LUISS

Corso di laurea in
Data Science & Management

Cattedra MACHINE LEARNING

End-to-End Portfolio Optimization: Integrating LLMs and Deep Learning in a Web Application for Smart and Personalized Investing

Prof. Giuseppe Francesco Italiano

RELATORE

Prof.Antonio Simeone

CORRELATORE

Matr. Leonardo Bosco - 758501

CANDIDATO

Anno Accademico 2024/2025

THESIS AKNOWLEDGMENTS

In this brief chapter I would like to express my thanks to all the people who supported me during this 2-year trip and that made this journey a truly memorable experience.

First of all, I would like to thank my professor Giuseppe Francesco Italiano that was my professor for 2 courses in my bachelor's degree and professor of one of my favorite courses in the master's degree: "Machine Learning". He was also the one who introduced me into the world of Data Science when I was in high school searching for a university course that I could truly love. I am thankful for the support during these years and during the drafting process of this thesis.

I would also like to express my gratitude to Davide Torre, teaching assistant of professor Italiano in the Machine Learning course since he helped me a lot in the thought process of what to choose as the thesis argument and supported me with continuous feedback on the code and the process underlying the web-application I developed. I am truly thankful for the patience and availability he had.

I am deeply thankful to my splendid family. My mom, who has been here for me throughout these years, always believing in me and encouraging me to do better and become a better person every day. And my dad, who won't be here to see this thesis or the end of this journey, but I am sure he would be proud of this achievement, as he was always and always will be my first and greatest supporter in my heart, together with my mom.

Furthermore, I would like to dedicate a special thought to my precious girlfriend, Francesca, who supported me and kept me company day and night while I was writing and coding, making this last part of this journey not only more bearable but also incredibly beautiful. Without her constant presence, sweetness, and love, reaching this milestone would not have felt as sweet.

Finally, my gratitude goes to all the friends I made along the way, as well as those who have been with me since the very beginning and continue to stand by my side.

Thank you.

ABSTRACT

In recent years, the global investment landscape has undergone a profound transformation, driven in part by technological advancements and the emergence of novel financial instruments. Cryptocurrencies, for instance, captured widespread attention due to their dramatic surges in popularity and market capitalization, only for persistent volatility and regulatory ambiguities to prompt renewed interest in more traditional investment avenues such as corporate stocks, government bonds, and exchange-traded funds (ETFs). Despite the relatively mature frameworks for risk management and return assessment associated with these conventional assets, equity markets remain inherently uncertain, thus requiring careful portfolio construction strategies.

In parallel, recent developments in machine learning, particularly deep learning, have led to sophisticated, data-driven methods that significantly enhance modern portfolio management. Predictive algorithms can now incorporate a multitude of market signals and historical data, forecasting both returns and underlying risk profiles with increasing levels of accuracy. A vital consideration in this process is the balance among volatility, correlations between assets, expected returns, and individual risk tolerance, factors that collectively form the cornerstone of portfolio optimization.

Building on this evolving landscape, the present work introduces a novel framework for a web-based application designed to integrate advanced artificial intelligence (AI) models into a user-friendly investment platform. Specifically, the system applies state-of-the-art methods to optimize capital allocation across multiple stock tickers, with an objective of mitigating risk, maximizing returns, and adapting to dynamic market conditions. Notably, this platform leverages a Large Language Model (LLM) as a conversational interface, thereby allowing users to submit natural language queries and receive comprehensive, explanatory feedback, ranging from raw portfolio weight distributions to interpretative commentary on factors such as volatility and historical performance. By employing a 1 trading year in the future test procedure, the system's performance is evaluated against standard financial metrics (annualized returns, Sharpe ratio, volatility, and maximum drawdown), ensuring a rigorous appraisal of both strengths and limitations. Ultimately, this thesis demonstrates how the convergence of AI-driven analytical techniques and intuitive, dialogue-based interfaces can empower both novice and experienced investors to make more informed decisions in a volatile and complex market environment.

INDEX

CHAPTER 1
INTRODUCTION

1.1 - Background and Motivation

The global investment ecosystem has experienced significant transformations over the last decade, influenced by regulatory shifts, rapidly increasing computational power, and the widespread adoption of innovative financial technologies. Participants in contemporary markets, ranging from individual retail traders to large-scale institutional investors, now have an extensive selection of instruments at their disposal, spanning equity, debt, commodities, foreign exchange, and cryptocurrencies. While digital assets such as Bitcoin and Ethereum initially attracted intense enthusiasm and media attention, the volatility and evolving regulatory status of these assets prompted many portfolio managers to refocus on more traditional forms of investment. In these established markets, investors benefit from a robust body of research on asset pricing and risk management, supplemented by well-known metrics such as the Sharpe ratio, maximum drawdown, and portfolio volatility. Nonetheless, even traditional equity and bond markets face a multitude of challenges stemming from macroeconomic indicators, geopolitical events, and internal corporate dynamics. Against this backdrop, the effort to balance optimal returns with effective risk management has grown increasingly complex and remains a fundamental goal for modern investment practices.

1.2 - Role of Artificial Intelligence in Modern Portfolio Management

In light of these complexities, artificial intelligence (AI) has emerged as a transformative approach to financial analysis, offering sophisticated ways to interpret the large and intricate datasets that characterize contemporary markets. Machine learning (ML) and deep learning techniques are particularly adept at identifying non-linear relationships, seasonal trends, and subtle patterns in market data. These capabilities contrast with traditional econometric methods, which often rely on more rigid assumptions or linear modeling. Within the domain of portfolio management, AI-driven models deliver a range of benefits. They improve predictive accuracy by leveraging extensive historical data and diverse sources of information, including macroeconomic indicators, corporate fundamentals, and social media sentiment. They also facilitate the automatic extraction of relevant features by exploiting architectures such as recurrent neural networks (RNNs) and Long Short-Term Memory (LSTM) units, which uncover temporal dependencies in data,

as well as convolutional neural networks (CNNs), which can interpret information in ways analogous to image recognition. Furthermore, AI models are inherently adaptive and can be retrained or fine-tuned to accommodate new data, enabling them to respond to changing market conditions. This responsiveness is particularly valuable in periods of sudden price movements, regime shifts, or heightened volatility.

1.3 - The Need for User-Centric Design

Despite the considerable potential of AI-based solutions in finance, significant challenges persist in translating sophisticated technical analyses into tools that are accessible and meaningful to end-users. Financial markets include participants with a broad spectrum of experience, from novices seeking introductory investment guidance to professional fund managers who require advanced analytics. The manner in which AI-generated insights are presented and interpreted is therefore as crucial as the accuracy of the underlying models. Traditionally, financial technology platforms have expected users to navigate complex dashboards and specialized terminology, creating barriers for those without extensive financial or technical backgrounds. Incorporating a natural language interface can mitigate these challenges by simplifying the query process and offering explanations in everyday language. The present work addresses this need by integrating a Large Language Model (LLM) that not only provides users with optimized asset weight distributions but also supplies rationales and interpretive commentary. Through a conversational interface, the system aspires to broaden access to powerful computational tools, encouraging a higher level of financial literacy and more informed decision-making.

1.4 - Proposed Framework

The thesis centers on the design and evaluation of a web-based AI-driven system for portfolio optimization, composed of two principal components. The first component predicts asset returns and volatility by employing an LSTM-based neural network to estimate both the expected returns and variance over a specified time horizon. These forecasts constitute a critical input for any subsequent allocation strategy because they inform the anticipated performance and risk of each asset in the portfolio. The second component focuses on allocating capital through convex optimization, making use of frameworks such as CVXPY to identify weight distributions that reconcile desired returns with defined risk constraints. This approach incorporates standard portfolio theory

requirements, including non-negativity of weights, budget constraints, and maximum asset weight thresholds, while remaining adaptable to user directives, such as concentrating on a predefined subset of stocks, all under the umbrella of a unique end-to-end process that we will discover in chapter 4. The introduction of an LLM-based interface sets this project apart from many traditional portfolio optimization tools. By allowing users to pose questions in everyday language (for example, “How should I allocate my budget among these five stocks?”), the system generates a detailed response that outlines suggested asset weightings, explains the various financial output metrics of interest and suggest the user to change tickers if the results are not optimal.

1.5 - Scope and Contributions

The thesis contributes to the academic and practical discourse in several ways. It demonstrates the effective integration of predictive modeling with convex optimization in one cohesive workflow, thereby uniting the strengths of data-driven AI methods with the reliability of classical portfolio theory. It also showcases how explanations generated by a language model can enhance the interpretability of complex financial tools, addressing a persistent gap in platforms that often provide numerical outputs without actionable or understandable context. Moreover, the solution is rigorously tested on a test set of 1 trading year (252 days), using standard metrics such as annualized returns, Sharpe ratio, volatility, and maximum drawdown to evaluate its performance under realistic conditions. In addition, a modular software structure allows for straightforward enhancements or substitutions of the predictive elements, optimization routines, and user interface components, facilitating ongoing adaptation as new analytical methods or technologies emerge. By combining advanced forecasting capabilities with an accessible conversational interface, this work aims to broaden the appeal and utility of AI-driven portfolio optimization, ultimately enabling a wider range of investors to make strategically sound decisions in rapidly evolving financial markets.

1.6 - Organization of the Thesis

Following this introduction, **Chapter 2** provides a literature review that synthesizes existing research on portfolio optimization, machine learning in financial forecasting, and user interface design in fintech applications. By examining established theoretical frameworks and

state-of-the-art approaches, it lays the groundwork for the subsequent methodological and practical contributions of this study.

In **Chapter 3**, the focus shifts to the datasets and preparatory steps involved in developing the predictive models. Attention is given to the rationale behind certain design choices, particularly those driven by computational constraints and time considerations, as well as to the manner in which data were preprocessed to ensure reliable model inputs.

Chapter 4 delves into the iterative process of trial and error that characterized the early experimentation phase. Various model architectures, optimizers, and objective formulations were tested before settling on the final LSTM-based forecasting framework and convex optimization approach. The chapter also discusses how these explorations led to refinements in both the algorithmic design and the overall methodology.

Chapter 5 presents the core experimental results, along with a comprehensive performance evaluation of the integrated system. Emphasis is placed on the strengths of the proposed solution, as well as on potential areas for enhancement, with a view to guiding future research and development. Subsequently, **Chapter 6** examines the development of the web-based application and its user interface, highlighting how usability and design considerations can facilitate broader adoption of AI-driven financial tools.

In **Chapter 7**, the thesis addresses the role of Large Language Models in both the input and output stages of the application. This discussion centers on why an LLM was incorporated—chiefly to offer interpretable explanations and natural language support—and how it can enhance user engagement and clarity. Finally, **Chapter 8** concludes the thesis by summarizing the key findings, outlining the practical implications of the proposed framework, and suggesting avenues for continued investigation.

Bringing these chapters together, the thesis seeks to make a meaningful contribution to the field of AI-driven finance. By uniting sophisticated predictive modeling with user-centric design and natural language interaction, it aspires to present a robust, empirically validated solution capable of guiding a diverse array of investors through an increasingly dynamic and complex market environment.

CHAPTER 2

LITERATURE REVIEW

This chapter provides a comprehensive review of the theoretical and technological underpinnings that inform AI-based portfolio optimization and user-centric design in financial technology. The discussion begins with Traditional Portfolio Theory, highlighting how foundational frameworks established key risk–return concepts. We then trace the evolution of Machine Learning in Finance, focusing on the early adoption of non-linear computational methods and the subsequent challenges arising from market non-stationarities. Next, we move on to Deep Learning for Time Series Forecasting, where we examine the practical advantages of Long Short-Term Memory (LSTM) architectures in capturing intricate temporal patterns in financial data. Following this, we delve into End-to-End Portfolio Optimization, an emergent paradigm in which forecasts and allocations are trained jointly under a unifying loss function geared toward real-world performance metrics. Finally, we discuss the growing role of Large Language Models (LLMs) in finance, particularly in improving user engagement and interpretability for sophisticated investment tools. By weaving these themes together, the chapter sets a cohesive stage for the methodological contributions presented later in the thesis.

2.1 - Traditional Portfolio Theory

The foundational work of Markowitz (1952) established the modern framework for portfolio selection, explicitly modeling the balance between expected return and variance. According to Modern Portfolio Theory (MPT), rational investors aim to position themselves on the so-called “efficient frontier,” which characterizes the portfolios that achieve the highest possible return for a given level of risk, or equivalently, the lowest risk for a given level of return. A particularly influential insight of MPT is the notion of diversification, wherein combining assets with different correlation structures can reduce overall portfolio volatility without substantially diminishing returns (Elton & Gruber, 1997).

Building upon these core ideas, Sharpe (1964) and Lintner (1965) introduced the Capital Asset Pricing Model (CAPM), which emphasizes systematic (market-wide) risk through a metric known as beta. CAPM posits a linear relationship between an asset’s expected return and its exposure to the broader market, thus offering an equilibrium-based view of risk–return trade-offs. While widely adopted, CAPM and similar factor models rely on fairly restrictive

assumptions about market efficiency, linearity, and stationarity. Merton (1980) further highlighted how real-world considerations, such as time-varying risk premia or difficulties in accurately estimating returns and covariances, can undermine the stability of these classical models.

Despite these critiques, traditional portfolio theory remains a cornerstone of quantitative finance. It not only offers a structured way to conceptualize the interplay between returns, variance, and correlation, but also underscores the profound influence of risk considerations in investment choices. This foundational perspective, however, provides only part of the picture, as many real-world assets exhibit non-linear relationships or structural regime shifts that cannot be fully captured through static, linear models alone. These limitations helped pave the way for more adaptive, data-driven techniques.

