Reinforcement Learning for Optimal Execution Algorithms

Master's Thesis by

Ethan Chemla

In Partial Fulfillment of the Requirements for the

Degree of

M2 - Artificial Intelligence, Systems and Data (IASD)

University Paris Dauphine - PSL
Paris, France

Within the Execution Algo Team
Under the supervision of the FX Trading Desk

2024 - 2025
Defended September 17, 2025

Abstract

In a context of increasingly complex and non-stationary financial markets, traditional algorithmic execution strategies struggle to maintain their performance. This thesis, conducted within the quantitative trading team at Natixis, addresses the challenge of moving beyond simple execution optimization to develop an autonomous trading agent capable of generating a consistent execution alpha. The core problem lies in the critical transition from prediction to action, a gap that this work aims to bridge through a complete and integrated Reinforcement Learning (RL) pipeline.

The methodology is structured in two distinct phases. First, we focus on building a robust predictive foundation through the engineering of original, theory-driven features. We introduce two novel concepts: the **Noise Range Percentage (NRP)**, a normalized and adaptive oscillator designed to capture temporary price deviations from its dynamic local trend based on mean-reversion hypotheses, and the **Adaptive Trend Finder (ATF)**, a mechanism that dynamically identifies the most statistically significant trend timescale. These features, along with more traditional indicators, are used to train advanced supervised models (XGBoost, ResNet) whose probabilistic predictions serve as high-level inputs for the second phase.

The second phase implements the end-to-end reinforcement learning framework. An agent based on the **Proximal Policy Optimization (PPO)** algorithm is trained within a custom-built optimal liquidation environment. Its objective is to intelligently liquidate a significant position over a finite horizon. The agent's learning is guided by a sophisticated reward function engineered to maximize the financial outperformance against an industry-standard **Time-Weighted Average Price (TWAP)** benchmark, while incorporating dynamic penalties for adverse market impact and incomplete liquidation. The agent's policy network leverages a hybrid **CNN-LSTM architecture** to effectively process the temporal nature of the market state.

The results first validate the predictive edge of our engineered features, which form a solid foundation for the RL agent. The final out-of-sample evaluation on an unseen dataset confirms the successful training of the PPO agent. The agent demonstrates the ability to learn a profitable and adaptive execution strategy, generating an average alpha of **+$42.19** per episode against the TWAP benchmark, achieving a **win rate of 55.57%** and a **99.5%** order completion rate. This thesis thus not only establishes a rigorous methodology for high-frequency signal extraction but also demonstrates its practical application by building a complete RL-based agent capable of translating these signals into a superior execution policy.

Contents

1 Introduction

1.1 General Introduction

Since the beginning of financial markets, the most valuable asset has always been information, and more specifically, the speed at which it could be obtained and exploited. The most famous anecdote remains that of the Rothschild family who, informed of Napoleon's defeat at Waterloo in 1815 before anyone else thanks to their own network of carrier pigeons, allegedly built part of their fortune on this decisive informational advantage. Later, the advent of the telegraph continued to widen the gap between those who could act instantly on news and others. This time advantage, measured in days and then in minutes, was the original engine of finance, long before speed was counted in fractions of a second. On the open outcry trading floors, the physical proximity of exchanges and the power of the voice remained manifestations of this fundamental need to act before others.

The real breakthrough occurred with the digital revolution. The creation of NASDAQ in 1971, the first fully electronic stock exchange, laid the foundations for a new paradigm. The successor to the carrier pigeon was the algorithm, capable of processing and reacting to information faster than any human being. This transition triggered a true race for latency, a fierce technological competition to gain a few microseconds. Colossal engineering projects emerged, such as the laying of nearly straight-line fiber optic cables between data centers in Chicago and New York to reduce the travel time of light. High-Frequency Trading (HFT) was born from this race, transforming markets into a complex ecosystem where algorithms interact at unimaginable speeds.

However, this large-scale automation revealed its own systemic fragility. The most significant event was the Flash Crash of May 6, 2010. In the span of a few minutes, a single massive selling algorithm on E-Mini S&P 500 futures contracts triggered a chain reaction, causing the near-instantaneous withdrawal of liquidity by thousands of other automated systems. The Dow Jones lost nearly 1000 points, or 9% of its value, before rebounding just as spectacularly. This event served as a brutal awakening: pure speed, devoid of a deep understanding of the market context, could generate extreme and uncontrollable instability.

From then on, the frontier of innovation shifted again. Faced with the decline of alpha, a phenomenon whereby trading strategies lose their effectiveness as they are discovered and replicated, simple speed optimization became insufficient. A new era opened, that of artificial intelligence. The most sophisticated players, such as quantitative hedge funds like Citadel or Millennium, and proprietary trading firms like Jane Street, no longer seek only to be the fastest. They now employ advanced machine learning and deep learning techniques to model non-linear market dynamics, understand complex volatility regimes, and develop strategies capable of learning and adapting in real time. The goal is no longer just to execute faster, but to decide smarter. It is in this context of searching for an informational alpha through adaptive strategies that this thesis is situated.

1.2 Professional and Strategic Context at Natixis

This thesis is anchored in the operational context of the quantitative trading team at Natixis, whose historical mission is to develop algorithmic strategies for the bank, particularly to optimize the execution of client orders. Traditionally, these strategies were designed to guarantee the best possible price, an essential function for an investment bank. However, the team has recently observed an erosion in the performance of these conventional approaches, faced with increasingly complex and competitive financial markets.

This observation catalyzed a dual awareness. On the one hand, competitive pressure from

other French banking players, who are increasingly integrating artificial intelligence into their quantitative research. On the other hand, a strategic internal desire at Natixis to accelerate its digital transformation and explore the potential offered by AI to tackle more difficult markets and create new sources of value. This thesis is therefore part of a project aimed at moving beyond pure execution optimization to develop a new source of revenue. The objective is to design an algorithm capable, while executing client orders within strict risk frameworks, of intelligently exploiting micro-market opportunities to capture an execution alpha.

1.3 Theoretical and Methodological Context

The design of such an algorithm faces major theoretical challenges inherent to financial data. These are characterized by at least three fundamental difficulties:

- **Noise:** Relevant information about future movements is drowned in a considerable volume of random fluctuations, making signals difficult to isolate.
- **Non-Stationarity:** The statistical properties of markets (volatility, correlations) are constantly evolving. A model that performs well in one market regime may become obsolete when the context changes.
- **Non-Linear Dependencies:** Cause-and-effect relationships are complex and cannot be captured by classical linear statistical models, requiring more sophisticated approaches.

Faced with these challenges, the team's approach has evolved into a two-step process: first, calibrating a probability of future movement using machine learning models; second, exploiting this prediction within a stochastic optimization framework. While this approach is already a significant advance, it retains a fundamental limitation: a prediction does not constitute a policy of action. It does not directly answer operational questions: should we act now? With what position size? Should we wait for a stronger signal at the risk of missing an opportunity?

It is to address this dilemma that reinforcement learning offers a paradigm shift. Rather than predicting the future to then decide, the reinforcement learning agent directly learns an optimal decision policy. By formalizing the problem as a Markov Decision Process (MDP), where an agent interacts with an environment (the market) by choosing actions (buy, sell, wait) to maximize a reward (the gain, adjusted for risk), we approach the problem holistically. This framework allows for the search for good enough strategies that, without claiming absolute theoretical optimality which is often inaccessible, offer superior robustness and performance in practice. This thesis will therefore explore these two paths: the sequential approach of prediction then optimization, and the integrated end-to-end approach of reinforcement learning.

1.4 Justification, Stakes and Motivation

The genesis of this project illustrates a research and development process at the heart of modern quantitative finance, structured around two distinct and complementary lines of work. The first axis, which constitutes a substantial part of the work carried out, focused on the challenge of predicting short-term returns. This task, fundamental in quantitative trading, required the development and validation of state-of-the-art deep learning models. The objective was to build a robust foundation capable of generating reliable predictive signals, a valuable contribution in itself.

The second axis of the project, of a more exploratory nature, arose from a collaboration with the structuring teams at Natixis working on energy markets. Faced with complex optimization problems, they rely on stochastic optimization methods. Inspired by this approach but seeking

a potentially more flexible path, it was proposed to test reinforcement learning (RL) as an alternative method to exploit the predictions from the first axis. This second phase therefore does not aim to solve a classic optimization equation, but to let a digital agent directly learn the best trading strategy. The gap that this thesis seeks to bridge is therefore twofold: first, that of reliable prediction, then that of the optimal transformation of this prediction into action.

The stake of this double exploration is significant. On the one hand, mastering Deep Learning techniques for prediction has become a standard to remain competitive. On the other hand, the application of deep reinforcement learning (Deep RL) to trading is a research area in full swing. While the theoretical concepts of RL have existed for decades, their combination with the power of deep neural networks has only generated massive interest in the scientific literature since 2018-2020. For Natixis, the strategic stake is therefore twofold: to consolidate its expertise on predictive models and to position itself at the forefront of research on autonomous decision-making models.

On a personal level, this subject represents a deep motivation to master the complete value chain of modern algorithmic trading, from prediction to action. The almost perfect relevance between the theoretical framework of RL and the reality of trading is a source of intellectual fascination. The main challenge now lies in the concrete implementation of these two technological bricks, adapting models from research to a professional environment, while facing technical constraints, particularly in terms of computing power.

1.5 Problem Statement

The financial market environment, characterized by increasing complexity and speed, constantly challenges traditional algorithmic trading approaches. These, often based on predefined rules or classic econometric models, demonstrate a certain rigidity and struggle to adapt dynamically to changes in market regime. Faced with this observation, the advent of machine learning techniques, and more particularly deep neural networks such as GRU or LSTM architectures, has opened a new era for the prediction of financial time series. These models offer an unprecedented ability to detect complex patterns and generate short-term return forecasts.

However, this new advance shifts the problem: a prediction, even a quality one, does not constitute a strategy. It raises a series of fundamental questions: how to translate this predictive information into an optimal sequence of operations? How to decide on the size of a position, the precise moment of an intervention or abstention, while managing the inherent risk and transaction costs? The simple application of a threshold to a prediction proves to be a largely insufficient answer to this challenge of sequential decision-making.

It is to address this transition from prediction to action that this thesis explores a hybrid approach, at the crossroads of two fields of artificial intelligence. The central problem of our study can thus be formulated as follows: *To what extent does the application of predictive models and a modeling based on reinforcement learning allow for the design of an autonomous trading agent, whose performance exceeds that of traditional algorithmic strategies?*

2 State of the Art

2.1 Introduction

Modeling and predicting financial time series represent one of the most complex yet most studied challenges in quantitative finance and data science. The ability to anticipate, even with a small margin, the future movements of an asset's price can generate considerable financial returns. However, the very nature of financial markets makes this task particularly difficult because,

unlike many physical or biological phenomena, for example, financial series are not the product of deterministic laws, but rather the aggregate result of the interactions of millions of agents, whose decisions are influenced by a constant flow of information and cognitive biases. This complexity manifests itself in unique statistical properties that systematically violate the assumptions of traditional econometric models, thus justifying the use of increasingly sophisticated approaches such as Machine Learning.

2.2 The Main Challenges

2.2.1 The Stochastic Nature of Financial Markets

First, non-stationarity is a predominant characteristic. A process is said to be stationary if its statistical properties, such as its mean and variance, remain constant over time. However, the price series of financial assets almost always exhibit trends, cycles, and seasonalities, which means their behavior is time-dependent. The basic model often used to describe price evolution, the random walk (Fama, 1965), is an example of a non-stationary process, as its variance increases linearly with time, making forecasts increasingly uncertain as the horizon lengthens. This non-stationarity implies that predictive models must be able to adapt constantly.

Second, the relationships between variables are non-linear. Linear models explain a predictable and proportional evolution of variables, a simplification that does not capture the reality of markets, influenced by complex factors such as investor behavior or unforeseen exogenous events. Numerous empirical studies have shown that financial series, although approximately uncorrelated, are not temporally independent, with the dependence manifesting through higher-order moments (Brock et al., 1996).

Third, conditional heteroskedasticity, and more specifically the phenomenon of volatility clustering, that is, the tendency for periods of high volatility to be followed by other periods of high volatility, and vice versa for periods of low volatility. This means that the variance of returns is not constant (homoscedastic), but is conditional on past information (Engle, 1982). The GARCH (Generalized Autoregressive Conditional Heteroskedasticity) family of models was specifically designed to capture this dynamic phenomenon of variance.

Finally, the distributions of financial returns exhibit fat tails. Extreme events, whether large increases or decreases, are much more likely than a normal (Gaussian) distribution would predict (Mandelbrot, 1963). This property has crucial implications for risk management, as it invalidates models that underestimate the probability of extreme market movements. As Li (2023) explains, this deviation from normality can be interpreted by the non-linear reactions of markets to events: they tend to underreact to minor news and overreact to major information, thus contributing to the empirically observed fat tails.

These properties should not be considered as isolated problems, but as interdependent facets of a complex and adaptive system. A change in market regime, which is a form of non-stationarity, can alter the nature of non-linearities and the intensity of volatility clustering. Therefore, an effective prediction model cannot be content with addressing only one of these characteristics. It must be intrinsically non-linear to capture complex relationships, heteroskedastic to model dynamic volatility, robust to extreme values to account for fat tails, and adaptive to manage non-stationarity and regime changes. This multifaceted requirement explains why classical econometric models quickly reach their limits and justifies the transition to deep learning architectures, which, by their nature, have greater freedom to learn and adapt to such dynamics (Ozbayoglu et al., 2020).

2.2.2 The Efficient Market Hypothesis and the Concept of Memory

At the heart of the debate on the predictability of financial markets is the Efficient Market Hypothesis (EMH). In its weak form, the EMH posits that all information contained in the history of prices is already fully reflected in current prices. The direct implication of this hypothesis is that asset prices follow a random walk, and that the best possible prediction for the future price is simply the current price. In such a scenario, any attempt to make abnormal profits based on the analysis of past prices would be futile. Influential historical studies, such as those by Meese and Rogoff (1983), have long supported this view by showing that the random walk outperformed structural econometric models and ARMA models of the time in predictive performance.

However, in apparent contradiction with the EMH, a growing body of empirical literature highlights the existence of long memory (or long-term dependence) in financial series. This phenomenon is characterized by the fact that past observations retain a statistically significant influence on future values, even over very long time horizons. This persistence is manifested by autocorrelation functions that decay very slowly, hyperbolically rather than exponentially, which is a sign of a non-trivial temporal structure.

The discovery of long memory did not originate in finance, but in hydrology. Hurst (1951), studying the floods of the Nile, developed a statistical analysis method known as R/S (Rescaled Range) analysis, which revealed cycles and abnormal persistence in the data series. Mandelbrot (1963) was the first to apply these ideas to financial markets, using Hurst's work to argue that price variations in cotton markets showed signs of long-term dependence. Mandelbrot showed that simple random walk models were inadequate to describe the complexity of price movements.

Empirically, one of the first articles to explicitly document long-term dependence in stock returns was by Greene and Fielitz (1977). By applying Hurst's R/S analysis to a large sample of stocks, they found significant evidence of long memory, concluding that returns were not independent. Later, the work of Andrew Lo and Craig MacKinlay (1988) dealt a serious blow to the random walk hypothesis. Using more robust statistical tests, they rejected the random walk hypothesis for weekly returns of US stocks, showing the existence of short-term positive autocorrelations. Although not focusing exclusively on long memory, their work paved the way for a broader questioning of market efficiency in its weak form.

This tension between the theory of efficiency and empirical evidence of predictability suggests that market efficiency is not a binary and static property, but rather a dynamic continuum. The fact that models like ARFIMA-GARCH can outperform the random walk indicates the existence of pockets of inefficiency or temporary predictable structures that these models are able to exploit. The phenomenon of long memory is precisely the statistical signature of these persistent inefficiencies. The success of a prediction strategy therefore depends on the model's ability to identify and adapt to these changing regimes of inefficiency. It is in this context that deep learning, with its ability to model complex dynamics and adapt to changing patterns, appears as a natural candidate to exploit these inefficiencies more systematically and robustly than traditional econometric models.

2.3 Part 1: Traditional Machine Learning Approaches

Historically, modeling attempts have relied on traditional statistical approaches, such as ARIMA (AutoRegressive Integrated Moving Average) or GARCH models. Although fundamental, these models are based on assumptions of linearity and stationarity that are often challenged by the real dynamics of the markets. The advent of machine learning (ML) and, more recently, deep learning (DL) marked a turning point. These techniques, capable of capturing complex non-linear relationships and hidden dependencies in vast datasets, have opened up new perspectives.

2.3.1 Support Vector Regression

Support Vector Regression (SVR) is an extension of Support Vector Machines (SVM) for regression problems. Its goal is to find a hyperplane that best fits the data, while tolerating a certain margin of error. The strength of SVR lies in the use of the kernel trick, which allows for the modeling of non-linear relationships by implicitly projecting the data into a higher-dimensional feature space, for example, with radial basis function (RBF) kernels.

A 2025 study empirically evaluated several ML algorithms, including SVR, to predict stock prices based on an alternative data source: insider trading transactions (Chakravorty & Elsayed, 2025). In this study, SVR with an RBF kernel proved to be the most performant model in terms of accuracy compared to decision trees or random forests. The study emphasizes that insider data, which reflects the transactions of company executives in their own company's stock, can provide valuable signals about market expectations. However, the authors conclude that no single data source is sufficient and that it is essential to integrate multiple sources, such as news or social media data, to improve the robustness of predictions.

Another study in 2023 directly compared SVR to more recent ensemble models like XGBoost and Random Forest, as well as a neural network (MLP), for short-term prediction (Rasekh et al., 2023). The data used were intraday prices (15-minute intervals) for volatile tech stocks (Tesla, Apple, Nvidia), enriched with numerous exogenous variables (market indices, interest rates, calendar variables). The results showed that XGBoost systematically outperforms the other models, including SVR, in terms of error metrics like RMSE and MAPE.

Once considered one of the top-performing models, SVR now serves as a robust baseline but is often surpassed by boosting models on large tabular datasets (Rasekh et al., 2023). Its success remains highly conditional on the quality of the input data and the choice of kernel. The study on insider trading shows that a theoretically less complex model like SVR can outperform other approaches if the input features contain a particularly strong and relevant signal. This highlights that the choice of model cannot be dissociated from the work of feature engineering. Moreover, the trade-off between accuracy and efficiency is a crucial factor for practical applications; SVR can be computationally expensive on very large datasets.

2.3.2 Ensemble Methods

These methods are based on the idea of combining the predictions of several simpler models, usually decision trees, to create a more powerful, robust, and accurate overall model. There are two main families: bagging, like Random Forests, which aims to reduce the model's variance, and boosting, like XGBoost or LightGBM, which aims to reduce bias by building models sequentially.

Random Forests

The Random Forest model builds a multitude of decision trees during the training phase (Breiman, 2001). Each tree is trained on a subsample of the data (obtained by bootstrap) and, at each node of the tree, the best split is sought among a random subset of features. The final prediction is obtained by aggregating the predictions of all the trees (by majority vote in classification or by averaging in regression). This double randomization process makes the model very robust to overfitting (Jain et al., 2024).

Recent research shows an evolution in the use of RFs, moving from a simple prediction model to a sophisticated component within larger pipelines. A 2025 study perfectly illustrates this trend by proposing a two-stage approach (Lee & Kim, 2025). Instead of using RF for the final prediction, the researchers employ it as a feature selection tool. A Random Forest Subset-based (RFS) is used to systematically test all combinations of features (high, low, volume, etc.)

and select the subset that minimizes the prediction error. This optimal subset of features is then used to train a more complex deep learning model (a BiGRU with an attention mechanism) on historical stock data. The RF acts here as an intelligent filter, reducing dimensionality and noise before passing the baton to a more powerful architecture.

Another innovation is presented in articles from 2024, which introduce the SARF (Sentiment-Augmented Random Forest) model (Jain et al., 2024; Yang et al., 2023; Wang et al., 2024). This approach enriches a traditional RF model with sentiment data generated by a Large Language Model (LLM) specialized in finance. The SARF model integrates sentiment scores, extracted from the analysis of financial news, as additional input features, alongside classic technical indicators. Experiments show that SARF significantly outperforms a standard RF and even an LSTM model, with an average accuracy improvement of 10%. These works demonstrate a shift in research; the goal is no longer simply to find the best model, but rather the best pipeline.

Gradient Boosting Machines

Gradient boosting algorithms build decision trees sequentially. Each new tree is trained to correct the residual errors of the previous trees. XGBoost (Extreme Gradient Boosting) popularized this approach by adding key improvements such as regularization to prevent overfitting and native handling of missing values. LightGBM (Light Gradient Boosting Machine) then optimized speed by using a leaf-wise growth strategy instead of the traditional level-wise growth, making it particularly effective on large datasets (Guennioui & El Beqqali, 2023).

A 2023 study proposes a hybrid Global XGBoost-LightGBM model that combines the predictions of these two powerful algorithms by averaging them (Guennioui & El Beqqali, 2023). The approach is described as global because the same hyperparameters are applied to a wide range of stocks in two very different markets: a mature market (USA) and an emerging market (Morocco). The results are striking: this ensemble not only outperforms the individual models but also deep learning models like LSTM, GRU, and CNN on this prediction task, while being faster to train.

Another research from 2025 tackles the highly volatile cryptocurrency market with a hybrid LSTM-XGBoost architecture (Agrawal et al., 2025). In this pipeline, an LSTM model is first used to capture the complex temporal dependencies in historical price series. The outputs of the LSTM (its predictions or hidden states) are then used as new features for an XGBoost model. The latter also integrates other variables, such as technical indicators or sentiment scores. The hybrid model consistently outperforms the LSTM and XGBoost models used in isolation.

These studies confirm that gradient boosting models are currently the champions for prediction problems on structured tabular data (Rasekh et al., 2023; Guennioui & El Beqqali, 2023). If the problem can be formulated with well-defined feature columns, XGBoost and LightGBM are the benchmarks to beat. The hybrid DL-ML architecture, like the LSTM-XGBoost model, represents a particularly winning strategy. It combines the best of both worlds: deep learning (LSTM) is used as an automated temporal feature extractor, solving the problem of manual engineering of sequential features. Classic machine learning (XGBoost) is then used as a powerful regressor on the set of structured features (those extracted by the LSTM and others). This approach recognizes that DL models excel on raw and unstructured data (sequences, text), while boosting models are king on clean tabular data. Hybridization is the bridge that connects these two worlds.

2.4 Part 2: Deep Learning Approaches

2.4.1 Recurrent Neural Networks for Sequential Data

Recurrent Neural Networks (RNNs) were the first architectures designed to process sequential data. However, their simple structure makes them prone to the vanishing gradient problem, which prevents them from learning long-term dependencies. To overcome this limitation, more sophisticated architectures were developed. LSTM (Long Short-Term Memory) and GRU (Gated Recurrent Unit) use gate mechanisms that control the flow of information through the network. The LSTM uses a forget gate, an input gate, and an output gate to maintain and update a separate cell state, acting as a long-term memory.

A 2025 study conducted a head-to-head comparison between a classic statistical model (ARIMA) and an LSTM for predicting the S&P 500 index (Patel, 2025; Nelson et al., 2017). The results were clear: the LSTM achieved an accuracy of 96.4% with a mean absolute error (MAE) of 175.9, while the ARIMA peaked at 89.8% accuracy with an MAE of 462.1. A particularly interesting fact from this study is that the LSTM trained solely on the raw historical price series outperformed an LSTM enriched with multiple technical indicators. This counter-intuitive result suggests that for such powerful models, additional features, which are often derivatives of the price series itself, can introduce noise and degrade performance rather than improve it. This highlights the ability of RNN models to extract complex patterns directly from raw data, questioning decades of feature engineering practices.

