaning it is non-stationary. It has some time-dependent structure.
- **Alternative Hypothesis** (H_1): the null hypothesis is rejected; it suggests the time series does not have a unit root, meaning it is stationary. It does not have a time-dependent structure.
- **p-value** > 0.05: fail to reject the null hypothesis (H_0), the data has a unit root and is non-stationary.
- **p-value** ≤ 0.05: reject the null hypothesis (H_0), the data does not have a unit root and is stationary.

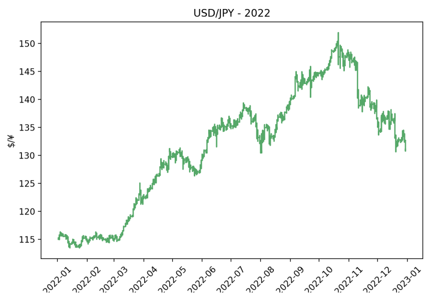
We extracted the ADF statistic from the test, the *p-value* and the 1%, 5% and 10% critical value (CV). The results are summarized in Table 8 and Table 9, respectively, for EUR/USD and USD/JPY. They show that the p-value > 0.05 and ADF statistic is greater than the critical values; hence we can reject the null hypothesis (H_0). As a consequence of this result, in our experiments, we used the first difference prices as a feature set of input for the RL algorithm.

Year	ADF Statistic	p-value	CV 1%	CV 5%	CV 10%
2018	-1.0578	0.731	-3.43%	-2.86%	-2.56%
2019	-2.6106	0.090	-3.43%	-2.86%	-2.56%
2020	-0.2360	0.934	-3.43%	-2.86%	-2.56%
2021	-1.07280	0.725	-3.43%	-2.86%	-2.56%
2022	-1.7057	0.428	-3.43%	-2.86%	-2.56%

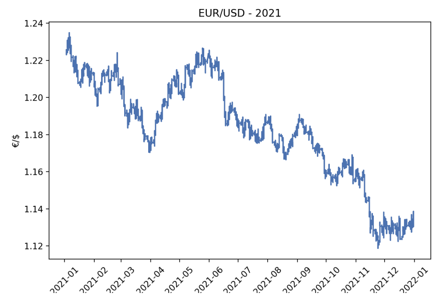
Table 8: Summary of Augmented Dickey-Fuller (ADF) Statistics, Critical Values and p-values for the pair EUR/USD.



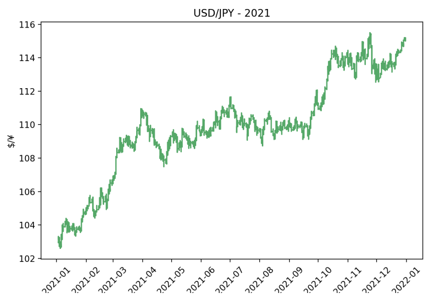
(a) EUR/USD, 2022 .



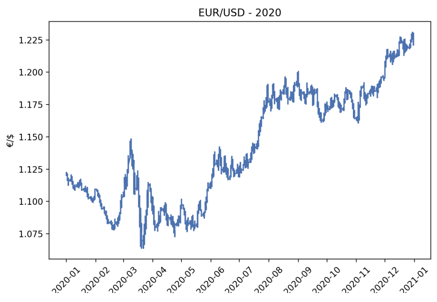
(b) USD/JPY, 2022



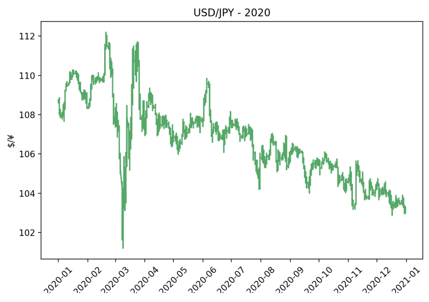
(c) EUR/USD, 2021 .



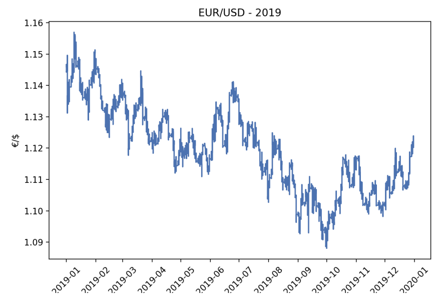
(d) USD/JPY, 2021



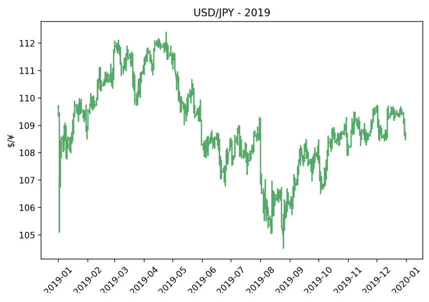
(e) EUR/USD, 2020.



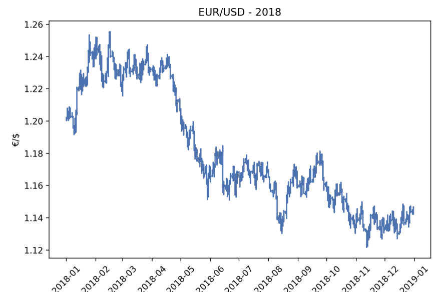
(f) USD/JPY, 2020



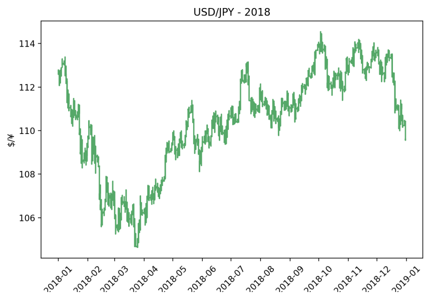
(g) EUR/USD, 2019



(h) USD/JPY, 2019



(i) EUR/USD, 2018 .



(j) USD/JPY, 2018

Figure 14: EUR/USD and USD/JPY exchange rates from 2018 to 2022.