2.2 - Machine Learning in Finance

Against the backdrop of mounting computational capabilities and the rise of large-scale financial databases, Machine Learning (ML) began to gain prominence as a tool for financial modeling. Initially, ML approaches were deployed to derive potentially non-linear relationships that conventional statistical models tended to overlook. Early work by Allen and Karjalainen (1999) showcased how genetic algorithms could discover technical trading rules, revealing complex patterns in market time series. These applications demonstrated the feasibility of ML in detecting nuanced signals beyond the scope of classical linear regressions also if results from this method showed that, after accounting for transaction costs, these rules did not generate returns superior to a simple “buy and hold” strategy during out-of-sample test periods, they demonstrated that models could “understand” where it would be convenient to stay or exit the market.

Nevertheless, these early ML methods often relied on labor-intensive feature engineering and conducted forecasting and portfolio construction in separate stages. For instance, an ML model might predict asset returns, but the subsequent optimization step to determine weights in a portfolio would remain disconnected, frequently performed by a conventional solver that optimizes standard criteria (e.g., mean–variance). This disjointed “two-stage” approach can lead to a mismatch between the objective of the forecasting model (e.g., mean squared error minimization) and the ultimate performance goals of the investor (e.g., maximizing the Sharpe

ratio). Moreover, non-stationary markets accentuated the overfitting problem, necessitating regular retraining or recalibration of these models.

Concurrently, risk management and capital allocation frameworks started embedding ML-derived forecasts into existing quantitative strategies. Although the synergy held promise, particularly in detecting hidden factors in large cross-sections of assets, practitioners realized that traditional metrics for forecast accuracy did not necessarily translate into superior portfolio performance. This realization intensified calls for more end-to-end solutions, wherein model training could incorporate downstream financial objectives rather than focusing on purely predictive performance.

2.3 - Deep Learning for Time Series Forecasting

Within the broader ML landscape, deep learning has proven especially adept at extracting features from complex, high-dimensional datasets, an attribute that naturally fits financial time series. Early deep learning architectures struggled with sequences, but Recurrent Neural Networks (RNNs) offered a first step toward modeling temporal structure. Still, these RNNs frequently suffered from vanishing and exploding gradients, hindering their ability to capture long-range dependencies.

The Long Short-Term Memory (LSTM) architecture, proposed by Hochreiter and Schmidhuber (1997), addressed these issues through a gating mechanism that modulates the flow of information. LSTMs can effectively "remember" significant events over extended periods, rendering them especially suitable for forecasting financial indicators that may display subtle cyclical or lagged behaviors (Zhang et al., 2017). Researchers have demonstrated that LSTMs often outperform both simpler neural networks and certain traditional econometric models in predicting returns or volatility (Fischer & Krauss, 2018).

Despite these successes, the bulk of deep learning studies in finance still center on predictive tasks, forecasting future prices, returns, or volatility, leaving the subsequent asset allocation choices to be determined separately. This scenario can yield robust error metrics at the model level, yet no guarantee exists that these forecasts will result in the optimal risk–return trade-off within the actual portfolio. In effect, there is a disconnect between forecast accuracy and practical portfolio performance, an issue that has spurred the exploration of end-to-end

portfolio optimization techniques that align the training process with ultimate investment outcomes.

2.4 - End-to-End Portfolio Optimization

End-to-end portfolio optimization aims to address the drawbacks of two-stage workflows by combining forecasting and allocation in a single, fully differentiable pipeline (Moody & Saffell, 1999; Deng et al., 2016). In this unified framework, the neural forecaster generates predictions, such as expected returns μ or volatility parameters σ, that feed directly into a differentiable optimizer. The optimizer then computes portfolio weights under constraints (e.g., a budget constraint or leverage limits) to maximize a pre-defined performance metric such as the Sharpe ratio. By allowing gradients to pass through both the forecaster and the optimizer, the entire system calibrates itself toward actual portfolio performance, rather than surrogate metrics like mean squared error of the forecast.

This approach has become more tractable thanks to advances in differentiable optimization libraries. Agrawal et al. (2019) showcased the feasibility of incorporating convex optimization problems into neural network pipelines through CvxpyLayers, which compile such problems into forms conducive to automatic differentiation. Building on these ideas, Zhang et al. (2020) demonstrated how a Markowitz-inspired layer (with constraints akin to budget and long-only conditions) could be trained end-to-end to improve out-of-sample Sharpe ratios.

While promising, end-to-end approaches face unique challenges. Model interpretability becomes more complicated, given that both the forecasting and the optimization processes are learned simultaneously in a “black box” setting. Overfitting risks also intensify if the system tailors itself too closely to historical market peculiarities, thereby reducing robustness to unforeseen conditions. Nonetheless, the capacity to optimize directly for metrics that matter to investors marks a substantial improvement over conventional siloed methodologies and sets the stage for deeper integration of predictive analytics and allocation logic.

2.5 - Large Language Models in Financial Applications

Parallel to these developments, Large Language Models (LLMs) have begun reshaping how users interact with complex financial analytics. GPT-type models (Radford et al., 2019; Brown et al., 2020) train on vast text corpora and exhibit strong capabilities in language

comprehension and generation. Their transformer-based architectures enable them to handle tasks as diverse as summarization, translation, and interactive dialogue.

In the financial realm, LLMs open the door to more user-friendly and interpretable systems. For instance, a platform might permit an investor to say, “Allocate half of my portfolio to tech stocks, and half to a broad index fund.” The LLM interprets this directive, checks it against the underlying optimization framework, and returns an allocation plan. Additionally, LLMs can generate plain-English explanations of why certain assets received specific weightings. However, these models can also produce “hallucinations,” meaning they may invent facts or propose solutions not grounded in real data. Consequently, robust validation mechanisms—such as function-calling approaches or carefully curated prompts, are essential to ensure reliability and accuracy in high-stakes domains like portfolio management.

2.6 - Synthesis and Directions

In surveying the trajectory from Traditional Portfolio Theory to contemporary methods that leverage deep learning and language models, a consistent theme emerges: financial markets demand increasingly adaptive, user-friendly solutions. Markowitz’s pioneering framework and its subsequent refinements revealed that diversification and risk–return balance are fundamental, yet they often underemphasize the market’s complex, non-linear patterns. Early machine learning efforts demonstrated the promise of algorithmic trading and predictive modeling, but a gap persisted between purely predictive accuracy and real-world portfolio performance. The rise of deep learning architectures, especially LSTMs, added crucial capacity for handling temporal dependencies in noisy financial series; however, siloed “two-stage” approaches, where forecasting and allocation are conducted separately, still risk suboptimal outcomes. By embedding convex optimization directly into a neural forecasting pipeline, the emerging paradigm of end-to-end portfolio optimization directly aligns predictive models with the metrics investors care about, such as Sharpe ratio and drawdown.

Parallel advancements in Large Language Models enable more intuitive user interactions and improved interpretability, addressing the frequent criticism that high-performing machine learning systems are often opaque to practitioners. In this thesis, these insights converge in a holistic solution: an AI-driven portfolio optimizer that integrates LSTM-based time-series forecasting with differentiable allocation layers and an LLM-enhanced interface. This framework not only refines risk–return outcomes relative to conventional methods but also

redefines accessibility, allowing both novice and expert users to engage in natural-language dialogues about asset selection, performance rationale, and risk metrics. By bridging sophisticated analytics with user-centric design, the proposed system stands to broaden the adoption of advanced portfolio optimization in financial services, paving the way for more transparent, adaptive, and data-driven investment decisions.

CHAPTER 3
EDA & DATA PREPARATION

In this chapter, we present the exploratory data analysis (EDA) conducted on our dataset and outline the series of preprocessing steps that precede the development of our forecasting and portfolio allocation system. The end goal is to ensure that the raw data, for a comprehensive set of stock tickers spanning multiple sectors, is transformed into a high-quality, time-aligned, and properly scaled format, enabling robust model training and evaluation. We begin by discussing how these data are sourced and inspected, followed by a detailed description of how daily returns are computed and further refined for modeling. By the chapter's conclusion, the reader will have a clear understanding of the data pipeline, including the reasoning behind particular design choices (e.g., scaling parameters, lookback windows, and train-test splits).

3.1 - Data Collection And Overview

The dataset for this project encompasses a broad array of publicly traded stocks, with a focus on a list of 20 high-capitalization tickers from various industries such as technology, finance, consumer staples, and energy, among others. These tickers were selected to capture a diverse cross-section of the market. On top of these primary assets, the ultimate system architecture also accommodates additional user-selected tickers (usually three or four). The overall idea is that the model, having learned from the entire set of 20 major stocks, will be capable of creating tailored weight allocations for whichever subset the user chooses. While this feature is not the focal point of the present chapter, it illustrates the broader flexibility of the system.

Data were downloaded using the yfinance library, which automates the retrieval of historical market data from Yahoo Finance API. Specifically, we focused on the adjusted closing prices (Adj Close), which account for dividends, stock splits, and other corporate actions, ensuring that the price series more accurately reflects the underlying value to investors. Our data spanned a date range from 2015-01-01 through 2025-01-01, yielding a long enough historical period to capture different market regimes and, in turn, provide the model with a variety of scenarios for improved generalization.

Once the price data was obtained, we performed an initial inspection to identify any missing records or potentially erroneous values. As is typical for large-cap stocks, missing rows were

primarily due to weekends, holidays, or occasional trading halts. These non-trading periods were dropped for simplicity and to not add any type of bias to the model. A final integrity check was then performed to confirm that the dataset did not exhibit irregularities or implausible price movements outside recognized market events.

3.2 – Computing Daily Returns and Initial Exploration

Raw stock prices present a convenient overview of performance over time, but they are not always the most effective inputs for predictive modeling. Instead, we computed daily returns for each ticker to represent the percentage change in value from one trading day to the next:

$$r_t = \frac{P_t - P_{t-1}}{P_{t-1}}$$

where P_t is the adjusted closing price on day t. This transformation normalized the data and provided a consistent measure of relative price movements. Using daily returns rather than raw prices, in fact, helps the model focus on fluctuations that are more directly comparable across assets with different price levels.

3.3 – Constructing Rolling Windows

With cleaned and validated return data in hand, we constructed a rolling window dataset for model training. This step is essential for capturing temporal dependencies in financial time series, an area in which machine learning models, especially recurrent networks that we are indeed going to use, tend to excel. Our approach involved specifying three parameters:

Lookback: The number of days of historical returns provided to the model as an input context. In our experimental setup, this was set to 90 days:

Gap: A short buffer period of 2 days between the last day in the lookback window and the start of the forecast horizon, ensuring that the model has no inadvertent overlap between the training window and the future it is tasked to predict.

Horizon: The length of the forecast period, configured to 30 days in our code, representing the range of future returns that the model must estimate. In our work the decision for the 30-days was chosen with the idea of evaluating the idea of changing weights every month.

For each valid index i in the daily returns array, we define an input X_i that contains 90 days of historical returns for all tickers, and a target y_i containing 30 days of "future" returns offset by the 2-day gap. These (X_i, y_i) samples form the core of our supervised learning dataset. We also stored corresponding timestamps for clarity in back testing and evaluation.
To manage the rolling windows and scaling steps, we utilized the DeepDow library, which provides InRAMDataset, RigidDataLoader, and Scale modules that simplify custom time-series transformations.

The DeepDow library provides unified methods for slicing and transforming time-series data in a manner that simplifies the construction of rolling windows, particularly through its InRAMDataset, RigidDataLoader, and Scale modules. InRAMDataset efficiently stores and slices the entire time series in memory according to user-defined lookback and horizon parameters, automatically generating the overlapping windows needed for both training and validation. RigidDataLoader ensures these windows are batched with consistent shapes and preserves the sequential order when necessary, reducing the complexity of ensuring uniform mini-batches across the time dimension.

Additionally, the Scale module and other transformations can be chained into the same pipeline, so that each window can be scaled or normalized on the fly, thereby eliminating the extra overhead of manually preprocessing slices and ensuring a clean and reproducible workflow.

Next, we employed a chronological split for training and testing, allocating all of the available samples (starting from the earliest) to training and only 252 trading days to testing, resembling a traditional trading year. This chronological approach respects the time-ordered nature of financial data, helping mitigate the risk of look-ahead bias wherein future information might inadvertently influence earlier model training.

3.4 – Scaling and Data Preparation Workflow

Deep Neural Networks often benefit from standardized input features especially to speed up backpropagation, since the algorithms used to update the Neural Network weights often work better with scaled data. Consequently, we applied a scaling procedure to the daily returns for all tickers. Specifically, we computed means and standard deviations for the training partition and used these statistics to normalize both the training and test partitions to avoid data

leakage into the test set. This transformation ensures that the daily returns from each asset possess near-zero means and unit variances, preventing any single ticker, particularly those with higher nominal volatility, from dominating the model's objective function.

After scaling, the final data preparation included reshaping the inputs to the format required by our chosen neural networks. More concretely, each input tensor X took the shape:

$$(batch\_size\ , 1, lookback, n\_assets),$$

which aligns with the expected input structure for one-dimensional LSTM cells that process temporal sequences. This design also facilitates mini-batch training, wherein smaller subsets of the data help stabilize gradient estimates, leading to more reliable and often quicker convergence.

Because the model ultimately focuses on allocating capital only to user-requested assets in addition to the core 20 major tickers, we needed to maintain flexibility in how we handle indexing. While the training process leverages the entire cross-sectional array of daily returns, constraints in the allocation layer selectively zero out weights for assets not requested by the user. This procedure is enforced through a simple mask in the convex optimization routine, ensuring that users can incorporate additional tickers they specify without disturbing the broader modeling logic that learns from the entire ticker universe. We will see more about this in the next chapter.

3.5 – Concluding Remarks on EDA and Data Preparation

The procedures outlined in this chapter represent the foundation upon which our subsequent modeling and optimization techniques depend. Data from multiple sources were integrated, cleaned, and shaped into coherent windows of historical returns ready for forecasting.