Other research focuses on the hybridization of recurrent architectures themselves. A 2025 study proposed a hybrid LSTM-GRU model (Ma et al., 2025). The architecture uses GRU layers first for their computational efficiency and their ability to capture shorter-term dependencies, followed by LSTM layers to refine the modeling of longer-term dependencies. Applied to a stock market with a set of 42 technical indicators, this hybrid model showed a 3% improvement in mean squared error (MSE) compared to previous approaches.

Research in this area goes further, even modifying the internal components of the cells. In 2025, Liu et al. introduced the MCI-GRU (Multi-Head Cross-Attention and Improved GRU) model (Liu et al., 2025). The researchers replace the traditional GRU's reset gate with an attention mechanism, which allows the model to dynamically assign weights according to the importance of past states. This innovation, in particular, allowed the model to surpass the state-of-the-art on the US and Chinese markets. Thus, research is no longer content with just stacking layers, but also seeks to improve existing architectures.

2.4.2 Convolutional Neural Networks

Initially designed for image processing, Convolutional Neural Networks (CNNs) have quickly gained popularity for time series analysis. For a 1D time series, convolutional filters are applied along the sequence to detect local patterns, in the same way they detect edges or textures in a 2D image. The advantage of CNNs is their efficiency in extracting these local features. However, their main limitation is their local receptive field, which makes them less suitable for modeling very long-term dependencies (Zhou et al., 2021).

Faced with this limitation, recent research uses CNNs not as standalone prediction models, but as feature extractors within hybrid architectures. A particularly popular approach is the CNN-Transformer hybrid, presented in several 2025 papers (Zhou et al., 2021). In this model, a CNN first processes the input sequence by identifying local patterns and transforming the raw sequence into a sequence of more abstract feature vectors. This output is then passed to a Transformer, which uses its global attention mechanism to model longer-term dependencies. Applied to intraday prediction on the S&P 500, this hybrid architecture has demonstrated its

superiority over statistical models and pure DL methods.

Another approach is undertaken by Wang et al. in 2023, who focus not on the model's architecture, but on the data representation. They propose the QGAF (Quantum Gramian Angular Field), a method consisting of using quantum computing principles to transform a 1D time series into a 2D image. A standard 2D CNN can then be trained on these images for classification or prediction. The results demonstrate performance well beyond classic image transformation methods. This technique suggests that a large part of future performance gains may come not from new architectures, but rather from the representation of financial data for existing models.

2.4.3 Transformer Models

The Transformer architecture, initially introduced for machine translation, has caused a revolution in many fields (Vaswani et al., 2017). Unlike RNNs that process data sequentially, which can lead to information loss over long periods, the self-attention of transformers allows for weighting the importance of each data point in a sequence relative to all other points in the same sequence. This ability to create direct connections between distant time points allows for very effective capture of long-range dependencies and offers the additional advantage of massive parallelization of training (Wen et al., 2025; Liu et al., 2025).

The flexibility and power of the Transformer have led to its rapid adoption, and it is becoming the new baseline architecture for deep learning in finance, gradually replacing RNNs (Sezer & Ozbayoglu, 2022; Wang & Li, 2025). The influence of the Transformer is such that it even extends to competing architectures, as shown by the MCI-GRU model which integrates an attention mechanism (Liu et al., 2025). However, a question arises: is the original Transformer architecture, with its encoder and decoder, optimal for time series prediction? A 2025 study compared different variants and concluded that there is not yet a clear answer, suggesting that simpler architectures, like an autoencoder model, might be sufficient or even more performant (Wen et al., 2025).

Cutting-edge research leverages the power of the Transformer to tackle increasingly granular data. A 2025 study presented TLOB (Transformers Limit Order Book), a model specifically designed for high-frequency limit order book data (Berti et al., 2025). The LOB contains extremely detailed and valuable information about supply and demand at different price levels. TLOB uses a dual attention mechanism to capture both temporal dependencies (the evolution of the LOB over time) and spatial dependencies (the relationships between different price levels at a given moment). This architecture represents a major shift in the approach to market prediction, moving from estimating general trends to modeling the dynamics of market microstructure in real time.

In parallel, a new type of Transformer was proposed in 2023: multimodal fusion. The Modality-aware Transformer (MAT) (Li et al., 2023) is a model capable of predicting a time series (say, interest rates) by simultaneously using numerical data and textual data (such as Federal Reserve reports). The MAT has specialized attention mechanisms (intra-modal and inter-modal) to learn the relationships within each data type and between different data types. This type of model could be particularly relevant in market finance with its heterogeneous data.

2.4.4 Sentiment Analysis

While time series processing architectures are improving, a significant part of research is now focused on exploiting textual data to extract predictive information. The idea behind sentiment analysis is simple: the tone of financial news and public opinion influence investor behavior, and therefore market movements. Integrating this textual information into predictive models has

become an entire field of research in quantitative finance.

A paper in this field is by Kalamkar et al. (2022), who examined how sentiment contained in news can help predict stock market volatility. Rather than simply classifying news as positive or negative, the authors used a BERT model pre-trained on a financial corpus, named FinBERT, to assign a sentiment score to news headlines. By adding this score as an exogenous variable to classic prediction models like GARCH, they obtained significantly better results. Their work proves that financial texts are not just background noise and hold signals for determining market risk.

Financial media is one source of data, but social networks like Twitter offer real-time data and directly reflect the sentiment of retail investors, who are also more likely to give in to their emotions. Arpit Mittal & Anshul Goel, two scientists from Stanford, sought to establish a correlation between public sentiment and market sentiment. To do this, they used Twitter data to predict public mood and, by combining this information with the closing values of the Dow Jones Industrial Average (DJIA) from previous days, they sought to anticipate stock market movements. Their analysis, conducted on data spanning from June to December 2009, showed that a self-organizing fuzzy neural network (SOFNN) model could achieve an accuracy of 75.56%.

For their analysis, the researchers developed their own sentiment classification tool, identifying four mood categories: Calm, Happy, Alert, and Kind. They discovered, through a Granger causality analysis, that the Calm and Happy moods were the most predictive of DJIA values, with a lag of 3 to 4 days. The integration of these two mood dimensions, in addition to historical DJIA data, yielded the best prediction results, confirming that adding other mood dimensions risked causing model overfitting.

2.5 Part 3: Reinforcement Learning Approaches

2.5.1 Recalls

Reinforcement Learning (RL) is a branch of machine learning inspired by behavioral psychology. An agent learns to navigate a complex and stochastic environment through trial and error. The agent's goal is to learn a strategy, called a policy, to maximize the rewards obtained during its exploration. This paradigm is fundamentally different from supervised learning, where labels are provided, and unsupervised learning, which seeks hidden structures. Here, learning is active and exploratory, guided only by a reward signal, which is sometimes sparse and delayed.

This approach is particularly relevant for sequential decision-making problems and is especially well-suited to our task of algorithmic trading. Financial markets are dynamic, non-stationary, and highly competitive environments with many actors interacting in real time. RL offers the possibility of an agent capable of continuously adapting its strategy according to new market conditions, learning to identify new opportunities and manage risk autonomously.

The Conceptual Framework: Markov Decision Processes (MDPs)

To mathematically model this learning process, RL relies on the framework of Markov Decision Processes (MDPs). This framework is more than just a formalism; it provides a precise language to break down the problem into manageable components:

- **State (S):** This is a snapshot of the environment, a complete representation of everything relevant for decision-making. In the context of trading, a state can range from a simple series of closing prices to a complex multidimensional construct including technical indicators (RSI, MACD, Bollinger Bands), order book data (market depth), or even

unstructured data like financial news sentiment. The major challenge is to design a state that is both informative enough to capture market dynamics and concise enough to avoid the curse of dimensionality.

- **Action (A):** This is the set of decisions the agent can make. The action space can be discrete (e.g., three actions: buy, sell, hold) or continuous. A continuous action space is more realistic for portfolio management, where the action could be allocate 25.7% of capital to this asset, offering much finer granularity but significantly increasing the complexity of the problem.
- **Reward (R):** This is the feedback signal that guides learning. The design of the reward function is one of the most difficult aspects. A simple reward based on profit and loss (P&L) can encourage excessively risky strategies. More sophisticated functions incorporate risk measures, such as the Sharpe ratio or the Sortino ratio, which penalize the volatility of returns. The agent thus learns to seek not the maximum profit at any cost, but the best risk-adjusted return.
- **Policy (π):** This corresponds directly to the agent's strategy: a policy maps a state to an action. It can be deterministic (for a given state, it always chooses the same action) or stochastic (it provides a probability distribution over possible actions). Stochastic policies are often preferred because they facilitate the exploration of new strategies.

The Bellman Principle and Fundamental Approaches

The theoretical foundation for solving MDPs is the Bellman equation (Bellman, 1957). Its intuition is deep and elegant: the value of being in a given situation is the sum of the immediate reward one can obtain and the average value of the future situations one can reach. It is this principle that allows a complex long-term planning problem to be broken down into a series of local and iterative decisions. When the rules of the environment are perfectly known (which is never the case in finance), algorithms like Value Iteration and Policy Iteration can calculate the optimal strategy. Their inability to function without a perfect model of the environment motivates the shift to model-free learning methods.

Monte Carlo and Temporal Difference Learning

These algorithms address this problem by generating countless episodes, i.e., sequences of decisions by the agent up to a terminal state, which allows for the approximation of state values.

- **Monte Carlo Methods:** These are the most trivial methods. The agent plays episodes and then analyzes the results to see which sequences of actions led to the best scores. Their main drawback is high variance: a good final result may be due to a single stroke of luck in the midst of a series of bad decisions, but the method will attribute credit to all actions in the sequence.
- **Temporal Difference (TD) Methods:** These represent a major advance. Instead of waiting for the end of an episode, the agent updates its strategy after each time step. It does this by comparing its prediction of a state's value with the reward it just received and its new prediction of the next state's value. This is learning by bootstrapping: improving one estimate from another estimate.
- **Q-Learning (Watkins, 1989):** This is the reference algorithm, said to be off-policy. It learns the optimal value of actions as if it were following the best possible policy, even if, for exploration, it performs non-optimal actions. It's like an amateur chess player learning a grandmaster's strategy by analyzing their games, while continuing to make their own

mistakes while playing. This aggressive approach allows it to converge to the true optimal policy.

- **SARSA (Rummery & Niranjan, 1994):** This algorithm is on-policy, meaning it learns the value of actions based on the policy it is currently following. This results in a more conservative agent, learning from the consequences of its own choices.

Deep Reinforcement Learning (Deep RL)

The rise of RL occurred notably with the application of deep learning to this field. The latter enabled the resolution of environments with continuous action spaces and high-dimensional state spaces, where classical RL approaches failed due to complexity. Among these deep RL techniques are:

- **Deep Q-Network (DQN) (Mnih et al., 2015):** This is one of the pioneering papers of deep RL. To stabilize learning, it introduces two mechanisms: the Replay Buffer and the target network.
- **Actor-Critic Methods (Konda & Tsitsiklis, 2000):** Composed of two estimators: an actor (the neural network representing the policy and choosing actions) and a critic (a second network that evaluates the actor's actions).
- **DDPG (Deep Deterministic Policy Gradient) (Lillicrap et al., 2015):** It merges the ideas of DQN with an actor-critic architecture. The actor produces a precise (deterministic) action rather than a probability distribution.
- **A2C (Advantage Actor-Critic) (Mnih et al., 2016):** A variant of Actor-Critic, this algorithm introduces the use of the advantage function. The critic calculates not the raw quality of an action, but its advantage: how much better this action was than the average expected action in that state.
- **PPO (Proximal Policy Optimization) (Schulman et al., 2017):** This is the most widely used method to date. The main idea is that the primary risk in learning is an update that is too large and completely breaks a good learned strategy. PPO solves this by ensuring that the new policy does not stray too far from the old one via a clipping mechanism.
- **SAC (Soft Actor-Critic) (Haarnoja et al., 2018):** One of the most advanced and state-of-the-art models currently. Its innovation is to integrate entropy directly into its objective. The agent is rewarded not only for maximizing its gains but also for maintaining a policy that is as random (high entropy) as possible.

2.5.2 Applications of Reinforcement Learning to Trading

The application of RL to algorithmic trading has seen rapid growth in recent years, driven by the successes of RL in other domains (Go, video games, robotics, LLMs) and by the availability of massive market data and computing power. The prize: a reinforcement-trained trading agent can theoretically learn lucrative strategies on its own by discovering patterns in financial data.

Pioneers and Classic Approaches Applied to Trading

The first experiments with RL in finance date back to the late 90s. For example, Moody & Saffell (1998) proposed a trading agent that directly optimized performance measures via a method called Recurrent Reinforcement Learning, and compared it to a basic Q-learning on

stock market data. Their RL agent already demonstrated the ability to exceed the performance of the S&P 500 index over a long period, showing the presence of exploitable predictive structure in prices. However, at that time, extracting a signal was much simpler than today, as there were fewer trading agents in the market. Other works from the same period explored variants: Neuneier (1996) applied multi-objective RL, and researchers like Dempster & Leemans (2006) used dynamic programming methods for FX (foreign exchange) trading. Nevertheless, these approaches remained limited by the available computing power and the lack of generalization via deep networks. In trading, a state can be represented by a vector of technical features (technical indicators calculated on prices, volumes, etc.) or even directly by price sequences (via LSTM, CNN, Transformer networks...). DQN for trading has been widely exploited; for example, Lucarelli et al. (2019) used DQN for cryptocurrency trading, with a reward based on the Sharpe Ratio.

Modern Advances (Deep RL)

Recent academic research is replete with applications of Deep Reinforcement Learning to financial markets, addressing various issues: portfolio allocation, market making, optimal order execution, or simple directional trading strategy on an asset.

- **Asset Trading:** A notable study is that of Jiang et al. (2017), who proposed a deep RL agent for portfolio trading: combining an LSTM network for the state and an actor-critic agent, they showed an outperformance of the agent compared to static benchmark strategies on crypto portfolios.
- **Market Making & High Frequency:** Market making (providing continuous buy and sell offers to capture the spread) is a well-studied area where RL applies well because it is a sequential problem by nature, with a subtle trade-off between profit (via the spread) and risk (inventory, price fluctuations). Classic market making approaches are beginning to be challenged by Reinforcement Learning approaches. Spooner et al. (2018) explored an RL agent for multi-asset market making. More recently, Kumar (2022) proposes a Deep Recurrent Q-Network (DRQN) agent for high-frequency market making trading. The agent uses an LSTM coupled with a DQN (to handle temporal dependence and partial observability of the order book) and interacts with a realistic market simulation. The results show that this agent surpasses a benchmark strategy from the field (based on temporal-difference) and reproduces realistic results observed on market data.
- **Optimal Execution:** Another area in finance where RL has proven its worth is in position execution. Optimal execution consists of breaking down a large order (e.g., selling 1 million shares) into a series of small transactions to minimize market impact and costs. Classic methods exist (stochastic control), but they assume simplified models. Researchers have formulated this as an MDP: the state can be the remaining part to sell and market conditions, the action the quantity to sell in the next interval. Nevmyvaka et al. (2006) were the first to take an interest in this field of application using Q-learning for order execution, and in 2021, Ning et al. applied double DQN to improve execution based on the order book.

Industrial Contributions

On the side of financial institutions, the adoption of RL has been more discreet in recent years. We see hedge funds and banks testing these methods, often internally. For example, J.P. Morgan has explored RL for managing the risk of derivatives, and Fidelity Investments has shown interest in RL for optimizing strategies. In 2022, Igor Halperin (a former quant at JPMorgan) mentioned working on RL techniques for asset allocation. The large high-frequency

trading firms keep their approaches secret, but we know from job offers and conferences that Citadel, Two Sigma, Jump Trading, and others are recruiting profiles skilled in implementing RL, a sign that they are experimenting in this area.

Regarding platforms, we are witnessing the emergence of dedicated frameworks aimed at encouraging reproducible research in Reinforcement Learning for Finance and facilitating experimentation for researchers and enthusiasts alike. Among the most notable:

- **FinRL** (Liu et al., 2020) is an open-source framework developed to standardize deep reinforcement learning approaches applied to finance.
- **TradeMaster** (Sun et al., 2023) extends this ambition by proposing a more modular platform, which emphasizes rigorous performance evaluation and comparison of algorithms in various scenarios.
- **TensorTrade** is another open-source initiative designed to train, test, and deploy trading agents via reinforcement learning.

These libraries play an essential role in democratizing RL in finance by lowering the technical barriers to experimentation, while promoting research and the reproducibility of results.

3 Methodology

The primary objective of this thesis is to develop a high-performance optimal execution algorithm. Before deploying complex models such as deep neural networks or reinforcement learning agents, a crucial preliminary step consists of identifying and validating the existence of predictive signals within the market's price data. Financial time series are notoriously noisy, and any robust strategy must be founded upon features that exhibit a statistically significant, even if weak, predictive power over future returns.

3.1 Data Description

This research is based on historical price and volume information for several major currency pairs on the Foreign Exchange (Forex) market, **covering the period from January 2022 to August 2025**. This high-frequency data is captured at a tick-by-tick level, often on a millisecond scale, providing a highly granular view of market activity.

However, working directly with data at such a high frequency presents significant challenges in terms of the required computational power and data storage. To create a dataset that is both tractable for analysis and sufficiently detailed to capture relevant micro-dynamics, we perform an aggregation step. The millisecond-level data is downsampled and structured into uniform **5-second intervals**.

Each of these 5-second intervals is then represented by a candlestick, a visualization that packs a significant amount of information into a single shape. As illustrated in Figure 1, a candlestick consists of a central body and two thin lines called wicks or shadows.

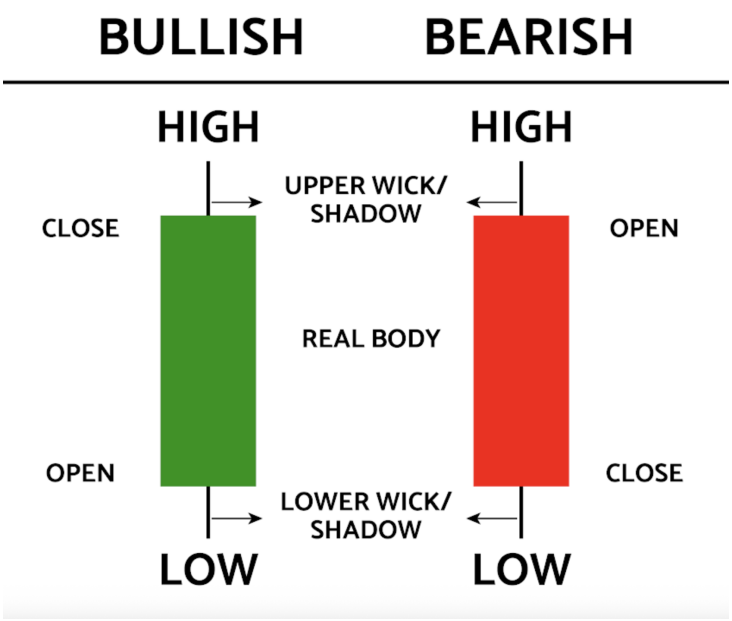


Figure 1: Structure of a bullish (green) and bearish (red) Japanese candlestick. The body represents the range between the open and close prices, while the wicks show the highest and lowest prices reached during the period.

The color of the body indicates the price direction: a green body signifies that the price closed higher than it opened, while a red body means the price closed lower than it opened. The five key data points encoded in each candlestick are:

- **Open (O):** The price of the first transaction at the beginning of the 5-second interval.
- **High (H):** The highest price reached at any point during the 5-second interval.
- **Low (L):** The lowest price reached at any point during the 5-second interval.
- **Close (C):** The price of the last transaction at the end of the 5-second interval.
- **Volume (V):** The total number of units of the asset traded during the 5-second interval.

While our dataset contains the full OHLCV information, for the initial feature engineering stages detailed in the following sections, our analysis is often focused on a single representative price series. In our work, this is referred to as the mid price, which serves as a proxy for the central tendency of the price within each interval. This simplification allows for the robust modeling of the underlying price dynamics.

Our analysis therefore began with the fundamental step of feature engineering. We started by exploring two of the most foundational and opposing hypotheses in quantitative trading: momentum (or drift) and mean reversion. This initial phase aimed to answer a simple question: can we construct simple, theory-driven features that have a demonstrable correlation with short-term future returns? This section details our first approach, which focused primarily on validating the mean reversion hypothesis, followed by an exploration of the momentum phenomenon.

3.2 Feature Engineering: A First Approach Based on Mean Reversion

3.2.1 Theoretical Framework of Mean Reversion

The concept of mean reversion supposes that asset prices, despite their random fluctuations, tend to oscillate around a long-term mean value. Any significant deviation from this mean is considered temporary and is expected to be corrected over time. This phenomenon is a foundation of statistical arbitrage, a class of strategies that employs statistical methods to identify and capitalize on such pricing contradictions (Ning, Chakraborty, & Lee, 2023).

Visually, mean reversion can be interpreted in two complementary ways. The first, more direct interpretation, applies to the price series itself. As shown in Figure 2, the price of an asset (blue line) often follows a general trend (purple line). Mean reversion strategies seek to exploit the temporary deviations from this trend. The red arrows illustrate these opportunities: when the price moves significantly above the trend, a trader might sell, anticipating a return downwards. Conversely, when the price drops well below the trend, a trader might buy, expecting a rebound.

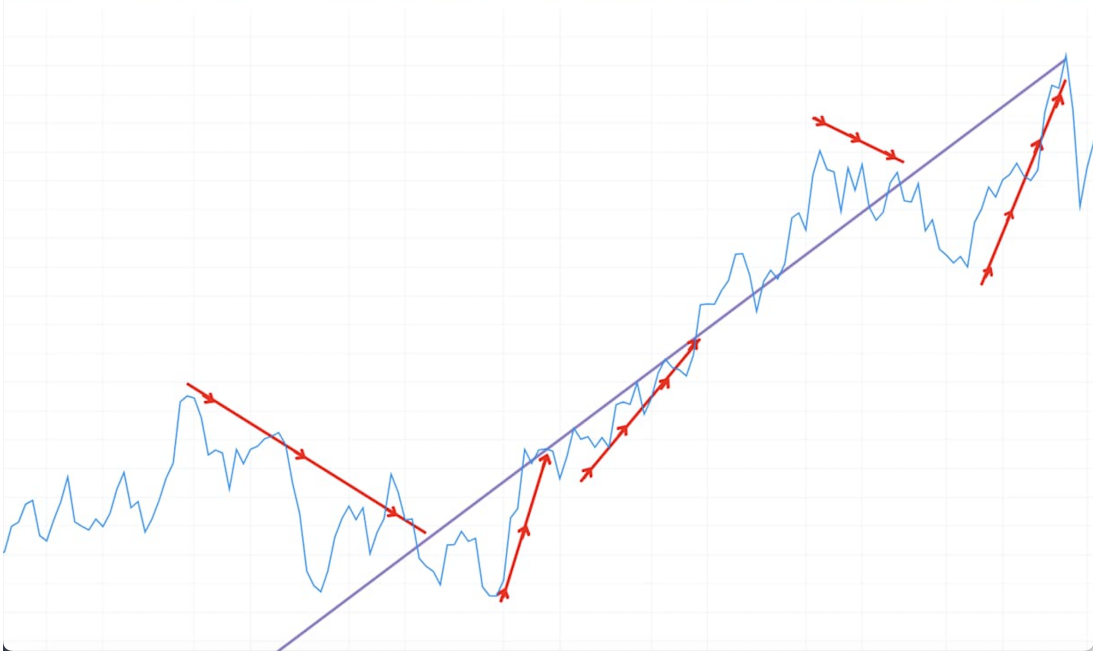


Figure 2: Illustration of mean reversion applied directly to a price series. The price oscillates around a central trend line, creating opportunities to trade the deviations. Source: City Index.

