Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

School of Computer Science and Statistics

An Investigation into Machine Learning-Based Asset Class Prediction for Investment Portfolios

John Wallace

April 2025

A Report submitted in partial fulfilment

of the requirements for the degree of

BA(Mod) in Science in Computer Science and Business

Acknowledgements

To Dr.Inmaculada Arnedillo-Sánchez, for her generosity with her time, her guidance, and her belief in this project. Her support and willingness to supervise my work were instrumental in making this project possible.

To Leonardo Mazza, my mentor throughout this journey thank you for your patience, wisdom, and encouragement. Your insights and steady guidance helped shape this project from its earliest stages, and I am deeply grateful for your support.

To all my friends and peers at Trinity College Dublin, thank you for the memories, the shared challenges, and the laughter over the past four years. I truly appreciate your patience and support, especially during the final stages of this project.

And finally, to my father and sister thank you for always being there. Your unwavering support made it possible for me to complete this project and my degree. I couldn't have done it without you.

Abstract

This project aims to demonstrate the practical application of machine learning models for financial time series forecasting and their potential to enhance investment decision-making. It focuses on the ability of machine learning models to be trained and tested on economic and market data to generate predictions specifically expected returns and volatility—which can then inform portfolio construction decisions. The project serves as a proof of concept for how such predictive models can contribute to more informed and potentially superior investment strategies.

The study evaluates both advanced and more traditional machine learning architectures, including the Temporal Fusion Transformer (TFT) and Long Short-Term Memory (LSTM) networks. These models are applied to monthly time series data using structured macroeconomic indicators and sentiment-driven variables. Each model is critically implemented and assessed based on its forecasting performance and its practical suitability for real-world use.

Crucially, this project aims to bridge the gap between predictive modeling in financial machine learning and classical portfolio allocation theory. By integrating the output of time series forecasts into a Mean-Variance Optimization (MVO) framework, the project explores how forward-looking, data-driven insights can enhance portfolio decision-making. It also contrasts this approach with more traditional investment philosophies, offering a perspective on how modern machine learning tools can complement established financial theory.

Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University and that unless otherwise stated, is my own work. Generative AI tools were used during the development of this project in a limited and supportive role. Specifically, AI was used to assist with code debugging, formatting and correcting grammar and spelling errors in the report. However, the core ideas, implementation decisions and overall structure of both the report and code were entirely my own. All intellectual contributions, unless otherwise stated, reflect my personal understanding and effort.

John Wallace

April 12, 2025

Contents

List of Tables

Chapter 1

Introduction

1.1 Motivation

This project represents the culmination of my studies in Computer Science and Business at Trinity College Dublin, integrating interdisciplinary academic knowledge with professional experience gained in the financial services industry.

1.1.1 Limitations of Traditional Financial Models

Since the adoption of quantitative methods in the 20th century, computational statistics and mathematical models have played a foundational role in the financial sector. Models such as the Capital Asset Pricing Model (CAPM) Perold (2004) and Modern Portfolio Theory (MPT) H. M. Markowitz (1991) have been instrumental in shaping investment theory. However, in today's increasingly complex and data driven markets, these models struggle to capture non linear dynamics, interdependencies and the stochastic behavior of economic systems. As a result, their predictive power in practical asset allocation has become limited.

1.1.2 The Challenge of Scale in Financial Data

Modern financial analysis relies on vast, high dimensional datasets encompassing economic indicators, sentiment measures and market data. Traditional, human led methods often fall short in extracting meaningful patterns from such complex information. As relationships between market variables become more intricate and evolve over time, there is a growing need for scalable, automated approaches that can synthesize this information effectively to support investment decisions.

1.1.3 The Rise of Machine Learning—But Only for the Few

Machine learning (ML) has emerged as a powerful solution for modeling non linearities and uncovering latent structures within financial data. Yet, access to ML based investment systems remains highly concentrated. Many state of the art models are proprietary, developed by top hedge funds and institutions that invest millions into research and infrastructure. Access is typically restricted to those with substantial capital—either to build in house systems or to buy into funds that employ these models.

Moreover, some of the most successful quantitative firms, such as Renaissance Technologies, have ceased accepting external capital for certain strategies, further reinforcing the exclusivity of machine learning in finance. As a result, smaller firms, independent quants and researchers are often left without access to the most advanced tools.

