COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI

Msc Big Data

Master's final project

Portfolio optimization with Machine Learning Algorithms

Author
Alberto Sáez-Royuela Ariza

Supervised by
Alejandro Polo Molina

Madrid
June 2025

Abstract

Portfolio optimization is a core problem in quantitative finance and has increasingly leveraged machine learning to improve forecasting and allocation. In this thesis, we show how deep neural networks can enhance risk-adjusted asset allocation by predicting optimal weights directly rather than modeling prices alone. We compare a classical Markowitz mean–variance allocator, using historical means and covariances, with a two-stage deep learning pipeline: first an LSTM network trained on 60-day windows to forecast next-day prices for five large-cap tech stocks, and second a feedforward NN that maps these forecasts to valid, long-only weight vectors via softmax.

Training uses data from 2015–2023, and both strategies are tested on 2024 under identical constraints. The LSTM+NN approach achieves a 35.40 % annualized return and a 1.67 Sharpe ratio, versus 22.18 % and 1.18 for Markowitz, while maintaining comparable volatility and displaying shallower drawdowns during volatility spikes. These results suggest that directly predicting allocations with deep learning yields more adaptive, robust portfolios in non-stationary markets.

I want to thank my family for their unconditional support and for always believing in me. I also thank my friends for their companionship and encouragement at all times, which made this project more bearable.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

A *portfolio* is a structured assembly of financial instruments, such as equities, bonds, commodities, ETFs and derivatives, designed to meet specific investment objectives. The primary goal of *portfolio optimization* is to determine the weight allocation that maximizes expected return for a given level of risk, or equivalently minimizes risk for a target return. This challenge is central to asset management, impacting the long-term wealth of pension funds, endowments and private investors alike.

Since Markowitz's Mean–Variance framework introduced in 1952 the concept of constructing an *efficient frontier* through quadratic optimization of expected return against portfolio variance [1], researchers sought to simplify practical implementation. Building on this foundation, Sharpe (1964) and Lintner (1965) developed the Capital Asset Pricing Model (CAPM), which reduced portfolio choice to a combination of the risk-free asset and a single market portfolio by linking each asset's expected return to its systematic risk (beta) [2], [3]. Together, these models form the bedrock of modern portfolio theory, prized for their closed-form solutions and intuitive economic interpretation.

However, real financial markets deviate substantially from the assumptions on which these frameworks are based. Asset returns frequently show *fat tails*, skewness and clustering of volatility, while correlations evolve over time and under stress exhibit abrupt regime shifts. Moreover, the estimation of expected returns and covariances from finite historical samples introduces substantial noise: portfolios optimized on these estimates often display extreme, non-intuitive weights and perform poorly out of sample [4].

Meanwhile, the proliferation of high-frequency market data, alternative information sources (e.g. news sentiment, macroeconomic indicators) and advances in computational power have encouraged the rise of *Machine Learning* (ML) in finance. Deep Neural Networks (DNNs) and Recurrent Neural Networks (RNNs) have demonstrated the ability to learn complex, non-linear relationships and capture temporal dependencies that classical models cannot [5], [6]. By leveraging these capabilities, it becomes possible to generate more accurate forecasts of returns and risk metrics, which can then feed into an optimized allocation routine to produce portfolios that adapt dynamically to evolving market conditions.

This document offers a concise yet comprehensive overview of our Master's Final Project. First, we review the traditional foundations of portfolio optimization in Section 2. In Section 3, we motivate the specific research gap and discuss the classical models. Section 4 presents the theoretical foundations of deep neural networks, and Section 5 describes our methodology and compares the classical approach with the proposed model. Moreover, the code of the project can be found at https://github.com/AlbertoSaezR/Portfolio-Optimization-

1.1 Literature Review

The foundation of modern portfolio optimization was laid by Harry Markowitz in 1952 [1], who introduced the idea that portfolio risk can be measured by the variance of returns and that, by carefully combining assets, an investor can achieve an "efficient frontier" of portfolios offering the highest expected return for each level of risk [1]. In practice, this means that instead of holding a single asset, investors should allocate across a diversified set of assets to reduce overall volatility without sacrificing returns. Building on this, the CAPM of Sharpe and Lintner in the 1960s provided a market-equilibrium framework. Under idealized assumptions, all investors hold combinations of the risk-free asset and a single market portfolio, and each asset's expected return is related to its systematic exposure ("beta") to that market portfolio [2], [3]. CAPM thus gave a clear economic interpretation of risk and return and became a cornerstone of financial theory and practice.

However, applying these models to real data revealed persistent challenges. Estimating expected returns and covariances from finite historical samples introduces considerable noise: portfolios optimized on these estimates often display extreme, non-intuitive weights and perform poorly out of sample [4]. To mitigate this, shrinkage techniques, such as Ledoit–Wolf, blend sample estimates with structured targets to produce more stable covariance matrices [7]. Moreover, robust optimization methods incorporate worst-case scenarios into the optimization, guarding against model misspecification [8]. The Black–Litterman approach further refines expected returns by combining market-implied equilibrium returns with investor-specified views, yielding allocations that align better with both theory and practitioner intuition [9]. Despite these enhancements, empirical studies have shown that when estimation error is high, even a simple 1/N equal-weight strategy, allocating the same weight $1/N$ to each of the N assets, can outperform more elaborate optimizers out of sample [10]. This underscores the importance of balancing theoretical optimality with the practical limitations of real-world data.

On the other hand, ML techniques have gained traction in finance by addressing the limitations of purely parametric models. Early work applied linear and regularized regression methods, such as Ridge and Lasso, to forecast asset returns and volatility [11]. These regression approaches stabilize coefficient estimates by penalizing large weights, thereby reducing overfitting. In parallel, tree-based ensemble methods like Random Forest and Gradient Boosting Machines were introduced to capture richer, non-linear relationships in financial time series without requiring Gaussian assumptions [12], [13]. By incorporating feature selection and regularization, these models mitigate overfitting and improve out-of-sample performance.

Building on these foundations, DNNs have demonstrated the ability to approximate highly non-linear functions when provided with sufficient data and model capacity [5]. In particular, fully connected architectures (such as multilayer perceptrons, or MLPs) and sequence-based models, like Temporal Convolutional Networks (TCNs), have outperformed shallow learners in forecasting price movements and volatility patterns on large-scale datasets [14], [15]. These deep architectures excel at learning complex relationships within historical market data that classical models often fail to capture.

At the same time, Reinforcement Learning (RL) has emerged as a paradigm for treating trading as a sequential decision-making problem [16], [17]. In this framework, an agent observes market states and learns buy, sell, or hold actions to maximize a cumulative reward that can account for transaction costs and risk penalties. More recent studies [18], have combined deep policy-gradient methods with feature-selection pipelines to outperform both classical and supervised-learning approaches in terms of risk-adjusted returns. By interacting with simulated market environments, RL agents adapt their strategies dynamically, reacting to changing conditions without relying on fixed distributional assumptions.

Although these ML methodologies have shown promise in forecasting, they do not abandon the strengths of classical financial theory. Hybrid approaches have therefore been developed to integrate data-driven predictions directly into portfolio optimization routines. For example, ML models, such as DNNs or ensemble learners, can generate refined estimates of expected returns μ and covariance matrices Σ, which are then used as inputs to the traditional Mean–Variance optimizer. By doing so, these "prediction-enhanced" methods reduce estimation noise and stabilize allocations, leading to improved out-of-sample performance [19].

Similarly, evolutionary search techniques guided by ML-based fitness functions offer a robust alternative for exploring the non-convex allocation landscape [20]. In this approach, Genetic Algorithms or Particle Swarm methods evaluate candidate portfolios using fitness scores derived from ML forecasts (such as predicted Sharpe ratios) and iteratively refine the allocation to identify robust weight vectors. This process captures non-linear interactions and adapts to changing market regimes without discarding established financial principles.

Finally, end-to-end allocation networks represent a unification of forecasting and optimization within a single neural architecture [21]. In these models, a neural network (NN) is trained to predict market dynamics and directly output portfolio weights, optimizing financial objectives (for instance, the Sharpe ratio or CVaR) through backpropagation. By embedding optimization logic into the network itself, end-to-end approaches bypass the need for a separate optimization step, allowing the model to learn allocation strategies that account for both return forecasts and risk metrics simultaneously.

In summary, the evolution of portfolio theory has moved from classical, parametric frameworks, such as Mean–Variance and CAPM, through enhancements like shrinkage, robust optimization, and Black–Litterman, to modern, data-driven methods that leverage ML and hybrid optimization. The next chapter will explore how these insights motivate a specific research gap and inform the design of a hybrid LSTM-based model to improve return and risk forecasts in dynamic market environments.

Chapter 2

Basic financial concepts

2.1 Financial Fundamentals and Investment Principles

In this chapter, we introduce the fundamental concepts of investment portfolios that underpin the analysis and methodology presented in subsequent sections. Specifically, we define what constitutes a portfolio, discuss the factors driving asset selection, and examine management strategies, asset allocation profiles, and diversification techniques. These financial principles form the foundation for the models and experiments developed later in this work.

2.1.1 What Is an Investment Portfolio and What Drives Its Selection?

First of all, an investment portfolio is a collection of financial assets owned by an individual, a company, or an investment fund. Such assets may include equities, bonds, commodities, mutual funds, ETFs, currencies, cryptocurrencies, and index trackers. Moreover, portfolios can also contain tangible assets, such as art, real estate, and land, depending on the investor's objectives and constraints.

The primary objective of any investment portfolio is to generate a return on the capital invested; to achieve this goal, proper diversification is essential. In particular, three key factors drive portfolio construction. First, investment goals determine how success is measured. For some investors, success means reaching a specified portfolio value, while for others it may be earning a steady annual dividend income. Second, the time horizon indicates the expected duration of the investment, which can range from a few months to several decades. Longer horizons often tolerate greater risk in pursuit of higher returns. Finally, the risk profile reflects an investor's willingness to accept volatility. In particular, higher-risk portfolios may offer the potential for greater returns but also carry a higher likelihood of significant drawdowns, whereas more conservative investors prefer lower volatility even if it means accepting more modest gains.

Depending on the balance of the three factors above, a portfolio's asset allocation will reflect the relationship between desired profitability, risk, and success timeframe.

2.1.2 Investment Management Types

Portfolio management can be carried out according to several distinct methodologies, each characterized by the level of investor involvement, the decision-making process, and the underlying investment philosophy. This section presents an overview of the principal approaches, active management, passive management, and collective investment schemes, and discusses their key features within the context of contemporary portfolio theory.

Regardless of who is in charge of the portfolio, there are three management methods, each with its own advantages and disadvantages. Each approach offers a distinct balance of decision-making autonomy, flexibility, and potential for excess return. The following subsections examine these management methods in greater detail, beginning with active portfolio management.

Active Portfolio Management

In active portfolio management, the manager takes on the responsibility of analyzing and selecting the securities and assets that constitute the portfolio at any given time. Thus, the manager decides what to buy, when to buy it, at what price, and when to sell. Depending on how market conditions align with the underlying investment hypothesis, this process may involve adjusting the portfolio daily, annually, or at any interval in between. Consequently, active management offers the flexibility to respond quickly to price fluctuations or emerging opportunities.

Moreover, the primary objective of active management is to achieve returns that exceed those of a chosen benchmark. For instance, a U.S.–based portfolio would aim to outperform a leading index such as the S&P 500. In contrast, a Spanish investor might measure success against the IBEX 35, while a U.K. investor could compare performance to the FTSE 100. Therefore, by continuously monitoring market movements and revising positions, active managers seek to generate higher risk-adjusted returns than a passive index-tracking approach.

Passive Portfolio Management

In contrast to the active approach discussed above, passive portfolio management is an investment strategy that aims to replicate the performance of a specific market index (either in whole or in part), such as the S&P 500, rather than trying to outperform it.