However, this approach has a major weakness: the trend itself is not stationary and can change, leading to false signals and this is why we focused on a second, more refined interpretation. We hypothesize that the price can be decomposed into two components: a non-stationary local trend and a stationary noise component, which represents the random fluctuations around this trend. Our main assumption is that it is this **noise** that is mean-reverting, not necessarily the price itself. This noise is a stochastic process whose mean is, by definition, centered at zero.

The figure 3 provides an illustration of such a stationary, zero-centered process. It displays a time series fluctuating over time around a constant mean. We can identify a central horizontal band representing this mean (zero, in the case of our noise component). The areas above and below are designated as Relative Highs and Relative Lows, respectively.

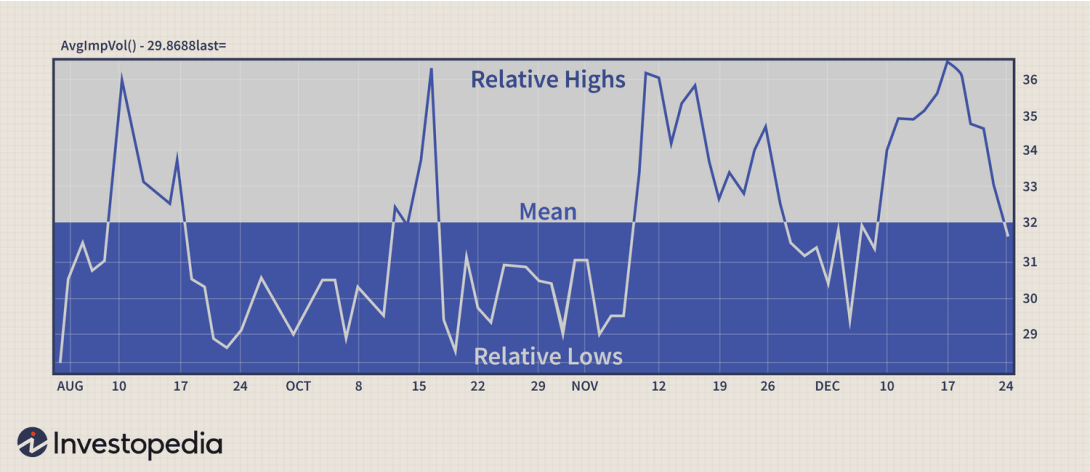


Figure 3: Visual representation of a mean-reverting process centered around a constant mean. The series oscillates between zones of relative highs and lows, with a tendency to return to the mean. This represents the behavior of the isolated noise component we seek to model. Source: Investopedia.

As illustrated, this isolated process does not drift away indefinitely. Instead, when it reaches the Relative Highs zone, it tends to subsequently fall back towards the mean (zero). Conversely, after dipping into the Relative Lows zone, it tends to rise back towards the mean. This gravitational pull towards a central value is the essence of the stationary mean reversion we aim to capture. Our strategy is therefore not to trade the price deviations from a visible trend, but to model and trade the deviations of this hidden noise component from its natural equilibrium of zero.

Mathematically, this behavior is often modeled as an Ornstein-Uhlenbeck (OU) process, which is the continuous-time analogue of a discrete autoregressive (AR(1)) process. The OU model is frequently used as a foundational assumption for developing optimal trading strategies in statistical arbitrage, as it provides a tractable mathematical description of the reverting behavior of this noise or spread (Ning, Chakraborty, & Lee, 2023). The value of this noise process, X_t, is described by the following stochastic differential equation:

$$dX_t = \theta(\mu - X_t)dt + \sigma dW_t$$

Where, in our specific case of modeling the noise component:

- μ is the long-term mean to which the process reverts. For our isolated noise component, we assume $\mu = 0$. This corresponds to the central line in Figure 3.
- θ is the rate or speed of reversion. A higher θ implies that the process returns to the mean more quickly after a deviation, indicating a stronger pull.
- σ is the volatility or magnitude of the random shocks that continuously push the process away from the mean.
- dW_t is a Wiener process, representing the random, unpredictable component of the movement.

The key component of this model is the drift term, $\theta(\mu - X_t)dt$. This term mathematically encodes the visual pull seen in the second figure. When the current value of the noise X_t is above its mean of zero (in the Relative Highs zone), the term becomes $-\theta X_t$, creating a negative drift that pushes it back down. Conversely, when X_t is below zero (in the Relative Lows zone), the term is positive, pushing it back up. This behavior is precisely what our mean-reversion strategy seeks to exploit.

3.2.2 A Personalized Approach: Dynamic Mean Reversion

The major challenge, therefore, is to effectively isolate this mean-reverting noise from the non-stationary price series. A major limitation of the classic Ornstein-Uhlenbeck model when applied directly to prices is the assumption of a static, long-term mean μ. On short time scales, the market does not revert to a fixed price level but rather follows a constantly evolving local trend. A price that seems high at 10:00 AM might be considered low at 10:05 AM if the market is in a strong uptrend. This highlights a critical point raised by Exley, Mehta, and Smith (2004): the frame of reference is paramount. What appears to be mean reversion in total returns might disappear when analyzing excess returns over a dynamic benchmark, underscoring the importance of defining the mean appropriately for the given context and timescale. Our approach directly tackles this by defining the mean as a dynamic, local trend. We hypothesize that the price reverts not to a constant level, but to this short-term trend line, which acts as a local fair value reference. By doing so, we can define the noise as the deviation from this trend. This aligns with the practical challenge in statistical arbitrage, where traders often model the spread between two co-moving assets, which by nature exhibits mean reversion around a dynamic, rather than a fixed, equilibrium (Ning, Chakraborty, & Lee, 2023).

1. **Defining the Local Trend:** At any given time t, we consider a look-back window of the last N price points, $\{P_{t-N+1}, ..., P_t\}$. We fit a simple linear regression model on these points against a time index $i \in [1, N]$. This gives us our local trend line, which is the line that best fits the data in this window:

$$\hat{P}_i = \beta_0 + \beta_1 \cdot i$$

The coefficients β_0 (intercept) and β_1 (slope) are determined using the Ordinary Least Squares (OLS) method. The principle of OLS is to find the values of β_0 and β_1 that minimize the Sum of Squared Errors between the actual prices P_i and the predicted prices $\hat{P}_i$ within the window. The general formula for the slope derived from this minimization is:

$$\beta_1 = \frac{N \sum_{i=1}^{N}(i \cdot P_i) - (\sum_{i=1}^{N} i)(\sum_{i=1}^{N} P_i)}{N \sum_{i=1}^{N} i^2 - (\sum_{i=1}^{N} i)^2}$$

This expression is equivalent to the more intuitive statistical formulation in terms of covariance and variance :

$$\beta_1 = \frac{\text{Cov}(P, t)}{\text{Var}(t)}$$

Where $\text{Cov}(P, t)$ is the covariance between the price series and the time index within the window, and $\text{Var}(t)$ is the variance of the time index. The intercept β_0 is then calculated to ensure that the line passes through the center of mass of the data points $(\bar{t}, \bar{P})$:

$$\beta_0 = \bar{P} - \beta_1 \bar{t}$$

Here, $\bar{P}$ and $\bar{t}$ represent the mean of the price and the time index over the window, respectively.

Adaptation to a Local Context: The key to our dynamic approach is that these formulas are not applied once to the entire dataset. Instead, they are adapted to the locality by being computed continuously within a sliding window of size N. At each new time step, the window slides forward by one point, and the coefficients β_0 and β_1 are recalculated using only the N most recent data points. This process generates a time-varying series of coefficients, $\beta_{0,t}$ and $\beta_{1,t}$, which in turn defines a new local trend line that is constantly updated to reflect the most immediate market behavior.

Figure 4 provides a clear illustration of this concept, applied directly to our dataset for the asset FGBLc1. The regression line (shown in red and extended in green) represents our dynamic estimate of the asset's fair value path, fitted to the most recent segment of the price data.

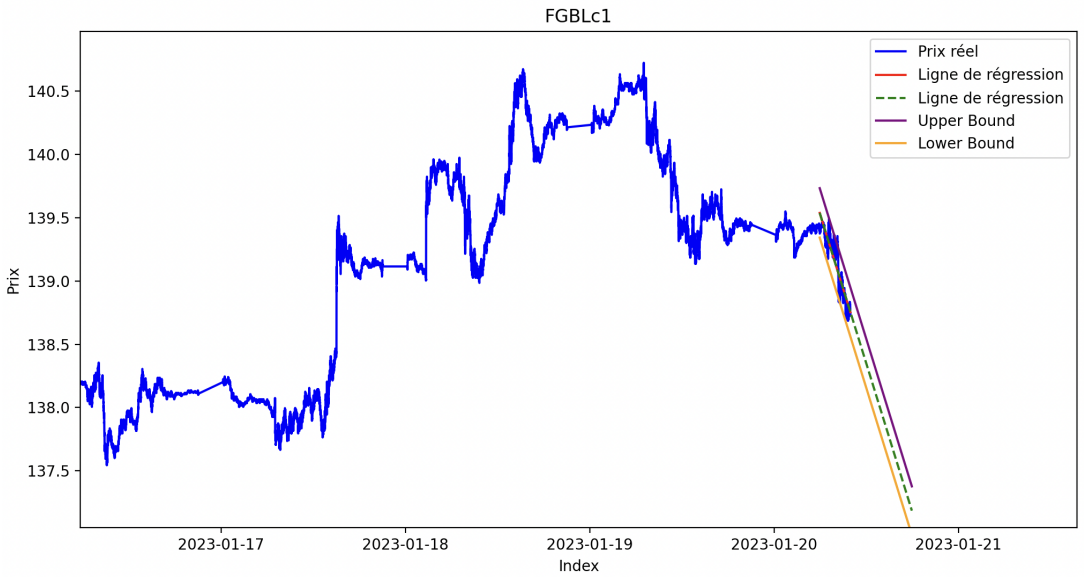


Figure 4: Example of a local linear regression fitted to the latest price data for the asset FGBLc1. The regression line (red/green) serves as the dynamic 'fair value', while the Upper (purple) and Lower (orange) bounds represent statistical fluctuation channels around this trend.

In addition to this central trend line, we construct statistical channels, an Upper Bound and a Lower Bound. As shown in the figure, these bounds define a region of expected fluctuation. They are calculated by taking the value on the trend line ($\hat{P}_t$) and adding or subtracting a multiple (k) of the price's recent volatility. This volatility, σ_N, is defined as the standard deviation of the price over the same look-back window N.

$$\text{Upper Bound}_t = \hat{P}_t + k \cdot \sigma_N$$

$$\text{Lower Bound}_t = \hat{P}_t - k \cdot \sigma_N$$

For the multiplier, we choose a standard value of $k = 2$. This choice is inspired by the properties of the normal distribution, where approximately 95% of data lies within two standard deviations of the mean. However, as financial price distributions are known to exhibit fat tails (leptokurtosis), this 95% figure should be interpreted as a heuristic for what constitutes a significant deviation, rather than a strict probabilistic guarantee. A price moving beyond these bounds is therefore considered statistically overextended.

2. **Calculating the Deviation (Noise):** The central part of our feature is the deviation, or noise, of the current price from this established local trend. We define our mean-reversion feature at time t, let's call it MR_t, as the residual of the last point from the regression line:

$$MR_t = P_t - \hat{P}_t = P_t - (\beta_0 + \beta_1 \cdot N)$$

This value, MR_t, quantifies how stretched or overextended the current price is relative to its own recent behavior.

3. **Transforming Noise into a Robust Oscillator:** The raw noise feature, MR_t, while useful, is unbounded and highly sensitive to outlier events. To create a more stable and normalized feature suitable for a machine learning model, we perform two subsequent processing steps:

   (a) **Defining a Robust Noise Channel:** To mitigate the impact of extreme outliers, we first apply a filtering step. At each time t, from the set of the last N noise values, $S_t = \{MR_{t-N+1}, ..., MR_t\}$, we discard the 10% of values that have the largest absolute magnitude. From the remaining 90% of the data points, which we denote as the filtered set S'_t, we then compute the minimum and maximum to serve as robust channel boundaries.

   $$\text{NoiseMin}_t = \min(S'_t) \quad \text{and} \quad \text{NoiseMax}_t = \max(S'_t)$$

   (b) **Normalizing the Noise into an Oscillator:** Finally, we normalize the most recent noise value (MR_t) within this robust channel to create a bounded oscillator, which we call NRP_t (Noise Range Percentage). This feature is analogous to a Stochastic Oscillator and indicates the relative position of the noise within its typical range.

   $$NRP_t = \frac{MR_t - \text{NoiseMin}_t}{\text{NoiseMax}_t - \text{NoiseMin}_t} \times 100$$

   This final feature, NRP_t, provides a consistent, bounded signal (mostly between 0 and 100) that is resilient to outliers.

4. **Formulating the Trading Hypothesis:** Based on this final, refined feature, our trading hypothesis becomes a direct application of the mean-reversion principle on the normalized noise component:

   - If NRP_t is high (approaching 100), the noise is at the top of its typical, robust range. We expect the noise to revert downwards towards its mean of zero, which implies a negative future return for the price itself.
   - If NRP_t is low (approaching 0), the noise is at the bottom of its typical, robust range. We expect the noise to revert upwards towards its mean of zero, which implies a positive future return for the price.

   The expected relationship between our feature NRP_t and the future return is therefore negative. This robust, inverse correlation is precisely the predictive signal our machine learning models will be trained to detect and exploit. This oscillator is designed to capture these temporary market inefficiencies in a more stable and reliable manner than the raw noise signal.

To visually summarize the entire feature engineering process, the following figures illustrate the transformation from raw price data to the final normalized oscillator for a look-back period of N=720.

Figure 5 shows the result of steps 2 and 3a. The top panel displays the raw noise component (MR_t) oscillating around its natural mean of zero. Superimposed are the robust upper and lower bounds, which form a dynamic channel that is resilient to outliers. The bottom panel shows the original price series, providing context for the noise's fluctuations.

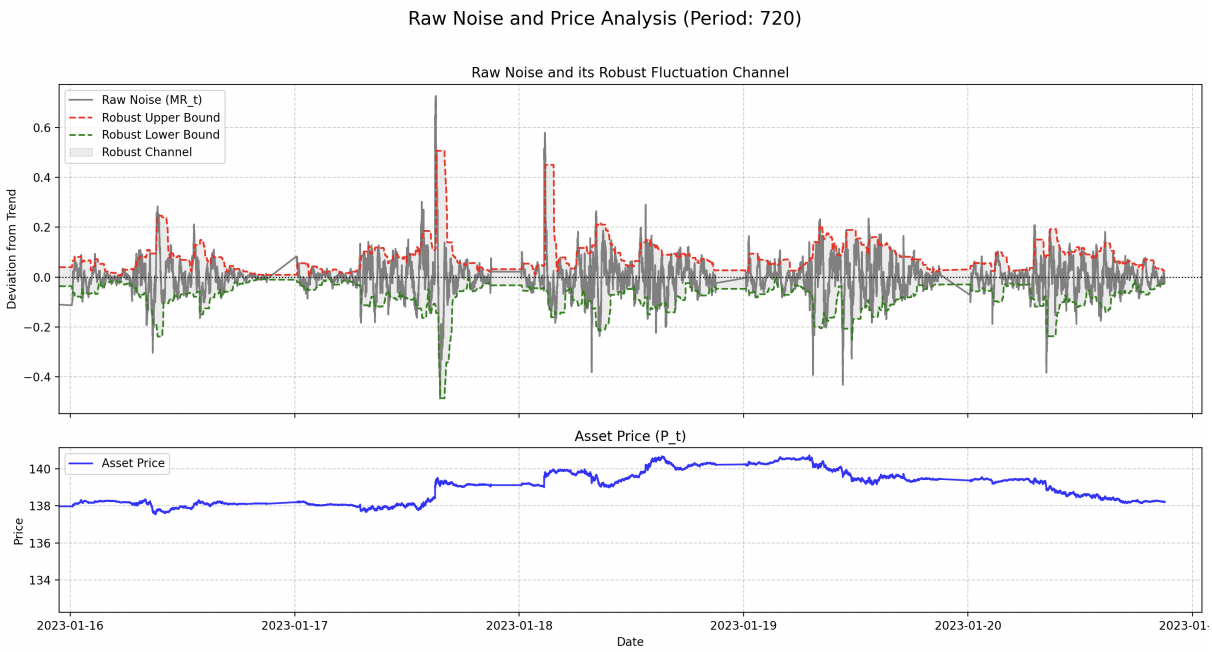


Figure 5: Visualization of the raw noise component and its robust channel. The top panel shows the noise (MR_t) and its calculated bounds. The bottom panel shows the corresponding asset price (P_t).

Finally, Figure 6 displays the final output of our process: the normalized oscillator (NRP_t) from step 3b. This feature is the direct result of normalizing the raw noise from the previous figure within its robust channel.

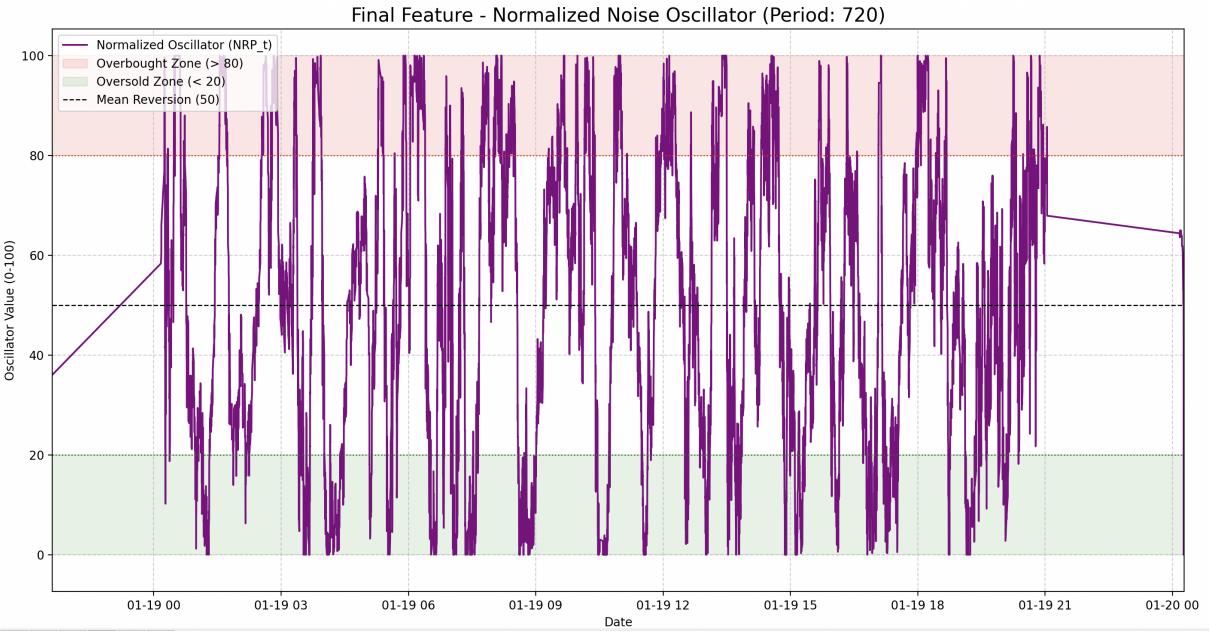


Figure 6: The final Normalized Noise Oscillator (NRP_t). The feature oscillates between 0 and 100, with the red and green zones indicating overbought and oversold conditions, respectively.

The graph clearly shows the bounded nature of the oscillator, which fluctuates between 0 and 100. The shaded areas highlight the overbought zone (above 80) and the oversold zone (below 20), where mean-reversion trading signals are generated, directly aligning with our trading

hypothesis.

3.3 An Adaptive Approach: The Adaptive Trend Finder (ATF)

The mean-reversion features developed previously rely on a fixed look-back period, N. While effective, this approach assumes that the market's memory is constant. In reality, the trend of an asset can unfold over short or long periods, and a model with fixed period may fail to capture the most relevant dynamic. In reality, an asset's trend can unfold over various timescales, and a fixed-period model may fail to capture the most relevant dynamic. To overcome this, we developed the Adaptive Trend Finder (ATF), a method designed to dynamically identify the most statistically significant trend at any given moment.

The main principle of the ATF is to test a wide range of look-back periods and select the one where the price trajectory is most accurately described by a log-linear regression model. The best fit is determined by the period yielding the highest Pearson correlation coefficient, ensuring we capture the most reliable ongoing trend.

3.3.1 Key Methodological Enhancements

The ATF's mathematical engine builds upon the linear regression framework established previously but introduces three main enhancements to enable its adaptive capabilities:

1. **Log-Linear Modeling:** Instead of modeling raw prices, the ATF operates on the logarithm of the price series, $y_t = \ln(P_t)$. This standard technique converts exponential trends into a linear form, allowing for a more robust regression fit. The same Ordinary Least Squares (OLS) method described earlier is then applied to fit the model:

$$y_t = \beta_0 + \beta_1 \cdot t + \epsilon_t$$

2. **Adaptive Selection via Pearson's R:** To objectively determine the best trend among the various periods tested, we introduce a goodness-of-fit criterion: the Pearson Correlation Coefficient, R. This coefficient measures the strength of the linear relationship between the actual log-prices (y_t) and the values predicted by the model ($\hat{y}_t = \beta_0 + \beta_1 \cdot t$). The formula adapted to our case is:

$$R = \frac{\sum_{t=1}^{N}(y_t - \bar{y})(\hat{y}_t - \bar{\hat{y}})}{\sqrt{\sum_{t=1}^{N}(y_t - \bar{y})^2 \cdot \sum_{t=1}^{N}(\hat{y}_t - \bar{\hat{y}})^2}}$$

   Where $\bar{y}$ and $\bar{\hat{y}}$ are the respective mean values of the actual and predicted log-prices over the window N. The period yielding the highest correlation is selected as the most statistically significant trend.

3. **A Refined Volatility Measure:** The ATF employs a more precise measure of volatility. Instead of using the standard deviation of the price itself (as in the first approach's channels), we calculate the standard deviation of the residuals (σ_ϵ) from the log-linear regression. This quantifies the volatility around the identified trend, not the overall market volatility.

$$\sigma_\epsilon = \sqrt{\frac{1}{N-1}\sum_{t=1}^{N}(y_t - \hat{y}_t)^2}$$

3.3.2 The Adaptive Selection Process and Channel Construction

The adaptive trend finder function manage this process to identify the optimal trend and build features around it.

1. **Iterative Search:** The function iterates through a predefined list of potential trend periods (e.g., 60, 180, 720, 2880 ticks). For each period, it calls the mathematical engine described above to compute its statistical properties.

2. **Selection of the Best Period:** The period that maximizes the Pearson correlation coefficient (R) is designated as the best period. This represents the time scale over which the asset's price evolution has been the most consistent and linear.

   Figure 7 provides a visual snapshot of this selection process at a single point in time. For the given Analysis Point, several regression lines corresponding to different look-back periods are computed. We can visually assess that the shorter-term trends (N=60) and longer-term trends (N=2880) do not fit the most recent price action as well as the intermediate trend. In this specific case, the ATF identified the regression over N=180 (shown in red) as the "BEST", as it offered the highest correlation with the recent price movement.