Year	ADF Statistic	p-value	CV 1%	CV 5%	CV 10%
2018	-1.6963	0.433	-3.43%	-2.86%	-2.56%
2019	-2.0373	0.270	-3.43%	-2.86%	-2.56%
2020	-1.8859	0.338	-3.43%	-2.86%	-2.56%
2021	-1.4446	0.560	-3.43%	-2.86%	-2.56%
2022	-1.5610	0.503	-3.43%	-2.86%	-2.56%

Table 9: Summary of Augmented Dickey-Fuller (ADF) Statistics, Critical Values and p-values for the pair USD/JPY.

5.3.2 Model Selection

As mentioned in the chapter, we trained our models in different conditions to provide a comprehensive and robust evaluation of the LV-FQI algorithm for learning intraday FX trading strategies.

The experiments were conducted on data from 2019 to 2022. To increase robustness, we used a training set composed of two consecutive years, 2020 and 2021, rather than a single year, to let the agents experience enough market conditions. Then, to select the models that better generalize over the following years, for each training dataset, we considered the years before and after the training set: the learned policies were validated in the former, 2019, and tested in the latter, 2022. We chose to consider shorter temporal distances, two years, in our analysis to address the non-stationarity of FX market observations. Longer distances may lead to the emergence of unobserved patterns, which can affect the agents' performance.

To select the best set of trading agents, we trained different versions of FQI model, each characterized by a certain action *persistence* (1, 5 and 10) and a set of different hyperparameters and a 1-minute-sampling frequency. Indeed, to choose the best FQI model, we can tune both the hyperparameters related to the regressor and the ones of the general algorithm. Due to its great performances in terms of accuracy, high scalability and computational efficiency when dealing with large training datasets, we chose *XGBoost* [29] as the regression mechanism for the Q-function at each iteration of FQI. According to what is suggested by [47], given a reasonable value for most hyperparameters, it is sufficient to tune the *min child weight* to regulate the model complexity. Generally, the higher the threshold is, the simpler the trained model becomes, as trees are forced to consider a greater number of samples to perform a split; hence complicated patterns are excluded. On the other hand, a low *min child weight* allows for more complex models, but it increases the risk of overfitting the training data. Along with the regression parameters, it is necessary to tune the number of FQI *iterations*. As it grows, the optimized horizon increases, allowing the model to learn longer-term patterns. However, iterating the Q-function fitting may lead to the propagation of approximation errors. As mentioned previously, with LV-FQI, we introduced an efficient method for integrating the *latent space* representation of VAE into the batch algorithm. Therefore, we extracted the latent variables from the two years of training, refer to Section 5.2, and included them as features set for the regressor among the others to improve the learning of the optimal policy.

To summarize, after extracting the latent variables, we tuned the XGBoost *min child weight* parameter and the number of FQI iterations for each value of persistence. We select a finite set of different *min child weight* values and train each model with up to 10 iterations of FQI, according to [79]. Moreover, we performed three different runs of the training algorithm to account for the XGBoost regression's randomness. Then, to determine the best set of FQI trading agents, we compare the performance on the validation set. For each persistence value, we first determined the best iteration for each *min child weight*. Then we selected the three models that attained the highest average cumulative return at the end of the validation year.

5.3.3 Algorithm Performance

In this sub-chapter, we focus on the performance comparison between the Latent-Variable FQI and its variation with persistence, compared with those achieved by standard FQI and Persistent FQI highlights the advantage of using latent variables as part of model building. All the experiments consider a fixed \$100K allocation and a fixed transaction cost of $\phi = 2 \cdot 10^{-5}$.

The results are reported in Table 10 for EUR/USD and Table 11 for USD/JPY. They show the performance in the validation year (2019) for the best three models after the model selection phase (5.3.2 in terms of *persistence*, *iterations* and *min child weight*. The other runs have been omitted for clarity and space.
Examining the performance metrics of the different algorithms, it becomes clear that the *latent variable* variation of FQI is the top performer not only in terms of P&L but also in terms of Sharpe Ratio and Maximum Drawdown. This means the algorithm obtains higher returns with lower risk than the others. Indeed, the

	Hyperparameters			Performance		
Algorithm	**Persistence**	**MinChildWeight**	**Iterations**	**P&L**	**SR**	**MDD**
FQI	1	40.000	5	6.5 ± 2.26	1.75	4.23
		60.000	**3**	**7.9 ± 2.5**	**1.8**	**5.19**
		80.000	4	0.9 ± 3.28	0.22	2.0
PFQI	5	**10.000**	**2**	**11.8 ± 0.6**	**2.88**	**0.6**
		20.000	3	6.6± 1.26	1.75	4.2
		40.000	2	8.5 ± 2.21	2.7	1.22
PFQI	10	500	1	2.2 ± 2.3	0.35	1.25
		5000	**2**	**15.4 ± 1.8**	**4.05**	**1.26**
		60.000	2	3.3 ± 2.4	0.65	2.8
LV-FQI	1	10.000	2	8.9 ± 2.43	1.42	0.95
		40.000	5	11.1± 1.3	2.5	44.9
		80.000	**2**	**26.8 ± 3.4**	**2.35**	**0.95**
LV-PFQI	5	10.000	3	16.6 ± 1.36	3.17	0.47
		20.000	3	14.1 ± 1.46	2.12	2.5
		40.000	**3**	**31.8 ± 2.2**	**3.0**	**1.08**
LV-PFQI	10	500	2	15.4 ± 3.25	4.1	30.6
		20.000	2	8.6 ± 2.24	2.49	3.28
		40.000	**2**	**15.8 ± 2.6**	1.38	1.2

Table 10: Performance for EUR/USD on validation year: 2019 of FQI variations. **SR** stands for Sharpe Ratio, **MDD** for Maximum Drawdown, **P&L** as mean ± standard deviation. The table shows the best three runs after the tuning for each combination and each model has run multiple times because of the randomness.

Sharpe Ratio (SR) is a measure of risk-adjusted return, which takes into account the volatility of returns, while Maximum Drawdown (MDD) is a measure of the largest percentage decline from a peak to a trough that an investment has experienced over a given period of time. The *latent variable* variation of FQI shows on average a higher SR than the other algorithms, indicating that it can generate higher returns with lower risk and a lower MDD; hence it can mitigate losses during market downturns and turbulent conditions. Furthermore, in Figure 15 and 17 a chart showing a comparative performance of the best run in validation for all the variations of FQI and manifest that our proposed approach nearly double the P&L, so generate high profits.