By adhering to a transparent pipeline involving careful data splitting, rolling window construction, and feature scaling, we reduce the risk of data leakage and ensure that each step is properly documented and reproducible. These decisions also facilitate experimentation with various neural network architectures and optimization routines, as described in upcoming chapters. Ultimately, a rigorous approach to EDA and data preparation positions us to produce

meaningful, consistent results that reflect genuine market behaviors, thereby bolstering the reliability of the model's subsequent predictions and allocations.

CHAPTER 4

METHODOLOGICAL FRAMEWORK AND MODEL EXPLORATION

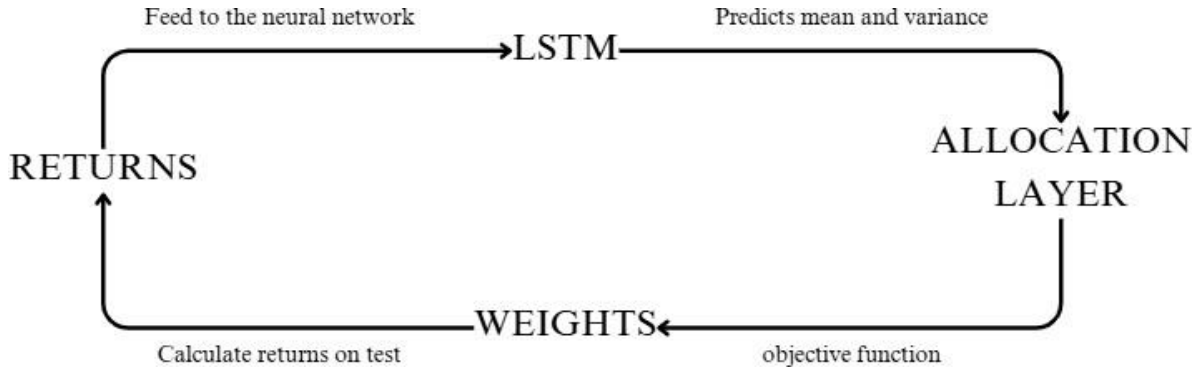
In the chapters preceding this point, we conducted an extensive exploration of our dataset and established a clear procedure for data preprocessing. This chapter focuses on the methodological backbone of our work: designing, training, and evaluating a neural network architecture that interfaces directly with a portfolio allocation module. The key innovation here lies in our end-to-end approach, wherein model parameters, both in the neural forecaster and the subsequent allocation layer, are updated jointly by maximizing a differentiable approximation of portfolio performance metrics such as the Sharpe ratio treating it as a convex optimization problem.

The chapter is structured as follows. First, we introduce our neural forecasting module, highlighting its architecture and motivation for using a recurrent structure like the Long Short-Term Memory (LSTM). Second, we detail the construction of two different allocation layers, one based on a mean-variance (Markowitz-style) objective and another that incorporates an entropy term, both embedded within the neural network via a differentiable convex optimization interface. Finally, we explain how these components are trained in tandem using a customized Sharpe ratio-based loss function, underscoring how this end-to-end formulation departs from traditional "two-stage" methods that separately optimize forecasts and allocations.

4.1 – Conceptual Overview of the End-to-End Architecture

Traditional quantitative investment workflows often separate the forecasting of returns or other relevant financial quantities from the subsequent portfolio construction. In such a two-stage approach, a machine learning model forecasts expected returns (and possibly variances or covariances), after which a portfolio optimization method (e.g., mean-variance, risk-parity, or others) is used to derive the allocation weights. While conceptually straightforward, this simple methodology does not account for potential misalignment between the loss function used to train the forecaster, often a mean squared error or likelihood-based objective, and the ultimate performance metric of interest such as portfolio returns, volatility, or the Sharpe ratio.

Our design explicitly merges the forecasting and allocation tasks into a single computational graph. The neural network (an LSTM in our work) processes historical input data, daily returns from a rolling window, and outputs parameters that feed directly into a differentiable convex optimization layer. This layer then produces portfolio weights. Crucially, the entire pipeline is trained by directly maximizing a metric related to risk-adjusted returns.



The benefits of this integrated approach are manifold:

Alignment of Objectives: The model learns forecasts in a manner that is directly pertinent to improving portfolio performance, rather than minimizing a generic error metric (such as MSE).
Risk-Return Trade-off: By embedding the portfolio optimization step within the training loop, the system can dynamically learn how "correct" forecasts translate into improved risk-return profiles.
Gradient Flow: Thanks to recent advances in differentiable optimization libraries, we can propagate gradients through the convex solver, enabling end-to-end parameter updates of both the forecaster and the allocation module.

4.2 - Neural Forecaster: LSTM-Based Architecture

Modern finance frequently relies on modeling time series data characterized by non-stationarities, sudden regime shifts, and high levels of noise. Traditional neural networks and basic Recurrent Neural Networks (RNNs) can struggle under these conditions, especially when the relevant temporal patterns extend hundreds of trading days. As a response to these issues, Long Short-Term Memory (LSTM) networks introduce carefully engineered gating mechanisms, permitting the model to retain pertinent information for longer horizons, and mitigating the well-documented "vanishing gradient" phenomenon.

In this section, we delve deeply into why LSTMs are appropriate for financial forecasting, how they structurally differ from simple RNNs, and how their outputs are ultimately shaped to provide the parameters needed in our downstream portfolio allocation layer.

4.2.1 Motivation for Using LSTMs

Standard RNNs attempt to process sequential data by maintaining a hidden state that is iteratively updated as new inputs arrive. While this architecture captures short-term patterns well, it frequently falters in identifying or remembering long-term dependencies, particularly important in financial contexts, where market sentiments, macroeconomic factors, or seasonal effects might exert influence weeks or months later.

A primary reason for this limitation is the vanishing (or exploding) gradient problem: as gradients are back-propagated through many timesteps, they can progressively shrink (or blow up), impeding effective learning. LSTM networks address this issue by including a specialized memory cell (the "cell state") and a collection of gates, namely the forget gate, input gate, and output gate, that regulate the flow of information.

Hochreiter and Schmidhuber originally proposed LSTMs to counteract the vanishing gradient through these gating mechanisms, effectively allowing certain aspects of the time series to be "forgotten," while retaining crucial signals for longer intervals.

In the context of financial forecasting:

Long-Term Dependencies: Asset prices can exhibit trends, momentum, or cyclical behaviors that span multiple months. LSTMs are designed to capture these extended temporal relationships through careful control of the memory cell.

Noise Reduction: Financial time series are noisy. By learning to selectively keep or discard information, LSTMs can filter out short-lived spikes or jitters.

Modular Design: Each LSTM cell processes inputs step-by-step, which aligns neatly with daily returns data. Such a design allows for robust mini-batch training strategies while preserving the notion of sequence.

As a result, LSTMs often outperform both simple feed-forward architectures and vanilla RNNs in tasks like stock return forecasting or volatility prediction, where the relevant patterns may span an unknown number of days.

4.2.2 - Network Structure and Input-Output Format

In our framework, the LSTM-based neural forecaster serves as the central component for modeling sequential patterns in financial data. This network ingests rolling windows of daily returns for multiple assets and transforms these inputs into predicted parameters essential for subsequent portfolio allocation. In our experiments we decided to use lookback windows of 90 days instead of 120 just for a computational power issue. The code is indeed very slow on a cpu-only machine so we had to constraint the model at the best of our ability.

A crucial design choice when building deep learning models for time series is how best to represent historical information. Here, as we said before, each sample is shaped as a 4D tensor:

$$(batch\_size, 1, lookback, n_{assets})$$

Although the second dimension (size =1) might initially seem superfluous, it provides room for future extensions, such as including multiple channels of data or additional transformations since LSTM can work with quantitative data but also with other types of data, we will talk about this in the conclusions. Internally, we "squeeze" this dimension so that each mini-batch entry essentially becomes a sequence of length $lookback$ with n_{assets} features per timestep.

From a financial perspective, each of those features corresponds to the daily return of an individual asset. Consequently, the model observes the cross-section of returns at each point in time, enabling it to learn how different assets may co-move or diverge over the lookback window. By the end of this sequence, the LSTM has traversed 90 days of historical returns, capturing both short- and potentially long-range dependencies.

Long Short-Term Memory networks are particularly well-suited to financial time series due to different problematics of traditional Recurrent Neural Networks that they are able to tackle thanks to their composition.

Traditional Recurrent Neural Networks (RNNs), in fact, often struggle with Vanishing Gradient / Exploding Gradient problem as well as, in which gradient magnitudes decay or grow

exponentially when attempting to learn long-range dependencies. LSTMs overcome this challenge by introducing a memory cell and three gating mechanisms, namely the forget gate, input gate, and output gate, that allow the network to regulate what information is retained or discarded over time.

When the model processes each rolling window of returns, it does so one day (or one timestep) at a time. At each step, it begins by computing the forget gate, which determines how much of the previous cell state should be preserved. Formally, the forget gate is defined by:

$$f_t = \sigma(U_f X_t + W_f H_{t-1} + b_f)$$

where X_t is the current input (the cross-sectional asset returns on day t), U_f is the weight associated with the input, H_{t-1} is the hidden state of the previous timestamp, W_f is the weight matrix associated with the hidden state and σ is the sigmoid activation function that squashes values into the interval $[0,1]$. If f_t is close to 1, the LSTM largely keeps the older information in its internal memory; if f_t is close to 0, it "forgets" that information. This mechanism is invaluable in finance, where certain historical data points become irrelevant after, for instance, a major news event or a regime shift. The term b_f represent biases for the respective gate.

Next, the model decides how much new information to incorporate from the current input via the input gate which is used to quantify the importance of the new information carried by the input. This is defined by the formula:

$$i_t = \sigma(U_i X_t + W_i H_{t-1} + b_i)$$

This gate provides a learned measure of how relevant the present-day signals are. In parallel, a candidate update C_t is computed using a tanh activation function:

$$\tilde{C}_t = \tanh(U_c X_t + W_c H_{t-1} + b_c)$$

The product $i_t \odot \tilde{C}_t$ (where $\odot$ is elementwise multiplication) represents the specific fraction of new information to be added to the memory cell. The cell state itself is updated by combining this addition with the fraction of old memory retained by the forget gate:

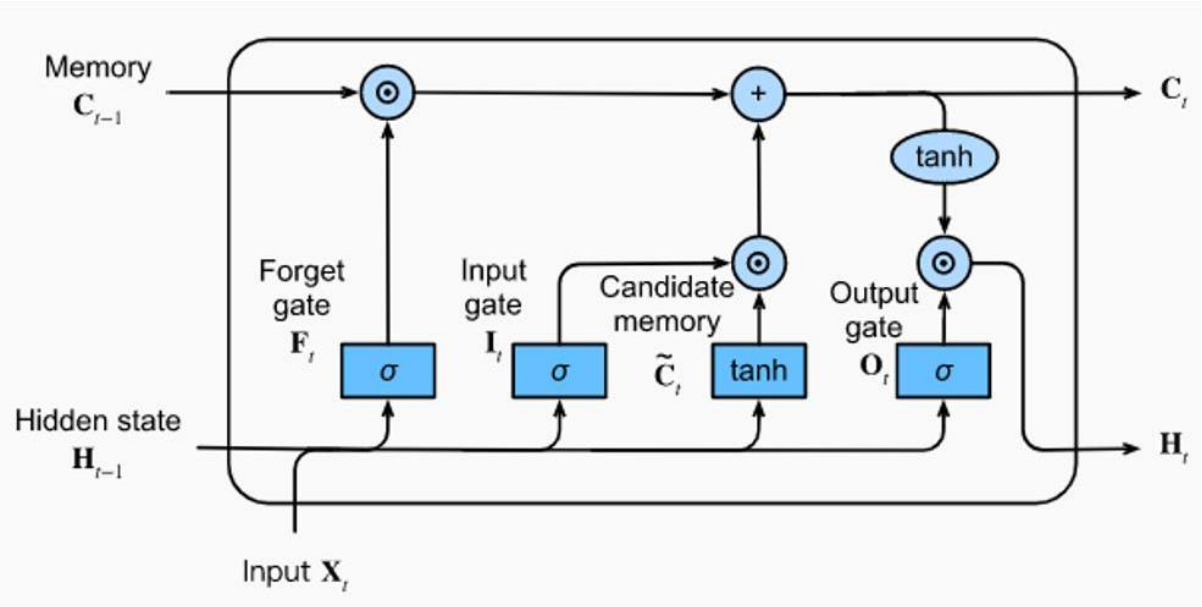
$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$

This updated cell stateC_t thus carries forward both older, still-relevant information and newly introduced signals from the current timestep, providing a flexible and powerful means of encoding longer-term dependencies in the data.

The output gate then controls what portion of the updated cell state is exposed as the hidden state HtH_tHt, which effectively functions as the immediate "summary" of everything the model has inferred thus far. Mathematically,

$$o_t = \sigma(U_o X_t + W_o H_{t-1} + b_o), \quad H_t = o_t \odot \tanh(C_t)$$

This gating logic ensures that at each timestep, the model can filter the cell state through a nonlinearity, revealing only the information deemed most relevant for the next computational steps.



An Intuitive Explanation of LSTM by Ottavio Calzone
(https://medium.com/@ottaviocalzone/an-intuitive-explanation-of-lstm-a035eb6ab42c)

After the LSTM processes the entire lookback window the network's final hidden state h_T encapsulates the salient signals gathered from that entire historical period. This final hidden state is then passed to a fully connected layer that outputs $2 \, x \, n_{assets}$ parameters. We interpret the first n_{assets} of these parameters as the model's estimates for the expected mean returns μ ̂, and the remaining n_{assets} as the log-variances $\log \sigma$ ̂$_i$. Exponentiating the latter produces strictly positive variance (or volatility) estimates,

$$\sigma_i^2 = \exp(\log \sigma_i^2).$$

In a classical finance context, mean and variance (or volatility) are of direct interest when constructing portfolios. By predicting these quantities, the LSTM provides precisely the inputs needed for an allocation layer that aims to balance expected returns against associated risk factors.