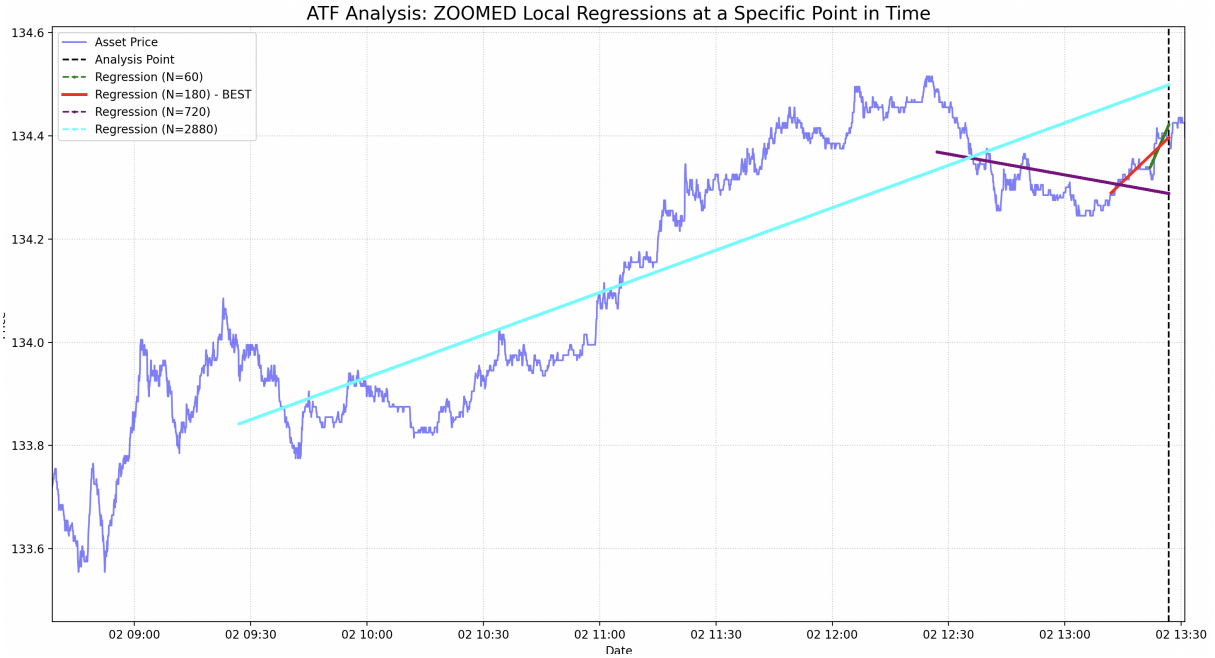


Figure 7: Visualization of the ATF's selection process. At the "Analysis Point" (dashed line), multiple regressions are tested. The ATF selects the one with the best statistical fit (N=180, red line) as the most representative local trend.

3. **Construction of Statistical Channels:** Once the best period (N_{best}) and its parameters are identified, we construct statistical channels around the trend line. These are projected boundaries based on volatility (σ_ϵ).

$$\hat{P}_{N_{best}} = e^{\beta_0 + \beta_1 \cdot N_{best}}$$

$$\text{Upper Bound} = \hat{P}_{N_{best}} \cdot e^{k \cdot \sigma_\epsilon} \quad , \quad \text{Lower Bound} = \hat{P}_{N_{best}} \cdot e^{-k \cdot \sigma_\epsilon}$$

   These channels define a region of expected fluctuation. Figures 8 and 9 illustrate the output of this process. Figure 8 shows a case where the ATF identified a strong, well-defined downtrend over a long period (N=2880), characterized by a high R-squared value

(0.595) and a clear channel. In contrast, Figure 9 shows a much weaker, short-term trend (N=60) with a lower R-squared (0.404) that the ATF might identify during less directional market phases.

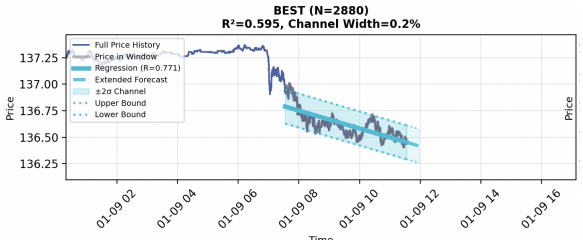


Figure 8: Example of a strong, long-term trend (N=2880) identified by the ATF. The price is well-contained within the statistical channel.

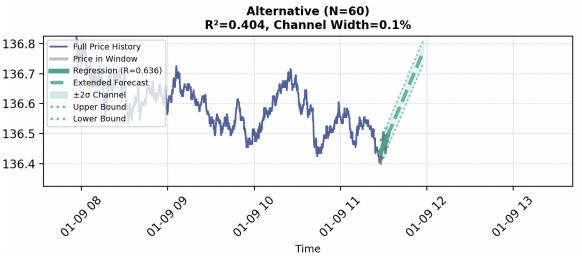


Figure 9: Example of a weaker, short-term trend (N=60) that could be identified in a different market context.

The final result is a hierarchical view of the market. As shown in Figure 10, the ATF identifies the dominant trend (N=2880, the wide cyan channel) that encompasses most of the price action. At the same time, it is aware of the alternative shorter-term trends (N=60, 180, 720) that are developing. This figure clearly shows these shorter-term trend forecasts (dashed lines) evolving within the boundaries of the main channel, illustrating the robustness of the adaptively-selected dominant trend.

Figure 10: Synthesis of the ATF analysis. The price evolves within the dominant channel (N=2880). The extensions of shorter-term trends (N=60, 180, 720) are shown to be mostly contained within this main channel.

3.3.3 From ATF Output to a Richer Feature Set

Beyond trend visualization, the ATF's primary strength is its function as an engine for dynamically generating a rich set of context-aware predictive features. At each time step, the optimal period (n^*) identified by the ATF provides a crucial piece of information about the market's current characteristic timescale, or memory. This information can be leveraged in two powerful ways:

- **Creating a Best Period Indicator:** The identity of the optimal period is in itself a valuable feature. We can transform this information into a format that a machine learning model can easily interpret, for instance by binarizing or one-hot encoding the set of all tested periods. This creates a feature that explicitly marks which timescale is currently

the most statistically relevant, allowing the model to learn different behaviors associated with different market regimes (e.g., short-term choppy vs. long-term trending).

- **Dynamic Parameterization of Technical Indicators:** More profoundly, the optimal period n^* can be used to dynamically calibrate a wide array of classic technical indicators. Instead of relying on fixed, arbitrary look-back periods (e.g., a 20-period Moving Average or a 14-period RSI), we can now compute these indicators using the market-derived best period. This allows us to generate a suite of adaptive features, such as:
  - Adaptive Moving Averages (MA, EMA) calculated over n^* periods.
  - Adaptive Bollinger Bands whose period and standard deviation are based on the ATF's regression over n^*.
  - Adaptive momentum oscillators like MACD or RSI, parameterized with n^* or fractions thereof.

This calibration step ensures our entire feature set is not based on static and preconceived notions of market timing, but is instead constantly calibrated to the most statistically relevant timescale identified by the market's own recent behavior. This provides our machine learning models with a much richer and more robust understanding of the current market context.

3.4 Feature Engineering: A Second Approach Based on Momentum

3.4.1 Theoretical Framework of Momentum

In stark contrast to mean reversion, the momentum hypothesis, often referred to as trend following or relative strength, is built on the idea that winners keep winning and losers keep losing. For someone unfamiliar with trading, this is akin to the concept of inertia in physics: an object in motion tends to stay in motion. In financial markets, this translates to the observation that stocks that have performed well in the recent past are likely to continue to perform well in the near future, while those that have performed poorly are likely to continue their downward trend.

Visually, the momentum phenomenon manifests as periods of trend continuation, as illustrated in Figure 11. After a series of declines (red candlesticks), the market shows a tendency to continue its downward trajectory. Conversely, after a period of gains (green candlesticks), the upward trend tends to persist. The figure also highlights a crucial point: these trends do not last forever and can undergo a trend reversal. The principal challenge for a momentum strategy is therefore not only to identify an existing trend but also to anticipate its continuation.

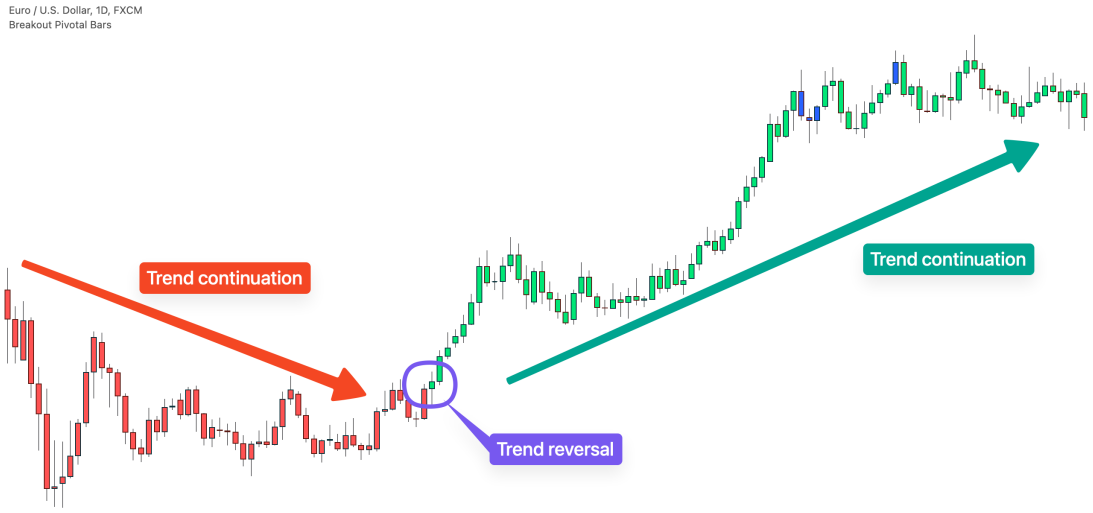


Figure 11: Visual illustration of the momentum phenomenon. The chart shows periods of Trend continuation both downwards (left) and upwards (right), separated by a Trend reversal.

This concept was empirically and rigorously documented in the seminal paper by Jegadeesh and Titman (1993), who demonstrated that strategies buying past winners and selling past losers generated significant positive returns. Their findings suggested that the market might underreact to firm-specific information, causing price adjustments to occur gradually over time rather than instantaneously, thus creating predictable trends. The robustness of this momentum effect was so profound that it quickly became recognized as a major market anomaly, one that could not be easily explained by traditional asset pricing models like the CAPM.

This led to the development of multi-factor models designed to better capture the sources of stock returns. Building directly on the work of Jegadeesh and Titman, Mark Carhart (1997) proposed a four-factor model that incorporated momentum as a fundamental risk factor, alongside the three factors previously identified by Fama and French (market risk, size, and value). The Carhart four-factor model explains the excess return of an asset with the following equation:

$$R_{it} - R_{ft} = \alpha_i + \beta_{i,MKT}(R_{Mt} - R_{ft}) + \beta_{i,SMB}SMB_t + \beta_{i,HML}HML_t + \beta_{i,MOM}MOM_t + \epsilon_{it}$$

Where:

- $R_{it} - R_{ft}$ is the excess return of asset i over the risk-free rate.
- α_i is the abnormal return, or alpha, of the asset.
- $(R_{Mt} - R_{ft})$ is the market risk premium.
- SMB_t (Small Minus Big) is the size factor, representing the excess return of small-cap stocks over large-cap stocks.
- HML_t (High Minus Low) is the value factor, representing the excess return of high book-to-market (value) stocks over low book-to-market (growth) stocks.
- **$\mathbf{MOM_t}$ (Momentum)** is the new factor introduced by Carhart. It is constructed monthly as the average return of the top 30% of stocks with the highest past returns minus the average return of the bottom 30% of stocks with the lowest past returns (often calculated over the prior 12 months, skipping the most recent month). This factor is also known as UMD (Up Minus Down) or PR1YR (Prior 1-Year return).

The inclusion of the MOM_t factor was a major breakthrough. Carhart (1997) showed that this factor alone could explain the hot hands phenomenon, or short-term persistence, observed in the performance of mutual funds. In essence, a stock's or a portfolio's sensitivity ($\beta_{i,MOM}$) to this momentum factor could account for a significant portion of its returns that were previously considered anomalous. This formalized momentum from a simple trading strategy into a systematic factor that captures a distinct dimension of risk and return in the market.

3.4.2 Constructing High-Frequency Momentum Features

To capture the directional bias, or drift, of the market on short time scales, we constructed a set of high-frequency momentum features. Unlike the classic long-term momentum factors found in academic literature, our indicators are designed to react quickly to the evolving market trend. Our methodology is based on three complementary approaches: return-based, regression-based, and oscillator-based momentum.

1. **Return-Based Momentum:** The most direct measure of momentum is the asset's past performance. We compute this as the cumulative return over a look-back window of size N. This feature, which we call return_backward, is defined as:

$$\text{Momentum}_{\text{ret},N}(t) = \frac{P_t - P_{t-N}}{P_{t-N}}$$

   Where P_t is the price at the current time t, and P_{t-N} is the price N periods in the past. A positive value indicates an upward trend over the window.

2. **Regression-Based Momentum (Slope):** A more robust measure of trend is the slope of the local regression line, which we already defined in our mean-reversion approach. The slope, $\beta_{1,t}$, represents the rate of price change over the window N and serves as a powerful drift indicator. A positive slope signifies a positive drift. It is calculated at each time step t over the preceding window of N points:

$$\beta_{1,t} = \frac{\text{Cov}(P, \text{time})}{\text{Var}(\text{time})}$$

A Note on Trend Quality: The R-Squared (R^2) Filter It is not enough to simply measure the slope of a trend; we must also assess its reliability. A market can have a positive slope over a period simply by chance, without a real underlying trend. To address this, we compute the Coefficient of Determination, or R-squared (R^2), for each local regression.

The R^2 is not a momentum indicator itself, but a crucial trend quality filter. It measures the proportion of the price's variance that is explained by the linear regression model. Its value ranges from 0 to 1.

$$R^2 = 1 - \frac{\sum_{i=1}^{N}(P_i - \hat{P}_i)^2}{\sum_{i=1}^{N}(P_i - \bar{P})^2} = 1 - \frac{\text{SS}_{\text{residual}}}{\text{SS}_{\text{total}}}$$

The purpose of the R^2 feature is to act as a confidence score for our regression-based indicators. A high R^2 (e.g., > 0.5) tells us that the trend is clear and linear, making the slope feature highly reliable. A low R^2 indicates that the price movement is noisy and non-linear, meaning the slope feature should be trusted less. In our final analysis framework, we use R^2 buckets to isolate conditions where the trend is most dependable.

3.4.3 Empirical Validation: Selecting the Best Indicators

To select the most predictive indicator for each market dynamic, we conducted a comprehensive correlation analysis. We computed the Pearson correlation coefficient between each candidate indicator (calculated over various fixed look-back periods) and the future return (calculated over various forward horizons). We also included the adaptive versions of these indicators, which are calibrated using the best period identified by the ATF. The results of this extensive analysis are synthesized in Figure 12.

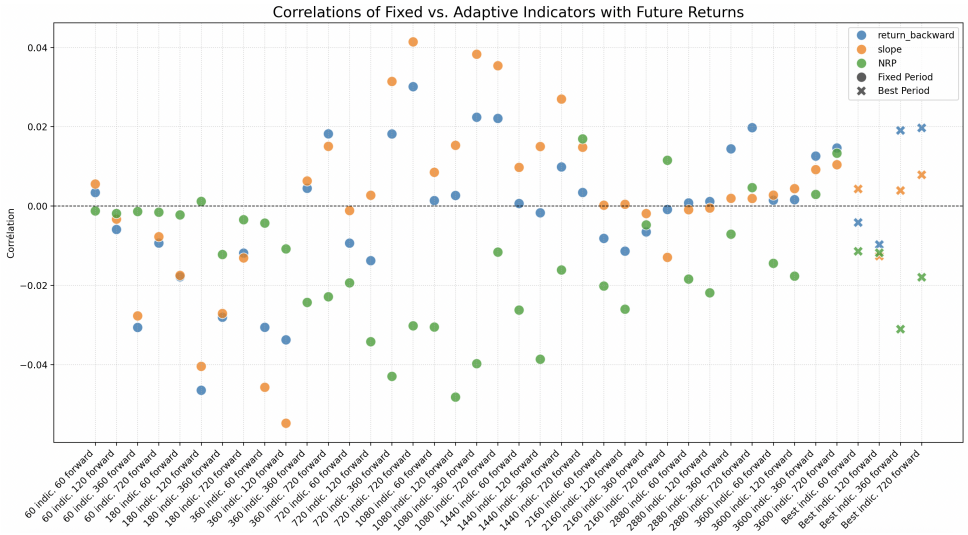


Figure 12: Correlations of Fixed vs. Adaptive Indicators with Future Returns. This scatter plot compares the predictive power of our candidate indicators, clearly showing the opposing nature of mean-reversion and momentum signals.

The analysis of this figure provides clear and compelling insights:

- **Confirmation of Theoretical Roles:** The results empirically validate the nature of our indicators. The momentum candidates, slope (orange dots) and return backward (blue dots), exhibit predominantly positive correlations, confirming that a positive past trend predicts a positive future return. Conversely, our mean-reversion candidate, NRP (green dots), shows an overwhelmingly negative correlation, confirming that a price that is too high relative to its mean tends to fall.
- **Identifying the Best Indicators:** The analysis highlights the most effective formulation for each concept. For **Momentum**, both the slope and return backward prove to be strong candidates, with the slope often reaching higher peaks of positive correlation. For **Mean Reversion**, the NRP provides a consistent and reliable negative correlation signal.
- **Robustness of the Adaptive Approach:** The adaptive indicators (crosses, labeled Best Period) demonstrate the value of our ATF methodology. The adaptive slope and return backward maintain a stable positive correlation, while the adaptive NRP maintains a stable negative one. This shows that the ATF successfully identifies relevant timescales, providing a robust signal that does not depend on a single, arbitrarily chosen fixed period.

3.5 Empirical Validation of Engineered Features

Once the features capturing mean reversion and momentum were engineered, the next step was to empirically validate their predictive power. A simple correlation analysis, while useful,

is often insufficient as it fails to capture the performance of signals under specific, interacting market conditions. To conduct a more granular and robust analysis, we designed a systematic framework to evaluate our hypotheses across a multi-dimensional state space. The goal is to move beyond asking does this feature work? to answering under which specific market conditions does this feature work best?.

3.5.1 Framework for Creating a Profitability Matrix

Our solution was to construct a comprehensive profitability matrix, defining a market context at any given time t by the simultaneous state of four key dimensions:

1. **The Mean-Reversion Signal:** The level of the noise oscillator, captured by its NRP value.
2. **The Noise Dynamic:** The immediate momentum of the noise, represented by its derivative (NRP_derivative), indicating whether the noise is currently increasing or decreasing.
3. **The Underlying Trend:** The direction of the local market trend, given by the sign of the regression slope (slope_binary).
4. **The Trend Quality:** The clarity or statistical significance of this trend, measured by its R-squared (R^2) value.

The validation process was then implemented through the following systematic steps:

1. State Space Discretization (Bucketing) To systematically analyze the interaction between these dimensions, the noise oscillator NRP and the trend quality R^2, were first discretized into categorical buckets. The NRP was segmented into twelve distinct states: one for values below 0%, ten for each 10-percentile increment from 0% to 100%, and one for values above 100%. A similar bucketing process was applied to the R^2 value. This transformation allows for the grouping of observations that fall within similar quantitative regimes.

2. Enumeration of Market Contexts A unique market context is defined as a specific combination of the discretized states of our four dimensions. For example, one such context could be formally described as: {NRP is in the >100% bucket, NRP_derivative is negative, slope_binary is positive, R^2 is in the 80%-90% bucket}. Our framework systematically enumerates every possible combination of these conditions that appears in the historical data.

3. Computation of Conditional Future Returns For each uniquely defined market context, the algorithm performs a full scan of the entire historical dataset. It identifies every timestamp t where this precise set of conditions was met. On this filtered subset of occurrences, we then compute the average future return over several predefined forward-looking horizons H (e.g., 60, 120, 360, 720, and 1440 ticks). The future return is calculated as:

$$\text{Mean Return}(H|\text{Context}) = \mathbb{E}\left[\frac{P_{t+H} - P_t}{P_t}\middle|\text{Context}_t\right]$$

This calculation is repeated for every defined context and every future horizon.

4. The Profitability Matrix The final output of this process is a comprehensive profitability matrix. This multi-dimensional table maps each specific market context to its historically observed average future profitability. This powerful analytical tool allows us to move beyond simple correlations and uncover the complex interplay between our engineered features, revealing the specific regimes where our theoretical hypotheses hold true, where they fail, and where they might even invert. The subsequent sections will present and analyze the results extracted from this matrix.

3.5.2 Refining the Signal: Calculating the Derivative of the Noise Oscillator

While the Noise Range Percentage (NRP) oscillator effectively identifies states of market overextension overbought (high NRP) and oversold (low NRP), it only provides a static snapshot of the market's condition. A high NRP value alone does not distinguish between a market that is still trending upwards and one that has just peaked and is beginning to reverse. To capture this crucial dynamic information and improve the timing of our signals, we introduce a second-order feature: the derivative, or momentum, of the NRP itself.

The Necessity of Smoothing Financial time series are inherently noisy. Directly computing the derivative of the raw NRP signal would result in a chaotic and unreliable indicator, prone to frequent sign changes due to minor fluctuations. To extract the underlying, meaningful dynamic of the NRP, a smoothing pre-processing step is essential. For this task, we employ the Savitzky-Golay filter, a sophisticated technique particularly well-suited for this purpose. The derivative is then calculated as the difference between the smoothed NRP value at time t and its value at a prior time step $t - k$. The lookback period k was chosen heuristically as $\lfloor \text{period}/10 \rfloor$, representing 10% of the analysis window's memory. This ensures that the comparison captures a significant change in the noise level, rather than short-term oscillations.

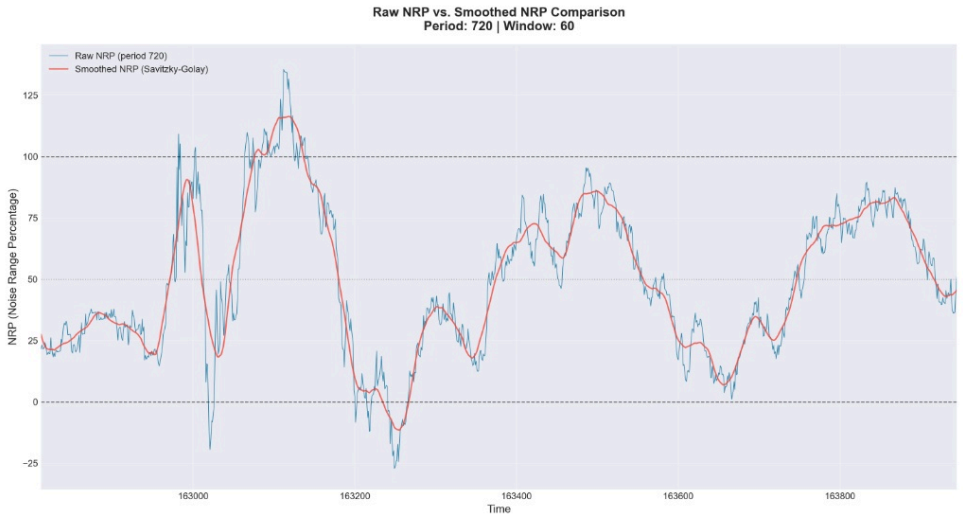


Figure 13: Comparison of the raw NRP and the Savitzky-Golay smoothed NRP. The smoothed signal (red) captures the underlying dynamic of the raw NRP (blue), reducing noise for a more reliable derivative calculation.