Another insight is that those with persistence equal to 5 and 10 outperform the ones trained with persistence equal to 1. The worse signal-to-noise ratio can explain the poor performances, which affects learning using high frequencies. Other than that, a higher persistence allows also to have computational advantages. Another interesting analysis can be done by looking at the feature importance; indeed, on one side, it is noticeable that the time feature becomes higher as the persistence increases. Also, the importance of the portfolio becomes less relevant for persistence equal to 10. On the other hand, the latent variables *latent variable* impact can be noticed on prediction. They indeed significantly impact the prediction because their inclusion allows the algorithm to capture hidden patterns and trends that may not be observable through other means. Finally, the feature importance analysis also highlights that recent variations are more important than past observations. This is consistent with the idea that the market constantly evolves, and recent information is more relevant to predicting future trends than historical data. The LV-FQI algorithm can adapt to changing market conditions and generate more accurate predictions by focusing on more recent variations.

Overall, using latent variables in the LV-FQI algorithm is essential because it allows the algorithm to capture more complex relationships between the input features and the output variable. In traditional regression models, only the observed features are used to make predictions; indeed, these models may not be able to fully capture the underlying dynamics of the system modelled. Latent variables, on the other hand, can capture unobserved factors that are believed to be essential for the prediction of the output variable. By incorporating latent variables into the FQI algorithm, the model can capture more complex patterns and relationships in the data, leading to more performant trading strategies. For example, in financial markets, there may be aspects such as market sentiment or the influence of large institutional investors that are not directly observable but can significantly impact market trends. In this way, these hidden variables can be captured and incorporated into the prediction model. Thus, using latent variables is important because it has consistently shown that it allows the model to capture hidden patterns and relationships in the data, leading to performances with more profit and less risk.

	Hyperparameters			Performance		
Algorithm	**Persistence**	**MinChildWeight**	**Iterations**	**P&L**	**SR**	**MDD**
FQI	1	20.000	3	-5.0 $\pm$ 2.4	-1.74	15.5
		60.000	**4**	**4.9 $\pm$ 3.0**	**0.3**	**6.1**
		80.000	1	3.65 $\pm$ 2.35	4.6	4.2
PFQI	5	**5000**	**3**	**8.6 $\pm$ 1.2**	**0.41**	**10.2**
		20.000	4	7.9 $\pm$ 1.2	1.1	2.4
		40.000	2	6.85 $\pm$ 3.6	0.60	8.52
PFQI	10	20.000	2	17.5 $\pm$ 2.1	1.8	1.4
		40.000	**1**	**21.2 $\pm$ 3.5**	**1.17**	**0.76**
		80.000	1	9.4 $\pm$ 3.2	0.7	6.2
LV-FQI	1	40.000	3	12.09 $\pm$ 2.74	1.15	7.90
		60.000	2	13.5 $\pm$ 1.8	1.91	3.1
		80.000	**3**	**18.6 $\pm$ 2.4**	**1.14**	**4.7**
LV-PFQI	5	**20.000**	**2**	**29.49 $\pm$ 1.8**	**1.62**	**1.67**
		40.000	1	22.2 $\pm$ 3.2	2.4	9.2
		80.000	2	20.1 $\pm$ 2.9	2.02	3.96
LV-PFQI	10	5000	4	19.95 $\pm$ 2.05	1.93	2.66
		40.000	**3**	**23.88 $\pm$ 3.1**	**1.014**	**1.27**
		80.000	1	8.2 $\pm$ 0.9	0.84	5.2

Table 11: Performance for USD/JPY on validation year: 2019 of FQI variations. **SR** stands for Sharpe Ratio, **MDD** for Maximum Drawdown, **P&L** as mean $\pm$ standard deviation. The table shows the best three runs after the tuning for each combination and each model has run multiple times because of the randomness.

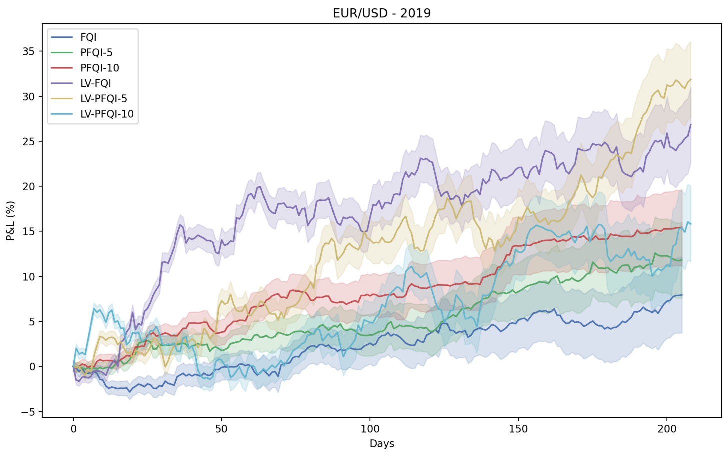
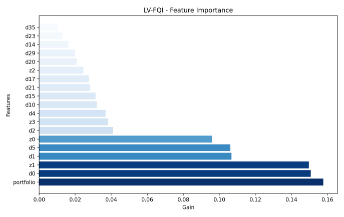
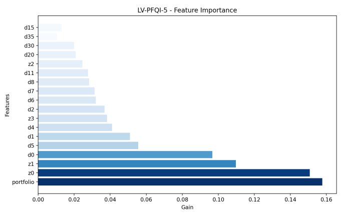


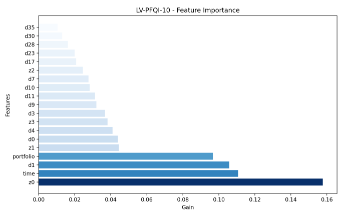
Figure 15: Comparison of the performance in terms of cumulative profit and loss (P&L) expressed in percentage for FQI variations on EUR/USD in the Validation year: 2019.