This approach involves selecting a diversified mix of assets that closely match the composition of the chosen index and then holding them over the long term. By doing so, investors can benefit from the general upward trend of the market while minimizing trading costs and the need for constant monitoring. Therefore, the objective is to achieve steady growth and reduce risk through broad diversification, making this strategy attractive for those who prefer a more hands-off investment style.

2.1.3 Asset Allocation Profiles

Beyond the choice between active and passive management, portfolio composition also depends on how assets are allocated to match an investor's objectives. The percentage of a portfolio's asset type weightings can suggest not only the investor's risk profile but also their time horizon. Hence, the different approaches that could be considered include:

- **Aggressive Focus Portfolio:** 70–90% in stocks, 10–30% in government bonds, 10% in cash to be able to take advantage of any investment that arises.

- **Balanced Portfolio:** Bond exposure is increased as stock percentage decreases. This leans more towards 50–70% in stocks, 30–50% in government bonds, with the remaining 10% held in cash/deposits/interest-bearing accounts.

- **Conservative Portfolio:** Usually found with investors who have a low risk appetite, a short time-horizon for capital drawdowns (or both), a conservative portfolio is usually no more than 20% in shares, with a heftier 50–60% in bonds, and more available cash than the other portfolios (usually around 20–30%).

- **Very Conservative Portfolio:** While uncommon, it's also possible to have a portfolio which consists of 0% stocks, 50% in fixed income (such as bonds) and 50% in deposits and remunerated accounts (cash).

2.1.4 Investment Portfolio Diversification and rebalancing

Diversification is a best-practice that involves spreading investments across various asset classes, sectors, and geographic regions to reduce risk. This approach mitigates risk by diversifying investments, thereby reducing the likelihood that underperformance in a single asset will significantly affect the overall portfolio. Portfolio diversification reduces risk, stabilizes returns and therefore helps to preserve capital.

Examples of Diversification

- **Assets:** The ideal is to spread the capital among several asset classes, such as stocks, indices, currencies, bonds, and commodities. The selection of the assets and their percentage would depend on the risk profile of the portfolio.
- **Markets:** Diversification can also happen within the same asset class. For example, the equity (stocks) portion can be distributed across different sectors (banks, energy, electricity, etc.).
- **Geography:** Invest in both domestic and international markets to reduce the risk linked to a single country's economic performance.

Furthermore, periodic rebalancing serves to realign a portfolio's weightings whenever certain investments outperform others and shift the original allocation. Over time, this drift can move a portfolio away from its intended risk–return profile. By rebalancing, sometimes called readjusting, investors sell portions of assets that have grown beyond target allocations and buy those that have fallen below. Hence, rebalancing ensures that the portfolio remains aligned with financial goals and the investor's tolerance for risk.

Chapter 3

Classical Models for Investment Portfolios

The objective of this section is to present the most influential classical portfolio-optimization frameworks, to analyse their underlying assumptions and limitations, and to motivate the adoption of DNNs for capturing complex, non-linear dependencies in order to improve return forecasts and risk management.

3.1 Mean–Variance Portfolio Optimization (Markowitz)

The Mean–Variance framework, introduced by Harry Markowitz in 1952 [1], represents the cornerstone of modern portfolio theory. Its primary insight is that investors care simultaneously about expected return and risk, where risk is quantified as the variance (or standard deviation) of portfolio returns. Under this paradigm, an investor seeks the portfolio weights $w = (w_1, \ldots, w_N)^\top$ that minimize portfolio variance for a given target return, or equivalently maximize expected return for a given level of risk.

Consequently, let

$$\mu = \big(\mu_1, \ldots, \mu_N\big)^\top \quad \text{and} \quad \Sigma = \big[\sigma_{ij}\big]_{i,j=1}^{N}$$

denote the vector of expected returns and the covariance matrix of returns for N assets, respectively. Then, the classical mean–variance optimization problem can be written as

$$\min_{w \in \mathbb{R}^N} \quad w^\top \Sigma\, w \quad \text{subject to} \quad w^\top \mu = \mu_p, \quad \mathbf{1}^\top w = 1, \qquad w \succeq 0,$$

where

- $w^\top \Sigma\, w$ is the portfolio variance

- $w^\top \mu = \mu_p$ enforces that the portfolio's expected return equals a pre-specified target μ_p
- $\mathbf{1}^\top w = 1$ ensures full investment of capital
- $w \succeq 0$ prohibits short-selling (non-negative weights).

Hence, by solving this quadratic program repeatedly for different levels of μ_p, one traces out the *efficient frontier*, which is the set of portfolios offering the maximum possible expected return for each level of risk.

Therefore, the economic interpretation of the model is as follows. The efficient frontier encapsulates the trade-off between risk and return: portfolios below the frontier are sub-optimal (they offer lower return for the same risk), while those above the frontier are unattainable under the model's assumptions. An investor's personal risk appetite then determines the appropriate point on this curve.

In this regard, the key assumptions and limitations of the Mean–Variance model are as follows:

1. *Returns are jointly normally distributed.* This justifies using variance as a complete measure of risk.
2. *Expected returns μ and covariances Σ are known and stationary.* In practice, these parameters must be estimated from historical data, often with significant noise.
3. *Investors are risk-averse and care only about mean and variance.* Other higher-moment preferences (skewness, kurtosis) are ignored.
4. *No transaction costs, taxes, or market frictions.* Trading is assumed frictionless.

The *estimation error* in μ and Σ is particularly problematic in high-dimensional settings (many assets, limited data), leading to extreme and unstable weight vectors that perform poorly out of sample. This "error maximization" property has motivated a rich literature on regularization (shrinkage), robust optimization, and Bayesian enhancements.

From a computational perspective, as the problem is a convex quadratic program, efficient solvers (e.g. CVXPY, OSQP) can handle portfolios of hundreds or thousands of assets in seconds. However, the quality of the resulting allocation is fundamentally tied to the reliability of the input estimates, underscoring the need for advanced ML methods to improve parameter forecasting.

3.2 Other Classical Models

Beyond the Mean–Variance framework, several influential models have shaped portfolio theory:

- **Capital Asset Pricing Model (CAPM)** [2], [3]: Links each asset's expected return to its sensitivity (β) to the market portfolio; underpins the Capital Market Line but struggles with empirical anomalies (e.g. size, value) [22].
- **Arbitrage Pricing Theory (APT)** [23]: Extends CAPM by expressing returns as a linear combination of multiple macroeconomic factors (e.g. GDP or Gross Domestic Product growth, inflation), allowing richer risk-factor structures.
- **Black–Litterman Model** [9]: Introduces a Bayesian blend of market-implied equilibrium returns and investor views, smoothing extreme allocations and improving intuitive portfolio adjustments.
- **Risk Parity** [24]: Equalizes risk contributions across assets by inversely weighting by volatility, often leading to overweighting of low-volatility bonds and underweighting of equities.
- **Conditional Value-at-Risk (CVaR) Optimization** [25]: Focuses on downside risk by minimizing expected losses beyond a specified quantile, offering explicit tail-risk control compared to variance-based approaches.

Each of these approaches addresses specific limitations of the original Mean–Variance model, whether by incorporating multiple risk factors, blending subjective views, balancing risk contributions, or focusing on tail risk, yet they remain sensitive to input estimation error and often rely on similar underlying data assumptions.

Having surveyed the ways in which classical frameworks address risk and return, yet remain constrained by distributional and estimation assumptions, we now turn to ML techniques that relax these assumptions and can capture richer data patterns.

Although the literature includes numerous extensions, such as multi-period optimization, dynamic CVaR models, and robust factor-uncertainty formulations, this project will focus exclusively on the classical Mean–Variance (Markowitz) method and its comparison with advanced ML models. All other models are beyond the scope of this TFM and may be explored in future research.

Chapter 4

Deep Neural Networks: Theoretical Foundations

4.1 Introduction to Deep Neural Networks

In the previous chapters, we have defined the financial optimization problem, reviewed classical portfolio models and explored the state of the art in data-driven techniques. We now turn to the core ML architectures that will underpin our hybrid optimization framework: DNNs, with a particular focus on RNNs and its gated variants LSTM.

DNNs are composed of multiple layers of interconnected computational units (neurons) that apply successive affine transformations and non-linear activations to input features. Thanks to the Universal Approximation Theorem [26], even relatively shallow networks can approximate arbitrary continuous functions. Furthermore, modern deep architectures, enabled by advances in optimization algorithms, regularization techniques, and GPU acceleration, can model extremely complex patterns in high-dimensional data [5], [26].

In quantitative finance, DNNs have demonstrated state-of-the-art performance for return and volatility forecasting, especially when combined with rich feature sets (technical indicators, macroeconomic variables, sentiment scores). However, financial time series exhibit strong temporal dependencies and regime shifts that feedforward networks cannot capture effectively. This motivates the adoption of sequence-aware models:

- **Recurrent Neural Networks (RNNs)** introduce hidden-state feedback loops, enabling the network to retain information across time steps and model sequential dependencies.
- **Long Short–Term Memory (LSTM)** and **Gated Recurrent Units**

(GRU) enhance standard RNNs with gating mechanisms that mitigate vanishing and exploding gradient issues, allowing for the learning of long-range temporal patterns.

In the following sections, we will first review the mathematical structure and training algorithms of feedforward DNNs to establish common ground. We will then delve deeply into RNNs and their gated variants, explaining their internal mechanisms, advantages and implementation considerations.

4.2 Feedforward (Multilayer Perceptron) Networks

A *feedforward network* (or Multilayer Perceptron (MLP) Figure 4.1) is organized into an input layer, one or more hidden layers, and an output layer. Each layer l computes

$$\mathbf{h}^{(l)} = \phi\big(W^{(l)}\,\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)}\big),$$

where $W^{(l)}$ and $\mathbf{b}^{(l)}$ are the weight matrix and bias vector, ϕ is a non-linear activation (e.g. ReLU, tanh, sigmoid), and $\mathbf{h}^{(0)}$ denotes the input features. Training proceeds by minimizing a suitable loss function (e.g. mean squared error for regression) via gradient-based methods such as stochastic gradient descent (SGD) or Adam, using backpropagation to compute parameter updates efficiently [27], [28].

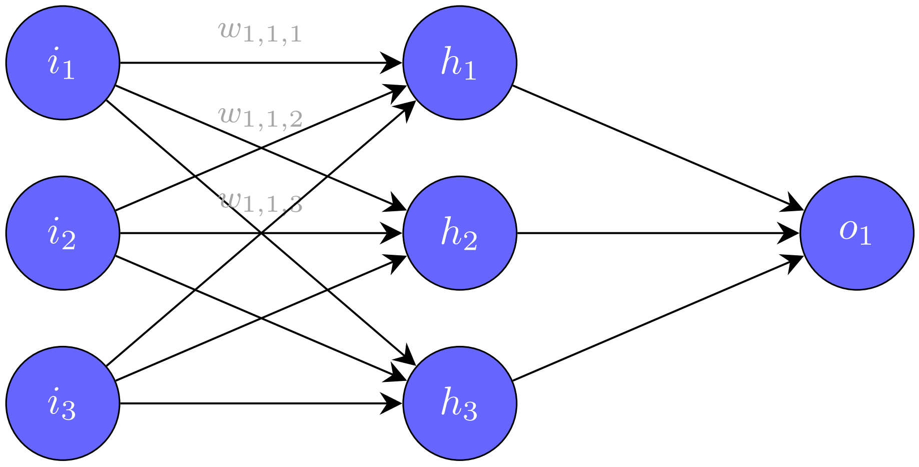

Figure 4.1: Multilayer Perceptron (MLP).