The Savitzky-Golay Filter Unlike a simple moving average which tends to flatten signal features, the Savitzky-Golay filter works by fitting a low-degree polynomial to successive subsets of the data points. The smoothed value for a given point is then obtained by evaluating this polynomial at the center of the window. This method excels at preserving the shape, height, and

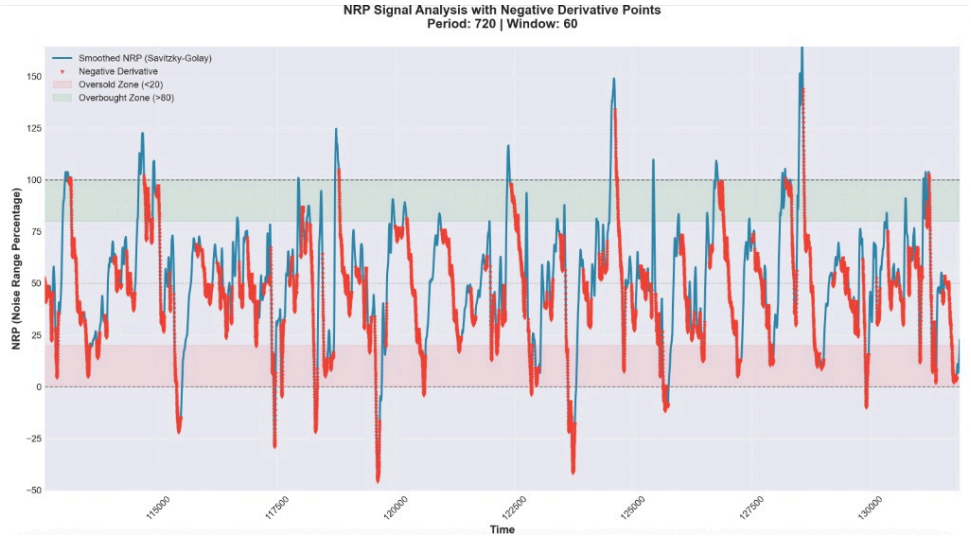


Figure 14: Identification of negative derivative points on the smoothed NRP signal. The red dots mark each moment where the oscillator's slope is negative, thereby capturing periods of noise decline. The oversold (<20%) and overbought (>80%) zones are also highlighted.

width of peaks and troughs in the signal, making it an ideal choice for the subsequent calculation of derivatives.

Mathematically, the Savitzky-Golay filter is a type of finite impulse response (FIR) filter. The calculation of each smoothed data point g_i is a linear combination of the surrounding raw data points f_{i+j}:

$$g_i = \sum_{j=-m}^{m} c_j f_{i+j}$$

Where f is the original signal (our NRP series), c_j are the filter coefficients, and the smoothing window consists of $2m + 1$ points. The coefficients c_j are pre-computed based on two key hyperparameters:

- **Window Length:** The number of consecutive data points used for each polynomial fit. In our implementation, we use a window length of 10.
- **Polynomial Order:** The degree of the polynomial to be fitted. We use a polyorder of 2, meaning a quadratic polynomial is used. This order is flexible enough to capture the curvature of peaks and troughs without overfitting to local noise.

Justification of a Non-Causal Approach for A Posteriori Analysis An essential aspect of the Savitzky-Golay filter, in its standard implementation, is its non-causal nature. To compute the smoothed value at time t, the filter uses a centered window that includes data points from the future (e.g., $t + 1, t + 2, ..., t + m$). In the context of creating a real-time predictive model for a trading simulation, the use of such a filter would be a fundamental methodological error, as it would introduce a lookahead bias.

However, the objective of our current analysis is different and specific. We are conducting an a posteriori empirical validation. The filter is not used to generate a tradable signal in a simulated environment, nor is it used to modify the underlying price data on which returns are calculated. Its sole purpose is to act as a labeling tool to retrospectively identify and filter for the data points of interest, specifically, those moments in the past where the underlying dynamic of the noise was positive or negative.

This process is conceptually equivalent to a manual, retrospective labeling. An analyst, looking at a complete historical chart, could identify the peaks and troughs and manually assign a negative derivative label to the points on the downward slope of a peak. The Savitzky-Golay filter merely automates this process of labeling with perfect hindsight, and arguably more consistently. One could even argue that a meticulous manual labeling would have been more effective, as it could have captured nuances that a fixed-parameter filter might miss. By using this non-causal tool, we are simply ensuring that our analysis of the profitability matrix is conditioned on the most accurate possible historical labels for the noise dynamic.

3.5.3 Analysis of the Profitability Matrix Results

To conduct a full and comparative analysis, the profitability matrix was constructed to display all forward-looking horizons simultaneously. For clarity and to facilitate a structured analysis, the results are split into two distinct tables. The first table focuses on **oversold and neutral conditions** (NRP Buckets 0, 1, and 5). The second table focuses on **conditions ranging from slightly overbought to extremely overbought** (NRP Buckets 6, 10, and 11). This separation allows for a detailed examination of the mean-reversion effect at both ends of the noise distribution, using the central buckets as a baseline.

Table 1: Profitability Matrix (Part 1): Oversold and Neutral Zones. Values are average returns in basis points (1 bps = 1% of 1).

Market Context		**NRP Bucket 0 (<0%)**					**NRP Bucket 1 (0-10%)**					**NRP Bucket 5 (40-50%)**				
Derivative	**Slope**	5m	10m	30m	1h	2h	5m	10m	30m	1h	2h	5m	10m	30m	1h	2h
-1	-1	0.15	0.57	0.78	1.17	1.21	0.04	0.25	0.25	0.21	0.27	-0.14	-0.22	-0.01	0.09	-0.11
	1	0.04	0.07	-0.39	-0.13	0.06	-0.17	-0.23	-0.31	-0.51	-0.22	-0.12	-0.14	-0.30	-0.49	-0.58
	All	0.04	0.19	-0.06	0.28	0.40	-0.13	-0.12	-0.20	-0.30	-0.09	-0.13	-0.17	-0.22	-0.29	-0.43
1	-1	0.58	0.73	2.09	2.69	1.91	0.83	0.96	1.24	1.21	0.82	0.19	0.18	0.23	0.05	-0.22
	1	0.83	0.38	-0.14	0.28	-1.21	0.25	0.18	-0.26	0.23	-0.14	0.16	0.07	-0.10	0.09	0.51
	All	0.67	0.72	1.70	2.42	1.61	0.48	0.51	0.37	0.60	0.27	0.18	0.14	0.10	0.07	0.10
All	-1	0.23	0.59	0.97	1.36	1.32	0.20	0.38	0.46	0.34	0.38	0.05	0.01	0.12	0.06	-0.17
	1	0.08	0.07	-0.39	-0.12	0.03	-0.10	-0.16	-0.26	-0.37	-0.18	-0.01	-0.05	-0.23	-0.28	-0.19
	All	0.10	0.21	0.05	0.39	0.46	-0.03	-0.03	-0.10	-0.16	-0.04	0.02	-0.03	-0.08	-0.13	-0.19

Table 2: Profitability Matrix (Part 2): Slightly Overbought to Extreme Overbought Zones. Values are average returns in basis points (bps).

Market Context		**NRP Bucket 6 (50-60%)**					**NRP Bucket 10 (90-100%)**					**NRP Bucket 11 (>100%)**				
Derivative	**Slope**	5m	10m	30m	1h	2h	5m	10m	30m	1h	2h	5m	10m	30m	1h	2h
-1	-1	-0.12	-0.10	-0.06	0.10	-0.01	-0.61	-0.75	-0.66	-0.88	-0.14	-0.99	-1.02	-0.89	-0.90	-2.07
	1	-0.11	-0.18	-0.46	-0.65	-0.70	-0.43	-0.35	-0.16	-0.89	-0.11	-0.78	-0.35	-0.20	-1.05	-0.88
	All	-0.11	-0.15	-0.30	-0.32	-0.40	-0.44	-0.41	-0.34	-0.90	-0.09	-0.79	-0.48	-0.36	-1.13	-1.09
1	-1	0.10	0.10	0.09	0.15	-0.06	-0.00	-0.03	-0.34	-0.50	-0.63	-0.15	-0.19	-0.22	-0.41	-0.65
	1	0.14	0.08	-0.03	-0.05	0.20	0.17	0.26	0.59	0.52	0.82	0.16	0.16	0.82	0.38	0.32
	All	0.11	0.10	0.06	0.07	0.03	0.06	0.07	-0.07	-0.24	-0.23	-0.02	-0.04	0.19	-0.07	-0.26
All	-1	0.01	0.03	0.03	0.12	-0.04	-0.07	-0.10	-0.37	-0.55	-0.55	-0.19	-0.25	-0.29	-0.51	-0.77
	1	-0.00	-0.07	-0.28	-0.40	-0.33	0.05	0.05	0.31	0.07	0.49	0.07	0.07	0.61	0.14	0.09
	All	0.01	-0.01	-0.10	-0.11	-0.17	-0.04	-0.03	-0.15	-0.37	-0.24	-0.10	-0.09	0.09	-0.21	-0.40

The analysis of these tables provides strong empirical support for our hypotheses regarding mean reversion, with the refined filtering leading to an even clearer and more potent predictive signal.

Confirmation of Mean Reversion in the Extremes The results validate the central premise that the predictive power of the NRP is concentrated in its extreme values.

- **Shorting Overbought Conditions (Table 2):** As hypothesized, extreme positive deviations of the noise are followed by negative returns. The columns for NRP Bucket 10 and 11 show consistently negative returns. The signal is now exceptionally strong, particularly for NRP Bucket 11 where, under a bearish trend (Slope = -1) and with a confirming negative derivative (Derivative = -1), the average return reaches a remarkable **-2.07 bps** at the H=1440 horizon.
- **Buying Oversold Conditions (Table 1):** Conversely, extreme negative deviations are followed by positive returns. The columns for NRP Bucket 0 show predominantly positive future returns. The peak profitability, found at the 720-tick horizon, is now even higher: buying an oversold market in a downtrend that has started to reverse (Slope=-1, Derivative=1) yields an impressive average return of **+2.69 bps**.

Analysis of the Central Buckets The central buckets, 5 (40-50%) and 6 (50-60%), serve as a crucial control group and illustrate the fine-grained nature of the mean-reversion effect.

- **The Neutral Zone (Bucket 5):** As shown in Table 1, the returns in this truly central bucket are consistently close to zero and show no clear, persistent pattern (e.g., the All row shows returns between +0.02 and -0.19 bps). This confirms that in the absence of a significant deviation, there is no predictive edge, which validates our approach.
- **The Tipping Point (Bucket 6):** Table 2 reveals a more subtle effect. In this bucket, which is just on the overbought side of the 50% median, we already see a slight but consistent negative bias emerge, particularly at longer horizons (the All row shows -0.10, -0.11, and -0.17 bps for H=360, 720, and 1440). This suggests that the gravitational pull of mean reversion begins to act immediately as the noise crosses its midpoint, even before it reaches extreme levels.

The NRP Derivative as a Powerful Confirmation Filter The refined results further accentuate the importance of the noise dynamic as a timing signal and this effect has more impact for the strong buy signals. Revisiting the case of NRP Bucket 0 and Slope -1 at the H=720 horizon in Table 1: the future return is **+2.69 bps** if the NRP has started to rise (Derivative = 1), more than double the **+1.17 bps** observed if the NRP is still falling (Derivative = -1). This amplified difference demonstrates that waiting for the noise to begin its reversion journey dramatically improves the signal's quality and profitability.

Effect of Time Horizon The presented tables allow us to directly observe how the predictive signals evolve over time. Generally, the magnitude of the predictable returns increases with the time horizon. The mean-reversion effect, while often immediate, seems to require a longer period to fully materialize and deliver its maximum profitability. For example, for the most profitable oversold conditions (NRP Bucket 0, Derivative 1), the peak returns are consistently found at the 360 and 720-tick horizons rather than at the shorter 60 and 120-tick horizons.

Conclusion In conclusion, this multi-dimensional and multi-horizon analysis, particularly after the application of a stricter filter, provides robust empirical evidence validating our feature engineering process. The NRP oscillator is a consistent predictor of future returns, particularly at its extremes. Its predictive power is substantially enhanced when combined with its own derivative, which acts as a timing filter. The effects are persistent and observable across a wide

range of time scales, confirming the existence of a strong and exploitable mean-reverting edge in the analyzed data.

3.6 Data Preprocessing

3.6.1 Kalman Filtering

The objective of this tool is to estimate the trend of a financial asset from observed prices at each moment, using an estimation of the underlying noise in the system's dynamics. It allows for a robust smoothing of the price series to improve the quality of a machine learning model's training thereafter.

The Principle of Causality

In time series analysis, respecting **causality** is paramount. This means that any prediction or estimation at a time t must be based **only on the information available up to time** t.

Ignoring this leads to a **data leak**, where the model unintentionally uses future information. Such a model would produce falsely optimistic results in backtesting and would fail completely in real conditions. The described approach is designed to ensure strict causality at every step.

General Operation

The Kalman filter estimates the hidden state $\mathbf{x}_k$ of a system from noisy measurements $\mathbf{z}_k$. It operates in two stages.

Step 1: Prediction

The filter predicts the next state and its uncertainty from the previous state.

$$\hat{\mathbf{x}}_{k|k-1} = \mathbf{F}_k \hat{\mathbf{x}}_{k-1|k-1} \quad \text{(State prediction)} \tag{1}$$

$$\mathbf{P}_{k|k-1} = \mathbf{F}_k \mathbf{P}_{k-1|k-1} \mathbf{F}_k^T + \mathbf{Q}_k \quad \text{(Uncertainty prediction)} \tag{2}$$

With:

- $\hat{\mathbf{x}}_{k|k-1}$: predicted state at time k based on information up to $k-1$.
- $\mathbf{F}_k$: state transition matrix, which models the system's evolution.
- $\mathbf{P}_{k|k-1}$: covariance of the prediction error.
- $\mathbf{Q}_k$: covariance of the process noise (model uncertainty).

Step 2: Update

The filter corrects its prediction using the new measurement $\mathbf{z}_k$.

$$\tilde{\mathbf{y}}_k = \mathbf{z}_k - \mathbf{H}_k \hat{\mathbf{x}}_{k|k-1} \quad \text{(Innovation: measurement-prediction discrepancy)} \tag{3}$$

$$\mathbf{K}_k = \mathbf{P}_{k|k-1} \mathbf{H}_k^T (\mathbf{H}_k \mathbf{P}_{k|k-1} \mathbf{H}_k^T + \mathbf{R}_k)^{-1} \quad \text{(Kalman Gain)} \tag{4}$$

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k \tilde{\mathbf{y}}_k \quad \text{(State update)} \tag{5}$$

$$\mathbf{P}_{k|k} = (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k) \mathbf{P}_{k|k-1} \quad \text{(Uncertainty update)} \tag{6}$$

With:

- $\mathbf{H}_k$: observation matrix, which links the state to the measurement.
- $\mathbf{R}_k$: covariance of the measurement noise (measurement uncertainty).

- $\mathbf{K}_k$: the Kalman gain, which weighs the importance of the innovation.

Adaptive Kalman Filter

A standard Kalman filter assumes that the noises $(\mathbf{Q}_k, \mathbf{R}_k)$ are constant. This is an unsuitable assumption for financial markets where volatility varies greatly.

We use a model where the state is:

$$\mathbf{x}_k = \begin{bmatrix} \text{price}_k \\ \text{velocity}_k \end{bmatrix}$$

The transition matrix $\mathbf{F}$ and observation matrix $\mathbf{H}$ are constant:

$$\mathbf{F} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}, \quad \mathbf{H} = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

$\mathbf{F}$ models that the next price is the current price plus the velocity. $\mathbf{H}$ indicates that we only directly observe the price. This parameterization is the same as in the base model.

The proposed approach is to make the process noise matrix $\mathbf{Q}_k$ **dynamic**. Instead of being fixed, it is updated at each time t based on the local market volatility.

Characterization of the Process Noise Matrix Q

In the context of the Kalman filter, the process noise covariance matrix, denoted $\mathbf{Q}$, is the parameter that measures the uncertainty inherent in the state transition dynamics model. It represents the covariance of the noise $\mathbf{w}_k$ in the state equation:

$$\mathbf{x}_k = \mathbf{F}_k \mathbf{x}_{k-1} + \mathbf{w}_k, \quad \text{with} \quad \mathbf{w}_k \sim \mathcal{N}(0, \mathbf{Q}_k)$$

For a constant velocity model where the state vector is defined by $\mathbf{x}_k = \begin{bmatrix} \text{price}_k & \text{velocity}_k \end{bmatrix}^T$, the matrix $\mathbf{Q}$ is:

$$\mathbf{Q} = \begin{bmatrix} \Sigma_{\text{price,price}} & \Sigma_{\text{price,velocity}} \\ \Sigma_{\text{price,velocity}} & \Sigma_{\text{velocity,velocity}} \end{bmatrix}$$

with $\Sigma_{\text{x,y}} = \text{cov(x,y)}$.

Q_{11} **: Variance of Position Noise.** Represents the variance of the noise directly affecting the first component of the state (the price), independently of the second (the velocity). It models stochastic disturbances that are not captured by the velocity dynamics; its value is assumed to be low.

Q_{22} **: Variance of Velocity Noise.** Determines the variance of the noise affecting the velocity component. The assumption of a constant velocity model is a first-order approximation of the real dynamics. Q_{22} thus captures unmodeled second-order and higher dynamics, particularly accelerations or random shocks to the underlying trend of the process.

Q_{12} **and** Q_{21} **: Covariance of Noises.** These off-diagonal terms represent the covariance between the process noises affecting the position and velocity, respectively. A common modeling assumption is the orthogonality of the process noise components, which leads to setting these terms to zero. The matrix $\mathbf{Q}$ is then assumed to be diagonal.

We then assume that the noises affecting the price and velocity are uncorrelated. This yields the diagonal matrix used in the model:

$$\mathbf{Q} = \begin{bmatrix} \sigma^2_{\text{price}} & 0 \\ 0 & \sigma^2_{\text{velocity}} \end{bmatrix}$$

For a non-adaptive filter, $\mathbf{Q}$ is assumed to be static. The methodology consists of postulating a sparse structure (usually diagonal) and estimating its constant components. This calibration can be performed by optimization methods, such as maximizing the likelihood of the filter's innovations.

The approach proposed in our work overcomes this limitation by considering $\mathbf{Q}$ as a time-varying matrix, $\mathbf{Q}_k$. Specifically, we link its component $Q_{22,k}$ to an observable measure of the market state: the local volatility σ_k^2, calculated over a rolling historical window.

1. Measurement of local volatility: We calculate the variance of log-returns over a past rolling window of w periods (arbitrarily chosen as 60):

$$\sigma_k^2 = \mathrm{Var}(\log(P_{t-w+1}), \ldots, \log(P_t))$$

2. Adaptation of the noise matrix $\mathbf{Q}_k$: The matrix $\mathbf{Q}_k$ represents the uncertainty of our movement model. We link the uncertainty on the velocity to the market volatility:

$$\mathbf{Q}_k = \begin{bmatrix} q_{\text{price}} & 0 \\ 0 & c \cdot \sigma_k^2 \end{bmatrix}$$

where q_{price} is a small, constant variance for the price, and c is a scaling factor.

Why this approach

Adapting $\mathbf{Q}_k$ allows the filter to change its behavior in real time:

- **In periods of high volatility**: σ_k^2 increases $\implies$ $\mathbf{Q}_k$ increases. The filter trusts its trend prediction (internal model) less and gives more weight to the new price measurement. It becomes more **responsive** and follows rapid movements.
- **In calm periods**: σ_k^2 decreases $\implies$ $\mathbf{Q}_k$ decreases. The filter trusts its trend prediction more. It considers small fluctuations as noise and produces a **smoother** signal.

The standard Kalman filter, with a fixed $\mathbf{Q}$, would be either too slow in a volatile market or too aggressive in a stable market. This adaptive approach allows for a more robust and realistic estimation of the asset's movement dynamics.

When this filter is applied to a time series $(y_t)_{t=1}^T$, we obtain a sequence of estimates $(\hat{x}_t)_{t=1}^T$, where each $\hat{x}_t = \mathbb{E}[x_t \mid y_{1:t}]$ is a vector representing the estimated state at time t.

$$\mathrm{x}_t = \begin{bmatrix} \text{position}_t \\ \text{velocity}_t \end{bmatrix}$$

The estimated price corresponds to the first component of x_t, so the filtered time series is given by:

$$(\hat{y}_t)_{t=1}^T = \left(x_t^{(1)}\right)_{t=1}^T$$

Thus, $\hat{y}_t$ denotes the position component of the state estimate x at t.

Several filters are then calculated for each series: Open, High, Low, and Close to smooth their values. Subsequently, the data is properly filled because gaps can occur, for example, a 30-second jump between consecutive ticks. To do this, missing values are filled with the value at the last timestamp. The volume, on the other hand, is uniformly distributed over the missing period.

3.6.2 Additional Feature Engineered

*Notations

- P_t: Closing price (Close) at time t.
- H_t: Highest price (High) at time t.
- L_t: Lowest price (Low) at time t.
- $\text{MA}_n(X)$: Simple Moving Average of series X over a period of n.
- $\text{EMA}_n(X)$: Exponential Moving Average of series X over a period of n.
- $\text{StdDev}_n(X)$: Rolling Standard Deviation of series X over a period of n.

List of Indicators

- **KF Deviation**: Difference between the initial price time series (y_t) and the series smoothed by the Kalman filter ($\hat{y}_t$).

$$\text{y}_t - \hat{y}_t$$

- **Velocity**: Measures the rate of change of the price's position.

$$\text{x}_t = \begin{bmatrix} \text{position}_t \\ \text{velocity}_t \end{bmatrix}$$

- **Log Return**: Measures the continuous rate of change of the price.

$$\text{log\_ret\_1}_t = \ln\left(\frac{P_t}{P_{t-1}}\right)$$

- **Rolling Volatility**: Standard deviation of logarithmic returns over a rolling period p, where $p \in \{360, 720, 1440\}$.

$$\text{volatility\_p}_t = \text{StdDev}_p(\text{log\_ret\_1})$$

- **Average True Range**: A measure of market volatility. It is defined as the moving average of the True Range (TR) over a period p, where $p \in \{360, 720, 1440\}$.

$$\text{TR}_t = \max(H_t - L_t, |H_t - P_{t-1}|, |L_t - P_{t-1}|)$$

$$\text{atr\_p}_t = \text{MA}_p(\text{TR}) \quad \text{(or a smoothed variant)}$$

- **Bollinger Bands Width**: Measures the relative width of the Bollinger Bands, indicating a contraction or expansion of volatility.

$$\text{bb\_width}_t = \frac{\text{Upper Band}_{20} - \text{Lower Band}_{20}}{\text{Middle Band}_{20}}$$

where the bands are defined as:

$$\begin{aligned} \text{Middle Band}_{20} &= \text{MA}_{20}(P) \\ \text{Upper Band}_{20} &= \text{Middle Band}_{20} + 2 \times \text{StdDev}_{20}(P) \\ \text{Lower Band}_{20} &= \text{Middle Band}_{20} - 2 \times \text{StdDev}_{20}(P) \end{aligned}$$

- **Relative Strength Index (RSI)**: A momentum oscillator that measures the speed and change of price movements.

$$\text{RSI}_{14} = 100 - \frac{100}{1 + RS}$$

where $RS = \frac{\text{Average of gains over 14 periods}}{\text{Average of losses over 14 periods}}$.