(a) LV-FQI feature importance.



(b) LV-PFQI-10 feature importance.



(c) LV-PFQI-10 feature importance.

Figure 16: Feature importance EUR/USD in Validation: 2019

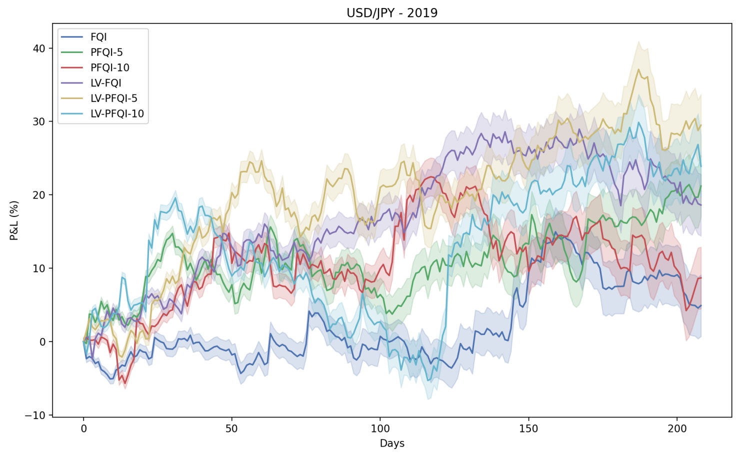
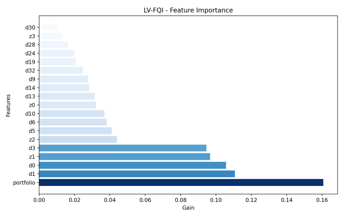
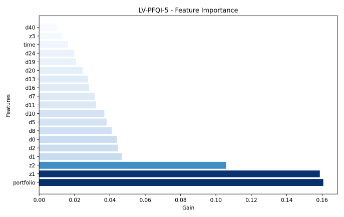


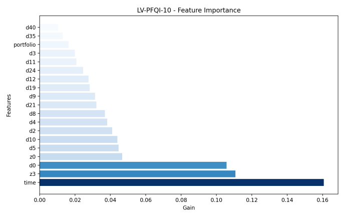
Figure 17: Comparison of the performance in terms of cumulative profit and loss (P&L) expressed in percentage for FQI variations on USD/JPY in the Validation year: 2019.



(a) LV-FQI feature importance.



(b) LV-PFQI-10 feature importance.



(c) LV-PFQI-10 feature importance.

Figure 18: Feature importance USD/JPY in Validation: 2019

5.4. Trading Strategies

This section presents the results of our Latent-Variable extension of FQI applied to both scenarios, named EUR/USD and USD/JPY, compared against different trading benchmarks in the Test Year 2022. The main objective is to assess the effectiveness of our proposed approach compared to various passive and active trading strategies. The passive strategy is called *Buy & Hold*. It consists in keeping a constant long portfolio position, instead *active strategies* based on technical indicators commonly used in algorithmic trading software or by human traders.

Here we provide a short description of active benchmark strategies:

(*i*) *Momentum*: it involves buying securities already showing an upward price trend and selling securities in a downward trend. The idea is to follow the market's momentum and take advantage of the trend. A momentum indicator, such as the Relative Strength Index (RSI) or Moving Average Convergence Divergence (MACD), often identifies potential buying or selling opportunities.

(*ii*) *MACD*: it is a popular momentum indicator used to identify potential trend reversals. This strategy involves buying when the MACD line crosses above the signal line and selling when it crosses below the signal line. The MACD is calculated by subtracting the 26-period exponential moving average (EMA) from the 12-period EMA.

(*iii*) *Mean-Reverting*: it involves buying securities trading below their historical average price and selling securities trading above their historical average price. The idea is that prices will eventually revert back to their mean. A mean-reverting strategy may use statistical tools such as Bollinger Bands, which are calculated based on the standard deviation of the price, to identify potential buying or selling opportunities.

The results obtained in this study indicate that the proposed algorithm, which utilizes latent variables with offline RL, is a promising approach for algorithmic trading in the FX market. Specifically, the LV-PFQI model with persistence of 5 and the LV-FQI model outperformed existing trading strategies, as demonstrated by the significantly higher cumulative P&L achieved in both trading scenarios, as shown in Fig 21 and 22. Moreover, these models showed consistent performance in the test year, indicating they can generalize well by beating the best PFQI model as shown in Fig. 20 and **??**. These findings suggest that utilizing latent variables can enhance the performance of offline RL algorithms in algorithmic trading, highlighting the potential of this approach for practical applications.

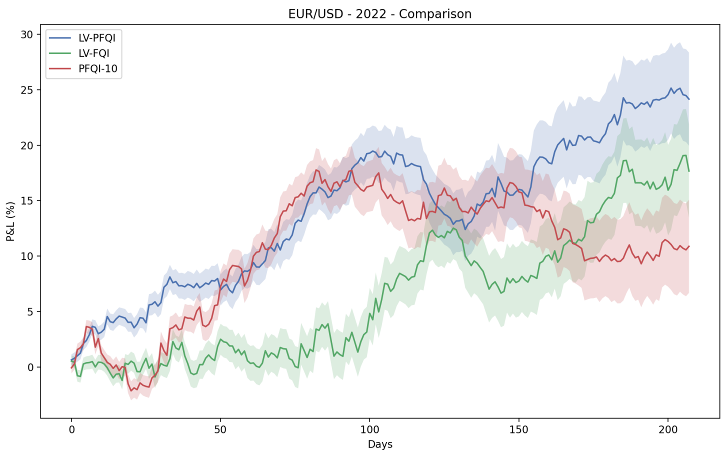


Figure 19: Comparison of the performance in terms of cumulative profit and loss (P&L) expressed in percentage for the best LV-FQI against the best PFQI model on EUR/USD in the Test year: 2022.