1.1.4 Bridging the Gap: Democratizing Predictive Finance

This project aims to address that gap by serving as a proof of concept: demonstrating that sophisticated ML driven investment tools can be developed with modest resources, without compromising on practicality or robustness. The goal is to contribute to academic research while expanding access to predictive modeling for a wider audience including smaller asset managers, academics and independent practitioners. By lowering the cost and complexity of entry, this work advocates for a more inclusive and transparent financial ecosystem.

1.1.5 A Trillion Dollar Industry in Need of Accessible Tools

As of June 2024, an estimated $132 trillion in assets was under professional management globally, (Lai et al. 2024) . While not all of this capital is allocated to public markets, the figure underscores the scale and influence of institutional investors. Democratizing access to data driven investment tools has the potential to redistribute analytical power, empowering more participants to make informed, evidence based decisions.

1.1.6 Beyond Trading: A Focus on Long Term Investing

Most existing research in ML for finance focuses on short term trading strategies, often aligning with speculative objectives and high frequency environments. While these studies offer valuable insights into market behavior, they frequently neglect the long term horizons central to portfolio management and strategic asset allocation.

This project adopts a different lens: it centers on investment oriented forecasting, targeting monthly predictions of asset class returns and volatility. The aim is to improve long term portfolio construction and risk management through a fusion of machine learning techniques and financial theory.

1.1.7 Streamlining the Financial Research Process

From my industry experience, I have observed that analysts and asset managers spend considerable time manually evaluating historical trends and macroeconomic data to guide investment decisions. This process is often inefficient and subjective. By integrating machine learning, this project seeks to automate and enhance forecasting workflows reducing time spent on manual research while increasing the consistency and reliability of insights.

1.1.8 An Integrative and Transparent Vision

The models developed in this project incorporate multiple interrelated variables—economic indicators, sentiment measures and market data to forecast key financial metrics such as return and volatility. Rather than modeling these factors in isolation, the models account for their interactions and evolving impact on asset class performance. In doing so, the project aims to deliver a practical and transparent framework that enhances decision making in asset management.

1.2 Research Statement and Objectives

The objective of this project is to develop a machine learning based system as a proof of concept to demonstrate the feasibility of using such tools for effective portfolio optimization, highlighting that advanced techniques can be applied without the need for complex or exclusive resources. The system will predict key financial outcomes, such as returns and volatility, over a defined time horizon and compare the performance of various advanced models. Through this approach, the project seeks to construct a structured and optimized investment portfolio and evaluate its performance relative to conventional portfolio management techniques.

The project has two primary objectives:

1. Model Performance Evaluation: Assess the predictive accuracy and effectiveness of two machine learning models in forecasting asset returns and volatility.
2. Portfolio Optimization: Utilise the predictions from these models to construct an investment portfolio and compare its performance against traditional portfolio allocation strategies.

By integrating widely used financial indicators with machine learning models, this project aims to enhance the accuracy of return forecasts for key asset classes, including equities, bonds and gold. The ultimate goal is to demonstrate the practical utility and cost effectiveness of these methodologies in improving portfolio performance.

1.2.1 Objective 1: Comparison of Models

This project will empirically evaluate the predictive capabilities of different machine learning models, assessing their performance across a set of predefined metrics.

1.2.2 Objective 2: Theoretical Investment Portfolio

Building on the foundation of the models' predictions, the project will construct a theoretical investment portfolio comprising various asset classes with optimal weightings based on predicted returns and volatility. The performance of this hypothetical portfolio will be compared against traditional static portfolio allocations to evaluate its effectiveness.

Chapter 2

Literature Review

This chapter explores the core fundamental concepts and challenges critical to understanding this project. It also examines existing research in the field of predictive models for time-series forecasting in financial markets.

2.1 Examination into Predictive Models

Initial research focused on various machine learning models currently in existence, with an emphasis on understanding deep learning techniques.

Machine Learning (ML) models employ regression and correlation techniques to generate predictions. Fundamentally, they all function similarly: each ML model takes a series of input variables and "learns" the importance (weights) of each variable in predicting the output. By analyzing past data sets where inputs and outputs are known, these models can predict unknown outputs with relative accuracy and confidence.