- **Average Directional Index (ADX)**: An indicator used to quantify the strength of a trend, regardless of its direction. It is composed of three lines: the ADX itself (which measures trend strength), and two directional indicators ($+DI_p$ and $-DI_p$) that measure trend direction. The typical period is $p = 14$.
  - *Directional Indicators*: $+DI_p$ (Positive) and $-DI_p$ (Negative) are calculated based on smoothed directional movements relative to the Average True Range (ATR).
  - *Directional Index (DX)*:

$$DX_p = 100 \times \frac{|(+DI_p) - (-DI_p)|}{(+DI_p) + (-DI_p)}$$

  - *ADX*: The ADX is a smoothed moving average of the DX.

$$ADX_p = \text{SMA}_p(DX)$$

- **Moving Average Convergence Divergence (MACD)**: A trend-following momentum indicator that shows the relationship between two exponential moving averages of the price.

- *MACD Line*:

$$\text{MACD Line} = \text{EMA}_{12}(P) - \text{EMA}_{26}(P)$$

- *Signal Line*:

$$\text{Signal Line} = \text{EMA}_9(\text{MACD Line})$$

- *Histogram*:

$$\text{Histogram} = \text{MACD Line} - \text{Signal Line}$$

- **Distance to Moving Average 720**: Relative deviation of the closing price from its medium-term moving average.

$$\text{dist\_ma\_720}_t = \frac{P_t - \text{MA}_{720}(P)}{\text{MA}_{720}(P)}$$

3.7 Learning Models

3.7.1 Prediction by Regression

To predict the return over a horizon of w time steps, we associate the features at time t (X_t) with the future return. The target variable y_t is defined as:

$$y_t = \log\left(\frac{P_{t+w}}{P_t}\right)$$

This formulation allows the model to be trained to predict the future return of the asset. The dataset is split into training, validation, and test sets over the period 2022-2024. The period of 2025 is kept for the final backtesting of the strategy.

$$X_{\text{train}} = \begin{bmatrix} x_{1,1} & \cdots & x_{1,m} \\ \vdots & \ddots & \vdots \\ x_{N,1} & \cdots & x_{N,m} \end{bmatrix} \qquad Y_{\text{train}} = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix} = \begin{bmatrix} \log(P_{1+w}/P_1) \\ \vdots \\ \log(P_{N+w}/P_N) \end{bmatrix}$$

For regression predictions, the model that performed best was **XGBoost**. The process, from data preparation to evaluation, was designed to respect the principle of causality and provide a realistic assessment of the model.

1. Data and Target Preparation Data from the years 2022, 2023, and 2024 are concatenated to form our dataset. The target variable y_t is defined as the future logarithmic return over a horizon of $w = 60$ periods (corresponding to 5 minutes with 5-second data), multiplied by 10,000 for better scaling (similar to basis points):

$$y_t = 10000 \times \log\left(\frac{P_{t+w}}{P_t}\right)$$

The dataset is then split chronologically to avoid any data leakage from the future:

- **Training Set**: The oldest 70% of the data.
- **Validation Set**: The next 15%.
- **Test Set**: The most recent 15%, strictly reserved for final evaluation.

2. Filtering: Focusing on Significant Movements Financial markets are inherently noisy. The majority of small price variations are difficult, if not impossible, to predict and can pollute the model's learning process. To improve the signal-to-noise ratio, we apply a filtering strategy:

1. A filtering threshold θ_{filter} is defined as the **50th percentile (the median) of the absolute value of the target** $|y_t|$ on the combined training and validation sets.
2. Only the samples (X_t, y_t) from the training and validation sets for which $|y_t| > \theta_{\text{filter}}$ are kept for the learning phase.

This approach forces the model to train on the largest and potentially more predictable market movements. **The test set is never filtered**, ensuring that the final performance is evaluated under realistic conditions where all types of movements are present.

3. Standardization of Features The features are standardized using Scikit-learn's StandardScaler. The scaler is fitted **only on the filtered training set** to calculate the mean μ and standard deviation σ of each feature. This transformation is then applied to the training, validation, and test sets.

$$z = \frac{x - \mu}{\sigma}$$

This step is crucial to prevent any statistical information leakage from the validation and test sets into the training process.

4. Model Training and Validation The XGBoost model is trained on the filtered and standardized training data. We use the validation set to implement an **early stopping** technique. Training stops automatically if the performance on the validation set does not improve for 10 consecutive rounds, which helps to find an optimal number of decision trees and prevent over-fitting.

5. Performance Evaluation The final model is evaluated on the complete (unfiltered) test set. Several metrics are used for a comprehensive analysis:

- **MAE (Mean Absolute Error) and RMSE (Root Mean Squared Error)**: These metrics measure the average prediction error.

$$\text{MAE} = \frac{1}{N}\sum_{t=1}^{N}|y_t - \hat{y}_t| \qquad \text{RMSE} = \sqrt{\frac{1}{N}\sum_{t=1}^{N}(y_t - \hat{y}_t)^2}$$

- **Directional Accuracy**: The predicted value is binarized using the sign function: $\text{sign}(\hat{y}_t) = \text{sign}(y_t)$, to evaluate the accuracy and compare it with the logistic regression model below.
- **Accuracy on Strong Predictions**: To simulate a more selective trading strategy, we evaluate directional accuracy only on a subset of predictions: those whose magnitude is greater than the median of positive predictions (for buys) or less than the median of negative predictions (for sells). This allows us to measure the reliability of the model when it emits a strong signal.

First predicting the future return values and then binarizing them into up or down has a superior advantage over a direct logistic regression model: the two-step approach allows for different penalization depending on the deviation between the prediction and the real value using the MSE loss function. Indeed, the binary prediction model is not penalized in the same way because its targets are binarized regardless of the return value, whether it is high or not. It is this aspect that the two-step approach enables: the first step with MSE refines the return prediction by penalizing large deviations more heavily, which can potentially lead to better return predictions, which in turn can improve the quality of the binarization if the model is well-calibrated. The direct logistic model does not benefit from this penalization based on the magnitude of the actual return.

3.7.2 Prediction by Classification

In addition to regression, a binary classification approach was developed to predict the direction of the asset's movement (Up or Down). This method transforms the return prediction problem into a classification task, which is often more robust to the noise of financial markets.

1. Formulation of the Binary Target

The continuous target variable $y_t = \log(P_{t+w}/P_t)$ is discretized into a binary label. The objective is to predict whether the price will increase or decrease over a horizon of w periods. The target y_t is defined as follows:

$$y_t = \begin{cases} 1 & \text{if } \log\left(\frac{P_{t+w}}{P_t}\right) > 0 \quad \text{(Up)} \\ 0 & \text{otherwise} \quad \text{(Down)} \end{cases}$$

2. Filtering of Training Data

To improve the signal-to-noise ratio, a filtering strategy similar to that of regression is applied. However, the threshold is not based on the median. We define a threshold θ_{filter} as the **58th percentile** of the absolute value of future log-returns on the training and validation data.

$$\theta_{\text{filter}} = \text{Quantile}_{0.58}\left(\left|\log\left(\frac{P_{t+w}}{P_t}\right)\right|\right)_{\text{train+val}}$$

Only the samples (X_t, y_t) for which $|y_t| > \theta_{\text{filter}}$ are kept for training. This approach allows the learning to focus on market movements with a slightly larger amplitude than the median, while still retaining a large portion of the data. The test set remains unfiltered for a realistic evaluation.

3. Model Architecture: Neural Network with Residual Blocks

For this task, a deep neural network was implemented with TensorFlow/Keras. The architecture is inspired by residual networks (ResNet) to facilitate the training of deep models and avoid the vanishing gradient problem.

The model consists of:

- A dense input layer, followed by Batch Norm and Dropout regularization.
- A series of **5 residual blocks**. Each block applies a non-linear transformation while maintaining a direct connection (shortcut) from its input to its output. The principle of a residual block is:
$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + \mathbf{x}$$
where $\mathcal{F}$ is the function learned by the layers of the block and $\mathbf{x}$ is the input to the block.
- An output layer with a single neuron and a **sigmoid** activation function, which produces a probability $\hat{p}(y_t = 1|X_t)$ between 0 and 1.

The 'swish' activation and 'he_normal' weight initialization are used throughout the network for their efficiency.

4. Training and Optimization The model is trained with advanced techniques to ensure robust convergence:

- **Optimizer**: RMSprop is used, with *weight decay* (L_2) regularization to penalize large feature weights and reduce overfitting.
- **Learning Rate Scheduling**: A CosineDecay is applied. The learning rate gradually decreases following a cosine curve, allowing for a finer learning rate as training progresses.
- **Loss Function**: Binary crossentropy is used.
- **Layer Initialization**: Using the He Normal distribution.
- **Early Stopping**: Training is stopped if the loss on the validation set does not improve for 5 consecutive epochs, restoring the weights of the best model.

5. Evaluation and Probability Calibration

The evaluation of the classification model goes beyond simple accuracy.

1. **Directional Accuracy**: This is the basic measure, where a prediction is considered correct if the predicted class (obtained by thresholding the probability at 0.5) matches the true market direction.
2. **Probability Calibration**: The raw probabilities from a neural network are often not well-calibrated (a prediction of 0.7 does not necessarily correspond to a 70% real chance of an increase). To correct this, an **isotonic regression** is fitted on the predictions of the *validation* set. This calibration tool learns a non-parametric, non-decreasing correction function.
$$\hat{p}_{\text{calibrated}} = \text{IsotonicRegression}(\hat{p}_{\text{raw}})$$
The isotonic regression model is then applied to the predictions of the *test* set to obtain more reliable probabilities. This calibration is crucial for our use case.

3.7.3 Isotonic Regression

Isotonic regression is a non-parametric method that fits a non-decreasing monotonic function to a set of data. In the context of probability calibration, the objective is to transform a sequence of raw probabilities $\hat{p}_{\text{raw}} = \{\hat{p}_1, \hat{p}_2, \ldots, \hat{p}_n\}$, which may not be well-calibrated, into a new sequence of probabilities $\hat{p}_{\text{calibrated}} = \{x_1, x_2, \ldots, x_n\}$ that is both close to the initial probabilities and perfectly monotonic.

Mathematical Formulation

Given a set of raw predictions $\hat{p}_1, \hat{p}_2, \ldots, \hat{p}_n$ and the corresponding observed outcomes $y_i \in \{0, 1\}$, isotonic regression seeks to find a monotonically increasing function f that solves the following optimization problem:

$$\min_{f \in \mathcal{M}} \sum_{i=1}^{n} w_i \left(f(\hat{p}_i) - y_i\right)^2$$

under the constraint:

$$f(\hat{p}_i) \leq f(\hat{p}_j) \quad \text{if} \quad \hat{p}_i \leq \hat{p}_j$$

where:

- y_i is the empirical probability (generally obtained by empirical frequency on bins),
- $\hat{p}_i$ is the probability predicted by the model before calibration,
- $f(\hat{p}_i)$ is the calibrated, monotonic, and increasing probability,
- w_i is a weight associated with each observation,
- $f(\hat{p}_i)$ is an estimate of: $\mathbb{P}(y_i = 1 \mid \hat{p}_i)$

This problem is equivalent to performing a linear regression under a monotonicity constraint.

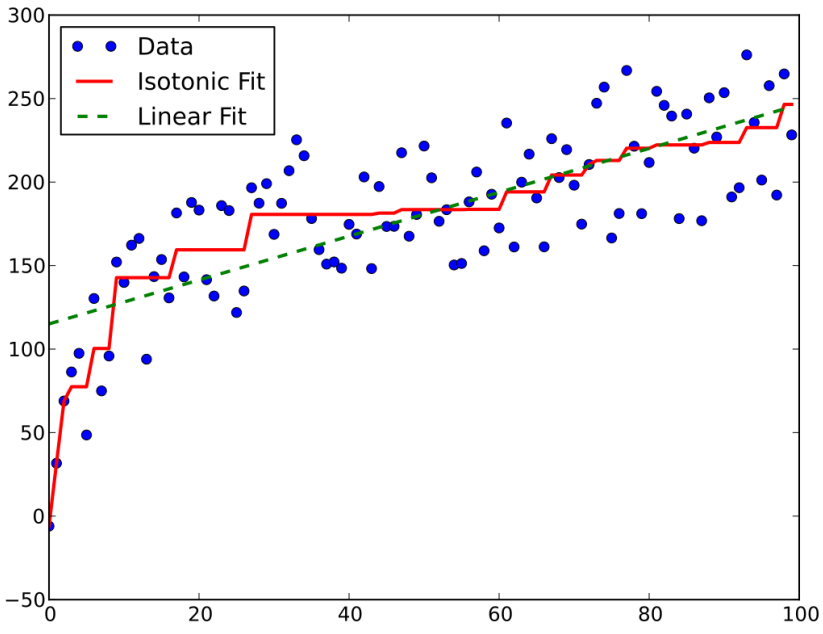


Figure 15: Isotonic Regression in green, Source : Wikipedia

PAVA Algorithm

The most common algorithm for solving an isotonic regression problem is the **Pool Adjacent Violators Algorithm (PAVA)**. It finds the monotonically increasing function that minimizes the weighted squared error with respect to the observations.

(a) **Initialization**: We start by sorting the data according to the increasing raw predictions $\hat{p}_i$. We associate each prediction with an individual block containing the target value y_i and a weight w_i (usually 1).

(b) **Violation Detection**: We traverse the sequence of blocks and identify any pair of adjacent blocks whose average violates the monotonically increasing constraint. In other words, if a block has a higher average than the next block, a violation is detected.

(c) **Correction (Pooling)**: When a violation is detected, the concerned blocks are merged into a new block B. A new weighted average value is then calculated:

$$\bar{y}_B = \frac{\sum_{j \in B} w_j y_j}{\sum_{j \in B} w_j}$$

This value is assigned to all points in the block:

$$f(\hat{p}_i) = \bar{y}_B \quad \text{for all } i \in B$$

(d) **Iteration**: After each merge, we re-check the monotonicity constraints with the neighboring blocks and repeat until the entire sequence is monotonically increasing.

Once the isotonic regression model is trained on the validation data, it can be applied to calibrate new predictions. The calibrated probability is obtained by applying the learned function to the raw probabilities:

$$\hat{p}_{\text{calibrated}} = \text{IsotonicRegression}(\hat{p}_{\text{raw}})$$

This step ensures that if the initial model predicts a probability $\hat{p}_a$ higher than a probability $\hat{p}_b$, the corresponding calibrated probability $\hat{p}_{\text{calibrated},a}$ will necessarily be greater than or equal to $\hat{p}_{\text{calibrated},b}$, thus ensuring ordinal consistency and improving the reliability of the predictions.

3. **Accuracy on Confidence Signals**: Using the calibrated probabilities, we simulate a more selective strategy. A decision is made only if the predicted probability is outside a neutral zone (for example, $\hat{p}_{\text{calibrated}} < 0.3$ or $\hat{p}_{\text{calibrated}} > 0.7$). The accuracy is then recalculated only on this subset of high-confidence predictions, giving a better insight into the potential performance in a realistic trading framework. Indeed, if the model's signal has a low probability, i.e., around 50%, we can decide to do nothing.

3.8 Deep Reinforcement Learning Implementation

3.8.1 Motivation for a RL Approach

While supervised learning models, as implemented in our data processing pipeline (Section 3.7), excel at prediction, the challenge of optimal trade execution is a problem of prescription. Our supervised ResNet models for instance, provide valuable probabilistic forecasts on market direction over various horizons (1, 5, 30, and 60 minutes). However, these predictions can be used to predict the market's movement but not the action that should be taken. Knowing that

the market has a 60% chance of rising does not determine the optimal fraction of an asset to sell, especially when considering the entire liquidation horizon.

This highlights the limitation of predictive models whichare not designed to handle a *sequence of actions* where each decision influences future outcomes. The execution problem is a dynamic feedback loop; selling a large quantity now impacts the market price, which alters the state and potential rewards for all subsequent actions. This task must therefore be framed as a problem of sequential optimal control under uncertainty. Our implementation formalizes this interactive loop within a custom gymnasium environment, the standard for such problems.

Reinforcement Learning (RL) represents a particularly well-suited computational framework for this domain. Instead of learning a mapping from features to a static target, an RL agent directly learns a policy (π)—a strategy that maps market states to optimal actions. Through trial-and-error interaction with the simulated environment, the agent's objective is shifted from minimizing prediction error to maximizing a cumulative reward. In our work, this reward is not a naive profit metric but a sophisticated signal representing the financial outperformance (*alpha*) against a TWAP benchmark, shaped by penalties for market impact and incomplete liquidation.

Actually, RL leverages the outputs of our predictive models not as final answers, but as rich features in a broader state representation, allowing it to learn the complex, dynamic strategy required for optimal execution. It does not learn what the market will do; it learns what an agent should do.

3.8.2 Formalizing the Problem as a Markov Decision Process (MDP)

To rigorously apply Reinforcement Learning, it is essential to frame the order execution problem as a Markov Decision Process (MDP). An MDP is a mathematical framework for modeling decision-making in situations where outcomes are partly random and partly under the control of a decision-maker. It is defined by a tuple (S, A, T, R), where we will focus on defining the state space (S), the action space (A), the transition dynamics (T), and the reward function (R). Precisely, the agent's objective is to liquidate a fixed initial position of \$1,000,000 over a finite execution horizon of 720 time steps. To ensure the agent learns a robust policy, each training episode is initialized at a random starting point within the historical dataset, and the position to be liquidated is randomly chosen to be either Long or Short.

The State (S)

The state is the complete set of information the agent receives from the environment to make its decision. A well-designed state must contain enough information to inform the policy without being overly complex. The state, generated at each step of the simulation, is a composite of market data and agent-specific portfolio data.

1. **Market Data (Temporal Dimension):** The agent receives a snapshot of the market over a recent time window. This is a matrix containing the values of all market features over a fixed historical window. These features are tripartite:
    - **Raw Market Data:** Price information such as Open, High, Low, Close and Volume.
    - **Engineered Technical Indicators:** The full suite of indicators engineered in sections 3.2 through 3.6, designed to capture market characteristics like momentum, mean-reversion, and volatility.
    - **Supervised Model Predictions:** Crucially, the state also includes the probabilistic outputs from the supervised models discussed in section 3.7. These predictions,

representing the likelihood of an upward or downward market move over multiple time horizons (specifically for 1, 5, 30, and 60-minute horizons), are fed directly to the RL agent as features.

Using a time window of 10 periods provides sufficient historical context for the model to detect temporal patterns. The inclusion of supervised predictions is a key design choice: it enriches the state with high-level, forward-looking information. Rather than forcing the RL agent to re-learn short-term directional patterns from raw data, we provide it with the predictive models. This form of knowledge transfer aims to guide the agent's decision-making process, allowing it to focus on the higher-level strategic task of managing inventory and timing trades over the entire execution horizon. This architecture inherently mitigates the impact of incorrect predictions as the supervised forecast is not a direct command, but rather one input among a rich set of features that includes raw market dynamics, technical indicators, the agent's own portfolio status... The agent's policy, therefore, learns to weigh the prediction against the other available evidence, preventing it from acting blindly on a potentially flawed signal and allowing it to correct its strategy based on the overall market context.

2. **Portfolio Data :** To contextualize the market data, we append five informational channels that describe the agent's current situation.
   - **PnL Channels:** Two channels provide the recent history of the agent's Profit and Loss (PnL) and that of the TWAP benchmark strategy. This is fundamental for the agent to assess its relative performance in real-time and adjust its strategy.
   - **Position Type:** A channel indicating whether the agent's task is to liquidate a Long position or a Short position.
   - **Time Remaining:** A ratio representing the fraction of time left before the end of the episode. This temporal signal is essential; it creates a sense of urgency and forces the agent to manage its inventory proactively as the deadline approaches.
   - **Position Remaining:** The fraction of the initial position that is yet to be liquidated. This serves as the most direct indicator of the agent's progress in its mission.

All features within the market data matrix are normalized to ensure they are on a comparable scale for the neural network input. The final observation structure is a 3D tensor, which is the standard input for neural network architectures like Convolutional Neural Networks (CNNs), adept at extracting features from grid-like data structures.

The Action (A)

The action is the decision the agent makes at each time step. The action space is continuous, represented by value between 0.0 and 1.0. This value means the fraction of the remaining position that the agent chooses to liquidate at that time step. An action of 0.1 means sell 10% of what is currently hold. This action is executed as a market order, assumed to be filled instantly and entirely at the current market price without partial fills. A crucial constraint is applied, capping the action at 30% of the remaining position. While the action space allows values up to 1.0, the effective action is subsequently clipped to 0.3 in the environment, meaning any output greater than this value is treated as 0.3. This is a realistic constraint that implicitly models market impact and prevents the agent from adopting unrealistic strategies, such as liquidating its entire position in a single tick.

The Transition Dynamics (T)

In a formal MDP, the transition function $T(s'|s, a)$ defines the probability of moving to a new state s' after taking action a in state s. In the context of financial markets, these dynamics are immensely complex, non-stationary, and unknown. It is impossible to explicitly model the transition matrix that governs the evolution of market prices and all derived indicators.

This intractability is precisely why a model-free Reinforcement Learning approach is necessary. Instead of attempting to learn a model of the market's dynamics, our agent will learn its policy directly from interaction. The simulation environment, built upon historical data, acts as a proxy for the true, unknown transition function. It is important to note that this simulation environment operates under the simplifying assumption that the agent's trades have no price impact on subsequent market data. The historical data is replayed irrespective of the agent's actions. Market impact is therefore modeled indirectly as a behavioral constraint via a penalty in the reward function, rather than as a dynamic feature of the environment itself. The agent treats the environment as a black box, learning to map states to optimal actions based on the rewards it receives, without needing to understand the underlying probability distribution of market movements.

The Reward (R)

The design of the reward function is the central component of the MDP. A naive approach rewarding the agent on its absolute PnL is fundamentally flawed, as it fails to distinguish between market-driven gains and genuine strategic alpha. To solve this, the reward must measure the value added by the agent's decisions compared to a passive baseline.

This differential approach completely reshapes the agent's incentives. It prevents a myopic, risk-averse strategy of crystallizing small gains in a trending market; in such a case, the agent's reward would be catastrophic compared to the baseline. Conversely, in a declining market, an agent that liquidates quickly is rewarded handsomely for its loss mitigation. The agent is thus forced to take calculated risks and develop a dynamic policy that constantly seeks to outperform the baseline, adapting to the perceived market trend. This well-posed reward signal provides a clearer learning gradient, which empirically leads to a more stable decrease in the critic's value loss during training, indicating that the agent is successfully learning to predict the long-term value of its actions.

The Time-Weighted Average Price (TWAP) strategy serves as our industry-standard benchmark. The benchmark is implemented by executing a fixed fraction of the total order at regular intervals of 30 time steps throughout the episode. To formally construct the reward, we must first define how PnL is calculated.