It should be emphasized that within the end-to-end training paradigm adopted here, the LSTM's predicted means and variances need not achieve perfect accuracy when evaluated solely through conventional forecasting metrics (e.g., mean squared error or likelihood-based scores). Rather, their fundamental role is to serve as intermediate parameters that feed directly into the subsequent portfolio optimization procedure. By designing the pipeline so that the Sharpe ratio becomes the loss function to be maximized, the model's parameters, from those governing the LSTM to those within the allocation layer, are jointly calibrated to maximize the ultimate investment objective. This coupling of forecast generation and portfolio construction contrasts sharply with traditional two-stage processes in which each module is optimized in isolation.

Within a conventional, forecast-centric paradigm, one might strive to minimize a predictive error metric without explicit regard for the quality of final portfolio returns. Such a disparity can lead to suboptimal overall performance, as the model may overemphasize aspects of return prediction that do not necessarily translate into improved portfolio-level results. In contrast, aligning the learning process directly with a portfolio performance metric places the focus on the features and temporal signals that genuinely enhance out-of-sample risk-adjusted returns. Consequently, the LSTM learns a representation of the financial time series that is more attuned to tangible trading outcomes, rather than purely numerical or statistical forecasting metrics.

In summary, the LSTM-based forecaster operates as an advanced mechanism for extracting temporal structure within cross-sectional daily returns. The gating architecture intrinsic to LSTMs ensures that historically pertinent information is preserved while ephemeral, potentially misleading signals are attenuated or discarded. Crucially, this mechanism yields data-driven forecasts of expected returns and volatilities that are specifically tailored to the downstream objective of building robust, risk-managed portfolios. By marrying the representational strengths of LSTMs with an end-to-end optimization criterion centered on portfolio performance, the system intrinsically calibrates its learned features and output

parameters to those market patterns most likely to contribute to consistent, risk-adjusted gains, thereby providing a unified framework that seamlessly integrates forecasting and allocation tasks.

4.3 – Allocation Layer

In the subsequent segments of this chapter, we continue our exposition of the end-to-end framework by describing how the portfolio allocation step integrates into the same computational graph as the previously discussed LSTM-based return forecasting module. In our case we are going to define two distinct allocation layers, a Markowitz-style layer that balances expected returns against variances, and an entropy-based layer that encourages weight diversification, and then we're going to introduce the various utility functions, performance metrics, and training procedures that tie the architecture together.

4.3.1 – Markowitz Allocation Layer

Following the LSTM module, one immediate step in many financial applications is to convert predictive distributions of returns (and possibly risks) into actionable portfolio weights. In order to accomplish that within our end-to-end paradigm, we begin with a mean-variance, or so-called Markowitz-inspired, objective function. The rationale for including a Markowitz-style layer emerges from decades of Modern Portfolio Theory, whose central theme is that it is often insufficient to merely maximize returns: the investment process must also include a term that penalizes portfolio volatility or variance.

Mathematically, let us denote the predicted mean of each asset's returns by

$$\mu \in \mathbb{R}^{n_{assets}},$$

and the predicted variance (or the diagonal of a variance–covariance matrix) by

$$\sigma^2 \in \mathbb{R}^{n_{assets}}_{\geq 0},$$

The layer then defines a portfolio weight vector

$$w \in \mathbb{R}^{n_{assets}}_{\geq 0},$$

where each w_i corresponds to the fraction of total capital allocated to asset i. We denote by α the hyperparameter that scales the magnitude of the variance penalty. The allocation layer solves a convex optimization problem of the form:

$$maximize\ (1-\alpha) * (\mu^{\intercal}\, w) - \alpha * \sum_{i=1}^{n_{assets}} \sigma_i^2\, w_i^2,$$

in which both returns and volatility are taken into account adjusted for a weight alpha that is personalized from 0 to 1. Near to 0 we give more weight to the returns and near to 1 we give more importance to volatility so having a more stable portfolio but at the expenses of less expected return. In our case alpha was set to 0.3 heuristically.

This function is subjected to the normalization constraint

$$\sum_{i=1}^{n_{assets}} w_i = 1,$$

non-negativity constraints,

$$w_i \geq 0 \text{ for each asset } i$$

and personalized max_weight constraints to avoid over concentration that were chosen heuristically

$$w_i \leq 0.80.$$

Furthermore, it is important to highlight that our model allows users full discretion in selecting the assets in which they wish to invest. This is achieved by leveraging the previously defined constraint to assign a weight of zero to all undesired assets, effectively excluding them from the feasible investment set.

A key innovation of our approach lies in integrating the portfolio optimization process within the neural network itself, rather than solving it externally in a traditional two-stage framework. Specifically, instead of computing the portfolio weights (w) outside the model, we employ a differentiable convex optimization layer to determine them endogenously. We will see how in the dedicated paragraph. This is implemented using the cvxpylayers library, which enables the construction of differentiable convex optimization layers in PyTorch that we will present in the next paragraphs.

In the code α can be tailored to encourage or discourage certain risk–return trade-offs: a higher α leads to portfolios with lower variance at the expense of potentially reduced expected returns, while a smaller α tends to focus more on maximizing returns, sometimes resulting in more concentrated weights. This provides a flexible mechanism to incorporate modern portfolio-

theoretic intuition into an end-to-end training loop, leveraging the best of classical finance but coupling it tightly with deep neural forecasting.

4.3.2 – Entropy-Based Allocation Layer

While the Markowitz paradigm is celebrated for balancing returns and risks, it often fails to enforce broad diversification unless one adds further constraints or carefully tunes the risk penalty. For instance, mean-variance portfolios sometimes allocate heavily to one or two assets that present the most favorable risk–return forecasts. In contrast, many market practitioners and academic studies recommend a degree of intentional diversification, thus mitigating tail risks, model uncertainty, and abrupt regime shifts.

To address this, the second differentiable allocation layer in our architecture embraces an *entropy-regularized* objective function, which promotes a more uniform spread of portfolio weights. The base formulation resembles a simple expected-return maximization,

$$\mu^{\top} w,$$

but incorporates an entropy term that encourages diversification. If we denote the entropy of the weight vector w by

$$H(w) = -\sum_{i=1}^{n_{assets}} entr(w_i),$$

then an entropy-augmented objective seeks to maximize

$$(1-\beta) * (\mu^{\top} w) + \beta * \sum_{i=1}^{n_{assets}} entr(w_i),$$

where $entr(w_i)$ is the standard entropy expression $-w_i \ln(w_i)$ in the solver's internal notation, and β is a scalar that sets the relative importance of diversification. Here the higher the β (between 0 and 1) the higher the concentration on entropy. Heuristically we defined for our code a $\beta = 0.7$. This objective remains a concave function in w, and under the previous constraints the problem is convex and solvable to global optimality.

Similar to the Markowitz layer, one can impose the usual constraints

$$\sum_{i=1}^{n_{assets}} w_i = 1,$$

$$w_i \geq 0 ,$$

$w_i = 0$ (if asset i is not permitted) and

$w_i \leq 0.35$ to incentivize the distribution of assets

In practice, the main difference between the Markowitz and entropy allocation layers lies in how they shape the resulting portfolio solutions. The Markowitz layer emphasizes a risk–return balance, which can be very sensitive to the forecasted variances σ2\sigma^2σ2. By contrast, the entropy-based approach de-emphasizes any explicit risk penalty and instead offers a strong incentive against over-concentration, thanks to the nature of the entropy function. This often leads to weight vectors that include more assets at smaller proportions, possibly reducing the portfolio's sensitivity to inaccurate forecasts of returns or volatilities.

From an end-to-end training perspective, this layer also feeds into the daily portfolio returns, enabling gradient signals to propagate backward. The difference is simply that the layer's gradient computations reflect the entropy-based objective rather than a variance-based one. This design allows the neural forecaster to adaptively discover which signals in the return history are most valuable for a diversified, robust approach to maximizing the eventual Sharpe ratio.

4.3.3 – Differentiable Convex Optimization with *cvxpylayers.torch*

A crucial ingredient that makes both the Markowitz and entropy-based allocation layers truly differentiable is the cvxpylayers.torch library. Traditional convex optimization toolkits, while capable of solving Markowitz and entropy-regularized problems, generally do not provide mechanisms for the seamless backpropagation of gradients into the parameters that define the optimization problem (e.g., the forecasted means and variances).

By contrast cvxpylayers combines the expressive power of the CVXPY domain-specific language for convex optimization with the computational graph capabilities of deep learning frameworks. Specifically, once we specify a convex optimization problem, such as the Markowitz or entropy objective with relevant constraints, cvxpylayers analyzes the problem structure and compiles it into an operator that can be placed directly into a PyTorch (or other supported) computational graph.

When we perform forward passes the predicted parameters μ and σ^2 become inputs to this operator that solves the convex problem, whether it be a mean-variance maximization or an entropy-augmented objective, and returns the optimal portfolio weights .

Then, for backward passes cvxpylayers applies the implicit function theorem to compute the partial derivatives of the optimal solution $w*$ with respect to each input parameter (i.e., μ and σ^2). These derivatives are propagated back through the remainder of the neural network, allowing the LSTM's internal parameters to update in ways that directly improve the final training objective (e.g., the Sharpe ratio).

Hence, instead of a static two-stage pipeline, we obtain a dynamically trainable pipeline whose every step, namely forecasting through LSTM, portfolio weighting, and performance evaluation, coexists in the same auto-differentiation framework. This arrangement not only simplifies development but also leads to more coherent optimization: if the neural net learns that small errors in forecasting σ^2 have a large detrimental impact on final portfolio outcomes (e.g., by over-allocating to volatile assets), it will re-tune its parameters accordingly. Conversely, if certain aspects of μ predictions prove especially useful for the optimization layer, the backpropagated gradients will reinforce those aspects automatically.

This synergy between modern deep learning architectures and classical convex optimization stands at the core of our end-to-end approach, bridging the gap between advanced statistical forecasting and the practical requirements of portfolio construction.

4.4 - Utility Functions and Performance Metrics

After either the Markowitz or the entropy allocation layer has provided an optimal weight vector www, we can measure the portfolio's subsequent performance over a particular horizon of daily returns. To that end, our framework includes a collection of functions for evaluating how effectively a set of portfolio weights would have performed on historical data. These functions operate on mini-batches of data, computing essential quantities such as portfolio returns, means, variances, or advanced measures like maximum drawdown.

A fundamental step, for instance, is to compute the daily portfolio returns once we have selected weights. If $r_{i,t}$ denotes the return of asset i on day t, and w_i denotes the portfolio weight of asset i, the daily portfolio return on day t is:

$$r_t^{portfolio} = \sum_{i=1}^{n_{assets}} w_i r_{i,t}$$

Aggregating these daily returns over a specified horizon permits us to calculate standard performance metrics such as the annualized return, annualized volatility, maximum drawdown, and the Sharpe ratio which is a measure of risk-adjusted return computed as:

$$Sharpe = \frac{\mathbb{E}[r_t^{portfolio}]}{\sigma(r_t^{portfolio})}$$

Where $\mathbb{E}[r_t^{portfolio}]$ is the expected return from the portfolio and denominator is the standard deviation of the returns in the portfolio. The rf member inside the formula was put to 0 because there is no risk free return when investing into stocks.

Maximizing this ratio can be interpreted as a preference for consistently positive returns with minimal fluctuations.

We will talk more about metrics in the dedicated chapter but we wanted to tackle the creation of the sharpe ratio to understand better the loss function explained in the next paragraph.

4.5 – Negative Sharpe Ratio and End-to-End Training

To seamlessly train the neural network and the allocation layer toward maximizing the Sharpe ratio, we introduce a differentiable loss function that corresponds to the negative of this metric. Most deep learning frameworks rely on gradient-based optimizers that proceed by *minimizing* an objective, so by negating the Sharpe ratio, we effectively implement "maximize Sharpe ratio" within the training pipeline:

$$\ell_{Sharpe} = -\frac{\overline{r^{portfolio}}}{\sigma(r^{portfolio}) + \varepsilon}$$

Where $\overline{r^{portfolio}}$ is the empirical mean of the daily returns within the mini-batch, $\sigma(r^{portfolio})$ is the square root of the variance, the standard deviation and ε is a small constant to avoid division by zero.

Putting all together, during each training iteration a batch of input sequences is passed through the LSTM, yielding predicted means μ^ and log-variances log(σ^2) for each asset. These predictions serve as inputs to the selected differentiable allocation layer (Markowitz or entropy), which then optimizes the portfolio allocation. The solver subsequently determines

the weight vector w that maximizes the corresponding objective function for each batch of predictions. With those weights w, the daily portfolio returns are computed over the relevant horizon. We then calculate the negative Sharpe ratio as our scalar loss. Gradients of this scalar loss are backpropagated through the allocation layer and the LSTM, thereby updating all parameters (including the LSTM gating parameters, fully connected layers, and any trainable coefficients in the allocation layer) in a way that directly aims to increase the Sharpe ratio in future iterations.

Such training proceeds for multiple epochs over the training set, where each epoch comprises multiple mini-batches of data and adjusting the batch size, learning rate, and number of epochs we ensure that the training converges to a suitable solution that generalizes well out-of-sample.

At last, the main function of the code which is responsible for the generation of the outputs given to the user will return a set of weights, which will be the last one in the series of output from the model, namely the one near the date the user is using the tool, various performance metrics which will see in the next chapter talking about evaluation of the model and finally the name of the allocation layer that gave the best results.

All these information will be fed to a generative AI (Gemini) which will make for the user a user-friendly explanation of the output explained in natural language. We will see this in the dedicated chapter.

4.6 - Concluding Remarks on the Allocation Layers

Through this unified structure, we manage to bring both classical finance concepts and modern machine learning techniques under one roof. The neural network focuses on discerning any exploitable patterns in daily returns and calibrates its forecasts to suit the precise needs of an allocation module that is itself learned in tandem. If, for example, the Markowitz penalty α reveals that certain forecast inaccuracies lead to large swings in allocated weights, the LSTM has an immediate incentive to refine its estimates of variance or expected returns in a way that mitigates this issue. Likewise, if the entropy-based module is employed, the LSTM is guided toward producing forecasts that allow for stable yet diversified allocations, thereby balancing the search for profitable assets with an inherent desire to avoid undue concentration.