Although most ML models were originally designed for language processing and classification tasks, advancements have been made to adapt these models for financial or time-series data prediction. This project builds upon these developments to focus specifically on time-series forecasting, (Machine Learning Models 2023).

2.1.1 Time-Series Predictive Models

Research into time-series prediction has examined both machine learning and statistical methods across financial and non-financial domains. By reviewing existing literature, a foundation of empirical data was established, enabling a focused effort on model types and comparisons between statistical and machine learning approaches.

2.1.2 Examination of Deep Learning Models

Neural networks and deep learning models are central to modern machine learning, with applications across finance, energy, and environmental science (Masood 2023). Kontopoulou et al. (2023) reviewed time-series forecasting methods in domains such as wind power, COVID-19 and oil production, finding that machine learning models generally outperformed traditional statistical approaches. Building on this, Mortezapour Shiri et al. (2024) highlighted the strengths of deep learning techniques—particularly LSTM and GRU models—for sequential data, while CNNs remained dominant in image-related tasks (see Table 2.3).

Kong et al. (2024) compared deep learning methods across solar energy forecasting, electricity usage and weather prediction. Table 2.1, adapted from their work, summarizes model performance using averaged metrics (MSE and MAE) to compare LSTM-based models, transformer architectures, and statistical methods like RLinear and DLinear. Transformers represent a recent leap in deep learning, delivering strong results across multiple domains. Ahmed et al. (2023) explored their adaptation for time-series analysis, while Lim et al. (2021) introduced the Temporal Fusion Transformer (TFT), designed to address the interpretability limitations noted by Wallaroo AI (2024). TFT's attention mechanisms and variable selection networks enabled it to outperform many prior models. Although Lim et al. (2021) focused on multi-horizon forecasts, their architecture provides a solid basis for short-term financial forecasting in this project.

Table 2.1: MSE and MAE comparison adapted from Table 2 in Kong et al. (2024)

Dataset	**P-sLSTM**		**LSTM**		**iTransformer**		**Rlinear**		**Dlinear**	
	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
Kong	0.2126	0.28065	0.38965	0.41635	0.216855	0.2828	0.30695	0.3345	0.25994	0.32602

2.2 Chosen Models Research

Financial time-series forecasting requires models that can capture complex temporal dependencies. This study compares two deep learning architectures: Long Short-Term Memory (LSTM) networks and the Temporal Fusion Transformer (TFT). LSTM is a strong baseline for sequential data, while TFT offers enhanced interpretability and performance through attention mechanisms.

Empirical research shows LSTM often outperforms traditional statistical models in modeling long-term dependencies (see Table 2.2). However, it can struggle with very long-range patterns and lacks built-in interpretability. TFT addresses these issues by leveraging attention layers and variable selection networks, making it well-suited for high-dimensional, multi-horizon forecasting (Lim et al. 2021).

This research evaluates the forecasting accuracy of both models in financial contexts, contributing to the literature on deep learning applications in time-series analysis.

Specific Analysis of the LSTM Model

A primary advantage of LSTM over traditional models such as ARIMA (Autoregressive Integrated Moving Average) and SARIMA (Seasonal ARIMA) lies in its capacity to capture nonlinear patterns and retain long-term dependencies within time-series data. Multiple empirical studies have demonstrated that LSTM consistently achieves lower forecasting errors across various performance metrics.

For instance, Dubey et al. (2021) compared the performance of ARIMA, SARIMA and LSTM using energy consumption data. The LSTM model achieved significantly lower error values in terms of RMSE, MAPE, and other key metrics (Table 2.2), demonstrating its superior predictive accuracy:
While ARIMA and SARIMA are limited to modeling linear trends and short-term dependencies, LSTM networks can learn complex, nonlinear relationships and adapt to more dynamic environments, such as those encountered in financial forecasting. Nevertheless, as highlighted by Mortezapour Shiri et al. (2024), LSTM models may

Table 2.2: Error Metrics Adapted from Table 8 in Dubey et al. (2021)

Metric	**SARIMA**	**LSTM**
RMSLE	0.06009	0.00101
RMSE	0.55065	0.23191
MASE	0.58524	0.32456
MAPE	5.23690	3.22134

experience performance degradation over longer forecasting horizons due to difficulties in capturing extended temporal dependencies.