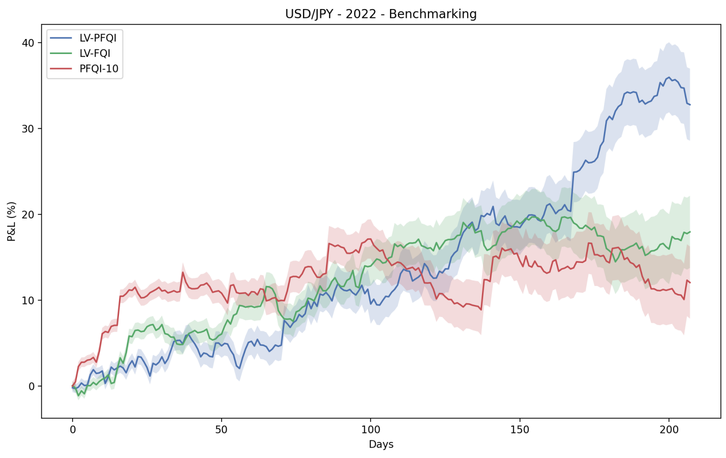


Figure 20: Comparison of the performance in terms of cumulative profit and loss (P&L) expressed in percentage for the best LV-FQI against the best PFQI model on USD/JPY in the Test year: 2022.

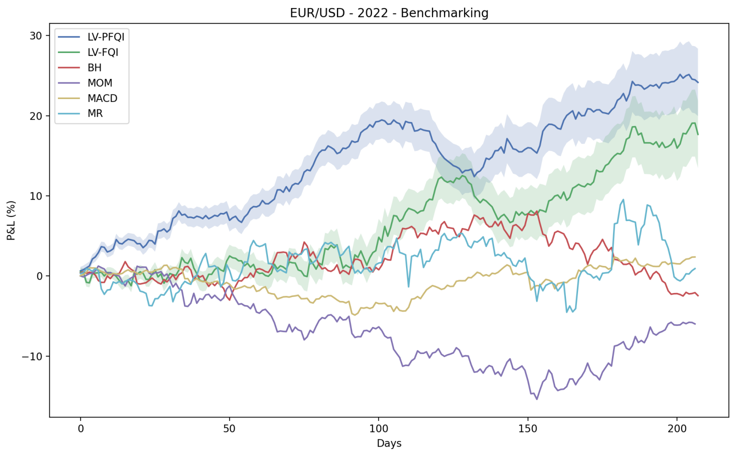


Figure 21: Comparison of the performance in terms of cumulative profit and loss (P&L) expressed in percentage for the best LV-FQI variations against the trading benchmarks on EUR/USD in the Test year: 2022.

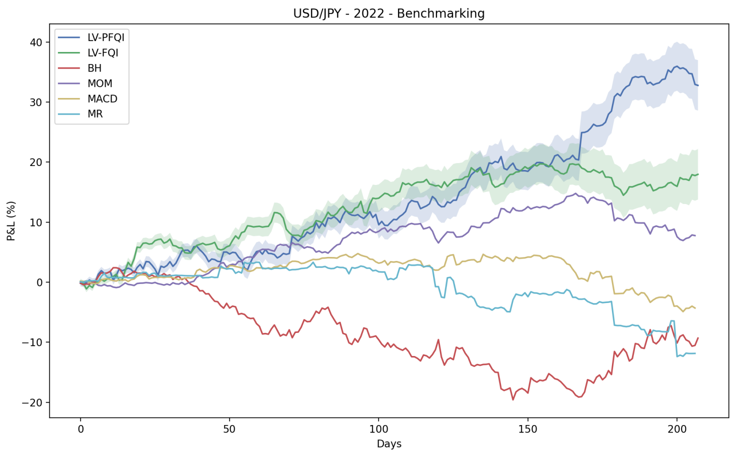


Figure 22: Comparison of the performance in terms of cumulative profit and loss (P&L) expressed in percentage for the best LV-FQI variations against the trading benchmarks on USD/JPY in the Test year: 2022.

6. Conclusions

The research and development of effective medium and high-frequency trading strategies is a significant challenge for AI in finance, particularly in the Forex Exchange market, where currency pair fluctuations are heavily influenced by noise. As exchange rates are a prime example of non-stationary and noisy time-series data, it is challenging to learn from them due to the unpredictability of financial markets. Indeed, any attempt to extract patterns using models designed for stationary time series is susceptible to errors.

To address this issue, we proposed a novel two-stage Reinforcement Learning (RL) framework to separate representation learning and task learning by relying on Variational Auto-encoders (VAE) to first explicitly acquire a latent representation and then train the RL agent by leveraging the features extracted from the latent space with a model-free and offline algorithm, Fitted Q-Iteration (FQI). It is a batch algorithm where the agent can evaluate the effects of his possible portfolio allocations on a historical dataset while observing the rates of the last hour. This raised interesting experimentation around the optimal trading frequency to find trading opportunities by trying different action persistence values. Furthermore, by utilizing latent variables to represent the underlying structure of the input data, we could improve the trading policy learned in the RL phase.

Our proposed algorithm, Latent Variable (Persistent) Fitted Q-Iteration, obtained robust results on synthetic data generated from classical financial models, where we validated the direct impact of the latent features on the performance. In addition, it obtained consistent results with an attractive performance profile compared to existing baseline RL agents, as well as both active and passive trading benchmark strategies, in terms of various metrics such as profit and loss, Sharpe ratio, and maximum drawdown. Precisely, we applied our approach to the EUR/USD and USD/JPY currency pairs, among the most liquid ones. This empirically proves two key points: firstly, the advantage of using latent variables in a batch RL algorithm to have a compact representation of the market non-linearity improving the results of previous work and overcoming the limitations of model-based approaches; secondly, that RL agents can outperform human and algorithmic tradings strategies. To the best of our knowledge, this is the first attempt that leverages the use of latent space added to a Batch-RL algorithm and further investigates the framework's effectiveness in the FX trading domain.