- **PnL Calculation:** At any time step t, the total PnL is the sum of the realized PnL from executed trades and the unrealized PnL of the remaining position, marked-to-market. We model a spot Foreign Exchange (FX) trading environment, where explicit brokerage fees are generally not applied. The primary transaction cost is the bid-ask spread, which the agent implicitly navigates by timing its trades. Therefore, the PnL is calculated gross of explicit fees.

$$\text{PnL}_t = \text{PnL}_t^{\text{realized}} + \text{PnL}_t^{\text{unrealized}}$$

  where:

  - $\text{PnL}_t^{\text{realized}} = (\bar{P}_{\text{exec},t} - P_{\text{entry}}) \cdot Q_{\text{sold},t} \cdot \delta$
  - $\text{PnL}_t^{\text{unrealized}} = (P_t - P_{\text{entry}}) \cdot Q_{\text{rem},t} \cdot \delta$

And the terms are: P_{entry} is the entry price, $\bar{P}_{\text{exec},t}$ is the average execution price of all trades up to time t, P_t is the current market price, $Q_{\text{sold},t}$ and $Q_{\text{rem},t}$ are the quantities sold and remaining, and δ is the position type (+1 for Long, -1 for Short). A similar calculation is performed for the TWAP benchmark's PnL.

This leads to our final reward function structure:

1. **The main Reward: Outperformance Against the Benchmark**
   The primary reward is a **differential reward**, measuring the agent's marginal outperformance at every step.

   - **Formula:**
   $$R_t = (\text{pnl}_t - \text{pnl}_{t-1}) - (\text{pnl\_twap}_t - \text{pnl\_twap}_{t-1})$$
   At each time step t, the agent is rewarded for the marginal gain in its PnL relative to the marginal gain of the TWAP strategy.

   Stepwise Reward vs. Terminal Reward

   Providing rewards incrementally at each step, rather than only at the end of the episode, is important for effective learning.

   - **Reasoning:** Stepwise rewards mitigate the *credit assignment problem*, where the agent would otherwise struggle to identify which specific actions led to success or failure.
   - **Advantage:** Immediate feedback allows the agent to adjust its policy incrementally, leading to faster convergence and more stable learning dynamics.
   - **Outcome:** By rewarding marginal outperformance at every step, the agent is guided consistently towards strategies that outperform the benchmark, rather than relying on a single delayed signal at the end.

2. **Penalties**
   To guide the agent towards realistic behavior, the final reward is adjusted by subtracting penalties. The final reward function is $R'_t = R_t - P_t^{\text{impact}} - P_t^{\text{incomplete}}$.

   - **Market Impact Penalty**: A penalty is subtracted at each step, quadratic in the size of the action and scaled by market volatility.
   $$P_t^{\text{impact}} = \lambda_{\text{dynamic}} \cdot a_t^2 \cdot V_0$$
   where a_t is the action, V_0 is the initial position value, and $\lambda_{\text{dynamic}} = \lambda_{\text{base}} \cdot (1 + k_\sigma \cdot \sigma_t)$ is a dynamic coefficient that increases with current market volatility σ_t (the 360-period historical volatility).
     - **Calibration of the Impact Parameter (λ_base):** Setting the base_lambda hyperparameter is not easy : a value that is too small will be ignored by the agent, while a value that is too large will suppress any meaningful trading activity. To address this, we employed a principled, data-driven calibration process to ensure the penalty is appropriately scaled relative to the reward signal.
       (a) **Estimate Reward Signal Magnitude:** First, we estimated the characteristic magnitude of the per-step reward, $|R_t|$. This was achieved by running a simple heuristic policy (a perturbed TWAP strategy) over 100 episodes and calculating the average absolute reward per step, denoted as $\mathbb{E}[|R_t|]$. This gives us a baseline for the magnitude of the learning signal.

(b) **Define a Target Penalty:** Next, we defined a target penalty magnitude for a reference action size. We posited that for a significant trade (e.g., $a_t = 0.1$, representing 10% of the remaining position), the resulting penalty should be a small but meaningful fraction (e.g., 8%) of the typical reward signal. The target penalty, P_{target}, is thus:

$$P_{\text{target}} = 0.08 \cdot \mathbb{E}[|R_t|]$$

(c) **Calculate λ_{base}:** Finally, by substituting the target penalty and reference action into the penalty formula ($P_{\text{target}} = \lambda_{\text{base}} \cdot a_t^2 \cdot V_0$), we can solve for λ_{base}:

$$\lambda_{\text{base}} = \frac{P_{\text{target}}}{a_t^2 \cdot V_0} = \frac{0.08 \cdot \mathbb{E}[|R_t|]}{(0.1)^2 \cdot V_0}$$

This data-driven approach provides a robust and justified starting point for λ_{base}, ensuring the market impact penalty is well-proportioned to the problem's reward landscape from the outset of training.

- **Incomplete Liquidation Penalty**: A terminal penalty is applied at the final time step T if any portion of the position remains ($Q_{\text{rem},T} > 0$).

$$P_T^{\text{incomplete}} = k_{\text{penalty}} \cdot \left(\frac{Q_{\text{rem},T}}{Q_0} \right)^2$$

where Q_0 is the initial position size and k_{penalty} is a hyperparameter **set to 10 to impose a significant cost on non-completion**. It is important to note that episodes always run for the full duration of 720 steps and end by truncation; early termination upon full liquidation is not permitted. This penalty is therefore here to encourage the agent to complete its mission within the allotted time. The quadratic nature of this penalty severely punishes large failures in liquidation.

3.8.3 Environment Preprocessing

To ensure a stable and efficient training process, the simulation environment undergoes several key preprocessing steps, managed primarily by wrappers from the Stable-Baselines3 library. These steps are standard practice in modern reinforcement learning for the success of PPO, especially when applied to noisy financial data.

- **Vectorization:** Instead of using a single environment, we create a *vectorized environment* using `SubprocVecEnv`, which runs 48 independent copies of the simulation in parallel. This large-scale parallelization is supported by a hardware architecture consisting of 2 Nvidia T4 GPUs.

  While the 48 simulation environments are executed in parallel across multiple CPU cores, the agent's neural network (the Actor-Critic policy) resides on one of the GPUs. This setup creates a highly efficient pipeline: the CPUs process the environment dynamics, while the GPU is dedicated to the computationally intensive task of neural network inference (generating actions for the batch of 48 states simultaneously) and training (updating the network weights). This CPU/GPU synergy is essential for two reasons. First, it dramatically accelerates data collection. Second, for on-policy algorithms like PPO, collecting experiences from a large, diverse batch of episodes (`n_steps * 48 environments`) helps to decorrelate the data, leading to more stable and robust policy updates.

- **Observation and Reward Normalization:** Neural networks, particularly the convolutional layers at the input of our Conv1D-LSTM model, are highly sensitive to the scale of their inputs. To address this, we use the `VecNormalize` wrapper, which performs two essential functions:

1. **Observation Normalization :** The wrapper computes a running average and standard deviation of all state observations (S) it encounters during training. It then uses these statistics to normalize each observation in real-time, ensuring the inputs to the neural network consistently have a mean of approximately 0 and a standard deviation of 1. This prevents features with vastly different scales (e.g., trading volume in millions vs. an RSI oscillator between 0 and 100) from disproportionately influencing the network's learning process, leading to faster and more stable convergence. For evaluation on a test set, these running statistics are frozen and saved after training. They are then loaded and applied to the test data without being updated, thus preventing any data leakage from the evaluation set and ensuring a fair assessment of the agent's performance.
2. **Reward Normalization :** Even more important, the rewards (R) are also scaled. Financial outcomes (PnL) can vary significantly in magnitude from one episode to the next depending on market volatility. Without normalization, a single episode with an unusually large PnL could generate excessively large gradients, destabilizing the agent's policy. Reward normalization scales the rewards to a consistent range, ensuring that the magnitude of policy updates remains stable and predictable throughout training.

- **Discount Factor (γ):** Within this wrapper, we also define the discount factor, a fundamental hyperparameter of the MDP. We set $\gamma = 0.999$. A value this close to 1 makes the agent *far-sighted*, meaning it heavily weighs future rewards when making a decision. For a long-horizon problem like ours (720 steps), this is essential. It encourages the agent to develop complex and patient strategies, tolerating a temporary underperformance against the benchmark to get a better opportunity later rather than settling for greedy actions that give small immediate gains.

3.8.4 Agent Architecture

Learning Algorithm: Proximal Policy Optimization

The choice of learning algorithm is crucial for ensuring stable and efficient training. We selected Proximal Policy Optimization (PPO), a state-of-the-art algorithm for a wide range of continuous control problems, for the following reasons:

- **Stability and Reliability:** One of PPO's main advantage is its stability, it achieves this through a clipped surrogate objective function, which constrains the size of policy updates at each training step. This mechanism prevents the policy from deviating too aggressively from the previous one, mitigating the risk of performance collapse which is a main concern in volatile financial environments.
- **Sample Efficiency and Performance:** PPO strikes an excellent balance between sample efficiency (how much data it needs to learn) and ease of implementation. It is known to achieve high performance on complex tasks without the extensive hyperparameter tuning required by other algorithms.
- **Suitability for the Environment:** As an on-policy algorithm, PPO is well-suited for learning from the continuous stream of experiences generated by our interactive market simulation.

The Actor-Critic Framework

PPO is an Actor-Critic algorithm, in which the agent is modeled using two distinct but interconnected neural networks: the Actor, which selects actions, and the Critic, which evaluates them.

- **The Actor (Policy Network):** The Actor is the decision-maker. Its role is to learn and represent the agent's policy (π).
  - **Input:** The current market and portfolio state, S_t.
  - **Output:** The parameters of a probability distribution over the action space. Since our action is a continuous value between 0.0 and 1.0, a Beta distribution is a natural choice. The Actor outputs the parameters α and β of this distribution, from which the final action a_t is sampled. This allows for stochasticity, which is essential for exploration.
- **The Critic (Value Network):** The Critic is the evaluator. It does not decide what to do, but instead learns to predict the expected quality of the states the agent visits.
  - **Input:** The current state, S_t.
  - **Output:** A single scalar value, $V(s_t)$, which is an estimate of the expected future cumulative reward (the return) from the current state.
  - **Utility:** The Critic's primary role is to reduce the variance of the policy gradient used to train the Actor. More precisely, its value estimates are used to compute a low-variance estimate of the Advantage function, typically using Generalized Advantage Estimation. The GAE signal, $\hat{A}_t^{\text{GAE}}$, indicates whether an action was better or worse than the average expectation for that state, providing a stable and reliable learning signal for updating the Actor's policy.

Mathematical Formulation of the Learning Process

The learning process is formalized through the interaction of the policy, the value function, and the advantage estimate, which are used to construct the PPO objective functions.

Policy and Value Functions

- **The Actor** network, with parameters θ, defines the stochastic policy $\pi_\theta(a_t|s_t)$. It maps a state s_t to a probability distribution over actions. In our case, this is a Beta distribution: $\pi_\theta(a_t|s_t) = \text{Beta}(\alpha(s_t;\theta), \beta(s_t;\theta))$.
- **The Critic** network, with parameters ϕ, learns an approximation of the state-value function, $V_\phi(s_t)$. This function estimates the expected cumulative discounted reward starting from state s_t:

$$V_\phi(s_t) \approx \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \,\middle|\, s_t \right]$$

  where $\gamma \in [0, 1]$ is the discount factor and r_{t+k} is the reward at a future step.

Generalized Advantage Estimation (GAE)

To get a stable estimate of how much better an action was than the average, we use Generalized Advantage Estimation. First, the Temporal Difference (TD) error δ_t is computed for each step:

$$\delta_t = r_t + \gamma V_\phi(s_{t+1}) - V_\phi(s_t)$$

The GAE is then the exponentially-weighted average of these TD errors:

$$\hat{A}_t^{\text{GAE}(\gamma,\lambda)} = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}$$

where:

- r_t: The reward received at time t.
- $V_\phi(s_t)$: The value of the current state, estimated by the Critic.
- $V_\phi(s_{t+1})$: The value of the next state.
- $\lambda \in [0, 1]$: A parameter that controls the bias-variance trade-off for the advantage estimate.

PPO Clipped Objective Functions

The Actor and Critic are trained by optimizing a composite objective.

- **Actor (Policy) Loss**: The Actor's parameters θ are updated using a clipped surrogate objective function that prevents overly large policy changes:

  $$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right]$$

  where:

  - $\hat{\mathbb{E}}_t$: Indicates the empirical average over a batch of samples.
  - $\hat{A}_t$: The advantage estimate from GAE.
  - ϵ: A small hyperparameter that defines the clipping range (e.g., 0.2).
  - $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ is the probability ratio between the current and old policies.

- **Critic (Value) Loss**: The Critic's parameters ϕ are updated by minimizing the mean-squared error between its predictions and a value target:

  $$L^{VF}(\phi) = \hat{\mathbb{E}}_t \left[(V_\phi(s_t) - V_t^{\text{target}})^2 \right]$$

  where V_t^{target}, the value target, is often calculated as $\hat{A}_t + V_\phi(s_t)$.

Network Implementation

Given that the state representation is a 3D tensor of shape (features, timesteps), a neural network architecture capable of extracting temporal patterns is required. We therefore implemented a hybrid **CNN-LSTM** model with a shared feature extraction backbone and two separate heads for the Actor and Critic. Let the input time series be denoted as $S_t \in \mathbb{R}^{C \times T}$, where C is the number of features and T is the number of timesteps.

1. **Shared Feature Extractor:** The network is designed to transform the time-series input into a feature vector :

   - The process begins with a first 1D convolutional layer containing **16 filters** with a kernel size of 3, followed by a Rectified Linear Unit (ReLU) activation function to introduce non-linearity, allowing the model to learn more complex patterns. For a given filter $W \in \mathbb{R}^{C \times k}$ and bias $b \in \mathbb{R}$, the output feature o_j at time position j is given by:

   $$o_j = b + \sum_{c=1}^{C} \sum_{i=1}^{k} W_{c,i} \cdot S_{c,j+i-1}$$

   The ReLU function is defined as:

   $$\text{ReLU}(x) = \max(0, x)$$

   A Dropout layer is then applied for regularization.
   - A second 1D convolutional layer with **32 filters** further processes these features, again followed by a ReLU activation and a Dropout layer.
   - The output of the convolutional block is then passed to a Long Short-Term Memory (LSTM) layer with **64 hidden units**. The LSTM's role is to model longer-term temporal dependencies across the sequence of features extracted by the CNN. Denoting the input sequence to the LSTM as $A = (a_1, ..., a_{T'})$, the cell's operations at each timestep t are:

   $$\begin{aligned}
   f_t &= \sigma(W_f \cdot [h_{t-1}, a_t] + b_f) && \text{(Forget Gate)} \\
   i_t &= \sigma(W_i \cdot [h_{t-1}, a_t] + b_i) && \text{(Input Gate)} \\
   \tilde{c}_t &= \tanh(W_c \cdot [h_{t-1}, a_t] + b_c) && \text{(Candidate Vector)} \\
   c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t && \text{(Cell State Update)} \\
   o_t &= \sigma(W_o \cdot [h_{t-1}, a_t] + b_o) && \text{(Output Gate)} \\
   h_t &= o_t \odot \tanh(c_t) && \text{(Hidden State Update)}
   \end{aligned}$$

   where σ is the sigmoid function and $\odot$ is the element-wise product.
   - The final hidden state of the LSTM at the last time step is then taken as the compact, high-level feature vector that summarizes the entire input sequence. This vector has a dimension of **64** and is denoted as $h_{final} = h_{T'}$.

2. **Decoupled Heads:** This feature vector, after an additional final Dropout layer is then fed into two separate Multi-Layer Perceptron (MLP) networks, known as heads. Each head consists of a hidden layer and an output layer.

   - The **Actor Head** uses a hidden layer of **64 neurons** (with ReLU activation) and an output layer that produces the two parameters (α and β) for the Beta distribution. The hidden layer computes $h_{actor} = \text{ReLU}(W_{actor} \cdot h_{final} + b_{actor})$. The output parameters are then derived as:

   $$\alpha = \text{softplus}(W_\alpha \cdot h_{actor} + b_\alpha) + 1$$

   $$\beta = \text{softplus}(W_\beta \cdot h_{actor} + b_\beta) + 1$$

   where the softplus function, defined as $\text{softplus}(x) = \ln(1 + e^x)$, is used to ensure that the parameters α and β are strictly positive, a mathematical requirement for the Beta distribution. The final action a_t is sampled from the resulting distribution: $a_t \sim \text{Beta}(\alpha, \beta)$.

- The **Critic Head** uses a similar hidden layer of **64 neurons** (with ReLU activation) and an output layer that produces the single scalar state-value, $V(s)$. The hidden layer computes $h_{critic} = \text{ReLU}(W_{critic} \cdot h_{final} + b_{critic})$, and the final value is given by:

$$V(s_t) = W_{value} \cdot h_{critic} + b_{value}$$

This shared-body, two-head architecture is more efficient in term of computation and favorise the learning of robust features that are beneficial for both the policy and the value function, while the use of Dropout provides essential regularization to prevent overfitting on financial time-series data.

3.8.5 Advanced Metrics via Monitoring Callbacks

Beyond the final evaluation metrics, it is insightful to use *callbacks* during the evaluation phase to compute indicators that contextualize the agent's performance. A callback is a function that is executed at specific moments (e.g., at every time step) and has access to the state of the environment and the agent's decisions. This allows us to calculate metrics that are not part of the reward function but offer a deeper analysis of the strategy's behavior. Beyond trend visualization, the ATF's primary strength is its function as an engine for dynamically generating a rich set of context-aware predictive features.

The Maximum Potential PnL :

An important aspect of strategy evaluation is assessing how the realized profit of the agent compares to the maximum profit theoretically attainable. For example, an agent generating 20 bps may appear effective, but this performance is less meaningful if market conditions allowed for 200 bps. To capture this notion, we introduce the Maximum Potential PnL metric.

- **Definition:** At each time step t where the agent makes a decision, we look into a future window of N periods (e.g., $N = 720$) and identify the highest price ($P_{\text{max future}}$) and the lowest price ($P_{\text{min future}}$).
  - The **Maximum Potential Long PnL** is the profit that would have been realized by buying at the current price P_t and selling at the future highest price $P_{\text{max future}}$.

$$\text{PnL}_{\text{max\_long},t} = (P_{\text{max future}} - P_t) \cdot Q$$

  - The **Maximum Potential Short PnL** is the profit that would have been realized by short-selling at the current price P_t and buying back at the future lowest price $P_{\text{min future}}$.

$$\text{PnL}_{\text{max\_short},t} = (P_t - P_{\text{min future}}) \cdot Q$$

  This metric (which assumes perfect foresight) establishes an upper bound for performance. It is not used to train the agent, but to evaluate it. By comparing the agent's realized PnL to the Maximum Potential PnL, we can calculate a Performance Capture Ratio, which measures the agent's effectiveness at extracting the market's potential.

The figure below illustrates the calculation of this metric for a single point in time.

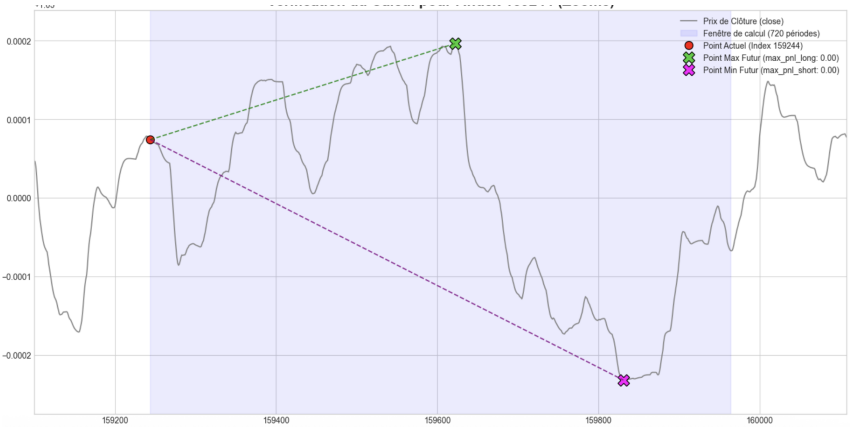


Figure 16: Illustration of the Maximum Potential PnL calculation. From a decision point, the metric identifies future price extrema within a defined window to quantify the maximum market opportunity.

- The **Current Point** (red circle) represents the time t when the agent must make a decision, at price P_t.
- The **Calculation Window** (blue shaded area) defines the future horizon of 720 periods over which the opportunity is measured.
- The **Future Max Point** (green X) is the highest point reached by the closing price within this window. The vertical price difference between this point and the current point represents the $\text{PnL}_{\text{max\_long}}$.
- The **Future Min Point** (purple X) is the lowest point reached. The vertical price difference between the current point and this point represents the $\text{PnL}_{\text{max\_short}}$.

By aggregating this information over an entire episode, one can visualize the dynamics of the available market opportunity and more accurately assess the relevance of the agent's decisions at each step.

To monitor how well the agent learns to capitalize on these market opportunities during training, a precise callback calculates this efficiency score at the end of each episode and logs it. This provides a dynamic view of the agent's learning progress, as shown in the figure below.

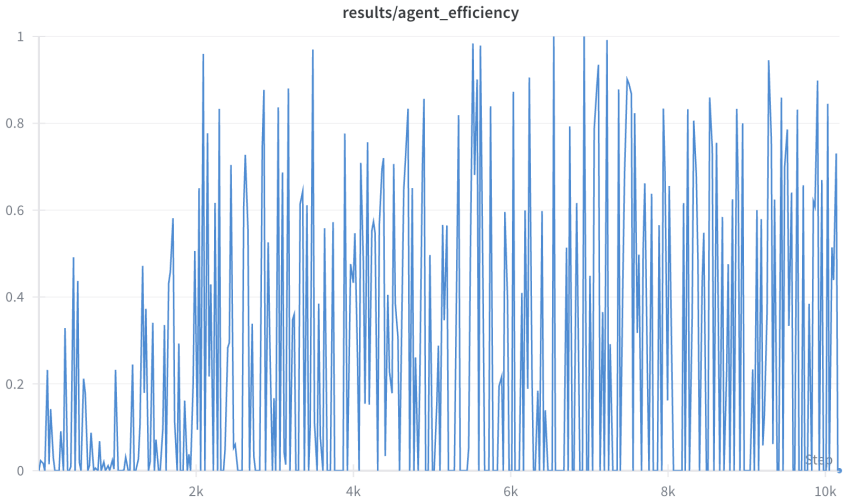


Figure 17: Evolution of the Agent's Efficiency per episode during the initial 10,000 training steps. The Y-axis represents the Performance Capture Ratio.

The "agent_efficiency" shown in Figure 17 is calculated at the end of each episode. First, the raw efficiency is computed as the ratio of the agent's PnL to the maximum potential PnL for that episode's position type (Long or Short):

$$\text{Efficiency}_{\text{raw}} = \frac{\text{PnL}_{\text{agent}}}{\text{PnL}_{\text{max\_potential}}}$$

where $\text{PnL}_{\text{agent}}$ is the final profit and loss of the agent for the episode, and $\text{PnL}_{\text{max\_potential}}$ is the perfect PnL (either 'max_pnl_long' or 'max_pnl_short'). To ensure the value remains a stable metric between 0 and 1, the algorithm then clips the result:

$$\text{Efficiency}_{\text{final}} = \text{clip}(\text{Efficiency}_{\text{raw}}, 0, 1)$$

This means that if the agent incurs a loss, its efficiency is floored at 0, and in the unlikely event it exceeds the theoretical maximum, its efficiency is capped at 1. The graph plots this final, clipped efficiency for each episode against the global training step. The high volatility is expected, as each episode presents a unique market scenario with varying difficulty. An upward trend in the average of these values over time is the primary indicator of successful learning.

Beyond the final evaluation metrics, it is insightful to use *callbacks* during the evaluation phase to compute indicators that contextualize the agent's performance. A callback is a function that is executed at specific moments (e.g., at every time step) and has access to the state of the environment and the agent's decisions. This allows us to calculate metrics that are not part of the reward function but offer a deeper analysis of the strategy's behavior.