A broad survey of deep learning architectures such as CNNs, RNNs, LSTMs and their bidirectional and gated variants reveals the importance of selecting an appropriate model for the task at hand. As shown in Table 2.3, adapted from Mortezapour Shiri et al. (2024), LSTM and GRU-based models generally exhibit superior accuracy and precision compared to simpler architectures. This underscores the value of choosing models capable of handling the specific complexities of financial time-series data.

Table 2.3: Accuracy and Precision Comparison Adapted from Table 2 in Mortezapour Shiri et al. (2024)

Model	**Accuracy (%)**	**Precision (%)**
CNN	89.18	93.78
RNN	91.73	93.56
LSTM	92.27	93.97
Bi-LSTM	92.21	93.91
GRU	92.24	94.00
Bi-GRU	91.26	92.99

Specific Analysis of the TFT Model

The Temporal Fusion Transformer (TFT) builds upon the strengths of LSTM architectures while incorporating several enhancements that make it especially well-suited for time-series forecasting. Notably, TFT enables multi-horizon forecasting and improves interpretability through attention mechanisms and dynamic variable selection layers. Although this study employs TFT for single-step forecasting, its architecture is designed to scale effectively across multiple time steps.

Lim et al. (2021) introduced the TFT model and benchmarked its performance against a range of forecasting methods, including ARIMA, DeepAR (Salinas, Flunkert, and Gasthaus 2020) and MQRNN, using datasets such as electricity consumption and freeway traffic occupancy rates (Yu, Rao, and Dhillon 2016). TFT achieved the lowest quantile loss in both P50 and P90 estimates, outperforming alternative models across datasets (Table 2.4):

Table 2.4: P50 and P90 Quantile Losses on Real-World Datasets (Adapted from Lim et al. (2021))

Prediction Method	**Electricity**	**Traffic**
ARIMA	0.154	0.233
ETS	0.102	0.236
TRMF	0.084	0.186
DEEPAR	0.075	0.161
DSSM	0.083	0.167
ConvTrans	0.059	0.122
Ridge	0.064	0.135
MLP	0.062	0.122
Seq2Seq	0.067	0.105
MQRNN	0.077	0.117
TFT	**0.055**	**0.096**

TFT's architecture allows it to identify and attend to the most relevant input features at each time step, significantly enhancing forecasting accuracy and model interpretability. This makes TFT particularly advantageous for financial datasets, where multiple interacting variables and dynamic regimes are common.

2.2.1 Summary of Model Selection

The selection of LSTM and TFT is driven by data-driven insights from existing research, which highlight their respective strengths and limitations in time-series forecasting.

- LSTM remains a widely researched recurrent neural network known for its ability to capture sequential dependencies effectively. Studies suggest it performs well in short to moderate forecasting horizons but struggles with long-range dependencies.

(Mortezapour Shiri et al. 2024).

- TFT, in contrast, incorporates attention mechanisms, variable selection networks and gating layers to improve interpretability and accuracy. Empirical evidence suggests that TFT consistently outperforms LSTM in multi-horizon forecasting tasks by addressing long-term dependencies more effectively, (Lim et al. 2021).

By comparing these models, this study aims to analyse whether attention-based architectures like TFT provide tangible improvements over recurrent models like LSTM in financial time-series forecasting. The findings will contribute to the ongoing discussion on the role of transformer-based models in financial applications and deep learning-driven asset forecasting.

2.2.2 Long Short-Term Memory Units (LSTM)

Long Short-Term Memory (LSTM) units are a type of ”Recurrent Neural Network (RNN)” first introduced by Hochreiter (1997). They were specifically designed to address the ”vanishing gradient problem”, which occurs when gradients diminish exponentially during backpropagation, making it difficult for standard RNNs to learn ”long-term dependencies” in sequential data.

LSTMs mitigate this issue by introducing ”memory cells” and ”gating mechanisms” (input, forget, and output gates), which enable them to ”selectively store and retrieve information” over extended sequences. This allows LSTMs to be more effective than traditional RNNs in capturing ”long-range dependencies” in data.