Although the research objectives have been achieved, improvements can still be made to our approach from both practical and theoretical perspectives. Future studies could develop a theoretical framework around the latent approach in RL and test the generality of the two-step method in other domains with high non-stationarity. To address non-stationarity, a mechanism that can detect regime shifts or handle price forecasting with non-stationary time series may be included in the feature space to improve performance. Instead, from an application perspective, the system can be made more realistic by incorporating the instantaneous spread and the order book dynamics to create more complex strategies, including limit and stop-loss orders and partial allocation of the portfolio amount. Additionally, the method can be applied to other asset classes, such as equities and commodities. Other interesting approaches to RL in trading that are emerging and worth investigating include hierarchical, multi-objective and robust RL.

References

[1] Bloomberg Execution Management System. `https://www.bloomberg.com/professional/product/trading/execution-management-system/`. Accessed on: April 5th, 2023.

[2] Cme group - ebs platform. `https://www.cmegroup.com/markets/ebs.html`. Accessed: April 5, 2023.

[3] Histdata.com. `https://www.histdata.com`. Accessed on April 9, 2023.

[4] Refinitiv - fx trading platforms. `https://www.refinitiv.com/en/trading-solutions/fx-trading-platforms#solutions`. Accessed: April 5, 2023.

[5] Jacob D Abernethy and Satyen Kale. Adaptive market making via online learning. In *NIPS*, pages 2058–2066. Citeseer, 2013.

[6] Robert Almgren and Neil Chriss. Optimal execution of portfolio transactions. *Journal of Risk*, 3:5–40, 2001.

[7] Hasan Alsabah, Agostino Capponi, Oscar Ruiz Lacedelli, and Michael Stern. Robo-advising: Learning investors' risk preferences via portfolio choices. *Journal of Financial Econometrics*, 19:369–392, 2021.

[8] Matteo Aquilina, Eric Budish, and Peter O'Neill. Quantifying the High-Frequency Trading "Arms Race"*. *The Quarterly Journal of Economics*, 137(1):493–564, 09 2021.

[9] V. Bacoyannis, V. Glukhov, T. Jin, J. Kochems, and D. R. Song. Idiosyncrasies and challenges of data driven learning in electronic trading. *arXiv preprint arXiv:1811.09549*, 2018.

[10] R.G. Bates, Michael Dempster, and Y.S. Romahi. Evolutionary reinforcement learning in fx order book and order flow analysis. pages 355 – 362, 04 2003.

[11] Richard Bellman. The theory of dynamic programming. *Bulletin of the American Mathematical Society*, 60(6):503–515, 1954.

[12] Christopher Berner, Greg Brockman, Ben Chan, Vicki Cheung, Piotr Dębiak, Collin Dennison, Damien Farhi, Quentin Fischer, Hafiz Hashme, Christopher Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

[13] Lorenzo Bisi, Pierre Liotet, Luca Sabbioni, Gianmarco Reho, Nico Montali, Marcello Restelli, and Cristiana Corno. Foreign exchange trading: A risk-averse batch reinforcement learning approach. In *Proceedings of the First ACM International Conference on AI in Finance (ICAIF '20)*, pages 26:1–26:8, New York, New York, USA, 2020. Association for Computing Machinery, ACM.

[14] Lorenzo Bisi, Luca Sabbioni, Edoardo Vittori, Matteo Papini, and Marcello Restelli. Risk-averse trust region optimization for reward-volatility reduction. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, IJCAI'20, 2021.

[15] Fischer Black and Myron Scholes. The pricing of options and corporate liabilities. *Journal of Political Economy*, 81(3):637–654, 1973.

[16] David M. Blei, Alp Kucukelbir, and Jon D. McAuliffe. Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 112(518):859–877, apr 2017.

[17] Andrea Bonarini, Claudio Caccia, Alessandro Lazaric, and Marcello Restelli. Batch reinforcement learning for controlling a mobile wheeled pendulum robot. In *IFIP International Conference on Artificial Intelligence in Theory and Practice*, pages 151–160. Springer, 2008.

[18] Gabriel Borrageiro, Nick Firoozye, and Paolo Barucca. Reinforcement learning for systematic FX trading. *IEEE Access*, 10:5024–5036, 2021.

[19] Leo Breiman, Jerome Friedman, Richard Olshen, and Charles Stone. *Classification and Regression Trees*. Chapman and Hall, 1984.

[20] Antonio Briola, Jeremy Turiel, Riccardo Marcaccioli, and Tomaso Aste. Deep reinforcement learning for active high frequency trading, 2021.

[21] Noam Brown and Tuomas Sandholm. Superhuman ai for multiplayer poker. *Science*, 365(6456):885–890, 2019.

[22] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.

[23] Serkan Cabi, Sergio Gómez Colmenarejo, Alexander Novikov, Ksenia Konyushkova, Scott Reed, Rae Jeong, Konrad Zolna, Yusuf Aytar, David Budden, Mel Vecerik, et al. Scaling data-driven robotics with reward sketching and batch reinforcement learning. *arXiv preprint arXiv:1909.12212*, 2019.

[24] Jing Cao, Jiajie Chen, John Hull, and Zili Poulos. Deep hedging of derivatives using reinforcement learning. *The Journal of Financial Data Science*, 3:10–27, 2021.

[25] João Carapuço, Rui Neves, and Nuno Horta. Reinforcement learning applied to forex trading. *Applied Soft Computing*, 73:783–794, 2018.

[26] Anthony R Cassandra. Acting optimally in partially observable stochastic domains. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, pages 1023–1028. AAAI Press, 1998.

[27] Andrea Castelletti, Francesca Pianosi, and Marcello Restelli. Tree-based fitted q-iteration for multi-objective markov decision problems. In *The 2012 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2012.

[28] Alain Chaboud, Benjamin Chiquoine, Erik Hjalmarsson, and Clara Vega. Rise of the machines: algorithmic trading in the foreign exchange market. *The Journal of Finance*, 69(5):2045–2084, 2009.

[29] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794. ACM, 2016.

[30] Junyoung Chung, Kyle Kastner, Laurent Dinh, Kratarth Goel, Aaron Courville, and Yoshua Bengio. A recurrent latent variable model for sequential data, 2016.

[31] Carole Comerton-Forde and Talis J Putnins. Dark trading and price discovery. *Journal of Financial Economics*, 118(1):70–92, 2015.