Consequently, to train the MLP, we minimize a loss function $L(\theta)$ (e.g. mean squared error) with respect to all parameters $\theta = \{W^{(l)}, b^{(l)}\}$. Backpropagation computes the gradient $\nabla_\theta L$ by applying the chain rule layer by layer in reverse order [27]. These gradients are then used in an optimizer, such as SGD or Adam, to update parameters

$$\theta \leftarrow \theta - \eta \, \nabla_\theta L,$$

where η is the learning rate [28]. This procedure will also serve as the foundation when we extend to RNNs and LSTMs, which use the same principles but propagate errors through time.

While MLPs excel at capturing static, non-linear mappings between inputs and outputs, they do not inherently handle temporal dependencies. Financial time series, however, exhibit sequential patterns, momentum, mean reversion and regime shifts, that require models with internal memory or specialized sequence architectures.

4.3 Recurrent Neural Networks

In contrast to feedforward architectures, which process each input independently, RNNs are specifically designed to handle sequential data by maintaining an internal hidden state that "remembers" information from previous time steps. This persistence of memory makes RNNs a natural fit for financial time series, which exhibit autocorrelation, momentum effects and regime shifts.

At each time step t, an RNN cell receives the current input vector $\mathbf{x}_t$ (e.g., asset returns or feature embeddings) and the previous hidden state $\mathbf{s}_{t-1}$, and updates its state and output via:

$$\mathbf{s}_t = \phi\big(W_{xh}\, \mathbf{x}_t + W_{hh}\, \mathbf{s}_{t-1} + \mathbf{b}_h\big), \quad \mathbf{y}_t = f\big(W_{hy}\, \mathbf{s}_t + \mathbf{b}_y\big),$$

where:

- W_{xh} and W_{hh} map inputs and prior state to the new state,
- W_{hy} maps the hidden state to the output $\mathbf{y}_t$,
- ϕ and f are non-linear activation functions (e.g. tanh, ReLU, sigmoid).

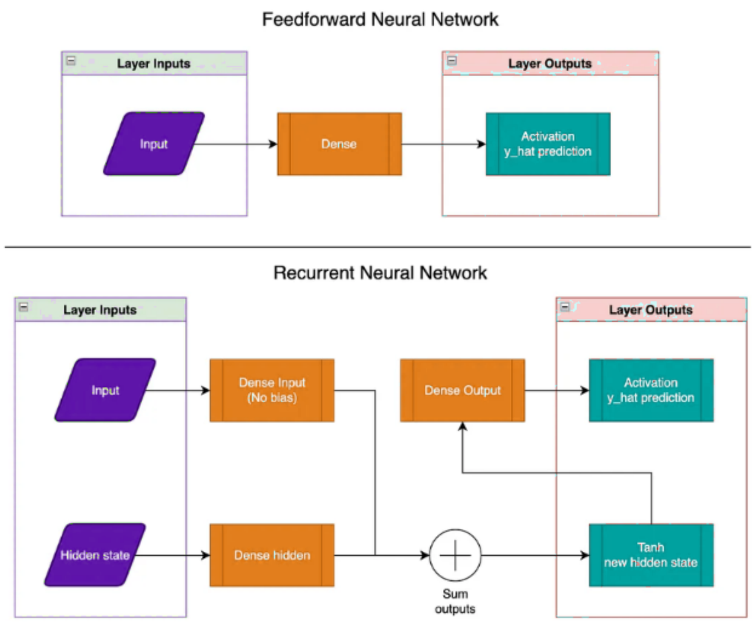

Figure 4.2: Comparison of a Feedforward NN (top) and a RNN (bottom). In the feedforward case, each input flows through a dense layer and then through an activation function to produce an output. In the recurrent case, the current input and the previous hidden state are both passed into dense transformations; their results are summed to form the new hidden state via a tanh activation, and that hidden state is also used (via a separate dense layer) to produce the output at each time step.

The top half of the Figure 4.2 illustrates a standard feedforward architecture: a single "Dense" (fully connected) layer transforms the input features into a hidden representation, which is then passed through an activation function (e.g. ReLU or sigmoid) to produce the network's output. As it can be observed, there is no mechanism for retaining information from previous inputs, each sample is processed independently.

In contrast, the bottom half depicts a simple RNN cell unrolled over a single time step. Two separate dense layers take as inputs (1) the current feature vector, labeled "Input" and (2) the previous time step's hidden state, labeled "Hidden state". Neither of these dense layers uses bias terms for this depiction, biases would be added in practice but are omitted here for clarity. The outputs of these two transformations are then added (the "Sum outputs" node) to yield the new hidden state. That combined signal is passed through a tanh activation (shown in red as "Tanh new hidden state") to form the updated hidden state for this time step. Finally, a separate dense transformation of the hidden state produces the network's output at the current time step, which is again passed through an activation (e.g. a softmax or identity, depending on the task).

Given the previously defined architecture, it is important to note that:

- *State Propagation:* Unlike the feedforward network, the RNN reuses its hidden state from the previous time step, enabling memory of past inputs and the modeling of temporal dependencies.
- *Dual Inputs:* At each time t, the RNN cell combines information from the new input vector $\mathbf{x}_t$ and the past hidden state $\mathbf{h}_{t-1}$. Formally, if we denote the weight matrices by W_{xh} (input-to-hidden), W_{hh} (hidden-to-hidden), and W_{hy} (hidden-to-output), then the new hidden state $\mathbf{h}_t$ is computed as

  $$\mathbf{h}_t = \tanh\bigl(W_{xh}\,\mathbf{x}_t \;+\; W_{hh}\,\mathbf{h}_{t-1}\bigr),$$

  and the output $\mathbf{y}_t$ is given by

  $$\mathbf{y}_t = f\bigl(W_{hy}\,\mathbf{h}_t\bigr).$$

  In our schematic, these operations correspond to the "Dense input" and "Dense hidden" blocks feeding into the sum node, followed by the "Tanh new hidden state," and finally the "Dense output" feeding into the activation "y_hat prediction."
- *Temporal Dynamics:* By carrying $\mathbf{h}_{t-1}$ forward, RNNs can learn patterns spread across multiple time steps, essential for financial series that exhibit momentum, volatility clustering, and structural breaks.
- *Training Implications:* During backpropagation through time (BPTT), gradients must flow both through the dense layers at each step and along the recurrent connections. This direct feedback loop often leads to vanishing or exploding gradients in vanilla RNNs, motivating gated variants (LSTM, GRU) described in the next section.

With this conceptual groundwork, we are now prepared to examine LSTM networks, which introduce explicit gating to control the flow of information and mitigate the training difficulties inherent in vanilla RNNs.

Why Not Feedforward?

Feedforward networks treat each time point in isolation, lacking any mechanism to encode temporal context. As a result, they cannot capture serial correlations or long–term dependencies, both of which are common in financial markets (momentum, mean reversion, volatility clustering). On the other hand, RNNs overcome this by propagating information forward through their hidden states, effectively creating a dynamic memory.

Origins and Limitations

RNNs were popularized in the 1990s but soon encountered training difficulties: during backpropagation through time, repeated multiplication by the same weight matrices causes gradients to either shrink towards zero (vanishing) or grow explosively (exploding), preventing the network from learning long–range dependencies [29]. In financial contexts, where a shock's effect may persist over many trading days, this limitation can severely reduce model effectiveness.

Setting the Stage for LSTM

To address these issues, gated architectures such as LSTM networks introduce explicit memory cells and learnable gating mechanisms that control the flow of information. In the next section, we will dissect the internal structure of LSTM cells, explain how their gates mitigate gradient problems, and illustrate why they are particularly well suited for modeling complex temporal patterns in asset returns and volatility.

4.4 Long Short–Term Memory Networks

Standard (vanilla) RNNs, despite their elegance, struggle to learn long-range dependencies due to the vanishing and exploding gradient problems during backpropagation through time [29]. In financial time series, where events from many days or even months ago may influence current asset behavior, this limitation severely hampers model performance. The LSTM architecture, introduced by Hochreiter and Schmidhuber (1997) [6], was specifically designed to mitigate these issues by incorporating an explicit memory cell and gating mechanisms that regulate the flow of information. The result is a recurrent unit capable of preserving and propagating relevant signals over long sequences, making LSTMs particularly well suited for applications such as volatility forecasting, regime detection, and return prediction in quantitative finance.

Overview of LSTM Components. An LSTM cell augments the hidden-state dynamics of a vanilla RNN with an internal *cell state* $\mathbf{c}_t \in \mathbb{R}^d$, which can be thought of as a "memory" that persists (largely unmodified) from one time step to the next, subject only to carefully controlled updates. Three distinct *gates*, all implemented via learned affine transformations followed by element-wise nonlinearities, control how information flows into, out of, and within the cell state:

- **Input Gate $\mathbf{i}_t \in (0,1)^d$:** Determines how much of the new candidate content $\mathbf{g}_t$ (sometimes called the "input modulation vector") to write into the cell state.
- **Forget Gate $\mathbf{f}_t \in (0,1)^d$:** Decides which portions of the previous cell state $\mathbf{c}_{t-1}$ should be "forgotten" (multiplied by small values) or retained (multiplied by values near 1).
- **Output Gate $\mathbf{o}_t \in (0,1)^d$:** Controls how much of the updated cell state $\mathbf{c}_t$ should influence the hidden-state output $\mathbf{h}_t$ at the current time step.

In addition, the cell computes a candidate update $\mathbf{g}_t \in (-1,1)^d$ (via a tanh nonlinearity) that represents new information extracted from the current input $\mathbf{x}_t$ and the previous hidden state $\mathbf{h}_{t-1}$. The final cell-state update is thus a convex combination (element-wise) of "old memory" and "new information," weighted by the forget and input gates, respectively:

$$\mathbf{c}_t = \mathbf{f}_t \mathbf{c}_{t-1} \; + \; \mathbf{i}_t \mathbf{g}_t.$$

Because $\mathbf{f}_t$ and $\mathbf{i}_t$ can be learned to take values near 1 or 0 (via their sigmoid activations), the network can choose to preserve (or discard) information in $\mathbf{c}_{t-1}$ independently for each dimension, thereby preventing gradients from vanishing or exploding when propagated over many time steps.

Mathematical Formulation

Formally, given an input vector $\mathbf{x}_t \in \mathbb{R}^n$ at time t and the previous hidden state $\mathbf{h}_{t-1} \in \mathbb{R}^d$, the LSTM cell computes:

$$\begin{aligned}
\mathbf{i}_t &= \sigma\big(W_{xi}\,\mathbf{x}_t \; + \; W_{hi}\,\mathbf{h}_{t-1} \; + \; \mathbf{b}_i\big), \quad |\mathbf{i}_t| \in (0,1)^d,\\
\mathbf{f}_t &= \sigma\big(W_{xf}\,\mathbf{x}_t \; + \; W_{hf}\,\mathbf{h}_{t-1} \; + \; \mathbf{b}_f\big), \quad |\mathbf{f}_t| \in (0,1)^d,\\
\mathbf{o}_t &= \sigma\big(W_{xo}\,\mathbf{x}_t \; + \; W_{ho}\,\mathbf{h}_{t-1} \; + \; \mathbf{b}_o\big), \quad |\mathbf{o}_t| \in (0,1)^d,\\
\mathbf{g}_t &= \tanh\big(W_{xg}\,\mathbf{x}_t \; + \; W_{hg}\,\mathbf{h}_{t-1} \; + \; \mathbf{b}_g\big), \quad \mathbf{g}_t \in (-1,1)^d,\\
\mathbf{c}_t &= \mathbf{f}_t \mathbf{c}_{t-1} \; + \; \mathbf{i}_t \mathbf{g}_t,\\
\mathbf{h}_t &= \mathbf{o}_t \tanh(\mathbf{c}_t),
\end{aligned}$$

where:

- $\sigma(\cdot)$ is the logistic sigmoid function applied element-wise, ensuring gate outputs lie in $(0,1)$.