Various LSTM variants have been introduced and explored, each designed for different applications in modern deep learning architectures:

- **Peephole LSTMs** – Allow gates to access memory cell states directly.
- **Bi-directional LSTMs** – Process sequences in both forward and backward directions.

- **Gated Recurrent Units (GRUs)** – A simplified alternative to LSTMs that reduces computational complexity.

For this project, we utilize publicly available implementations of the ”LSTM model” from the ”PyTorch Forecasting Library” Beitler (2020).

LSTM Architecture

The architecture employed in this project is a vanilla (base) LSTM model. The key advancements introduced by LSTM lie in its incorporation of a memory cell and gating mechanisms, which effectively address the limitations of traditional recurrent neural networks in capturing long-term dependencies. As discussed by Staudemeyer and Morris (2019), these mechanisms enable LSTMs to retain and update relevant information over extended sequences, making them particularly well-suited for time series forecasting tasks such as the one in this study.

Memory Cell The ”memory cell” is the core component of an LSTM network and is responsible for ”storing and maintaining long-term dependencies” in sequential data. Unlike standard RNNs, which overwrite information at each time step, ”LSTM memory cells selectively retain or forget information” over long sequences.

Gating Mechanisms Gating mechanisms regulate the flow of information ”to and from” the LSTM memory cell, ensuring that relevant information is retained while irrelevant information is discarded.

- **Forget Gate**
  - Determines which past information should be discarded.
  - Helps eliminate irrelevant or outdated memory.
- **Input Gate**
  - Controls which new information should be stored in the memory cell.
  - Works alongside the cell input activation to update stored memory.

- **Output Gate**
  - Decides how much of the memory cell's information should be passed to the next time step.
  - Produces the "hidden state output", which is used for predictions.

Activation Functions Activation functions regulate "how much information is passed" through the LSTM and determine the output at each step.

- **Sigmoid Activation** (σ)
  - Used in the gates (input, forget, and output) to squash values between "0 and 1".
  - Effectively decides "how much information should pass through".
- **Tanh Activation** (tanh)
  - Applied to the "cell state" to regulate values between "-1 and 1".
  - Helps balance outputs and "preserves gradients" through time.

Key Model Features Adjusted During Implementation

During the course of the project, there are predefined architectural features of the LSTM model were explored and adjusted to better tailor the model to the forecasting task.

Sequence Length The sequence length refers to the number of time steps the model looks back in the input data to make a prediction. It determines how much past information the model considers before forecasting the next step.

Number of Layers The number of layers refers to the number of stacked LSTM layers in the network. A deeper model with multiple layers allows for learning more complex temporal dependencies.

Hidden Size The hidden size represents the number of hidden units (neurons) in each LSTM layer. It defines the dimensionality of the hidden state and cell state vectors, impacting the model's capacity to learn patterns from the data.

Dropout Dropout is a regularization technique used to prevent overfitting by randomly setting a fraction of neurons to zero during training. This forces the model to learn more robust features rather than over-relying on specific neurons.

2.2.3 Temporal Fusion Transformer (TFT) Model

The Temporal Fusion Transformer (TFT) was first proposed by Lim et al. (2021). It is an attention-based deep learning model optimized for both performance and interpretability in multi-horizon forecasting. TFT introduces a novel architecture that integrates self-attention with recurrent layers enabling it to capture short- and long-term dependencies while preserving interpretability.

The model leverages specialized components including recurrent layers for local temporal processing, self-attention for long-range dependencies, variable selection mechanisms and gating layers to suppress irrelevant information. Since its introduction, TFT has been evaluated across various real-world datasets, consistently outperforming traditional forecasting models.

For this project, we utilize publicly available implementations of the Temporal Fusion Transformer (TFT) model from the PyTorch Forecasting library (Beitler 2020).

Underlying Architecture

TFT consists of several intricate components, each playing a role in processing temporal data. Key input types include ”static covariates”, ”known inputs” and ”observed inputs”.

The main architectural elements of TFT are:

Self-Attention mechanism Is used to capture long-range dependencies in time-series data. It enables the model to focus on the most important time steps while making predictions.