Meanwhile, the *cvxpylayers.torch* integration ensures that every step in this pipeline, ranging from raw market data input to final performance evaluation, is differentiable. Rather than the LSTM being judged by how well it predicts returns in isolation, it is judged by how much value

it adds when those predictions feed into an actual portfolio. By eliminating the historical disconnect between “forecasting accuracy” in one stage and “optimal allocation” in another, practitioners can achieve an improved synergy that can reduce the risk of overfitting or forecast misalignment. The direct gradient flow from the final portfolio returns back through the solver into the forecaster ensures that each forecast is evaluated not just by how well it predicts future returns in an abstract sense, but by how effectively it enhances the final risk-adjusted returns of the portfolio.

CHAPTER 5

EVALUATION AND EXPERIMENTAL RESULTS

Having laid out the end-to-end neural architecture, encompassing both the LSTM forecaster and the differentiable allocation layer(s), we now turn to the crucial task of evaluating the model's out-of-sample performance. This chapter addresses how portfolio returns are calculated, which performance metrics are used to gauge effectiveness, and how these metrics allow us to compare our approach against a classic Markowitz benchmark. By the end, we will see how end-to-end learning, anchored in Sharpe ratio maximization, translates into tangible differences in portfolio outcomes.

5.1 - Portfolio Evaluation Function

A central element of our portfolio backtest is the evaluation function, which maps a given set of portfolio weights to the resulting time series of daily returns, and ultimately to higher-level metrics such as annual returns, volatility, Sharpe ratio, and maximum drawdown. In our framework, this evaluation function is invoked repeatedly for each mini-batch during training (to compute the negative Sharpe loss used as loss function by the LSTM model) and on the entire test set (to produce final performance statistics).

The test set, covering the final 252 trading days in our dataset, serves as an independent validation window that enables an objective assessment of the learned portfolio allocation strategy. We didn't approach the problem with a normal 80%-20% train-test split since we decided to evaluate our results on an entire trading year (252 days remaining excluding weekends and public holidays).

This specific period is reserved to measure the out-of-sample performance of the investment strategy. Unlike the training phase, where weight updates occur iteratively based on the loss function, evaluation on the test set operates in a strictly forward manner: given past price movements, the model predicts asset-level return distributions and derives optimal portfolio weights, which are then applied to compute realized portfolio returns over the horizon. It's important to emphasize that we are not using the final year's data for training, thus avoiding look-ahead bias.

To ensure consistency in performance evaluation, portfolio returns are computed using the weight allocations derived from the test set inference. The process follows a structured pipeline in which the trained LSTM-based forecaster that we presented in chapter 4 receives a rolling

window of past observed return (the 90 days of lookback), it processes this sequence and outputs a forecast, namely the expected mean return vector and log-variance vector of asset returns for the next period of 30 days (horizon), the log-variance is exponentiated to obtain the variance. The predicted mean and variance are passed to the allocation layer that solves the optimization problem under the constraints we imposed and outputs the resulting optimal portfolio weights for the first horizon of the test set that are stored for application.

Once the first set of portfolio weights is obtained, it computed the first 30 days' returns and the input is rolled forward, the model must iterate through the entire test set using a rolling window approach.

The oldest chunk of 30 days in the input window is dropped and the newest observed returns (from the first horizon of the test set) is added. The entire sequence is again fed to the LSTM model and the process repeats.

To improve stability and reduce transaction costs, instead of updating weights on a rolling daily basis, the optimization is performed at fixed 30-day intervals, ensuring that once weights are set for a given period, they remain unchanged until the next rebalance date. This approach gives back slightly worse results than a normal daily redistribution of budget but maintains adaptability while preventing excessive trading, aligning with practical investment constraints.

While the forecasted weights are applied to the actual asset returns in the test set we are generating a time series of realized daily portfolio returns that we will use to compute key financial risk and reward metrics.

5.2 - Performance Metrics Computation

We decided to evaluate our portfolio selection through 5 main metrics that are extremely useful in finance: the Equity Curve, the Annual Return, the Annualized Volatility, the Sharpe Ratio (also used for training) and the maximum Drawdown

Starting from the first one, an intuitive way to visualize the trajectory of the portfolio value through time is the equity curve. This cumulative representation tracks how an initial investment would evolve under compounding returns. Mathematically, if $r_{portfolio,t}$ is the daily return at day t, then the equity curve up to day T is computed by:

$$E_t = \prod_{i=1}^{t} (1 + r_{portfolio,i}), \qquad t = 1, \ldots, T$$

Here, E_t can be interpreted as the growth factor: a value greater than 1 indicates net growth, while a value below 1 signals net decline.

It is important also to mention that this equity curve will be our only graphic choice given by the model as output for the user.

For the second metric, since the backtest period is exactly 252 trading days (approximately one trading year), the total return realized at the end of day T closely approximates the annual return that is the metric of interest. By definition:

$$Annual\ Return = E_t - 1$$

where E_t is the final value of the equity curve after T days. This measure answers a fundamental question: How much would the portfolio have gained or lost, in percentage terms, over the course of one year?

While return quantifies the upside potential, volatility captures the risk or uncertainty. In financial contexts, volatility is often expressed in annualized form. Concretely, we calculate the standard deviation of daily returns and scale it to a yearly figure by multiplying by $\sqrt{252}$. Hence:

$$\sigma_{annual} = \sigma_{daily} \times \sqrt{252}$$

where σ_{daily} is the standard deviation of the daily return series. Volatility serves as a key gauge of the variability of returns: higher volatility implies more pronounced swings in daily performance.

Going forward**,** a direct way to quantify risk-adjusted performance is the Sharpe Ratio, which considers both the reward (annual return) and the risk (annualized volatility). Assuming a zero risk-free rate for simplicity, the Sharpe Ratio is given by:

$$Sharpe = \frac{Annual\ Return}{Annual\ Volatility}$$

A higher Sharpe Ratio indicates that each unit of risk taken by the strategy generates a higher level of return, thus signaling more efficient use of risk capital.

Finally we chose the Maximum Drawdown since portfolio management also requires understanding worst-case declines. The MDD measures the largest peak-to-trough drop in the equity curve throughout the evaluation window. Defining the equity curve at day t as E_t and the running maximum of the curve up to t as $max_{s \leq t}\ E_s$, the daily drawdown at t is:

$$D_t = \frac{E_t - max_{s \leq t}\ E_s}{max_{s \leq t}\ E_s}$$

The Maximum Drawdown is then:

$$MDD = \min_t D_t$$

capturing the steepest percentage decline from a historical peak to a subsequent trough. This statistic highlights the depth of potential losses and is crucial for evaluating downside risk.

5.3 – Comparison of our model performance with a standard Markowitz

To rigorously evaluate the effectiveness of our end-to-end portfolio optimization model, we conducted a comprehensive comparative analysis against the traditional Markowitz mean-variance framework. This evaluation was performed by selecting a universe of 30 widely recognized financial assets and systematically generating randomized four-asset portfolios in each iteration. These asset combinations were then fed into both models, which computed the same set of performance metrics, including annual return, annual volatility, Sharpe ratio, and maximum drawdown. The results of 20 iterations and 20 different combinations of assets were systematically recorded and organized into a comparative table (Fig. 1), facilitating a clear and objective assessment of the models' relative performance. This methodology ensures that our analysis captures a broad spectrum of market conditions and asset interactions, thereby providing robust insights into the advantages of our approach over conventional optimization techniques.

The empirical results demonstrate a clear advantage of our approach, which consistently achieves superior risk-adjusted returns, lower volatility, and improved drawdown control. While classical Markowitz optimization (with mean and variance calculated naively using the

mean of the past 90 days) is widely recognized for its theoretical foundation, it suffers from critical limitations, including sensitivity to estimation errors in the covariance matrix, excessive concentration in a small number of assets, and an inability to dynamically adapt to changing market conditions. Our model overcomes these deficiencies by leveraging a more robust asset allocation strategy that enhances diversification while preserving return potential.

A comparative analysis across different test cases highlights these advantages. For instance, in step 0, the classical Markowitz approach yields a Sharpe ratio of 1.637, whereas our model achieves 2.0278, signifying a substantial improvement in risk-adjusted performance. Furthermore, our model effectively mitigates concentration risk by distributing capital more efficiently across multiple assets. Unlike the classical approach, which allocates 90.31% of capital to AMZN and only 9.69% to GOOGL, our model balances allocations among AMZN (35%), GOOGL (35%), and the S&P 500 index (^GSPC, 29.24%), thereby reducing reliance on a single asset and increasing portfolio resilience. This improved allocation strategy also contributes to a lower maximum drawdown (-0.1501 vs. -0.1917 for the classical model), underscoring the risk management capabilities of our approach.

Behind the greater stability brought by our model by the form of an higher Sharpe ratio, our model significantly reduces portfolio volatility, in fact in the same example we can see how the annual volatility is 0.26 in the classical Markowitz approach and 0.19 in our end to end approach.

Perhaps the most significant shortcoming of the classical Markowitz approach, as revealed by our experiments, is its frequent tendency to over-allocate capital to a single asset. In steps 2, 4, 9, 11, and 19, the classical optimizer assigns 100% of capital to a single stock, an allocation that, while mathematically optimal under historical conditions, introduces substantial concentration risk. Our model avoids such pitfalls by maintaining a more balanced portfolio structure, ensuring that risk is distributed across multiple assets rather than being concentrated in one or two securities.

In conclusion, The empirical evidence strongly supports the superiority of our end-to-end AI-driven portfolio optimization model over classical mean-variance optimization. Unlike traditional Markowitz approaches that rely solely on historical return data and static covariance estimates, our model incorporates advanced AI techniques to predict future returns and dynamically adjust asset allocations based on evolving market conditions. This predictive capability allows our model to optimize portfolio construction in a more forward-looking

manner, mitigating the risks associated with estimation errors and market instability that often undermine classical optimization strategies.

By leveraging AI-driven forecasting techniques, our model delivers higher risk-adjusted returns while maintaining lower volatility and enhanced drawdown management. The integration of intelligent allocation mechanisms ensures that portfolios remain well-diversified and resilient, avoiding the pitfalls of concentration risk that frequently arise in static optimization frameworks. Furthermore, the adaptability of our approach enables it to respond proactively to shifts in market dynamics, improving its ability to capture opportunities and hedge against downside risks more effectively than conventional models.

5.4 – Comparison of our model performance with a two steps model

To further test our end-to-end model performance we decided to create another model similar in structure with the only difference being the fact that the second will have a two-separate-stage training in which the forecaster (LSTM) will have as loss function a simple MSE (mean-square-error) and the output of the model (mean and variance) will be fed to the allocator that statistically distributes weights.

The empirical results showed in the table in Fig.2 from our comparative analysis between our end-to-end portfolio optimization model and the two-stage forecast model demonstrate a clear and consistent advantage of our approach across multiple performance metrics performing better than the two steps model 26 times out of 35 different combinations of tickers (74.28%).

The two-stage model, which combines an LSTM-based forecaster trained with mean squared error (MSE) and a classical Markowitz optimization layer, exhibits several limitations that our end-to-end model successfully addresses. Specifically, the two-stage approach relies on a decoupled process where the forecasting and optimization steps are performed independently, leading to suboptimal asset allocations due to the lack of feedback between the two stages. This often results in higher volatility, lower risk-adjusted returns, and less effective drawdown management compared to our integrated model.

Our end-to-end model, which leverages a unified framework combining AI-driven forecasting and dynamic allocation, consistently outperforms the two-stage model in terms of Sharpe ratio, annual return, and maximum drawdown. For example, as we can see in step 1 of Fig.2, the two-stage model achieves a Sharpe ratio of 1.3703, while our end-to-end model attains a higher

Sharpe ratio of 3.5594 denoting a marked superiority of our model highlighted by the superior risk-adjusted performance of our approach. Furthermore, our model demonstrates better diversification and capital allocation, avoiding the over-concentration in single assets that often plagues the two-stage model.

As we said in our work, the two steps model perform worse due to the fact that the forecasting and optimization steps are performed independently, so the forecaster predicts future returns and variances without considering how these predictions will be used in the optimization process. This decoupling can lead to a mismatch between the forecasted parameters and the actual portfolio construction process. For example, the forecaster might predict high returns for a particular asset, but the optimizer might allocate too much capital to that asset, leading to over-concentration and increased risk. This lack of feedback between the forecasting and allocation layer can result in suboptimal asset allocations since the first isn't informed of the objective that needs to be reached. Instead our end-to-end model trains the forecaster with the sharpe ratio as loss function, resulting in more resilient portfolios.

In conclusion, the empirical evidence underscores the superiority of our end-to-end portfolio optimization model also over the two-stage forecast model. By integrating AI-driven forecasting with dynamic allocation mechanisms, our model not only enhances risk-adjusted returns but also improves diversification, reduces volatility, and mitigates drawdowns. These findings highlight the transformative potential of end-to-end AI-driven approaches in modern portfolio management, offering a robust alternative to traditional decoupled optimization frameworks. As financial markets continue to evolve, the ability to leverage advanced AI techniques for forward-looking portfolio construction will remain a critical advantage for achieving long-term investment success.

CHAPTER 6

WEB-APPLICATION DEVELOPMENT

In addition to designing and testing the underlying machine learning models, a key objective of this work is to deliver a simple and intuitive way for users, whether novice investors or experienced fund managers, to interact with these sophisticated portfolio-optimization capabilities. As discussed throughout earlier chapters, even the most advanced AI-driven models lose much of their power if end-users find them difficult to interpret or cumbersome to operate. This chapter outlines how our web-based platform (fig.3) was built with the dual goals of ensuring technical robustness and offering a clean, user-friendly interface. We describe how the front-end is structured, highlight the importance of intuitive page layouts and interactive elements, and illustrate how user input seamlessly triggers the back-end logic to produce meaningful outputs.