- $\tanh(\cdot)$ is the hyperbolic tangent, producing values in $(-1, 1)$.
- $W_{xi}, W_{xf}, W_{xo}, W_{xg} \in \mathbb{R}^{d \times n}$ are input-to-gate weight matrices.
- $W_{hi}, W_{hf}, W_{ho}, W_{hg} \in \mathbb{R}^{d \times d}$ are hidden-to-gate (recurrent) weight matrices.
- $\mathbf{b}_i, \mathbf{b}_f, \mathbf{b}_o, \mathbf{b}_g \in \mathbb{R}^d$ are bias vectors for each gate and the candidate cell update.

The hidden-state $\mathbf{h}_t$ itself can be used directly as either:

- An *intermediate output* at time t, when the LSTM is configured to return sequences (`return_sequences=True`), or
- A *final summary* if one only needs $\mathbf{h}_T$ after processing T time steps (i.e. `return_sequences=False`).

Intuition Behind Each Gate

- $\mathbf{i}_t$ (input gate): By scaling the candidate $\mathbf{g}_t$ element-wise, the network decides which features from the new information "deserve" entry into the cell memory. For example, if a particular element of $\mathbf{i}_t$ is near 0, the corresponding dimension of $\mathbf{g}_t$ is ignored in updating $\mathbf{c}_t$.
- $\mathbf{f}_t$ (forget gate): Controls the extent to which the previous memory $\mathbf{c}_{t-1}$ is retained. If an element of $\mathbf{f}_t$ is near 1, that feature in $\mathbf{c}_{t-1}$ is preserved; if it is near 0, that portion of memory is reset. This mechanism allows the LSTM to "forget" outdated information (e.g. a past volatility regime that no longer applies).
- $\mathbf{o}_t$ (output gate): Determines which parts of the updated memory $\mathbf{c}_t$ should influence the hidden state $\mathbf{h}_t$ (and hence any subsequent outputs or further recurrence). By gating $\tanh(\mathbf{c}_t)$, the network can mask irrelevant memory components when producing $\mathbf{h}_t$.
- $\mathbf{g}_t$ (candidate cell update): Represents new content extracted from $\mathbf{x}_t$ and $\mathbf{h}_{t-1}$ that might be useful to store in memory. Its values are squashed between -1 and 1 by tanh, ensuring numerical stability.

Gradient Flow and Long-Range Dependencies

Because $\mathbf{c}_t$ can be updated largely via element-wise multiplication by $\mathbf{f}_t \approx 1$, gradients can propagate backward through many time steps without vanishing. In particular, the cell-state update $\mathbf{c}_t = \mathbf{f}_t \mathbf{c}_{t-1} + \cdots$ implies

$$\frac{\partial \mathbf{c}_t}{\partial \mathbf{c}_{t-1}} = \mathbf{f}_t \mathbf{I}_d,$$

which, when $\mathbf{f}_t$ elements are near 1, allows $\frac{\partial \mathbf{c}_t}{\partial \mathbf{c}_{t-1}} \approx \mathbf{I}_d$. Consequently, the "constant error carousel" effect (Hochreiter & Schmidhuber, 1997) preserves error signals across long intervals, enabling the learning of dependencies spanning dozens or even hundreds of time steps [6].

LSTM Variants and Extensions

Although the original LSTM formulation included additional "peephole connections" (allowing gates to inspect $\mathbf{c}_{t-1}$ directly) and different activation choices, the "vanilla LSTM" as presented above has become the de facto standard in many libraries (TensorFlow, PyTorch, Keras). Several useful variants include:

- **Peephole LSTM:** Adds direct connections from $\mathbf{c}_{t-1}$ to each gate (i.e. additional weights W_{ci}, W_{cf}, W_{co}) so that gates can condition decisions on the previous cell state explicitly. This can improve performance on tasks with precise timing requirements.
- **Coupled Input-Forget LSTM:** Ties the input and forget gates via $\mathbf{f}_t = 1 - \mathbf{i}_t$, reducing parameter count and sometimes improving generalization.
- **Gated Recurrent Unit (GRU):** A simplified variant that merges the input and forget gates into a single "update gate" and discards the explicit cell state, leading to fewer parameters and often comparable performance in practice [30].

Application of LSTM to Financial Time Series

In a financial portfolio optimization context, we typically frame the problem as: given a historical sequence of feature vectors $\{\mathbf{x}_{t-T+1}, \mathbf{x}_{t-T+2}, \ldots, \mathbf{x}_t\}$ (e.g. returns, technical indicators, sentiment scores over the last T days), predict a target variable at time $t+1$ (e.g. next-day return, volatility, or covariance matrix component). An LSTM network can be stacked with one or more layers (possibly interleaved with dropout or batch-normalization layers) so that:

1. The first LSTM layer processes the input sequence and outputs a sequence of hidden-state vectors $\{\mathbf{h}_1^{(1)}, \mathbf{h}_2^{(1)}, \ldots, \mathbf{h}_T^{(1)}\}$.

LSTM Cell Dynamics

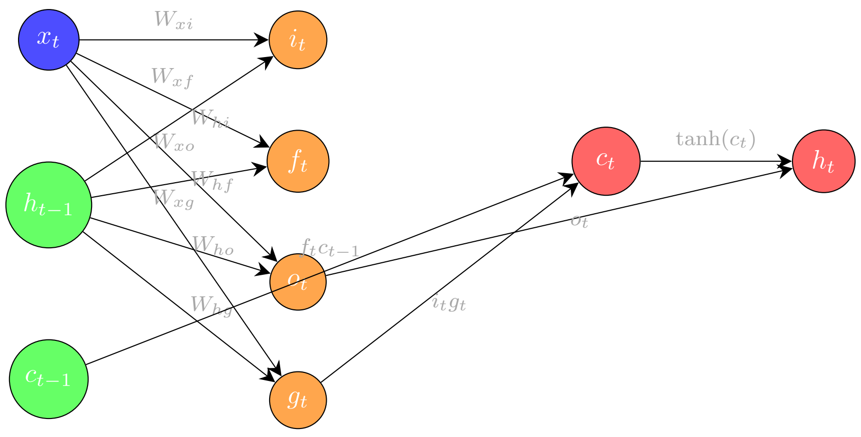

Figure 4.3: Internal dynamics of an LSTM cell: the input vector x_t (blue) and the previous hidden state h_{t-1} (green) feed four gates (orange): input (i_t), forget (f_t), output (o_t), and candidate (g_t). The cell state c_t (red) is formed by combining $f_t c_{t-1}$ and $i_t g_t$, and then produces the new hidden state h_t.

2. If `return_sequences=True`, this entire sequence is passed to a second LSTM layer; if `return_sequences=False`, only the final hidden state $\mathbf{h}_T^{(1)}$ is forwarded to subsequent dense layers.

3. A final dense (fully connected) layer maps the chosen LSTM output(s) to the desired prediction $\hat{y}_{t+1}$. For example, a linear activation could predict a scalar next-day return, or a more elaborate head could output a vector of expected returns for N assets.

4. During training, a suitable loss function is minimized (e.g. mean squared error between $\hat{y}_{t+1}$ and the true y_{t+1}), with gradients computed via backpropagation through time (BPTT), which now flows freely through the "constant error carousel" of the LSTM cell state.

Figure 4.3 illustrates the internal structure of a single LSTM cell, highlighting how the gates interact with the cell state and hidden state.

Interpretation of the LSTM Cell Diagram

In Figure 4.3, observe that each gate (input, forget, output) receives the same concatenated input $[\mathbf{x}_t; \mathbf{h}_{t-1}]$, but uses its own learned weights and bias. The candidate cell update $\mathbf{g}_t$ is similarly computed from $[\mathbf{x}_t; \mathbf{h}_{t-1}]$ but passes through a tanh nonlinearity. The forget gate $\mathbf{f}_t$ then scales $\mathbf{c}_{t-1}$ element-wise, while the input gate $\mathbf{i}_t$ scales $\mathbf{g}_t$, and their sum yields $\mathbf{c}_t$. Finally, the output gate $\mathbf{o}_t$ determines which parts of $\mathbf{c}_t$ are exposed to form $\mathbf{h}_t$. By adjusting these gates dynamically at each time step, the LSTM can (1) retain salient financial signals across many days, (2) filter out noise or obsolete patterns, and (3) produce context-aware predictions.

Training Considerations and Regularization.

Training an LSTM on financial data requires careful attention to overfitting and time series validation. Common practices include:

- *Dropout on Recurrent Connections:* Applying dropout within the LSTM cell (e.g. variational dropout) to prevent co-adaption of gates.
- *Weight Decay:* Introducing an ℓ_2 penalty on all weights to encourage smaller parameters and reduce variance.
- *Early Stopping:* Monitoring validation loss on a rolling-window split (walk-forward validation) and halting training when performance plateaus.
- *Gradient Clipping:* Clipping gradients to a fixed norm to avoid exploding gradients, especially important when learning long sequences.

These regularization techniques, combined with hyperparameter tuning (e.g. selecting the number of LSTM layers, hidden-state dimensionality d, learning rate, batch size), ensure that the model generalizes effectively to unseen market regimes and does not simply memorize spurious patterns.

In summary, LSTM networks represent a powerful extension of vanilla RNNs, addressing fundamental training difficulties and enabling the modeling of long-term dependencies in sequential data. In the context of portfolio optimization, they allow us to generate more accurate forecasts of future returns, volatilities, and covariances by effectively "remembering" relevant financial signals over extended horizons. In Chapter 5, we will implement an LSTM-based predictor and integrate its forecasts into a NN to optimize portfolio weights, so a construction of a portfolio that adapt dynamically to evolving market conditions will be made.

Chapter 5

Comparative Methodology: Classical vs. Machine Learning Portfolio Optimization

5.1 Introduction

As already mentioned, in the context of modern quantitative finance, portfolio optimization has long been a central problem, attracting considerable attention from both academia and industry [4], [7], [9], [14]. The classical approach, as formalized by Harry Markowitz [1] in his seminal mean-variance optimization framework, has become a foundational pillar of asset allocation and risk management. Nevertheless, recent advances in computational power, the availability of high-frequency data, and the rapid evolution of ML algorithms have opened new horizons for financial modeling and portfolio construction.

Consequently, the objective of this chapter is to present a rigorous and comprehensive comparison between two fundamentally different paradigms for portfolio optimization: the traditional Markowitz mean-variance approach and a modern data-driven solution based on deep learning techniques. The ML approach specifically combines LSTM networks for time series forecasting with fully connected NNs for direct portfolio weight assignment.

This comparative study is motivated by several important considerations. First, the assumptions underlying the classical approach, such as the use of historical means and covariances as stable predictors of future asset behavior, are often violated in practice due to non-stationarities, structural breaks, and rapidly evolving market regimes. Second, traditional models are limited in their ability to capture complex, nonlinear dependencies and temporal patterns inherent in financial markets. Finally, the rise of ML, and deep learning in particular, provides the possibility of learning adaptive allocation strategies directly from historical data, potentially outperforming rigid parametric models in out-of-sample environments.

Therefore, in this chapter, we propose and implement a robust experimental framework designed to evaluate the strengths and limitations of both approaches. First of all, we begin by detailing the process of data acquisition, cleaning, and transformation, which is critical to any quantitative analysis. To ensure reliable results, we divide the data into separate training and test sets, using the training set for model fitting and the test set for evaluating their performance. Next, we formalize the implementation of the Markowitz optimization, followed by the design and training of a hybrid LSTM and fully connected NN model aimed at directly learning optimal portfolio weights from historical return patterns. Finally, each method is evaluated under identical market conditions, using the same set of assets and the same out-of-sample test period, thereby ensuring the fairness of the comparison.