PNL Difference and Smoothed Learning Curves

In addition to tracking efficiency against a perfect agent, we also track the agent's performance against the TWAP benchmark. The same callback computes also the final PnL difference ($\text{PnL}_{\text{agent}} - \text{PnL}_{\text{TWAP}}$) at the end of each episode. However, this raw per-episode metric is highly volatile due to the stochastic nature of the market, making it difficult to assess the learning trend.

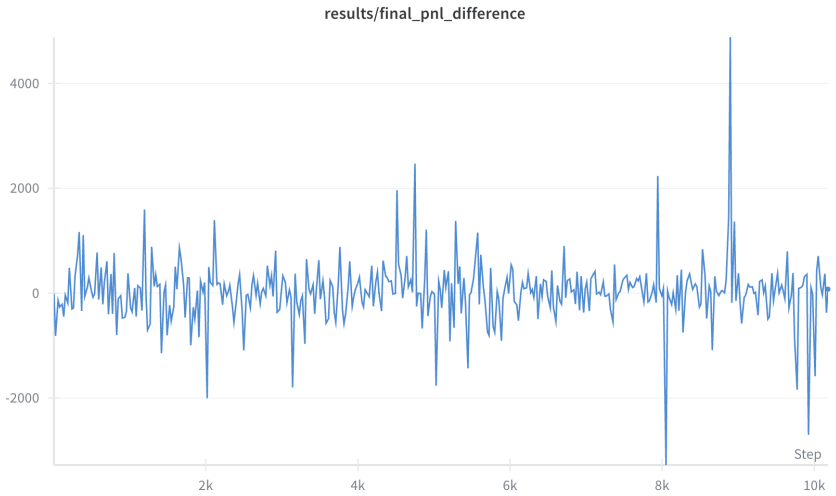
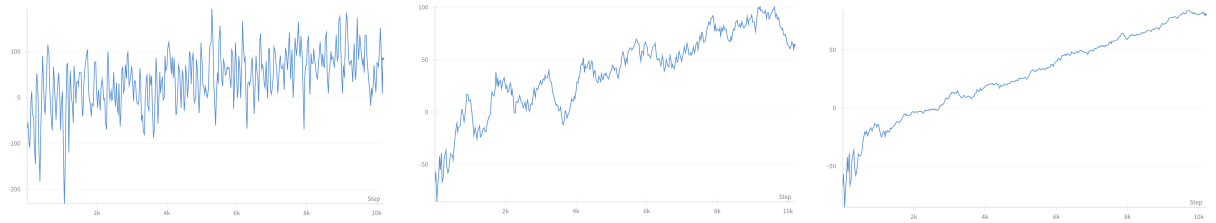


Figure 18: Raw final pnl difference per episode during the initial training steps. The high volatility, with performance fluctuating significantly from one episode to the next, masks any underlying learning trend and necessitates the use of smoothing techniques.

To get a clearer view of the agent's learning and performance, we apply moving averages to this PnL difference signal. This smoothing technique filters out short-term noise and reveals the underlying performance trend. Figure 19 shows the effect of this smoothing with three different window sizes.



(a) 100-episode window (b) 1000-episode window (c) 5000-episode window

Figure 19: Comparison of the smoothed PnL difference vs. TWAP using different moving average windows. As the window size (N) increases, the curve becomes smoother, revealing the agent's true learning trajectory.

The moving average is calculated with the following formula:

$$\bar{\Delta}_{\text{PnL}}(N) = \frac{1}{N} \sum_{i=k-N+1}^{k} \Delta_{\text{PnL},i}$$

where k is the index of the most recently completed episode, N is the window size (100, 1000, or 5000), and $\Delta_{\text{PnL},i}$ is the PnL difference for episode i.

As the figures clearly demonstrate, increasing the window size reveals a more stable and interpretable learning curve. While the 100-episode window is still quite volatile, the 1000-episode window shows a distinct upward trend, and the 5000-episode window confirms a robust and consistent ability for the agent to learn how to outperform the TWAP benchmark.

Visualizing the Agent's Liquidation Strategy

While numerical metrics like the PnL difference and efficiency tell us the final outcome of an episode, they do not reveal the underlying strategy the agent employed to achieve it. To gain a qualitative understanding of the agent's learned behavior, build trust in its decisions, and ensure it is not adopting unintended or overly risky strategies, it is essential to visualize its actions over the entire execution horizon.

This callback provides this crucial visual insight. At a fixed frequency during training, it captures the agent's inventory decay curve for a recently completed episode. It then generates a plot that contrasts two distinct trajectories:

1. **The Agent's Trajectory:** A continuous curve representing the agent's remaining position at each time step. Its shape reveals the agent's dynamic decisions.
2. **The TWAP Trajectory:** A **step-wise curve** representing the benchmark's behavior, where a fixed quantity is sold at regular intervals. This accurately models a real-world TWAP composed of child orders.

The implementation further enriches this visual by shading the areas between the two curves. It is essential to interpret these zones correctly: they represent the agent's **liquidation pace** relative to the benchmark, not direct PnL outperformance.

- A **green shaded area** signifies that the agent's remaining position is smaller than the TWAP's. This indicates a more **aggressive stance**, where the agent is liquidating faster than the benchmark.
- A **red shaded area** signifies that the agent's remaining position is larger than the TWAP's. This indicates a more **passive stance**, where the agent is liquidating slower than the benchmark.

The profitability of such a stance is entirely context-dependent. An aggressive (green) stance is beneficial in a falling market but detrimental in a rising one. The plot's title, which displays the final PnL difference and is **color-coded (green for a win, red for a loss)**, serves as the verdict, confirming whether the agent's chosen sequence of aggressive and passive stances was financially successful for that specific episode. To illustrate how the agent's strategy evolves and adapts, Figure 20 presents three examples of liquidation trajectories from different moments during training.

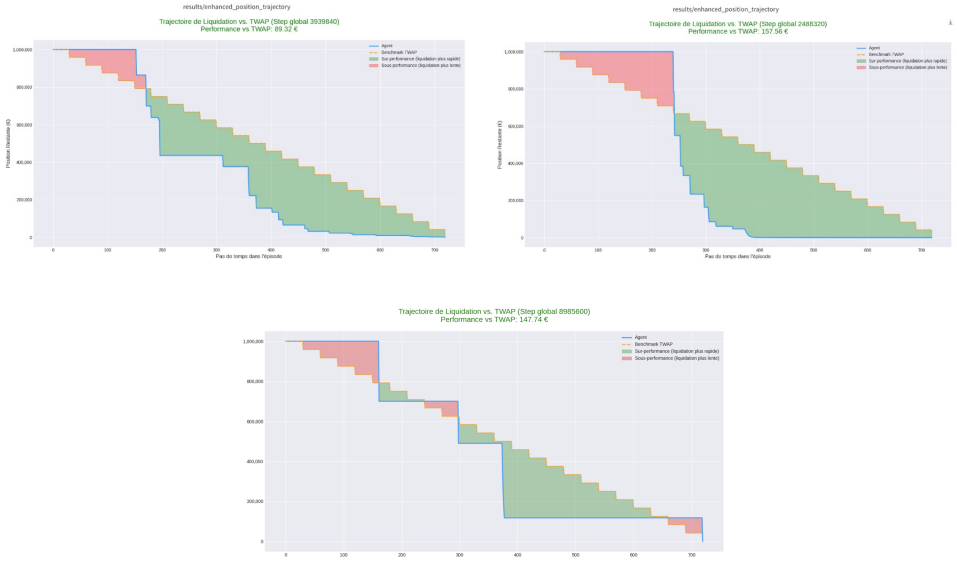


Figure 20: Examples of the agent's liquidation trajectories. Green shaded areas denote an aggressive stance (faster liquidation than TWAP), while red areas denote a passive stance (slower liquidation).

These examples highlight the agent's ability to learn and refine its strategic stance over time, adapting its selling pace to market conditions to financially outperform the static, step-wise TWAP benchmark.

3.8.6 Final Evaluation

The agent's training demonstrated a growing ability to outperform the TWAP benchmark strategy within the simulated environment. The final step of our work is to rigorously evaluate the final model's performance on a completely unseen dataset, to validate its generalization capability and to estimate its potential alpha under realistic conditions.

Saving the Agent and Normalization Statistics

At the conclusion of the training phase, which generated 10 million time steps for convergence, we selected the best-performing agent. The selection criterion was based on the highest and most stable average performance observed via the monitoring callbacks over the last 1000 training episodes.

The final saved agent does not only contain the neural network weights of the actor-critic but also includes the complete state of the vectorized normalization environment. This encapsulation ensures that during evaluation, the test data will be normalized using the statistics from the training set, thus preventing any data leakage and ensuring causal and correct assessment.

Test Protocol

The final evaluation was conducted on a strict out-of-sample and out-of-time dataset covering the entire month of **February 2025**. This period was never used during the training phases, which used data of January 2025, thus ensuring a robust assessment of the agent's generalization capabilities.

The evaluation was as follows:

1. **Model Loading:** The trained PPO agent and the VecNormalize environment with its frozen statistics are loaded.
2. **Deterministic Evaluation:** Unlike training where actions are sampled from the actor's probability distribution to encourage exploration, evaluation is performed in a deterministic mode. The agent systematically chooses the action with the highest probability, which corresponds to exploiting the learned policy.
3. **Execution of Test Episodes:** We executed a total of **3000 complete liquidation episodes** by aggregating two independent evaluation runs. For each episode, a starting point is randomly chosen from the February 2025 test dataset, and the position type (Long or Short) is also randomized.
4. **Metrics Collection:** For each episode, we recorded the agent's final PnL, the TWAP strategy's PnL, and whether the liquidation was completed.

Results Analysis

The aggregated evaluation results over the 3000 test episodes confirm that the agent has learned a robust and profitable execution strategy.

- **Positive and Significant Alpha:** The agent succeeded in generating an average PnL difference of **+42.19 $** per episode against the TWAP benchmark. This demonstrates a clear and consistent ability to add value. However, the standard deviation of this difference is very high at **321.55 $**, which indicates a wide dispersion of outcomes and an asymmetric return profile rather than consistent small gains.
- **Win Rate:** Over the 3000 episodes, the agent outperformed the TWAP strategy in **55.57% of cases**. This win rate which is above 50%, proves that the agent possesses a genuine predictive edge.
- **Liquidation Reliability:** The agent liquidated 99.5% of its initial position in **100% of the episodes**. The quadratic penalty for incomplete liquidation was thus effective in teaching it to always complete its mission within the allotted time.

Beyond the direct PnL, we also analyzed the agent's efficiency, as shown in Figure 21, the metric is distributed across a wide range, with a mean capture rate around 30%. This confirms that while the agent does not achieve perfect execution, it consistently extracts a meaningful fraction of the available market alpha.

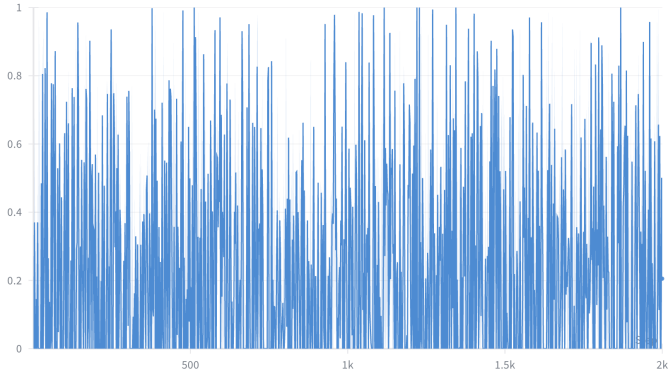


Figure 21: Distribution of the agent's efficiency on the February 2025 test set. Efficiency is measured as the ratio of the agent's realized PnL to the maximum potential PnL for each episode.

The qualitative analysis of the liquidation trajectories revealed an intelligent strategic behavior. The agent learned to exploit the predictions from the supervised models: it adopted a **passive stance** (liquidating slower than TWAP) when the models predicted a high probability of a favorable trend continuation, thereby maximizing gains. Conversely, it became very **aggressive** (liquidating faster than TWAP) when mean-reversion signals and its internal models predicted an imminent market reversal. This adaptive strategy, while highly profitable on average, naturally leads to a wide distribution of outcomes, as confirmed by the high standard deviation of the PnL difference. This highlights the inherent trade-off in the learned strategy: its profitability is fundamentally linked to the accuracy of its underlying signals within a given market regime, leading to high-reward but also high-risk outcomes.

4 Conclusion

This master's thesis undertook a central challenge in modern quantitative finance: to move beyond mere market prediction to the prescription of an optimal action policy. The guiding vision of this work was to build a robust methodological bridge between these two realms. To achieve this, we adopted a two-stage approach, which forms the backbone of our contribution. The first phase was dedicated to the engineering of original features and the rigorous validation of market hypotheses (mean reversion and momentum) to construct a reliable predictive foundation. The second, more ambitious phase involved using these signals not as an end in themselves, but as the sensory input for an autonomous agent, trained via reinforcement learning to navigate a complex liquidation environment. The main contribution of this work, therefore, lies in the demonstration of a complete and functional pipeline, spanning from the extraction of alpha signals to the implementation of an adaptive execution strategy.

Empirically, we demonstrated that the engineered features, particularly the normalized NRP oscillator and the adaptive ATF approach, possess statistically significant predictive power. These signals provided a solid foundation for our supervised learning models. The PPO agent, trained within our simulated liquidation environment, validated our end-to-end approach. The final out-of-sample evaluation confirmed its ability to learn a profitable strategy, generating an average alpha of **+$42.19**/million $ to execute, per episode against the TWAP benchmark,

with a win rate of **55.57%**. This result proves that it is possible to train an agent capable of intelligently leveraging predictions to outperform standard algorithmic strategies.

It is, however, essential to acknowledge the limitations of our approach, primarily the assumption of a simulation environment with no market impact. This simplification, while necessary for our study, naturally paves the way for several exciting avenues for research.

Future Work

The positive outcomes of this project open up new avenues for research and development:

- **Exploring Other RL Algorithms:** Although PPO provided excellent results, more recent off-policy algorithms like **Soft Actor-Critic (SAC)** could offer better sample efficiency and potentially converge to even more performant strategies.
- **Improving Environment Realism:** The main assumption of our simulation is the absence of market impact. A major step forward would be to develop a more complex simulation environment, for example based on Agent-Based Models, where the order book would react dynamically to the agent's transactions.
- **Real-World Deployment:** The next logical step would be to connect the agent to a live trading environment, first in a paper trading setup to validate its behavior without financial risk, and then potentially in production with strict risk limits. This would allow for confronting the learned strategy with the real and unpredictable dynamics of the market.
- **Multi-Asset Extension:** The current framework focuses on a single asset. Extending it to an allocation and execution problem on a portfolio of several assets is a complex but very promising research direction.

In conclusion, this work establishes a solid foundation and demonstrates that reinforcement learning, when fueled by rigorous feature engineering, is a viable and extremely promising approach to solving the challenges of optimal execution in modern financial markets.

5 Bibliography

References

[1] Agrawal, S., et al. (2025). Crypto price prediction using LSTM+XGBoost. *IEEE Access, 13*, 11245-11258.

[2] Bellman, R. (1957). *Dynamic Programming*. Princeton University Press.

[3] Berti, L., & Kasneci, G. (2025). *TLOB: A novel transformer model with dual attention for stock price trend prediction with limit order book data*. Technical University of Munich.

[4] Breiman, L. (2001). Random forests. *Machine Learning, 45*(1), 5-32.

[5] Brock, W. A., Dechert, W. D., Scheinkman, J. A., & LeBaron, B. (1996). A test for independence based on the correlation dimension. *Econometric Reviews, 15*(3), 197–235.

[6] Carhart, M. M. (1997). On persistence in mutual fund performance. *The Journal of Finance, 52*(1), 57-82.

[7] Chakravorty, A., & Elsayed, N. (2025). *A comparative study of machine learning algorithms for stock price prediction using insider trading data* (arXiv:2502.08728). arXiv.

[8] Dempster, M. A. H., & Leemans, V. (2006). An automated FX trading system using adaptive reinforcement learning. *Intelligent Systems in Accounting, Finance and Management, 14*(1-2), 3-15.

[9] Engle, R. F. (1982). Autoregressive conditional heteroskedasticity with estimates of the variance of United Kingdom inflation. *Econometrica, 50*(4), 987–1007.

[10] Exley, J., Mehta, S., & Smith, A. (2004). *Mean Reversion.* Presented to the Faculty & Institute of Actuaries Finance and Investment Conference, Brussels.

[11] Fama, E. F. (1965). The behavior of stock-market prices. *The Journal of Business, 38*(1), 34–105.

[12] Greene, M. T., & Fielitz, B. D. (1977). Long-term dependence in common stock returns. *Journal of Financial Economics, 4*(3), 339–349.

[13] Guennioui, A., & El Beqqali, O. (2023). A global XGBoost-LightGBM ensemble model for stock price prediction. *Procedia Computer Science, 220*, 243-252.

[14] Haarnoja, T., Zhou, A., Abbeel, P., & Levine, S. (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *Proceedings of the 35th International Conference on Machine Learning, 84*, 1861-1870.

[15] Hurst, H. E. (1951). Long-term storage capacity of reservoirs. *Transactions of the American Society of Civil Engineers, 116*(1), 770–799.

[16] Jain, A., et al. (2024). *SARF: Sentiment-augmented random forest for stock price movement prediction* (arXiv:2402.01328). arXiv.

[17] Jegadeesh, N., & Titman, S. (1993). Returns to buying winners and selling losers: Implications for stock market efficiency. *The Journal of Finance, 48*(1), 65-91.

[18] Jiang, Z., Xu, D., & Liang, J. (2017). *A deep reinforcement learning framework for the financial portfolio management problem* (arXiv:1706.10059). arXiv.

[19] Kalamkar, P., Banerjee, A., & Desarkar, M. S. (2022). Enhancing GARCH model's volatility prediction using news sentiment. *Expert Systems with Applications, 207*, 117866.

[20] Konda, V. R., & Tsitsiklis, J. N. (2000). Actor-critic algorithms. *Advances in Neural Information Processing Systems, 12.*

[21] Kumar, P. (2022). *Deep reinforcement learning for high-frequency market making* (arXiv:2209.09386). arXiv.

[22] Lee, J., & Kim, D. (2025). A two-stage forecasting model using random forest subset-based feature selection and BiGRU with attention mechanism: Application to stock indices. *Journal of Big Data, 12*(1), 14.

[23] Li, S., et al. (2023). *MAT: A modality-aware transformer for multimodal time series forecasting* (arXiv:2310.09741). arXiv.

[24] Li, Y. (2023). *A heavy-tail and non-linear story of asset price comovement* (BIS Working Papers, No 1092). Bank for International Settlements.

[25] Lillicrap, T. P., et al. (2015). *Continuous control with deep reinforcement learning* (arXiv:1509.02971). arXiv.

[26] Liu, X., et al. (2021). FinRL: A deep reinforcement learning framework to automate trading in quantitative finance. *Proceedings of the 2nd ACM International Conference on AI in Finance*, 1-9.

[27] Liu, Z., et al. (2025). MCI-GRU: A novel stock prediction model based on multi-head cross-attention and improved GRU. *Information Sciences, 655*, 120567.

[28] Lo, A. W., & MacKinlay, A. C. (1988). Stock market prices do not follow random walks: Evidence from a simple specification test. *The Review of Financial Studies, 1*(1), 41–66.

[29] Lucarelli, G., & Borrotti, M. (2019). A deep reinforcement learning approach for automated cryptocurrency trading. In G. A. Papakostas, & E. Keramopoulos (Eds.), *Artificial Intelligence Applications and Innovations* (pp. 247-258). Springer.

[30] Ma, T., et al. (2025). A hybrid LSTM-GRU model for stock price prediction on the Tehran stock exchange. *Applied Soft Computing, 154*, 110456.

[31] Mandelbrot, B. (1963). The variation of certain speculative prices. *The Journal of Business, 36*(4), 394–419.

[32] Meese, R. A., & Rogoff, K. (1983). Empirical exchange rate models of the 1970s: Do they fit out of sample? *Journal of International Economics, 14*(1-2), 3–24.

[33] Mittal, A., & Goel, A. (2011). *Stock prediction using Twitter sentiment analysis*. Stanford University.

[34] Mnih, V., et al. (2016). Asynchronous methods for deep reinforcement learning. *Proceedings of the 33rd International Conference on Machine Learning, 48*, 1928-1937.

[35] Mnih, V., et al. (2015). Human-level control through deep reinforcement learning. *Nature, 518*(7540), 529–533.

[36] Moody, J., & Saffell, M. (1998). Reinforcement learning for trading. *Advances in Neural Information Processing Systems, 11*.

[37] Nelson, D. M., Pereira, A. C. M., & de Oliveira, R. A. (2017). Stock market's price movement prediction with LSTM neural networks. *2017 International Joint Conference on Neural Networks (IJCNN)*, 1-8.

[38] Neuneier, R. (1996). Optimal asset allocation using adaptive dynamic programming. *Advances in Neural Information Processing Systems, 8*.

[39] Nevmyvaka, Y., Feng, Y., & Kearns, M. (2006). Reinforcement learning for optimized trade execution. *Proceedings of the 23rd International Conference on Machine Learning*, 673–680.

[40] Ning, B., Lin, F. H. T., & Jaimungal, S. (2018). Double deep Q-learning for optimal execution. *Applied Mathematical Finance, 28*(4), 361–380.

[41] Ning, B., Chakraborty, P., & Lee, K. (2023). *Optimal entry and exit with signature in statistical arbitrage* (arXiv:2309.16008). arXiv.

[42] Ozbayoglu, A. M., Gudelek, M. U., & Sezer, O. B. (2020). Deep learning for financial applications: A survey. *Applied Soft Computing, 93*, 106384.

[43] Patel, R. (2025). A comparative analysis of ARIMA and LSTM models for stock market prediction. *International Journal of Financial Studies, 13*(1), 22.

[44] Rasekh, A., et al. (2023). A comparative study of machine learning models for short-term stock price prediction. *Expert Systems with Applications, 215*, 119345.

[45] Rummery, G. A., & Niranjan, M. (1994). *On-line Q-learning using connectionist systems* (Technical Report CUED/F-INFENG/TR 166). Cambridge University Engineering Department.

[46] Schulman, J., et al. (2017). *Proximal policy optimization algorithms* (arXiv:1707.06347). arXiv.

[47] Sezer, O. B., & Ozbayoglu, A. M. (2022). Explainable AI for financial time series prediction: A survey. *Financial Innovation, 8*(1), 1-35.

[48] Spooner, T., et al. (2018). Market making via reinforcement learning. *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, 434–442.

[49] Sun, S., et al. (2023). *TradeMaster: A holistic quantitative trading platform empowered by reinforcement learning.* Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track.

[50] Vaswani, A., et al. (2017). Attention is all you need. *Advances in Neural Information Processing Systems, 30*, 5998–6008.

[51] Wang, Y., et al. (2023). QGAF: A quantum-inspired method for encoding time series as images. *IEEE Transactions on Knowledge and Data Engineering.*

[52] Wang, Z., et al. (2024). *Can ChatGPT-like models be a good stock-specific news sentiment analyzer?* (arXiv:2403.01838). arXiv.

[53] Watkins, C. J. C. H. (1989). *Learning from delayed rewards* [Doctoral dissertation, King's College, Cambridge].

[54] Wen, Q., et al. (2022). *Transformers in time series: A survey* (arXiv:2202.07125). arXiv.

[55] Zhou, H., et al. (2021). Informer: Beyond efficient transformer for long sequence time-series forecasting. *Proceedings of the AAAI Conference on Artificial Intelligence, 35*(12), 11106-11115.