6.1 - Rationale for a Simple, Intuitive Interface

Given the complexity of AI-based portfolio optimization, it is easy for developers to overload an interface with too many controls, technical terms, or data visualizations. Such approaches risk alienating users who have limited financial knowledge or who are new to machine learning. Our focus instead is on simplicity: by limiting the number of mandatory input fields and presenting outputs in clear, everyday language, thanks to Large Language Model (LLM), that we will describe in the next chapter, the platform lowers the barriers to entry. This aligns with our overarching mission to make advanced investment analytics accessible to all.

In designing the front-end, we followed several principles common to effective user experience (UX) design:

Using a minimalist design, we present only the most critical input fields and a concise display of model outputs. Unnecessary data or functionality can clutter the page and confuse the user.

Through a clear Call-to-Action the user can easily identify how to interact with the system, for example, by typing a question or request in a chat-like interface following the instructions on the screen. This lowers the learning curve for navigating the tool.

Furthermore, while we aim to maintain a minimalist design, a concise chart (in our case an equity curve) can be more impactful than long tables of numerical data to support the results of our model.

Thus, we use carefully chosen plots that highlight key performance metrics or historical performance.
For users with limited financial or technical expertise, text-based explanations generated by Gemini are provided in natural language to help them understand why a model suggests specific allocations, or how the portfolio itself went on the test set and the generative AI will also give a comment on the main metrics and suggest, in case the portfolio didn't perform well, to try again with other tickers.

Finally for easy reset of the model to let the user write a new prompt and have a new output there will be a simple reset button that will delete the trained model and the train answer and give again the word to the user.

6.2 – Technology Stack: Streamlit

To achieve the above UX goals, we employed the Streamlit framework for Python, a tool that excels in building interactive web applications for data science and machine learning tasks. Its declarative style allows developers to concentrate on the logic and results rather than the details of front-end implementation. By automatically handling the rendering of widgets such as text inputs, buttons, and charts, Streamlit greatly accelerates prototyping. It also offers streamlined state management through st.session_state, which keeps track of user inputs and model outputs across multiple interactions, avoiding the need for complex backend solutions. Additionally, Streamlit integrates smoothly with libraries like Altair, Plotly, and Matplotlib, enabling easy embedding of charts and enhancing the overall data visualization experience.

6.3 – Conversational Interaction

One of the most novel features of this platform is its chat-driven interface, which addresses a critical gap in many fintech tools: interpretable and natural language feedback. Instead of forcing the user to navigate through complex forms, we use a text box at the bottom of the page where the user can submit questions, for example:

"How should I allocate my portfolio among these five stocks?"

Internally, the application interprets these requests, triggers the relevant model logic, and returns answers in a conversational style. While some advanced generative AI components are used in the back-end, from the user's perspective, it simply feels like chatting with an intelligent assistant with the only difference that the user will wait for 1-2 minutes to wait for the model to train.

The answer, as said before will be in natural language, following the language of the user that wrote the prompt and will report the suggested weigths, the metrics of interest and a graph of the equity curve as shown in Fig.4.

In the top-right we can find a "↻" button that let the user reset the entire application in a way to pose a new question with different tickers of interest.

6.4 – Concluding Remarks on Web-Application Development

This chapter has shown how an emphasis on user-centric design principles and a streamlined code structure can transform a technically complex system, encompassing LSTM forecasting, convex optimization, and sophisticated portfolio metrics, into a platform easily navigable by diverse users. The decision to incorporate conversational interaction via a chat-like interface exemplifies a broader shift in fintech applications, moving away from dense dashboards and specialized interfaces toward natural language–driven tools that build trust and expand accessibility.

Overall, the result is a robust, flexible application that bridges advanced AI-driven portfolio optimization with clear, human-readable guidance. By keeping the user experience in the spotlight, we ensure that our solution addresses its central mission: enabling intelligent, data-driven investment decisions without sacrificing clarity or simplicity.

CHAPTER 7

LLM INTEGRATION FOR INPUT AND OUTPUT

In the previous chapter, we presented the web application's front-end design and illustrated how a user-friendly interface can significantly improve the overall investment experience. We also mentioned that, behind this intuitive interface, the portfolio optimization model from Chapter 4 handles the core analytical tasks. An important question, however, arises: how does the model actually receive the specific inputs, namely the stock tickers in which a user wants to invest, when these inputs are stated in casual, everyday language?

This is precisely where a Large Language Model (LLM)—in our case, Gemini—enters the picture. Instead of burdening the user with rigid data-entry forms or specialized financial jargon, the application allows people to interact through natural language prompts. A user might type, for instance, "I'd like to invest in Apple and Google" or "Allocate my funds between Tesla, Netflix, and Microsoft." The LLM takes these free-form statements, parses them, and translates them into structured instructions, which the underlying computational framework (the LSTM model and allocation layers from Chapter 4) can then act upon.

By doing so, Gemini functions as an intelligent intermediary. It processes queries in input for the correct tickers and generates a refined command that the back-end system can interpret seamlessly and generates comments in output so that the user will be able to understand the quantitative results of the model. This approach not only increases accessibility for users with limited financial or technical backgrounds, but also significantly reduces friction in data entry. Rather than juggling ticker symbols and learning how to operate a specialized interface, users can simply converse in natural language, leaving the behind-the-scenes translation to Gemini.

In this chapter, we explore Gemini's role in greater detail, discussing how its conversational capabilities integrate with the front-end interface to enable a fluid, chat-like exchange. We delve into how the generative AI model identifies the relevant assets from a prompt and how these extracted elements are channeled into the algorithms described in earlier chapters.

7.1 – Introduction to Large Language Models (LLMs)

In recent years, Large Language Models (LLMs) have emerged as a central technology for understanding and generating human-like text. Modern LLMs, often based on Transformer architectures, harness attention mechanisms that enable them to relate every token in a sequence to all other tokens. This design allows for more efficient parallelization in training

and a remarkable capacity to capture contextual nuances over extended input spans. Through exposure to vast corpora of text, these models learn to handle diverse topics, languages, and even specialized domains, ranging from everyday conversation to fields like law or finance. Their few-shot and zero-shot reasoning abilities mean they can often respond coherently to new or unexpected queries as long as the prompt is suitably structured.

Such adaptability is particularly valuable in finance, where investors might phrase their objectives in many ways—some direct, others vague or partially defined. A strong LLM can process seemingly casual instructions (for instance, "I want to put my money into Apple and maybe a broad market ETF") and extract the relevant financial elements (AAPL, SPY, or ^GSPC). In turn, it can facilitate a conversational format that allows users to refine and iterate on their requests. By bridging specialized computational routines with plain language, the LLM helps overcome barriers to entry, ensuring that sophisticated financial analytics can be delivered to both new and experienced market participants in an accessible manner.

7.2 - Gemini

Among the various available LLMs, we adopt Google's Gemini for its strong balance between linguistic fluency and developer-oriented functionality and also for its 300$ of free credit to start learning and experimenting with its functions. Gemini is grounded in a large-scale Transformer infrastructure and benefits from comprehensive pretraining, enabling it to parse both colloquial expressions and technical financial terms. It also incorporates mechanisms for function calling, where the model can return structured data aligned with specific developer-defined schemas. These features eliminate much of the ambiguity typically associated with natural language processing, allowing the system to interpret user text as a clear set of instructions for subsequent forecasting or allocation routines. Gemini further provides built-in safety filters to maintain focus on relevant content and support multilingual interactions, broadening its scope to users who may prefer communicating in languages other than English.

7.3 - The Application of LLM on Our Web App

The primary task of the LLM in our portfolio optimization tool is to link plain-language queries with the advanced analytics described in earlier chapters. Rather than forcing users to navigate a list of tickers or fill out complex forms, the web application presents a simple chat interface where they can type statements like, "I'd like to invest in Tesla and Microsoft." Under the hood, Gemini interprets this free-form text and calls a function defined by the developer. This

function is configured to receive the user's prompt and output a structured object containing the extracted tickers. The model accomplishes this through its function-calling interface, which ensures that whenever it detects references to companies, it packages them in a JSON-like format rather than leaving them embedded in a longer textual response.

Once this function returns, the web application forwards the resulting ticker made into a list to our back-end model. At this stage, the system applies the AI-driven forecast (our end-to-end model) and then conducts the convex optimization step. The solver arrives at a set of weights that reflect the objective of the optimization problem we saw previously, together with the additional constraints we described before. These results, which include numeric allocations and performance indicators, are then handed back to Gemini in a second prompt for explanation. This second prompt instructs the model to generate a narrative summarizing the final weight distribution, clarifying its rationale, and commenting on how risk metrics (like Sharpe ratio, annualized volatility, or drawdowns) might inform the user's decision.

By leveraging function calls in both directions, first for extracting meaningful inputs and then for generating explanatory outputs, the system encloses its core logic within a dialogue cycle. On the input side, Gemini's function calling helps standardize user requests. Even if a user writes slightly ambiguous statements or if the user doesn't know the code of the tickers he wants to invest in, the model can often interpret them correctly and respond with a neat list of recognized symbols. On the output side, Gemini's capability to produce structured narrative ensures that performance metrics and allocation strategies are distilled into a reader-friendly commentary that can highlight key points, identify potential risks, and recommend further diversifications.

Behind the scenes, each function call defines its expected input and output schemas. During the first call, the model knows it must produce the ticker symbols in an array. During the second call, it transforms allocation weights and risk metrics into a coherent explanatory paragraph or two. Because these steps are executed under well-defined schemas, the system avoids the usual pitfalls of ad hoc text parsing. Developers can thus focus on the financial logic and user experience, rather than writing extensive rules for extracting data from plain text. Function calling effectively gives the model a blueprint for what is considered "valid" output in each stage of the conversation, reducing the chances of misunderstanding or hallucination in sensitive domains like finance.

7.4 – Concluding remarks

Integrating an LLM into our application substantially lowers the user's cognitive load when specifying assets or interpreting model outputs. Rather than searching for the right ticker or decoding cryptic allocation tables, users receive a guided, conversation-like experience. The use of Gemini, with its robust function-calling interface, provides a structured way to handle both the extraction of tickers from user text and the generation of narrative explanations around portfolio results. This yields a workflow that is not only more accessible but also more transparent, since every recommended allocation is accompanied by a natural language commentary that clarifies its purpose and associated risks.

Nevertheless, a few limitations should be acknowledged. LLMs can occasionally produce off-topic or factually incorrect statements, so safeguarding features like function schemas, safety filters, and post-processing checks remain important for maintaining consistent quality. Model inference also comes with higher computational and economic costs than purely rule-based systems, so balancing real-time performance with prompt complexity is a crucial design consideration. Despite these caveats, the benefits of a language-driven interface—improved usability, greater interpretability, and more interactive investment dialogues—make a compelling case for further refining LLM-based approaches in portfolio management tools.

CHAPTER 8

DISCUSSIONS AND CONCLUSIONS

This thesis introduced a comprehensive framework for AI-enhanced portfolio management, with a particular focus on melding predictive modeling, optimization, and user accessibility. At the core of this framework is a Long Short-Term Memory (LSTM) network designed to generate forecasts of asset returns. These forecasts are then passed to a differentiable convex optimization layer that determines how capital should be distributed among the selected assets. The entire process is packaged in a web-based application, which draws on a Large Language Model (LLM) interface to make advanced financial tools more approachable to a broad range of users. By embedding portfolio construction into an end-to-end training process and relying on a conversational interface, the methodology aims to tap into deep learning's predictive capabilities while demystifying complex financial analytics for both beginner and experienced investors.

One of the primary achievements of this work lies in demonstrating how traditional, sequential approaches to forecasting and portfolio allocation can be improved when folded into a single computational pipeline. Typically, machine learning models generate return predictions in isolation, and a separate optimization procedure later decides the portfolio weights. In contrast, the present thesis integrates these steps so that the LSTM's training objective is directly tied to a portfolio performance metric. Rather than simply minimizing a prediction error, the LSTM learns patterns in the data that ultimately enhance the final risk–return profile of the portfolio. In empirical tests, this unified approach not only delivered higher risk-adjusted returns compared to a classic mean-variance optimizer but also lowered overall volatility and reduced the problem of concentrating too heavily in a few assets. Still, in any practical investing context—where liquidity constraints, transaction costs, and regulatory issues loom large, it would be prudent to refine the framework further to handle these real-world complexities.

The thesis also makes a case for broadening the data fed into the LSTM beyond mere returns and volatility. Although daily price fluctuations and risk indicators capture much of the market's short-term movement, relevant information often resides outside of these conventional data streams. News reports, social media sentiment, macroeconomic indicators, and company announcements can all shape investor expectations. Because LSTMs excel at tracking temporal patterns and contextual signals, a natural direction for future research involves merging these diverse inputs into a single, richer dataset. Such an expansion might,

for example, allow the model to detect early warning signs of regime shifts, periods of excessive optimism, or impending volatility not yet fully mirrored in asset prices. However, effectively merging these parallel data streams would necessitate careful data alignment and more sophisticated preprocessing. Despite the potential hurdles, the benefits of a more holistic forecast, particularly in volatile or information-driven markets, could be profound.

An additional contribution of this work is demonstrating how cutting-edge financial analytics can be delivered via a web application that uses natural language as the primary mode of interaction. Through the integration of Google's Gemini LLM, the system allows users to express their allocation preferences in familiar, everyday language: for instance, "Allocate my budget among Tesla, Microsoft, and a broad market index." In response, the application identifies the requested assets, processes them through the trained model, and returns a clear explanation of how the funds are apportioned and why. This focus on accessible language helps demystify quantitative strategies for individuals who may lack a formal background in finance or data science. It also reduces the friction often experienced when interacting with specialized software or complex user interfaces.