Throughout the chapter, we place particular emphasis on performance metrics that are widely recognized in the field, including annualized return, volatility, and the Sharpe ratio [2]. The ultimate goal is to provide a statistically sound and practically meaningful assessment of whether advanced ML models can deliver superior portfolio performance when benchmarked against established classical methods. This work thus aims to contribute to the ongoing discourse regarding the role of artificial intelligence in the future of asset management, offering insights that may inform both researchers and practitioners interested in the design of next-generation portfolio optimization systems.

5.2 Data Acquisition and Preprocessing

5.2.1 Data Sources and Universe Selection

The foundation of any quantitative portfolio study lies in the careful selection and preparation of the data. In this work, we focus on a representative set of large-cap technology equities, specifically the stocks of Apple (AAPL), Microsoft (MSFT), Google (GOOG), Amazon (AMZN), and Meta Platforms (META). This selection is motivated by the liquidity, coverage, and economic relevance of these assets, as well as their frequent inclusion in contemporary portfolio construction research.

Daily historical price data for each selected asset was retrieved from Yahoo Finance, a widely used and reputable source for financial time series. The data spans from January 1, 2015, to the latest available trading day in 2024 (December 30th), thereby ensuring that both the training and testing phases of the study are based on a sufficiently long and recent time horizon. By starting in 2015, we capture multiple market regimes, including periods of both volatility and stability, which is essential for the robust evaluation of forecasting and allocation models.

Before feeding the data into the LSTM-based model, the raw price series undergoes several preprocessing steps to ensure consistency, reliability, and suitability for quantitative modeling. First, prices are aligned to trading days across all assets so that missing values, caused by non-trading days or asset-specific events, do not distort the analysis; when a price is absent, it is forward-filled to maintain continuity in the time series. Next, any residual gaps remaining after this alignment are also imputed via forward-fill, based on the assumption that price changes occur only on trading days and temporary data gaps do not reflect actual market movements. After handling missing values, daily log-returns are computed from the adjusted closing prices, since log-returns are preferred in quantitative finance for their time-additive property and because they better satisfy the normality assumptions of many models. Finally, for the deep learning components, particularly the LSTM networks, all price and return series are normalized using a Min–Max scaling approach. In particular, the parameters for normalization (minimum and maximum values) are fitted exclusively on the training set and then applied to the test set, thus preventing any information leakage and preserving the integrity of the out-of-sample evaluation.

Finally, to ensure a fair and meaningful comparison between the portfolio optimization methods, both strategies are constructed and evaluated under identical market conditions and data constraints. The dataset is strictly divided into a training period (2015–2023) and a test period (2024), with all model parameters determined using only the training data. Consequently, no information from the test period is used during model development or feature scaling. Both approaches operate on the same asset set and the same preprocessed return series. Additionally, all portfolio allocations are subject to long-only, fully-invested constraints, and no transaction costs or liquidity restrictions are included in the initial analysis. This rigorous protocol is designed to guarantee that any observed differences in out-of-sample performance arise solely from the optimization methodology, rather than from discrepancies in data treatment or evaluation procedures.

To conclude, it is important to mention that these preprocessing and evaluation steps are crucial to ensuring that the subsequent ML models operate on clean, consistent, and stationary data, and that the performance results of both classical and machine learning-based approaches can be meaningfully compared in a robust, out-of-sample setting.

5.3 Training methodology

The focus of this chapter is to detail the architecture, training process, and parameter choices underlying each method, with particular emphasis on the design, tuning, and implementation of the LSTM and NN components. It is important to note that both methods are evaluated under identical data splits, constraints, and evaluation protocols, as described in the preceding section.

Therefore, the experimental framework developed in this chapter is designed to provide a fair, transparent, and statistically robust comparison between two fundamentally distinct approaches to portfolio optimization. The first approach is the classical mean-variance optimization method introduced by Markowitz [1], which uses historical statistics of returns and covariances to derive a single set of optimal portfolio weights, subsequently held constant throughout the evaluation period. On the other hand, the second approach leverages advanced ML techniques, combining a LSTM network for financial time series forecasting with a fully connected NN trained to assign portfolio weights dinamically, based on the LSTM predicted value of the assets.

While the theoretical foundations and architectural details of each model have been thoroughly addressed in previous chapters, it is important here to clarify their operational differences in empirical application. The classical Markowitz model computes a single set of portfolio weights at the outset of the test period, estimating an annualized mean vector and covariance matrix of daily log-returns from the training window (2015–2023) and then maximizing the Sharpe ratio under long-only, fully invested constraints (weights ≥ 0 and summing to 1) via Sequential Least Squares Programming (SLSQP). Once determined, these weights remain fixed throughout the entire evaluation window, providing a static benchmark. In contrast, the proposed LSTM+NN pipeline continually updates portfolio weights on a rolling basis: first, a LSTM network forecasts future asset returns; then a fully connected NN learns to map those predicted return sequences directly to optimal weights. This dynamic recalibration during the test period is the core objective of our study, enabling a direct empirical comparison between static mean-variance optimization and a ML-based strategy that leverages recurrent neural forecasts. A detailed explanation of the LSTM+NN model that underlies our proposed approach is provided below.

LSTM-based Prediction and Dynamic Allocation

The core of the proposed methodology lies in a two-stage deep learning pipeline designed to provide adaptive and data-driven portfolio allocation. The first stage consists of a LSTM network, specifically constructed to model temporal dependencies and patterns in financial time series. At each point in time, the LSTM receives as input a sliding window comprising the most recent n daily values (e.g., closing prices or returns) for all assets in the portfolio. This input is thus a matrix of dimensions $n \times d$, where d is the number of assets under consideration. The LSTM processes this sequential data and produces, for each asset, a prediction of the future price or return over the desired forecast horizon.

The second stage leverages these predictions to make allocation decisions. Specifically, the sequence of predicted future returns (or prices) for all assets, typically spanning the next m days, is concatenated into a single input vector. This vector encapsulates both the cross-sectional and temporal predictive information inferred by the LSTM. It is then fed into a fully connected (feedforward) NN, which is trained to map this multi-asset, multi-period forecast into a set of portfolio weights for the next trading day. The output of the network is a vector of d weights, constrained (via a softmax activation function) to be non-negative and to sum to one, thereby ensuring compliance with long-only, fully-invested portfolio requirements.

During training, the NN learns to assign weights that maximize a risk-adjusted reward, typically the Sharpe ratio or another utility-based criterion, across the training set, using the historical realized returns as targets. In the test phase, for each day, the latest LSTM predictions are used to dynamically update the allocation in a rolling fashion, enabling the portfolio to react continuously to newly observed data and anticipated market movements.

This LSTM+NN framework, Figure 5.1, represents a significant improvement from the classical static approach of Markowitz. Instead of holding constant weights over the entire evaluation period, the portfolio composition is recalibrated at each step, allowing for real-time adaptation to shifting risk-return profiles and market dynamics. By integrating both time series prediction and allocation within a unified, end-to-end trainable pipeline, the method leverages the strengths of deep learning to address the inherently dynamic nature of financial markets.

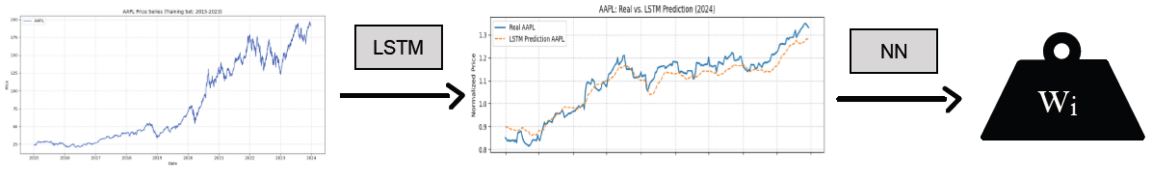

Figure 5.1: Schematic Overview of the Two-Stage Deep Learning Framework for Adaptive Portfolio Allocation

5.3.1 Model Hyperparameters

To ensure full reproducibility and transparency, this section describes in detail the hyperparameter configurations and training procedures employed for both the classical Markowitz approach and the ML pipeline.

For Markowitz Optimization, daily log-returns were computed for each asset, and annualized by multiplying by 252 (the typical number of trading days in a year). The annualized covariance matrix was calculated analogously. The portfolio optimization sought to maximize the Sharpe ratio under the constraints of long-only positions (all portfolio weights ≥ 0) and full investment (weights sum to one). The optimization problem was solved using the Sequential Least Squares Programming (SLSQP) algorithm, as implemented in `scipy.optimize.minimize`, with an equal-weight initial guess. The Sharpe ratio was calculated assuming a risk-free rate of zero.

On the other hand, the LSTM network was trained to predict the next-day closing price of each asset based on a rolling input window of the previous 60 trading days. The network architecture consisted of a single LSTM layer with 64 units, followed by a dropout layer (rate: 0.2) and a dense output layer with linear activation to match the number of assets. Training was performed using mean squared error (MSE) as the loss function and the Adam optimizer (learning rate = 0.001). The batch size was set to 32, and training ran for up to 50 epochs, with early stopping applied if the validation loss failed to improve for five consecutive epochs. A validation split of 20% was used, and all input features were normalized via Min–Max scaling, fitted exclusively on the training set to avoid information leakage. Model development was conducted in Keras using the TensorFlow backend.

Since the overall model architecture has been described previously, we focus here on the specific configuration of the NN component responsible for portfolio weight assignment. The fully connected NN received as input the concatenated predicted returns for all assets over the rolling 60-day window. The architecture included two dense layers (128 and 64 units, respectively, each with ReLU activation), each followed by dropout (rate: 0.2), and an output layer with softmax activation to ensure that portfolio weights are non-negative and sum to one. The network was trained to minimize mean squared error between the predicted weights and target weights (derived from the rolling-window Markowitz solution during training). Optimization again used Adam (learning rate = 0.001), batch size of 32, up to 50 epochs, with early stopping based on validation loss. Dropout provided regularization, and a 20% validation split was applied.

All random seeds were fixed prior to training to ensure reproducibility. Hyperparameters were selected based on empirical validation set performance within the training period, and no information from the test period was used at any stage of model selection or evaluation.

Component	**Parameter**	**Value / Setting**
Markowitz	Algorithm	SLSQP (scipy.optimize)
	Risk-free rate	0
	Initial weights	Equal allocation
	Constraints	Long-only, weights sum to 1
LSTM	Input window	60 days
	LSTM units	64
	Dropout rate	0.2
	Output activation	Linear
	Loss function	MSE
	Optimizer	Adam (lr=0.001)
	Batch size	32
	Epochs	50 (early stopping)
	Validation split	20%
NN for weights	Input shape	60 days $\times$ No. assets
	Dense layers	[128, 64] units, ReLU
	Dropout rate	0.2
	Output activation	Softmax
	Loss function	MSE
	Optimizer	Adam (lr=0.001)
	Batch size	32
	Epochs	50 (early stopping)
	Validation split	20%

Table 5.1: Summary of main hyperparameter choices

5.3.2 Performance Metrics

To enable a transparent and rigorous evaluation of portfolio performance, three key metrics are reported for each strategy over the test period (2024):

- **Annualized Return:** The mean daily return of the portfolio, multiplied by the typical number of trading days in a year (252), provides an annualized measure of profitability. This metric enables a direct assessment of the capital growth potential offered by each approach.

- **Annualized Volatility:** The standard deviation of daily portfolio returns, scaled by the square root of 252, yields the annualized volatility. This serves as a proxy for risk, quantifying the variability of portfolio returns over time.

- **Sharpe Ratio:** Defined as the ratio of annualized return to annualized volatility, the Sharpe ratio provides a normalized measure of risk-adjusted performance. It indicates how efficiently each approach transforms risk into return and is widely used as a benchmark in portfolio management.