Gating Mechanism The gating mechanism regulates information flow within the network, ensuring efficient learning by dynamically activating or suppressing specific pathways. The two core gating components include:

- **Gated Linear Units (GLUs):** These apply a gating function (e.g., sigmoid activation) to selectively retain or discard input signals.
- **Gated Residual Networks (GRNs):** These introduce skip connections, allowing layers to bypass certain computations, thereby improving gradient flow and training stability.

Variable Selection TFT employs an ”attention-based variable selection network”, which dynamically identifies the most relevant input features at each time step. This ensures that the model focuses on informative signals while ignoring less useful inputs.

Static Covariate Encoders Static covariate encoders process ”time-invariant features”, such as asset class or sector classification, which remain constant over time. These static features enhance forecasting accuracy by providing context to the model.

Interpretable Multi-Head Attention Unlike traditional multi-head attention, TFT's attention mechanism is specifically designed for interpretability, enabling the model to highlight key time steps that contribute most to the final forecast.

Temporal Fusion Decoder The ”Temporal Fusion Decoder” extracts meaningful patterns from historical data and predicts future values using the following layers:

- **Locality Enhancement (Sequence-to-Sequence Layer)**
  - Identifies critical points in the time series (e.g., trend shifts, anomalies).
  - Uses an LSTM-based encoder-decoder framework to process past and future inputs separately.

- **Static Enrichment Layer**
  - Integrates static covariates with dynamic time-series data.
  - Improves forecast consistency by incorporating broader dataset characteristics.
- **Temporal Self-Attention Layer**
  - Captures long-term dependencies via a modified self-attention mechanism.
  - Uses decoder masking to ensure causality in forecasts.
- **Position-Wise Feed-Forward Layer**
  - Applies non-linear transformations to refine self-attention outputs.
  - Uses residual connections to enhance model stability.

Key Model Features Adjusted During Implementation

For the context of this project, there are a set number of architectural features that vary throughout.

Max Encoder Length Defines how many past time steps the model uses to learn patterns before making predictions.

Hidden Size Controls the model's complexity and learning capacity by determining the number of neurons in key network layers.

Attention Head Size Determines how many separate attention mechanisms are used in the multi-head attention layers.

LSTM Layers Used for processing sequential time-series data, capturing long-term dependencies and trends before feeding the data into the attention mechanism.

Hidden Continuous Size Defines the number of hidden units used for encoding continuous input features before they are fed into the main model.

Dropout Prevents overfitting by randomly dropping some connections during training, improving the model's ability to generalize to unseen data.

Time-Varying Known Reals Specifies continuous numerical features that are known ahead of time for all forecasted periods.

Time-Varying Known Specifies categorical features that are known in advance for all forecasted time steps.

2.3 Financial Theory

The focus of this project is on the application of advanced machine learning models for time series prediction within financial markets. It is essential to establish the underlying financial theory that guides this analysis. For this project, the emphasis is on the U.S. financial markets for the following reasons.

- **First:**The U.S. market is one of the most liquid and easily tradable in the world, offering significant opportunities for analysis and forecasting.
- **Second:**It is among the best-researched and most closely monitored markets globally, which provides a wealth of reliable data.

The accessibility and depth of this data make the U.S. financial markets an ideal subject for this study, allowing for the use of sophisticated machine learning techniques to explore and predict market trends.

2.3.1 Asset Classes

The underlying predictions for this project are categorized into three broad classifications: Equities, Bonds and Gold.

Equity

BlackRock (2025) defines an equity investment as “money invested in a company through the purchase of shares or stock in that company.”

To improve manageability and computational efficiency, this project utilizes a proxy for equity markets rather than forecasting returns for individual securities(Companies). This strategy avoids the complexity of modeling each stock separately, instead enabling a consolidated prediction for a broad set of assets. An Exchange-Traded Fund (ETF) is employed for this purpose, as it aggregates a group of securities into a single investment instrument (Chen 2024).

Given the project's focus on U.S. markets, an equity proxy reflecting the overall performance of publicly listed U.S. companies was selected. Specifically, the S&P 500 index is used as the representative equity component of the portfolio. The S&P 500 is a market capitalization-weighted index that comprises 500 of the largest publicly traded companies in the United States, (Kenton 2024). In this weighting scheme, each company's influence on the index is proportional to its total market value, calculated using the following formula:

$$\text{Weight of Company} = \frac{\text{Market Capitalization of Company}}{\sum \text{Market Capitalization of all Companies in Index}} \tag{2.1}$$

This approach ensures that larger companies with greater market capitalization have a proportionally higher influence on the index's movements.