Notwithstanding these successes, the research brought to light a number of considerations that will likely influence future iterations. First, a significant computational bottleneck arose during training when using solely CPU resources, especially with deeper lookback windows or an increasing number of assets. Switching to GPUs or even multi-GPU clusters would speed up both training and inference, making it more feasible to explore deeper models or larger cross-sections of the market. Second, although the Gemini-based chat interface is well suited to parsing simple sentences, a more advanced conversation flow could prove valuable for users with intricate requirements, such as limiting exposure to specific market sectors or imposing constraints on portfolio turnover. By defining more comprehensive function calls within the LLM integration, the system could respond to a wider array of requests and produce increasingly sophisticated, tailor-made outputs.

Another area for refinement is the rebalancing schedule. In this study, a monthly rebalancing frequency was chosen to balance the goal of maintaining current market alignment against the desire to minimize excessive transaction costs. Yet certain market conditions might call for triggers based on factors like volatility, risk thresholds, or abrupt shifts in predicted returns. Implementing such adaptive mechanisms would improve the resilience of the strategy, albeit at the cost of a more elaborate workflow for monitoring and re-optimizing the portfolio.

Furthermore an important issue pertains to the transparency of deep learning systems. Although the LLM-generated explanations provide a user-friendly summary of the portfolio's logic and performance, they do not fully reveal the underlying rationale of the LSTM itself. In highly regulated environments or when user trust is paramount, it is vital to complement these narrative summaries with additional diagnostics. Feature attribution and model interpretability techniques, such as gradient-based saliency maps, attention heatmaps, or local interpretation methods, could help stakeholders understand which factors most strongly influence an allocation decision. Such transparency could also foster greater trust in automated strategies and satisfy any regulatory demands for explainability in algorithmic trading.

In conclusion, this thesis shows that advanced machine learning approaches can be effectively unified with user-centric design to produce an AI-driven portfolio optimization system that is both powerful and accessible. By training the LSTM forecaster and the optimization module jointly, the model evolves with a focus on genuinely meaningful signals, while the conversational interface offers broad accessibility to users of varying expertise. Looking ahead, scaling up the computational resources, integrating additional risk measures, and refining the LLM-driven conversation to accommodate intricate financial contexts all represent important next steps. If implemented, these enhancements could render the tool even more valuable, enabling it to adapt to a rapidly evolving marketplace while remaining both transparent and user-friendly. Through such developments, AI-driven portfolio optimizers have a strong chance of taking on a prominent role in navigating the complexities of modern finance, thus serving a diverse range of investors with minimal technical barriers.

CHAPTER 9

APPENDIX

Fig.1

Step	Tickers	Classical MZ (Note)	MZ AR	MZ Vol	MZ Sharpe	MZ MDD	MZ Weights	Best (Entropy/Markowitz)	E2E AR	E2E Vol	E2E Sharpe	E2E MDD	E2E Weights	Best model
0	^GSPC, IWM, GOOGL, AMZN	Classical MZ (My model)	0,4401	0,2688	1,637	-0,1917	AMZN=0,9031, GOOGL=0,0969	Best (Entropy)	0,4019	0,1982	2,0278	-0,1501	AMZN=0,3500, GOOGL=0,3500, ^GSPC=0,2924	E2E
1	^VIX, IWM, ^RUT, QQQ	Classical MZ (My model)	0,583	0,2883	2,0222	-0,1364	QQQ=0,7575, ^VIX=0,2425	Best (Entropy)	0,3595	0,1285	2,7979	-0,0494	QQQ=0,3500, ^RUT=0,2791, IWM=0,2318, ^VIX=0,1390	E2E
2	^IXIC, UNH, BRK-B, ^DJI	Classical MZ (My model)	-0,0241	0,2725	-0,0885	-0,2205	UNH=1,0000	Best (Entropy)	0,2226	0,1332	1,6707	-0,067	UNH=0,3500, ^IXIC=0,3500, ^DJI=0,1761, BRK-B=0,1239	E2E
3	^IXIC, MSFT, AMZN, JPM	Classical MZ (My model)	0,2589	0,2144	1,2073	-0,1711	MSFT=0,5785, AMZN=0,4215	Best (Entropy)	0,37	0,1733	2,1347	-0,133	MSFT=0,3500, ^IXIC=0,2850, JPM=0,2625, AMZN=0,1024	E2E
4	PG, MSFT, DIA, SPY	Classical MZ (My model)	0,1293	0,1992	0,6489	-0,1549	MSFT=1,0000	Best (Entropy)	0,2023	0,1135	1,7823	-0,0843	PG=0,3500, SPY=0,2525, MSFT=0,2223, DIA=0,1752	E2E
5	BRK-B, AMZN, JNJ, UNH	Classical MZ (My model)	0,2757	0,1919	1,4365	-0,0933	AMZN=0,6321, UNH=0,3679	Best (Markowitz)	0,3787	0,1677	2,2584	-0,0644	AMZN=0,8000, UNH=0,0979, JNJ=0,0551, BRK-B=0,0470	E2E
6	GC=F, SI=F, ^GSPC, VTI	Classical MZ (My model)	0,2453	0,1168	2,1005	-0,0775	VTI=0,8648, GC=F=0,1352	Best (Entropy)	0,2853	0,1048	2,7213	-0,0674	GC=F=0,3500, ^GSPC=0,3469, VTI=0,2829, SI=F=0,0202	E2E
7	DIA, BRK-B, TSLA, GOOGL	Classical MZ (My model)	0,564	0,3864	1,4595	-0,2211	TSLA=0,5209, GOOGL=0,4791	Best (Entropy)	0,3355	0,1884	1,7811	-0,1373	GOOGL=0,3500, BRK-B=0,3403, DIA=0,1668, TSLA=0,1429	E2E
8	MSFT, VTI, TSLA, ^RUT	Classical MZ (My model)	0,3625	0,3109	1,166	-0,195	MSFT=0,6004, TSLA=0,3996	Best (Markowitz)	0,2922	0,2076	1,4077	-0,1539	MSFT=0,6454, VTI=0,2323, ^RUT=0,0671, TSLA=0,0552	E2E
9	NVDA, ^RUT, META, SI=F	Classical MZ (My model)	1,7125	0,524	3,2682	-0,2705	NVDA=1,0000	Best (Entropy)	0,9482	0,2792	3,3955	-0,1451	NVDA=0,3500, META=0,3353, SI=F=0,1849, ^RUT=0,1298	E2E
10	HD, GOOGL, GC=F, ^DJI	Classical MZ (My model)	0,2924	0,1928	1,5167	-0,1218	GOOGL=0,6154, HD=0,3846	Best (Entropy)	0,2907	0,1288	2,2579	-0,0736	GOOGL=0,3500, GC=F=0,2583, HD=0,2404, ^DJI=0,1513	E2E
11	PG, ^RUT, USDJPY=X, NVDA	Classical MZ (My model)	1,7125	0,524	3,2682	-0,2705	NVDA=1,0000	Best (Entropy)	0,562	0,1665	3,3754	-0,1118	NVDA=0,3500, PG=0,3500, USDJPY=X=0,2747, ^RUT=0,0253	E2E
12	PG, GOOG, CL=F, SPY	Classical MZ (My model)	0,3562	0,2766	1,2874	-0,2228	GOOG=1,0000	Best (Entropy)	0,2376	0,127	1,8705	-0,1086	GOOG=0,2832, CL=F=0,2618, SPY=0,2459, PG=0,2091	E2E
13	UNH, MSFT, DIA, ^IXIC	Classical MZ (My model)	0,1064	0,1605	0,6626	-0,096	MSFT=0,7858, UNH=0,2142	Best (Entropy)	0,2016	0,1297	1,5538	-0,0636	^IXIC=0,3500, MSFT=0,3500, UNH=0,1583, DIA=0,1417	E2E
14	QQQ, AMZN, VTI, GOOG	Classical MZ (My model)	0,439	0,2671	1,6438	-0,1911	AMZN=0,8906, GOOG=0,1094	Best (Entropy)	0,3655	0,174	2,1009	-0,1239	GOOGL=0,3500, VTI=0,3434, QQQ=0,2990	E2E
15	SI=F, AMZN, ^RUT, NVDA	Classical MZ (My model)	1,7125	0,524	3,2682	-0,2705	NVDA=1,0000	Best (Markowitz)	1,649	0,4086	4,0352	-0,249	NVDA=0,8000, AMZN=0,1876, SI=F=0,0123	E2E
16	JNJ, SPY, CL=F, META	Classical MZ (My model)	0,4631	0,226	2,0489	-0,1123	META=0,5173, SPY=0,4792	Best (Markowitz)	0,3474	0,1586	2,1903	-0,1016	SPY=0,3150, JNJ=0,2765, META=0,2400, CL=F=0,1685	E2E
17	META, MSFT, QQQ, GOOGL	Classical MZ (My model)	0,1293	0,1992	0,6489	-0,1549	MSFT=1,0000	Best (Entropy)	0,3509	0,1955	1,7943	-0,1543	MSFT=0,3500, GOOGL=0,3296, QQQ=0,1969, META=0,1234	E2E
18	HD, SPY, AAPL, PG	Classical MZ (My model)	0,2981	0,2134	1,397	-0,1451	AAPL=0,9265, HD=0,0735	Best (Entropy)	0,2841	0,1185	2,3965	-0,0597	HD=0,3500, AAPL=0,3500, SPY=0,2445, PG=0,0555	E2E
19	AAPL, JNJ, NVDA, GOOGL	Classical MZ (Base Markowitz)	1,7125	0,524	3,2682	-0,2705	NVDA=1,0000	Best (Entropy)	0,6706	0,2287	2,9321	-0,1666	JNJ=0,3237, GOOGL=0,3039, AAPL=0,2308, NVDA=0,1416	Classical Markowitz

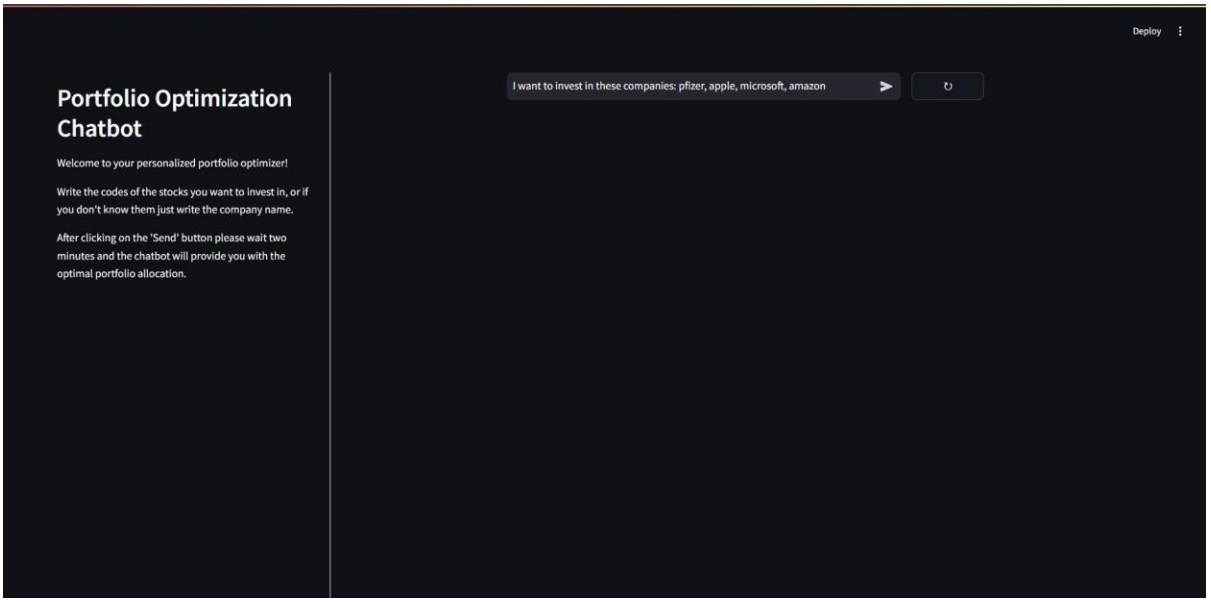
Comparison between our end-to-end model and a classical Markowitz approach