These metrics are computed for both the classical Markowitz approach and the LSTM+NN-based strategy, using strictly out-of-sample data. For clarity and reproducibility, all figures and tables report metrics rounded to two decimal places.

Moreover, while annualized return, volatility, and Sharpe ratio capture average performance and overall variability, they do not reflect the severity and duration of potential losses in adverse market conditions. In addition to standard performance measures, it is critical to assess the risk profile of portfolio strategies through the analysis of drawdown dynamics. The *drawdown* at any time t is defined as the percentage loss relative to the most recent historical maximum of the cumulative portfolio value, and is formally given by

$$\text{Drawdown}(t) = \frac{V(t) - \max_{s \leq t} V(s)}{\max_{s \leq t} V(s)} \tag{5.1}$$

where $V(t)$ denotes the cumulative wealth of the portfolio at time t. A drawdown of zero indicates that the portfolio has reached a new maximum, while negative values reflect periods of decline from the previous peak.

5.4 Empirical Results

A crucial factor of the success of any ML-based portfolio strategy is the predictive accuracy of its underlying time series forecasting model. Figures 5.2, 5.3, 5.4. 5.5. 5.6 provide a comprehensive visualization of the LSTM network's predictive performance for each of the five constituent assets in the test period (2024). For each asset, the figure displays both the normalized actual price trajectory and the corresponding predictions generated by the LSTM model.

The visual comparison reveals the extent to which the LSTM is able to capture the short-term dynamics and overall trends present in the real price series. In general, the predicted series closely follows the real data, successfully replicating major movements and exhibiting low deviation in periods of stable market conditions. Nonetheless, occasional discrepancies and lags are observed, particularly during abrupt price reversals or periods of heightened volatility, which is characteristic of the challenges faced by deep learning models in highly non-stationary environments.

This figures serve not only to validate the predictive capability of the LSTM architecture, but also to contextualize the results obtained in the subsequent portfolio allocation stage. Since the assignment of portfolio weights by the downstream

NN relies on these forecasts, the fidelity of the LSTM predictions is directly linked to the quality of the resulting investment decisions and, ultimately, the out-of-sample performance of the proposed ML-based portfolio strategy.

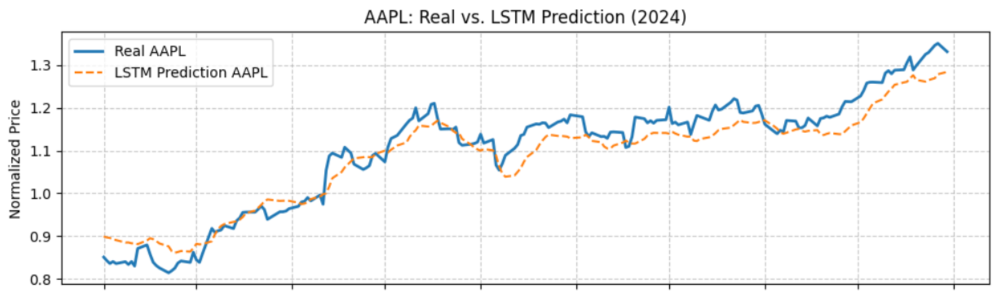

Figure 5.2: APPLE. Comparison of LSTM-predicted and real normalized prices for all assets during 2024.

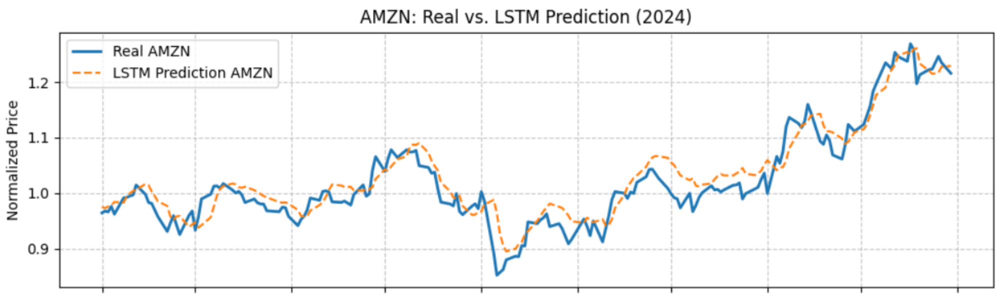

Figure 5.3: AMAZON. Comparison of LSTM-predicted and real normalized prices for all assets during 2024.

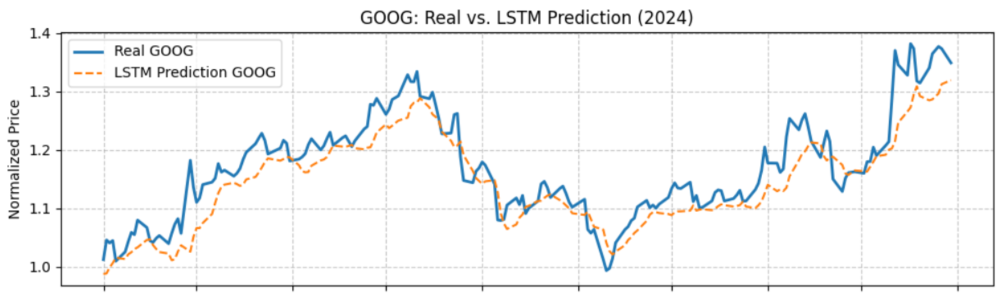

Figure 5.4: GOOGLE. Comparison of LSTM-predicted and real normalized prices for all assets during 2024.

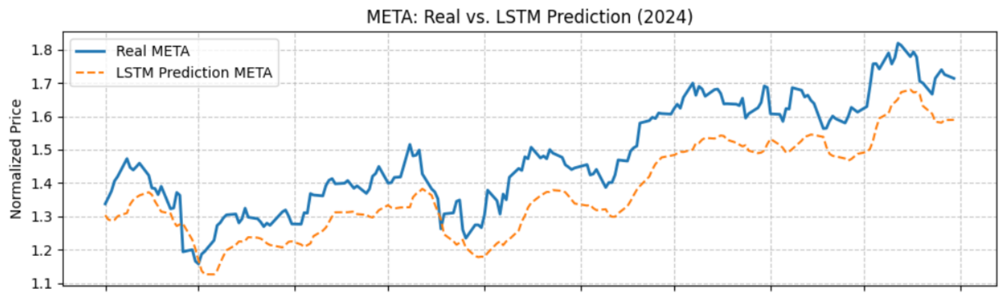

Figure 5.5: META. Comparison of LSTM-predicted and real normalized prices for all assets during 2024.

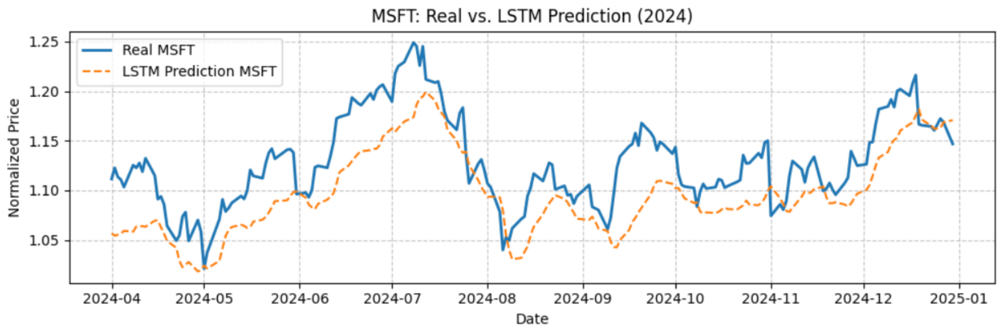

Figure 5.6: MICROSOFT. Comparison of LSTM-predicted and real normalized prices for all assets during 2024.

Accorded to the previous images, the errors in LSTM predictions and mean-variance are summarized in Table 5.2, the LSTM model achieves mean squared errors that are orders of magnitude lower than those obtained using the historical mean as a naive benchmark for next-day price prediction. This substantial reduction in prediction error highlights the limitations of static, mean-based forecasting approaches when applied to non-stationary and trending financial time series. In contrast, the LSTM's architecture, which leverages recent sequential information, enables the model to more accurately capture the dynamic patterns present in the market data, resulting in markedly improved predictive performance for all assets in the test set.

Stock	**LSTM MSE**	**Historical Mean MSE**
AAPL	62.17	18125.36
AMZN	46.00	9856.72
GOOG	65.49	9714.71
META	758.91	116798.00
MSFT	239.36	71707.54

Table 5.2: Mean Squared Error (MSE) comparison for next-day price prediction: LSTM vs. Historical Mean (2024 Test Set)

Given the fundamental objective of this study is to compare the effectiveness of the two portfolio optimization methodologies, it is essential to present their key performance indicators side by side. Table 5.3 summarizes the out-of-sample performance of both portfolio optimization methods during the 2024 test period. The results highlight the annualized return, annualized volatility, and Sharpe ratio achieved by each approach. It presents the out-of-sample performance metrics for both the classical Markowitz strategy and the proposed LSTM+NN approach during the 2024 evaluation period. The results demonstrate that the LSTM+NN strategy achieves a substantially higher annualized return (35.40%) compared to the Markowitz benchmark (22.18%), while maintaining a comparable level of risk, as reflected by annualized volatility (21.18% for LSTM+NN versus 18.86% for Markowitz). Most notably, the Sharpe ratio, a key measure of risk-adjusted performance, is significantly higher for the LSTM+NN portfolio (1.67) than for the Markowitz portfolio (1.18).

Method	**Profit (%)**	**Annualized Volatility (%)**	**Sharpe Ratio**
Markowitz	22.18	18.86	1.18
LSTM+NN	35.40	21.18	1.67

Table 5.3: Out-of-sample performance comparison: Markowitz vs. LSTM+NN (2024)

These findings indicate that the dynamic, data-driven allocation enabled by the LSTM and NN models can more effectively capture evolving market trends and respond adaptively to new information, resulting in superior risk-adjusted returns. In contrast, the static Markowitz portfolio, constrained by its reliance on historical average returns and covariances, is less able to exploit changes in market conditions during the test period. This empirical evidence underscores the value of integrating advanced ML techniques into portfolio optimization, particularly in environments characterized by non-stationarity and structural shifts in asset return dynamics. Figure 5.7 provides a visual summary of the same metrics, facilitating a more intuitive comparison of the relative performance of each method.

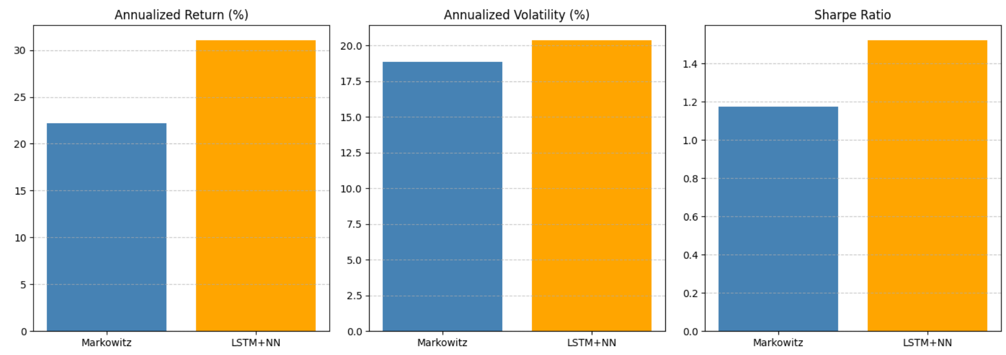

Figure 5.7: Comparison of annualized return, volatility, and Sharpe ratio for Markowitz and LSTM+NN portfolios (2024).