[32] James Cumming, Dalal Alrajeh, and Luke Dickens. An investigation into the use of reinforcement learning techniques within the algorithmic trading domain. Master's thesis, Imperial College London, 2015.

[33] M. A. Dempster, T. W. Payne, Y. Romahi, and G. W. Thompson. Computational learning techniques for intraday fx trading using popular technical indicators. *IEEE Transactions on Neural Networks*, 12(4):744–754, 2001.

[34] M.A.H. Dempster and V. Leemans. An automated fx trading system using adaptive reinforcement learning. *Expert Systems with Applications*, 30(3):543–552, 2006.

[35] Michael A.H. Dempster and Youssef S. Romahi. Intraday fx trading: An evolutionary reinforcement learning approach. In *Proceedings of the International Conference on Intelligent Data Engineering and Automated Learning*, pages 347–358. Springer, 2002.

[36] Abhyuday Desai, Cynthia Freeman, Zuhui Wang, and Ian Beaver. Timevae: A variational auto-encoder for multivariate time series generation, 2021.

[37] Jiaming Du, Miao Jin, Petter N Kolm, Guy Ritter, Yanqin Wang, and Bin Zhang. Deep reinforcement learning for option replication and hedging. *The Journal of Financial Data Science*, 2:44–57, 2020.

[38] Gabriel Dulac-Arnold, Daniel Mankowitz, and Todd Hester. Challenges of real-world reinforcement learning. *arXiv preprint arXiv:1904.12901*, 2019.

[39] Carlo D'Eramo, Marcello Restelli, and Alessandro Nuara. Estimating maximum expected value through gaussian approximation. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1032–1040, New York, New York, USA, 20–22 Jun 2016. PMLR.

[40] Damien Ernst, Pierre Geurts, and Louis Wehenkel. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6(Apr):503–556, 2005.

[41] Eugene F Fama. The behavior of stock market prices. *The Journal of Business*, 38(1):34–105, 1965.

[42] Leonardo Kanashiro Felizardo, Francisco Caio Lima Paiva, Anna Helena Reali Costa, and Emilio Del-Moral-Hernandez. Reinforcement learning applied to trading systems: A survey. 2022.

[43] Bank for International Settlements. 2022 triennial central bank survey. `https://www.bis.org/statistics/rpfx22.htm`, 2022. Accessed: April 2, 2023.

[44] Yuwei Fu, Wu Di, and Benoit Boulet. Batch reinforcement learning in the real world: A survey. In *Offline Reinforcement Learning Workshop at Neural Information Processing Systems*, pages 1–71, 2020.

[45] Kuzman Ganchev, Yuriy Nevmyvaka, Michael Kearns, and Jennifer Wortman Vaughan. Censored exploration and the dark pool problem. *Communications of the ACM*, 53(10):99–107, 2010.

[46] Xiu Gao, Shatin Hongkong, and Laiwan Chan. An algorithm for trading and portfolio management using q-learning and sharpe ratio maximization. 02 2001.

[47] Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Machine Learning*, 63(Apr):3–42, 2006.

[48] Carl Gold. Fx trading via recurrent reinforcement learning. pages 363 – 370, 04 2003.

[49] Geoffrey J Gordon. Approximate solutions to markov decision processes. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA SCHOOL OF COMPUTER SCIENCE, 1999.

[50] David Hendricks and David Wilcox. A reinforcement learning extension to the almgren-chriss framework for optimal trade execution. In *2014 IEEE Conference on Computational Intelligence for Financial Engineering & Economics (CIFEr)*, pages 457–464. IEEE, 2014.

[51] Tin Kam Ho. Random decision forests. In *Proceedings of 3rd International Conference on Document Analysis and Recognition*, volume 1, pages 278–282. IEEE, 1995.

[52] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[53] Andriy Hryshko and Tom Downs. System for foreign exchange trading using genetic algorithms and reinforcement learning. *International Journal of Systems Science*, 35(10):763–774, 2004.

[54] Chien Yi Huang. Financial trading as a game: A deep reinforcement learning approach. *arXiv preprint arXiv:1807.02787*, 2018.

[55] John C Hull. *Options, Futures, and Other Derivatives.* Pearson Education Limited, 10th edition, 2017.

[56] Vasanth Kalingeri. Latent variable modelling using variational autoencoders: A survey, 2022.

[57] Robert L. Kessler. *Algorithmic Trading Methods: Applications Using Advanced Statistics, Optimization, and Machine Learning Techniques.* Academic Press, 2nd edition, 2020.

[58] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *International Conference on Learning Representations (ICLR)*, 2014.

[59] Diederik P Kingma and Max Welling. An introduction to variational autoencoders. *arXiv preprint arXiv:1906.02691*, 2019.

[60] Sascha Lange, Thomas Gabel, and Martin Riedmiller. Batch reinforcement learning. In *Reinforcement Learning*, pages 45–73. Springer, 2012.

[61] Erwan Lecarpentier, Guillaume Infantes, Charles Lesire, and Emmanuel Rachelson. Open loop execution of tree-search algorithms. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, pages 2362–2368. International Joint Conferences on Artificial Intelligence Organization, 2018.

[62] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[63] Charles-Albert Lehalle and Sophie Laruelle, editors. *Market Microstructure in Practice.* World Scientific Books. World Scientific Publishing Co. Pte. Ltd., 2018.

[64] Sergey Levine and Vladlen Koltun. Guided policy search. In *International Conference on Machine Learning*, pages 1–9. PMLR, 2013.

[65] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

[66] Yang Liu, Qi Liu, Hongke Zhao, Zhen Pan, and Chuanren Liu. Adaptive quantitative trading: An imitative deep reinforcement learning approach. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34:2128–2135, 04 2020.

[67] Andrew W Lo. The adaptive markets hypothesis. *The Journal of Portfolio Management*, 30(5):15–29, Jan 2004.

[68] Nehal Metawa, Mohamed Elhoseny, Aboul Ella Hassanien, and Mohammad Kamrul Hassan, editors. *Expert Systems in Finance: Smart Financial Applications in Big Data Environments.* Routledge, 1st edition, 2019.