Fig.2

Step	Tickers	Two-step AR	Two-step Vol	Two-step Sharpe	Two-step MDD	Two-step Weights	E2E Best Approac	E2E AR	E2E Vol	E2E Sharp	E2E MDD	E2E Weights	Winner
0	^GSPC, IWM, GOOGL, AMZN	0.3471	0.1770	19.616	-0.1213	AMZN 0.310226, IWM 0.254246, ^GSPC 0.243804, GOOGL 0.191724	entropy	0.3872	0.1964	19.719	-0.1496	GOOGL 0.35, AMZN 0.35, ^GSPC 0.285052, IWM 0.014948	E2E
1	^VIX, IWM, ^RUT, QQQ	0.4097	0.2990	13.703	-0.1789	^VIX 0.296454, QQQ 0.273953, IWM 0.224874, ^RUT 0.204719	markowitz	0.5349	0.1503	35.594	-0.0543	QQQ 0.800003, IWM 0.175379, ^VIX 0.024618	E2E
2	PG, MSFT, DIA, SPY	0.2243	0.1008	22.254	-0.0589	PG 0.276514, SPY 0.269029, MSFT 0.240397, DIA 0.214060	entropy	0.2538	0.1119	22.671	-0.0616	MSFT 0.350003, DIA 0.332020, SPY 0.301641, PG 0.016336	E2E
3	BRK-B, AMZN, JNJ, UNH	0.1886	0.1195	15.784	-0.0572	BRK-B 0.283671, AMZN 0.248970, JNJ 0.237379, UNH 0.229981	markowitz	0.2331	0.1660	14.042	-0.0829	UNH 0.317003, BRK-B 0.286266, JNJ 0.198635, AMZN 0.198095	Two-step
4	GC=F, SI=F, ^GSPC, VTI	0.2987	0.1358	21.999	-0.0859	VTI 0.297197, ^GSPC 0.248659, SI=F 0.239009, GC=F 0.215134	markowitz	0.3241	0.1146	28.275	-0.0701	VTI 0.799998, ^GSPC 0.200008	E2E
5	DIA, BRK-B, TSLA, GOOGL	0.4256	0.2071	20.551	-0.1160	DIA 0.258700, TSLA 0.252788, BRK-B 0.251702, GOOGL 0.236809	entropy	0.5783	0.2555	22.631	-0.1392	TSLA 0.350001, GOOGL 0.349999, BRK-B 0.161812, DIA 0.138188	E2E
6	MSFT, VTI, TSLA, ^RUT	0.3673	0.2132	17.228	-0.1315	VTI 0.325714, TSLA 0.245358, MSFT 0.237375, ^RUT 0.191553	entropy	0.2984	0.2291	13.027	-0.1266	MSFT 0.35000, VTI 0.31862, TSLA 0.17999, ^RUT 0.15139	Two-step
7	NVDA, ^RUT, META, SI=F	0.7719	0.2488	31.022	-0.1298	NVDA 0.283722, META 0.268065, SI=F 0.241863, ^RUT 0.206350	entropy	0.8609	0.2745	31.358	-0.1322	NVDA 0.350001, META 0.349994, SI=F 0.198356, ^RUT 0.101650	E2E
8	HD, GOOGL, GC=F, ^DJI	0.2779	0.1195	23.251	-0.0752	GOOGL 0.257832, GC=F 0.250268, ^DJI 0.246257, HD 0.245644	entropy	0.2384	0.1287	18.514	-0.0807	HD 0.349989, GC=F 0.311997, GOOGL 0.182701, ^DJI 0.155313	Two-step
9	PG, ^RUT, USDJPY=X, NVDA	0.4747	0.1579	30.064	-0.0953	NVDA 0.267000, ^RUT 0.254665, PG 0.247831, USDJPY=X 0.230505	markowitz	12.327	0.2935	41.999	-0.1589	PG 0.551981, USDJPY=X 0.233235, NVDA 0.214828	E2E
10	PG, GOOG, CL=F, SPY	0.2267	0.1162	19.510	-0.0947	PG 0.270018, GOOG 0.257340, SPY 0.249877, CL=F 0.222765	entropy	0.2599	0.1169	22.232	-0.0807	GOOG 0.349992, PG 0.349992, SPY 0.295637	E2E
11	UNH, MSFT, DIA, ^IXIC	0.1925	0.1291	14.909	-0.0704	^IXIC 0.294936, DIA 0.238836, MSFT 0.233649, UNH 0.232578	entropy	0.1688	0.1354	12.465	-0.0601	UNH 0.350000, MSFT 0.350000, DIA 0.246088, ^IXIC 0.053912	Two-step
12	QQQ, AMZN, VTI, GOOG	0.3696	0.1796	20.580	-0.1400	VTI 0.280581, AMZN 0.254872, QQQ 0.233039, GOOG 0.231509	markowitz	0.4007	0.2172	18.445	-0.1739	AMZN 0.800012, QQQ 0.157590, VTI 0.042391	Two-step
13	SI=F, AMZN, ^RUT, NVDA	0.6009	0.2215	27.129	-0.1478	^RUT 0.270364, SI=F 0.269216, NVDA 0.232251, AMZN 0.228170	entropy	0.8171	0.2721	30.034	-0.1755	AMZN 0.349999, NVDA 0.349998, SI=F 0.262244, ^RUT 0.037759	E2E
14	JNJ, SPY, CL=F, META	0.2649	0.1345	19.697	-0.0865	META 0.278759, JNJ 0.261553, SPY 0.234446, CL=F 0.225243	entropy	0.3252	0.1485	21.893	-0.0924	META 0.350002, SPY 0.349998, JNJ 0.220668, CL=F 0.079332	E2E
15	AAPL, JNJ, NVDA, GOOGL	0.5633	0.1896	29.707	-0.1207	NVDA 0.258765, JNJ 0.252212, AAPL 0.248838, GOOGL 0.240185	markowitz	0.9289	0.2967	31.304	-0.1708	AAPL 0.575966, NVDA 0.297289, JNJ 0.126739	E2E
16	CL=F, MSFT, GC=F, V	0.1882	0.1146	16.420	-0.0806	V 0.270267, CL=F 0.254953, GC=F 0.240416, MSFT 0.234364	entropy	0.2509	0.1123	22.342	-0.0698	MSFT 0.349999, GC=F 0.349999, V 0.293588	E2E
17	AAPL, JPM, AMZN, GC=F	0.3806	0.1337	28.459	-0.1004	GC=F 0.353774, AAPL 0.223197, AMZN 0.214525, JPM 0.208504	markowitz	0.5734	0.1478	38.799	-0.0560	JPM 0.502069, GC=F 0.333230, AMZN 0.155364	E2E
18	^GSPC, META, USDJPY=X, UNH	0.2564	0.1232	20.824	-0.0437	UNH 0.275470, META 0.250358, USDJPY=X 0.245341, ^GSPC 0.228832	entropy	0.2651	0.1334	19.872	-0.0477	META 0.273917, USDJPY=X 0.252279, UNH 0.251652, ^GSPC 0.222152	Two-step
19	^IXIC, ^VIX, GC=F, VTI	0.4914	0.2859	17.189	-0.1654	^VIX 0.272606, ^IXIC 0.261541, GC=F 0.252780, VTI 0.213072	markowitz	0.4850	0.1217	39.863	-0.0540	VTI 0.578012, ^IXIC 0.347634, ^VIX 0.074350	E2E
20	USDJPY=X, AMZN, DIA, ^DJI	0.2485	0.1178	21.090	-0.0906	AMZN 0.269602, DIA 0.263916, USDJPY=X 0.234137, ^DJI 0.232345	markowitz	0.3340	0.1500	22.271	-0.0734	AMZN 0.716811, USDJPY=X 0.141900, DIA 0.141248	E2E
21	IWM, VTI, EURUSD=X, BRK-B	0.1614	0.0976	16.539	-0.0571	EURUSD=X 0.281609, BRK-B 0.269175, IWM 0.226992, VTI 0.222225	markowitz	0.3096	0.0968	31.975	-0.0707	VTI 0.800057, IWM 0.086892, BRK-B 0.064458, EURUSD=X 0.048592	E2E
22	^DJI, TSLA, PG, V	0.3597	0.1952	18.423	-0.0963	TSLA 0.265727, ^DJI 0.257593, V 0.241782, PG 0.234898	markowitz	0.4796	0.2502	19.167	-0.1236	TSLA 0.522669, PG 0.242933, V 0.234397	E2E
23	SPY, META, ^GSPC, ^VIX	0.6506	0.2298	28.309	-0.1175	META 0.407617, SPY 0.277018, ^GSPC 0.195231, ^VIX 0.120134	entropy	0.4247	0.1057	40.190	-0.0534	^GSPC 0.349999, SPY 0.349999, META 0.207519, ^VIX 0.092484	E2E
24	^RUT, GC=F, USDJPY=X, ^GSPC	0.2026	0.0978	20.709	-0.0759	^RUT 0.274467, USDJPY=X 0.249633, ^GSPC 0.245571, GC=F 0.230328	markowitz	0.2424	0.0973	24.908	-0.0615	GC=F 0.585164, ^GSPC 0.414849	E2E
25	AAPL, PG, IWM, TSLA	0.3817	0.2068	18.453	-0.1281	AAPL 0.268700, IWM 0.261300, TSLA 0.242031, PG 0.227969	entropy	0.5057	0.2338	21.633	-0.1315	TSLA 0.349999, AAPL 0.313295, PG 0.176697, IWM 0.160009	E2E
26	V, ^VIX, AAPL, QQQ	0.4479	0.2785	16.083	-0.1703	V 0.258672, QQQ 0.250144, ^VIX 0.249125, AAPL 0.242059	entropy	0.3617	0.1342	26.959	-0.0719	V 0.349970, QQQ 0.284145, AAPL 0.193929, ^VIX 0.171956	E2E
27	V, SI=F, CL=F, AMZN	0.2675	0.1527	17.520	-0.1247	V 0.275965, AMZN 0.248810, SI=F 0.245698, CL=F 0.229527	entropy	0.3652	0.1635	22.335	-0.1275	V 0.349998, AMZN 0.349998, SI=F 0.298043	E2E
28	VTI, MSFT, QQQ, GC=F	0.2804	0.1265	22.159	-0.0882	QQQ 0.266735, GC=F 0.253993, VTI 0.251979, MSFT 0.227293	entropy	0.2569	0.1424	18.046	-0.1163	MSFT 0.350000, QQQ 0.350000, VTI 0.263704, GC=F 0.036295	Two-step
29	IWM, ^IXIC, ^RUT, PG	0.2172	0.1392	15.600	-0.0785	IWM 0.294994, ^RUT 0.248539, PG 0.230933, ^IXIC 0.225535	entropy	0.2578	0.1285	20.061	-0.0840	PG 0.350000, ^IXIC 0.350000, IWM 0.233481, ^RUT 0.066519	E2E
30	PG, CL=F, NVDA, MSFT	0.4182	0.1675	24.974	-0.1191	CL=F 0.286361, PG 0.270158, MSFT 0.226992, NVDA 0.216489	entropy	0.6358	0.2173	29.267	-0.1411	MSFT 0.349996, NVDA 0.343669, PG 0.302838	E2E
31	USDJPY=X, ^VIX, EURUSD=X, PG	0.2317	0.3273	0.7077	-0.1900	PG 0.297312, ^VIX 0.249961, EURUSD=X 0.245300, USDJPY=X 0.207427	markowitz	0.1073	0.0784	13.685	-0.0335	USDJPY=X 0.465975, PG 0.305686, EURUSD=X 0.210656, ^VIX 0.017683	E2E
32	GOOG, META, JNJ, JPM	0.3827	0.1471	26.019	-0.0636	JPM 0.295765, GOOG 0.255731, JNJ 0.250708, META 0.197795	entropy	0.3191	0.1517	21.041	-0.0809	GOOG 0.350009, JPM 0.350007, JNJ 0.239820, META 0.060164	Two-step
33	TSLA, V, NVDA, GOOGL	0.7845	0.2727	28.768	-0.1864	GOOGL 0.272349, TSLA 0.248572, NVDA 0.242068, V 0.237010	entropy	0.5410	0.2504	21.608	-0.1523	GOOGL 0.349989, V 0.349987, NVDA 0.219161, TSLA 0.080863	Two-step
34	CL=F, ^IXIC, AAPL, DIA	0.2165	0.1315	16.461	-0.0983	^IXIC 0.314976, AAPL 0.234144, DIA 0.227128, CL=F 0.223751	entropy	0.2730	0.1423	19.181	-0.0951	AAPL 0.349996, DIA 0.332659, ^IXIC 0.313313	E2E

Comparison between our end-to-end model and a two-step model with the forecaster and allocation separated

Fig.3



Our web-application when opened with an example of input

Fig.4

Our web-application with the output of the model explained

CHAPTER 10

BIBLIOGRAPHY

- **Calzone, O. (2018, May 15)**. *An intuitive explanation of LSTM*. *Medium*. https://medium.com/@ottaviocalzone/an-intuitive-explanation-of-lstm-a035eb6ab42c
- **Agrawal, A., Amos, B., Barratt, S., Boyd, S., Diamond, S., & Kolter, J. Z. (2019).** Differentiable convex optimization layers. *Advances in Neural Information Processing Systems*, 32.
- **Allen, F., & Karjalainen, R. (1999).** Using genetic algorithms to find technical trading rules. *Journal of Financial Economics, 51*(2), 245–271.
- **Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., … & Amodei, D. (2020).** Language models are few-shot learners. *Advances in Neural Information Processing Systems, 33*, 1877–1901.
- **Deng, Y., Bao, F., Kong, Y., Ren, Z., & Dai, Q. (2016).** Deep direct reinforcement learning for financial signal representation and trading. *IEEE Transactions on Neural Networks and Learning Systems, 28*(3), 653–664.
- **Elton, E. J., & Gruber, M. J. (1997).** Modern portfolio theory, 1950 to date. *Journal of Banking & Finance, 21*(11–12), 1743–1759.
- **Fischer, T., & Krauss, C. (2018).** Deep learning with long short-term memory networks for financial market predictions. *European Journal of Operational Research, 270*(2), 654–669.
- **Hochreiter, S., & Schmidhuber, J. (1997).** Long short-term memory. *Neural Computation, 9*(8), 1735–1780.
- **Lintner, J. (1965).** The valuation of risk assets and the selection of risky investments in stock portfolios and capital budgets. *The Review of Economics and Statistics, 47*(1), 13–37.
- **Markowitz, H. (1952).** Portfolio selection. *The Journal of Finance, 7*(1), 77–91.
- **Merton, R. C. (1980).** On estimating the expected return on the market. *Journal of Financial Economics, 8*(4), 323–361.
- **Moody, J., & Saffell, M. (1999).** Reinforcement learning for trading. In *Advances in Neural Information Processing Systems* (pp. 917–923).
- **Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019).** Language models are unsupervised multitask learners. *OpenAI Blog*.

- **Sharpe, W. F. (1964).** Capital asset prices: A theory of market equilibrium under conditions of risk. *The Journal of Finance, 19*(3), 425–442.
- **Zhang, X., Wu, Y., & Zhang, Y. (2017).** Stock market prediction of S&P 500 via combination of long-term and short-term trading signals. *Neurocomputing, 213*, 72–82.
- **Zhang, Y., Li, Z., & Wang, X. (2020).** End-to-end portfolio optimization using deep neural networks: A Markowitz-based approach. *The Journal of Finance and Data Science, 6*(4), 300–315.