Furthermore, Figure 5.8 illustrates the evolution of drawdown for both the Markowitz and LSTM+NN strategies over the evaluation window. This metric provides insight into the depth and duration of losses experienced by each portfolio, which are not captured by return and volatility statistics alone. A strategy with smaller or shorter drawdowns can be considered more resilient, as it recovers from market downturns more quickly and exposes the investor to less severe capital losses. From the figure, it can be observed that the LSTM+NN strategy generally experiences shallower drawdowns compared to the Markowitz benchmark throughout most of the evaluation window. While both portfolios exhibit declines from local peaks, the drawdowns of the LSTM+NN approach tend to be less severe and recover more quickly to new highs. In contrast, the Markowitz portfolio is subject to deeper and more prolonged periods of decline, particularly during market downturns observed in late summer and early autumn. These results suggest that the dynamic, data-driven allocation enabled by the LSTM+NN model provides improved downside risk management, allowing the portfolio to recover faster from adverse movements and maintain capital preservation more effectively than the static Markowitz allocation. This analysis complements the evaluation of returns and Sharpe ratios by offering a more comprehensive picture of portfolio robustness under adverse market conditions.

Continuing with the comparative analysis of the models presented, figure 5.9 presents the evolution of cumulative wealth for an initial investment of $1 under both portfolio strategies during the out-of-sample test period (2024). In this context, the vertical axis reflects the value of a hypothetical portfolio that starts with one unit of capital at the beginning of the year and is updated daily according to the returns generated by each method. A value greater than 1 at the end of the

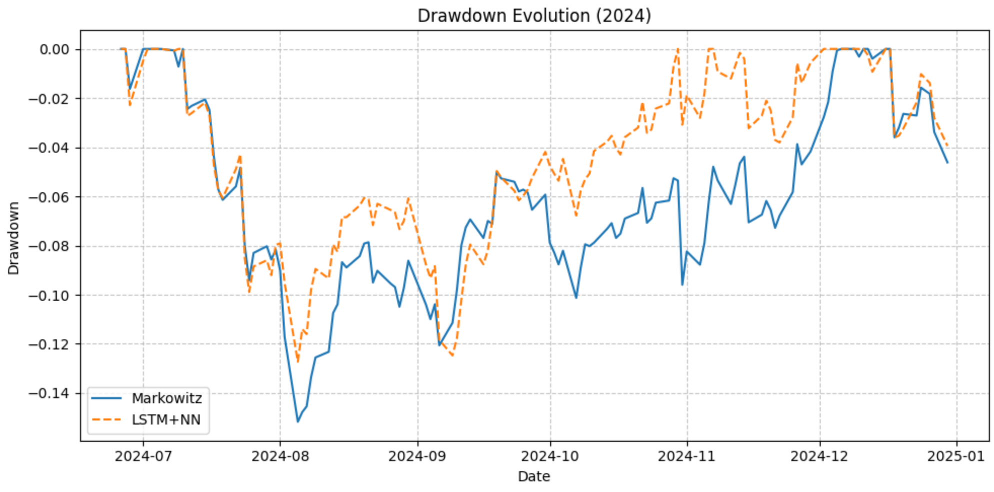

Figure 5.8: Drawdown evolution for the Markowitz and LSTM+NN portfolios during the 2024 evaluation window.

period indicates that the initial investment has grown, whereas a value less than 1 signifies a loss relative to the starting amount. For example, if the LSTM+NN portfolio reaches a final value of 1.15, this means that the investment has appreciated by 15% over the year, net of compounding. Conversely, a final value of 0.95 would represent a cumulative loss of 5%. This visualization allows for an intuitive comparison of not only the total returns achieved by each approach, but also their behavior throughout different market phases. The superior end-of-period value attained by the LSTM+NN strategy demonstrates its enhanced capacity to capture and exploit market opportunities, resulting in a higher growth of invested capital relative to the classical Markowitz benchmark.

The empirical results presented before reveal clear distinctions between the classical Markowitz approach and the advanced LSTM+NN-based strategy for portfolio optimization. As evidenced by the comparative metrics, the LSTM+NN approach demonstrates a capacity for dynamic adaptation to evolving market conditions, a property that is unattainable by the static allocation inherent to the Markowitz framework. Consequently, the dynamic rebalancing enabled by the ML approach allows for a continuous response to the latest patterns in asset returns, potentially capturing short-term market trends and reducing exposure to adverse regimes. This adaptivity is reflected in the observed improvement in annualized returns and Sharpe ratio for the LSTM+NN portfolio, provided that the underlying predictive models (notably the LSTM) achieve sufficient accuracy in forecasting future returns. Conversely, the Markowitz strategy, relying solely on historical statistics, is inherently less responsive to abrupt market changes, which can lead

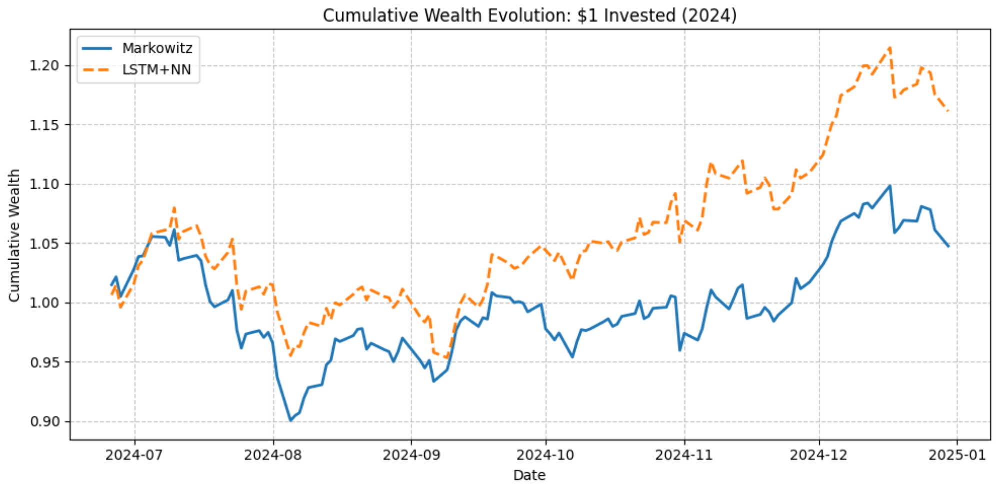

Figure 5.9: Cumulative wealth evolution: \$1 invested according to Markowitz and LSTM+NN strategies during 2024.

to suboptimal performance in highly volatile or non-stationary environments.

However, it is important to recognize that the effectiveness of the LSTM+NN approach is not guaranteed in all scenarios. The performance gains are contingent on the predictive skill of the LSTM model and the ability of the NN to generalize the relationship between historical return patterns and optimal portfolio weights. If the LSTM fails to capture meaningful predictive signals, as may occur in highly efficient or noisy markets, the dynamic portfolio may underperform, or even introduce additional risk relative to the classical benchmark.

Chapter 6

Conclusions and Future Work

6.1 Conclusion

This work has provided a comprehensive and rigorous comparative analysis between two fundamentally distinct approaches to portfolio optimization: the classical mean-variance methodology pioneered by Markowitz, and a modern data-driven pipeline based on LSTM networks combined with fully connected NNs for direct and dynamic allocation of portfolio weights. Through a carefully controlled experimental design, ensuring identical data, constraints, and strictly out-of-sample evaluation, the study delivers robust evidence on the relative strengths and weaknesses of each approach.

The empirical findings clearly demonstrate the potential of machine learning techniques to enhance portfolio performance. The LSTM+NN strategy achieved significantly higher risk-adjusted returns than the static Markowitz benchmark in the out-of-sample period, largely due to its ability to dynamically adapt to new information and evolving market conditions. This adaptivity enables the model to capture short-term trends and respond promptly to changing asset relationships, delivering both superior returns and improved downside risk management as measured by the Sharpe ratio and drawdown analysis.

At the same time, this study highlights several practical challenges and limitations associated with the deployment of advanced machine learning models in portfolio management. Notably, the complexity and lower interpretability of LSTM-based architectures can present hurdles for regulatory approval and institutional adoption. There is also an inherent risk of overfitting, particularly when the available data may not be fully representative of future regimes. The success of the approach is critically dependent on the predictive skill of the LSTM component as poor forecasts can directly degrade portfolio outcomes. Moreover, the analysis has abstracted away from real-world frictions such as transaction costs, liquidity constraints, and operational risk, all of which could affect practical implementation and realized performance.

Despite these challenges, there exist promising avenues for addressing the identified limitations. Model complexity and interpretability may be improved through techniques such as attention mechanisms or model distillation. The risk of overfitting can be further mitigated by adopting more advanced regularization strategies, ensembling methods, or cross-validation across multiple market regimes. To reduce sensitivity to hyperparameter choices and initialization, robust optimization or Bayesian optimization techniques may be employed. Additionally, transaction costs and other implementation frictions should be explicitly modeled and incorporated into the training and evaluation process in future work.

6.2 Future Work

Building on the encouraging results presented in this thesis, several directions for future research emerge. First, extending the analysis to incorporate transaction costs, bid-ask spreads, and liquidity constraints would provide a more realistic assessment of practical investability. Second, exploring alternative ML architectures, such as transformers, temporal convolutional networks, or hybrid models, could further enhance predictive accuracy and allocation robustness. Third, evaluating the generalizability of the approach across different asset classes, international markets, and varying macroeconomic environments would yield valuable insights into its broader applicability. Finally, integrating interpretability techniques and model risk analysis will be essential for bridging the gap between advanced data-driven models and their real-world deployment in institutional portfolio management.

In summary, this thesis demonstrates the significant promise of ML, and particularly RNN, for advancing the field of portfolio optimization. By systematically comparing classical and modern approaches, it provides both empirical evidence and methodological guidance for future research at the intersection of quantitative finance and artificial intelligence.

Appendix A

Glossary of Financial Definitions

This appendix compiles and organizes key financial terms used throughout this TFM. Each definition is stated concisely to ensure the reader possesses the necessary background before entering the machine-learning–driven portfolio optimization methodology.

A.1 Basic Concepts

Asset: Any resource with economic value that can generate future returns. Examples include equities (stocks), bonds, commodities, currencies, and real estate.

Share (Stock): A unit of ownership in a company or corporation. Shareholders can earn returns via price appreciation and dividends.

Bond: A fixed-income security representing a loan from an investor to a borrower (typically a government or corporation). Bonds pay periodic interest (coupon payments) and return the principal at maturity.

Exchange-Traded Fund (ETF): A security that tracks a particular index, commodity, bond, or basket of assets. ETFs trade on exchanges like individual stocks and offer diversified exposure with intraday liquidity.

Index: A statistical measure of the performance of a group of assets (e.g. the S&P 500, FTSE 100, IBEX 35). Indices serve as benchmarks for portfolio performance and are often replicated by index-tracking funds.

A.2 Return and Risk

Return: The gain or loss on an investment over a specified period, typically expressed as a percentage of the initial investment. Common measures include:

- *Simple (Arithmetic) Return:* $R_t = \dfrac{P_t - P_{t-1}}{P_{t-1}}$, where P_t is price at time t.
- *Log Return:* $r_t = \ln\bigl(P_t/P_{t-1}\bigr)$. Log returns are additive over time, simplifying continuous-time modeling.

Expected Return (μ): The anticipated average return of an asset or portfolio, usually computed as the historical mean or a forecasted value from a predictive model.

Volatility: A statistical measure of the dispersion of returns, typically proxied by the standard deviation of returns. High volatility implies larger fluctuations and greater risk.

Covariance (σ_{ij}): A measure of how two assets' returns move together. If assets i and j have returns r_i and r_j, then

$$\mathrm{Cov}(r_i, r_j) = \mathbb{E}\bigl[(r_i - \mu_i)\,(r_j - \mu_j)\bigr].$$

A positive covariance indicates that assets tend to move in the same direction; negative covariance implies inverse movements.