Bonds

A bond is a financial instrument in which investors lend money to an entity at a specified interest rate for a predetermined period. The issuing entity repays the principal along with periodic interest payments to investors, (Fernando 2024).

For this project, the bond component is represented by the Bloomberg U.S. Aggregate Bond Index, a broad-based benchmark for intermediate-term investment-grade bonds traded in the United States. This index serves as a measure of bond market performance and is constructed using a diverse set of fixed-income securities, including U.S. Treasury securities, corporate bonds, mortgage-backed securities (MBS), asset-backed securities

(ABS) and municipal bonds, among other investment-grade debt instruments issued in the U.S.

Similar to how the S&P 500 index is commonly used as a proxy for the U.S. equity markets, the Bloomberg U.S. Aggregate Bond Index serves as a representative measure of the U.S. debt markets and is treated as a tradeable security.

The composition of this index provides insight into its allocation across different fixed-income instruments. As shown in Table 2.5, U.S. Treasury securities constitute the largest component, followed by mortgage-backed securities and corporate bonds, reflecting the index's focus on investment-grade debt.

Issuer	**Weight (%)**
United States Treasury	44.55
Federal National Mortgage Association	11.22
Government National Mortgage Association II	5.80
Federal Home Loan Mortgage Corporation	5.52
Uniform MBS	1.52
JPMorgan Chase & Co.	0.60
Bank of America Corp.	0.59
Government National Mortgage Association I	0.49
Morgan Stanley	0.45
Goldman Sachs Group Inc.	0.39

Table 2.5: Composition of the Bloomberg U.S. Aggregate Bond Index as of February 26, 2025.

Debt, particularly U.S. Treasury bonds. Sovereign debt securities are issued by governments and are generally considered low-risk investments due to their backing by national governments. U.S. Treasury bonds serve as a benchmark for fixed-income securities and play a critical role in diversified investment portfolios.

Gold

Gold has historically played a crucial role in investment portfolios, primarily as a store of value and a hedge against inflation.Gold is widely regarded as a safe-haven asset during periods of economic uncertainty. Its intrinsic value, limited supply and historical significance contribute to its role as a wealth preservation instrument. Empirical research highlights gold's effectiveness as a hedge against inflation and its comparative stability relative to other asset classes during financial crises, (Pattnaik et al. 2023).

In this project, gold is incorporated as an asset class due to its diversification benefits and its historically inverse correlation with equities during periods of market distress. For the purpose of this study, historical gold prices are sourced from Gold Futures spot prices, where the tradable security specification is 100 troy ounces, with pricing denominated in U.S. dollars and cents per troy ounce, (CME Group 2025).

2.3.2 Chosen Input Features

This section explores the research supporting each economic and financial indicator used in the predictive model, evaluating their relevance and importance based on academic literature. The selection of input features for the models was based on two key criteria:

- (1) The availability of data, particularly from publicly accessible sources.
- (2) The abundance of historical data to enable the development of a robust and well-trained model.

The chosen features were selected based on their significance to market participants. The indicators can be broadly classified into two categories: lagging indicators and forward-looking indicators. For context, all the indicators used in this study pertain to measuring conditions in the United States.

Lagging Indicators

Lagging indicators are metrics that are backward-looking and often refer to the current or previous state of the economy.

The key lagging indicators used in this study are:

- **Consumer Price Index (CPI)**: The CPI is a cost-of-living index that measures inflation in the United States by tracking price fluctuations in a basket of goods and services. Wynne and Sigalla 1994 provides an in-depth analysis of its construction and limitations. Inflation is crucial to economic activity and since the USA's GDP is largely driven by consumption, changes in purchasing power directly impact the broader economy and corporate revenues.
- **Unemployment Rate**: The unemployment rate represents the percentage of the working population that is unemployed. High unemployment is generally detrimental to economic growth and often negatively impacts the stock market. The significance of this indicator is explored in Chi (2020) and in Gonzalo and Taamouti (2017).
- **Interest Rates (1-Year and 10-Year)**: Interest rates affect borrowing costs for firms, which in turn influence corporate expansion and economic growth. Uddin and Alam (2010) highlights the importance of interest rates in stock market predictions.
- **Asset Class Past Month Returns and Volatility**: Past returns and volatility for each asset class are used as predictors of future returns and volatility. Ross (2023) supports the use of past performance in predictive models.