[69] Alberto Maria Metelli, Flavio Mazzolini, Lorenzo Bisi, Luca Sabbioni, and Marcello Restelli. Control frequency adaptation via action persistence in batch reinforcement learning. In *Proceedings of the 37th International Conference on Machine Learning*, ICML'20. JMLR.org, 2020.

[70] Tom M. Mitchell. *Machine Learning.* McGraw-Hill Education, 1997.

[71] John Moody and Mark Saffell. Learning to trade via direct reinforcement. *IEEE Transactions on Neural Networks*, 12(4):875–889, 2001.

[72] Bo Ning, Francis HT Ling, and Sebastian Jaimungal. Double deep q-learning for optimal execution. *arXiv preprint arXiv:1812.06600*, 2018.

[73] Maureen O'Hara. High frequency market microstructure. *Journal of Financial Economics*, 116(2):257–270, 2015.

[74] Parag C Pendharkar and Patrick Cusatis. Trading financial indices with reinforcement learning agents. *Expert Systems with Applications*, 103:1–13, 2018.

[75] Xue Bin Peng, Aviral Kumar, Grace Zhang, and Sergey Levine. Advantage-weighted regression: Simple and scalable off-policy reinforcement learning. *arXiv preprint arXiv:1910.00177*, 2019.

[76] Tidor-Vlad Pricope. Deep reinforcement learning in quantitative algorithmic trading: A review. *arXiv preprint arXiv:2106.00123*, 2021.

[77] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming.* Wiley & Sons, 2014.

[78] Antonio Riva, Lorenzo Bisi, Pierre Liotet, Luca Sabbioni, Edoardo Vittori, Marco Pinciroli, Michele Trapletti, and Marcello Restelli. Learning fx trading strategies with fqi and persistent actions. In *Proceedings of the Second ACM International Conference on AI in Finance*, pages 1–9, 2021.

[79] Antonio Riva, Lorenzo Bisi, Pierre Liotet, Luca Sabbioni, Edoardo Vittori, Marco Pinciroli, Michele Trapletti, and Marcello Restelli. Addressing non-stationarity in fx trading with online model selection of offline rl experts. In *Proceedings of the Third ACM International Conference on AI in Finance*, ICAIF '22, page 394–402, New York, NY, USA, 2022. Association for Computing Machinery.

[80] Diederik M Roijers, Peter Vamplew, Shimon Whiteson, and Richard Dazeley. A survey of multi-objective sequential decision-making. *Journal of Artificial Intelligence Research*, 48:67–113, 2013.

[81] Paul A Samuelson. Rational theory of warrant pricing. *Industrial Management Review*, 6(2):13–31, 1965.

[82] David Silver, Aja Huang, Chris Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[83] Sutta Sornmayura. Robust forex trading with deep q network (dqn). *ABAC Journal*, 39(1), 2019.

[84] Thomas Spooner, John Fearnley, Rahul Savani, and Andreas Koukorinis. Market making via reinforcement learning. In *International Foundation for Autonomous Agents and Multiagent Systems, AAMAS '18*, pages 434–442. International Foundation for Autonomous Agents and Multiagent Systems, 2018.

[85] Shuo Sun, Rundong Wang, and Bo An. Reinforcement learning for quantitative trading. *ACM Trans. Intell. Syst. Technol.*, 14(3), mar 2023.

[86] Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction.* The MIT Press, 2018.

[87] Csaba Szepesvári. *Algorithms for reinforcement learning*, volume 4 of *Synthesis lectures on artificial intelligence and machine learning.* Morgan & Claypool Publishers, 2010.

[88] Thibaut Théate and Damien Ernst. An application of deep reinforcement learning to algorithmic trading. *Expert Systems with Applications*, 173:114632, jul 2021.

[89] José F. Torres, Dalil Hadjout, Abderrazak Sebaa, Francisco Martínez-Álvarez, and Alicia Troncoso. Deep learning for time series forecasting: A survey. *Big Data*, 9(1):3–21, 2021. PMID: 33275484.

[90] Oldrich Vasicek. An equilibrium characterization of the term structure. *Journal of financial economics*, 5(2):177–188, 1977.

[91] Ashish Vaswani, Noam Shazeer, Niki Parmar, and et al. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

[92] Edoardo Vittori, Amarildo Likmeta, and Marcello Restelli. Monte carlo tree search for trading and hedging. In *Proceedings of the Second ACM International Conference on AI in Finance*, ICAIF '21, New York, NY, USA, 2021. Association for Computing Machinery.

[93] Christopher JCH Watkins. *Learning from Delayed Rewards.* PhD thesis, University of Cambridge, Cambridge, England, 1989.

[94] Chen-Yu Wei, Yi-Te Hong, and Chi-Jen Lu. Tracking the best expert in non-stationary stochastic environments. *CoRR*, abs/1712.00578, 2017.

[95] Moritz Wiese, Robert Knobloch, Ralf Korn, and Paul Kretschmer. Quant gans: Deep generation of financial time series. *Quantitative Finance*, 20:1419–1440, 2020.

[96] Hongyang Yang, Xiao-Yang Liu, Shan Zhong, and Walid Anwar. Deep reinforcement learning for automated stock trading: An ensemble strategy. In *ICAIF '20: ACM International Conference on AI in Finance*, 2020.

[97] Philip Yu, Jun Shian Lee, Igor Kulyatin, Zhe Shi, and Sanjoy Dasgupta. Model-based deep reinforcement learning for dynamic portfolio optimization. *arXiv preprint arXiv:1901.08740*, 2019.

[98] Konstantinos Saitas Zarkias, Nikolaos Passalis, Avraam Tsantekidis, and Anastasios Tefas. Deep reinforcement learning for financial trading using price trailing. In *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3067–3071, 2019.

[99] Zihao Zhang, Stefan Zohren, and Stephen Roberts. Deep reinforcement learning for trading, 2019.

[100] Ding Zhou, Jiajie Chen, and Qi Gu. Provable multi-objective reinforcement learning with generative models. *arXiv preprint arXiv:2011.10134*, 2020.