Correlation (ρ_{ij}): The normalized form of covariance, bounded between -1 and 1,

$$\rho_{ij} = \frac{\mathrm{Cov}(r_i, r_j)}{\sigma_i\,\sigma_j},$$

where σ_i and σ_j are the standard deviations of r_i and r_j, respectively.

Risk (Portfolio Variance): For a portfolio with weight vector w and covariance matrix Σ,

$$\mathrm{Var}(\text{portfolio}) \;=\; w^{\top}\Sigma\,w.$$

Risk, in the mean–variance framework, is identified with this variance or its square root (standard deviation).

A.3 Portfolio Construction

Portfolio: A collection of financial assets (e.g. stocks, bonds, ETFs) held by an investor. The portfolio is defined by a weight vector $w = (w_1, \ldots, w_N)$, where w_i is the fraction of the total capital invested in asset i.

Portfolio Weight (w_i): The proportion of total portfolio value allocated to asset i. All weights sum to one:

$$\sum_{i=1}^{N} w_i = 1.$$

Diversification: The practice of allocating capital across multiple, low-correlated assets to reduce idiosyncratic risk. Proper diversification smooths returns and limits drawdowns by ensuring that poor performance in one asset is offset by others.

Efficient Frontier: The set of portfolios that maximize expected return for a given level of risk (variance), or equivalently, that minimize risk for a given expected return. Under Markowitz's Mean–Variance framework, the efficient frontier is traced by solving:

$$\min_w \; w^\top \Sigma w \quad \text{s.t.} \quad w^\top \mu = \mu_p, \quad \mathbf{1}^\top w = 1, \quad w \succeq 0.$$

Capital Allocation Line (CAL): The line that represents combinations of a risk-free asset and a risky portfolio. Its slope (the Sharpe ratio) indicates the best risk–return trade-off achievable by mixing the risk-free asset with the market portfolio.

Sharpe Ratio: A measure of risk-adjusted return defined as

$$\text{Sharpe Ratio} = \frac{\mathbb{E}[R_p] - R_f}{\sigma_p},$$

where R_p and σ_p are the portfolio's expected return and standard deviation, and R_f is the risk-free rate. Higher Sharpe ratios indicate more reward per unit of risk.

A.4 Classical Optimization Models

Mean–Variance Optimization (MVO): The foundational framework introduced by Markowitz [1]. Investors choose weights w to solve

$$\min_w \; w^\top \Sigma w \quad \text{s.t.} \quad w^\top \mu = \mu_p, \quad \mathbf{1}^\top w = 1, \quad w \succeq 0,$$

yielding the efficient frontier of optimal risk–return trade-offs.

Capital Asset Pricing Model (CAPM): Developed by Sharpe (1964) and Lintner (1965) [2], [3], CAPM links an asset's expected return $E[R_i]$ to its systematic risk (β_i):

$$E[R_i] \;=\; R_f + \beta_i \left(E[R_m] - R_f\right), \quad \beta_i = \frac{\mathrm{Cov}(R_i, R_m)}{\mathrm{Var}(R_m)}.$$

Here R_f is the risk-free rate and R_m is the market return. CAPM assumes market efficiency, homogeneous expectations and single-period investment horizons.

Arbitrage Pricing Theory (APT): Proposed by Ross (1976) [23], APT models an asset's return as a linear function of multiple macroeconomic factors F_k:

$$R_i = \alpha_i + \sum_{k=1}^{K} \beta_{ik}\, F_k + \epsilon_i,$$

where β_{ik} are factor sensitivities. APT does not require a single market portfolio but assumes no arbitrage and a large number of assets.

Black–Litterman Model: Introduced by Black and Litterman (1992) [9], this Bayesian framework combines market-implied equilibrium returns (the "prior") with the investor's subjective views to generate posterior expected returns:

$$\mu_{\mathrm{BL}} = \Big[(\tau\Sigma)^{-1} + P^{\top}\Omega^{-1}P\Big]^{-1}\Big[(\tau\Sigma)^{-1}\,\pi + P^{\top}\Omega^{-1}\,q\Big],$$

where π are equilibrium returns, P encodes view exposures, q are view returns, Ω is view-error covariance, and τ is a scaling factor.

Risk Parity: Allocates portfolio weights so that each asset (or asset class) contributes equally to total portfolio risk. If σ_i is the volatility of asset i, a simple risk-parity rule might allocate

$$w_i \propto \frac{1/\sigma_i}{\sum_{j=1}^{N}(1/\sigma_j)},$$

effectively overweighting low-volatility assets to equalize risk contributions [24].

Conditional Value-at-Risk (CVaR) Optimization: Instead of minimizing variance, CVaR optimization seeks to minimize the expected loss beyond a specified Value-at-Risk (VaR) quantile α. If $L(w)$ is the portfolio loss, then CVaR_α is

$$\mathrm{CVaR}_\alpha(w) \;=\; \mathbb{E}\big[L(w) \mid L(w) \geq \mathrm{VaR}_\alpha(w)\big],$$

and one solves

$$\min_w \; \mathrm{CVaR}_\alpha(w) \quad \text{s.t. } \mathbf{1}^\top w = 1,\; w \succeq 0$$

to explicitly control tail risk [25].

A.5 Portfolio Metrics and Constraints

Turnover: The proportion of portfolio value traded (bought or sold) in a rebalancing period. High turnover increases transaction costs and may erode net returns.

Maximum Weight Constraint: A limit on the weight w_i of any single asset to prevent concentration risk, often written as

$$w_i \leq w_{\max}, \quad \forall i.$$

Long-Only Constraint: Prohibits short selling by requiring $w_i \geq 0$. A *long-only* portfolio holds only non-negative positions.

Leverage: The use of borrowed funds to increase exposure. In a leveraged portfolio, $\sum_i |w_i| > 1$. Leverage magnifies both gains and losses.

Benchmark (Market Index): A standard against which portfolio performance is measured. Common benchmarks include the S&P 500, MSCI World, and IBEX 35.

Tracking Error: The standard deviation of the difference between portfolio returns and benchmark returns. A low tracking error indicates returns closely follow the benchmark.

Information Ratio: Measures alpha generation relative to tracking error:

$$\text{Information Ratio} \;=\; \frac{\mathbb{E}[R_p - R_b]}{\sigma(R_p - R_b)},$$

where R_p is portfolio return, R_b is benchmark return, and $\sigma(\cdot)$ denotes standard deviation.

A.6 Machine-Learning–Related Terms

Feature Engineering: The process of creating input variables (features) from raw data (e.g. returns, technical indicators, macro variables, sentiment scores) that improve a model's predictive power.

Hyperparameter: A model parameter set externally (not learned during training) such as learning rate, number of hidden layers, dropout rate, and batch size. Hyperparameter tuning (e.g. with Optuna or Ray Tune) searches for the optimal configuration.

Backpropagation Through Time (BPTT): Extension of backpropagation for sequence models (RNNs, LSTMs), which computes gradients by unrolling the network across time steps and applying the chain rule. BPTT can suffer from vanishing/exploding gradients without gating mechanisms.

Early Stopping: A regularization technique that halts training when validation performance ceases to improve, preventing overfitting.

Dropout: A technique that randomly "drops" (sets to zero) a fraction of neurons during each training iteration, forcing the network to learn redundant representations and reducing overfitting.

Batch Normalization: A layer that normalizes inputs to each mini-batch to have zero mean and unit variance, stabilizing training and allowing higher learning rates.

Cross-Validation (Rolling Window): A method of evaluating time-series models by repeatedly training on a window of past data and validating on the subsequent period ("walk-forward" testing), preserving temporal ordering to avoid look-ahead bias.

Bibliography

[1] H. Markowitz, "Portfolio selection," *The Journal of Finance*, vol. 7, no. 1, pp. 77–91, 1952.

[2] W. F. Sharpe, "Capital asset prices: A theory of market equilibrium under conditions of risk," *Journal of Finance*, vol. 19, no. 3, pp. 425–442, 1964.

[3] J. Lintner, "The valuation of risk assets and the selection of risky investments in stock portfolios and capital budgets," *Review of Economics and Statistics*, vol. 47, no. 1, pp. 13–37, 1965.

[4] R. O. Michaud, "The markowitz optimization enigma: Is 'optimized' optimal?" *Financial Analysts Journal*, vol. 45, no. 1, pp. 31–42, 1989.

[5] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[6] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[7] O. Ledoit and M. Wolf, "A well-conditioned estimator for large-dimensional covariance matrices," *Journal of Multivariate Analysis*, vol. 88, no. 2, pp. 365–411, 2004.

[8] D. Goldfarb and G. Iyengar, "Robust portfolio selection problems," *Mathematics of Operations Research*, vol. 28, no. 1, pp. 1–38, 2003.

[9] F. Black and R. Litterman, "Global portfolio optimization," *Financial Analysts Journal*, vol. 48, no. 5, pp. 28–43, 1992.

[10] V. DeMiguel, L. Garlappi, and R. Uppal, "Optimal versus naive diversification: How inefficient is the 1/n portfolio strategy?" *Review of Financial Studies*, vol. 22, no. 5, pp. 1915–1953, 2009.

[11] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 58, no. 1, pp. 267–288, 1996.

[12] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[13] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *Annals of Statistics*, vol. 29, no. 5, pp. 1189–1232, 2001.

[14] T. Fischer and C. Krauss, "Deep learning with long short-term memory networks for financial market predictions," *European Journal of Operational Research*, vol. 270, no. 2, pp. 654–669, 2018.

[15] S. Bai, J. Z. Kolter, and V. Koltun, "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling," *arXiv preprint arXiv:1803.01271*, 2018.

[16] J. Moody and M. Saffell, "Learning to trade via direct reinforcement," *IEEE Transactions on Neural Networks*, vol. 12, no. 4, pp. 875–889, 2001.

[17] Y. Li, X. Zhao, J. Wang, and W. Zhang, "Deep reinforcement learning for automated stock trading: An ensemble strategy," *IEEE Access*, vol. 7, pp. 23 028–23 038, 2019.

[18] M. F. Dixon, I. Halperin, and P. Bilokon, "Machine learning in finance: From theory to practice," *Springer*, 2018.

[19] J. Heaton, N. G. Polson, and J. H. Witte, "Deep learning in finance," *arXiv preprint arXiv:1602.06561*, 2017.

[20] X. Chen and Q. Lin, "Hybrid portfolio optimization using evolutionary algorithms and neural networks," *Journal of Computational Finance*, vol. 24, no. 3, pp. 1–29, 2020.

[21] X. Ding, T. Zhang, S. Wan, and X. Yu, "End-to-end portfolio optimization using reinforcement learning," *Applied Soft Computing*, vol. 98, p. 106 627, 2021.

[22] E. F. Fama and K. R. French, "The cross-section of expected stock returns," *Journal of Finance*, vol. 47, no. 2, pp. 427–465, 1992.

[23] S. A. Ross, "The arbitrage theory of capital asset pricing," *Journal of Economic Theory*, vol. 13, no. 3, pp. 341–360, 1976.

[24] E. E. Qian, "Risk parity portfolios: Efficient portfolios through true diversification," PanAgora Asset Management, Tech. Rep., 2005.

[25] R. T. Rockafellar and S. Uryasev, "Optimization of conditional value-at-risk," *Journal of Risk*, vol. 2, no. 3, pp. 21–42, 2000.

[26] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989.

[27] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[28] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, 2015.

[29] Y. Bengio, P. Simard, and P. Frasconi, “Learning long-term dependencies with gradient descent is difficult,” *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, 1994.

[30] K. Cho, B. van Merriënboer, Ç. Gülçehre, *et al.*, “Learning phrase representations using rnn encoder–decoder for statistical machine translation,” in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Association for Computational Linguistics, 2014, pp. 1724–1734.