Forward-Looking Metrics

Forward-looking metrics aim to gauge anticipated future expectations of various economic and financial indicators. These metrics play a crucial role in assessing potential market conditions and consequently, the expected performance of financial markets.

The key forward-looking indicators used in this study are:

- **Forecasted Consumer Price Index (CPI)**: Inflation expectations influence market performance. Chiang (2023) finds a consistently negative relationship between real stock market returns and expected inflation, highlighting the importance of including forecasted CPI in market modeling.
- **Purchasing Managers' Index (PMI)**: PMI is a key monthly indicator of economic health, widely used to gauge the direction of manufacturing activity. Özkan and

Kiriş (2020) examined the role of PMI in Turkish financial markets and found that, when structural breaks are accounted for, PMI Granger-causes stock indices such as the BIST-100 and BIST-Metal Goods Sector index. This supports the use of PMI as a leading indicator capable of influencing investor sentiment and forecasting market performance.

The inclusion of these indicators allows the system to account for both macroeconomic changes in the U.S. economy and their impact on financial assets. By incorporating key economic and financial metrics—such as inflation expectations, interest rates and unemployment surprises—the models aim to capture shifts in macroeconomic conditions and their relationship to asset performance, while also accounting for the behaviour and sentiment of market participants. Empirical evidence supporting the relevance of these indicators is provided in Appendix A, where prior studies and regression analyses (e.g., Chiang (2023), Uddin and Alam (2010), Chi (2020), Gonzalo and Taamouti (2017), Ross (2023)) demonstrate statistically significant relationships between these variables and stock market returns or volatility.

2.3.3 Portfolio Theory

The second objective of this project is to utilize the insights generated from the predictive model and apply them to a practical use case in constructing an investment portfolio.

Over the years, researchers have proposed various theories on how to construct a portfolio. For the context of this project, the focus was on the following portfolio construction methods:

- **Mean-Variance Optimization (MVO)**: A mathematical framework pioneered by H. Markowitz (1952), MVO models individual security returns as random variables and uses their expected values and variances to quantify both investment return and investment risk. This method seeks to optimize the trade-off between risk and return to construct an efficient portfolio. For a more in-depth review, please refer to Zhang, Li, and Guo (2018).
- **Black-Litterman Model**: Developed in 1990 by Black and Litterman (1992), this model combines concepts from the Capital Asset Pricing Model (CAPM)

and Markowitz's mean-variance optimization framework. It allows investors to incorporate their own views on expected returns while balancing these against the equilibrium market returns to determine optimal portfolio weights.

For the purpose of this project, the Mean-Variance Optimization (MVO) approach was chosen over the Black-Litterman model for the following reasons:

- Unlike the Black-Litterman model, MVO does not rely on a predefined market equilibrium portfolio or the estimation of implied returns. This makes it more suitable in contexts where only forecasts of expected returns and volatilities are available, without requiring prior assumptions about the market's overall composition.
- MVO allows the use of predictable metrics while also incorporating the correlation between assets, which provides a structured approach to portfolio diversification.

2.3.4 Mathematical Formulation of Mean-Variance Optimization with Regularization

The goal of Mean-Variance Optimization (MVO), introduced by H. Markowitz (1952), is to allocate assets in a portfolio to maximize expected return while managing risk. In this project, we expand upon the classical MVO framework by incorporating regularization and confidence score adjustments for both returns and volatilities, inspired by practical implementation using convex optimization in Python.

Portfolio Expected Return

The expected return of the portfolio is:

$$E(R_p) = \mathbf{w}^T \boldsymbol{\mu} \tag{2.2}$$

where:

- $\mathbf{w} = (w_1, w_2, ..., w_n)$ represents the portfolio weights.
- $\boldsymbol{\mu} = (E(R_1), E(R_2), ..., E(R_n))$ represents the expected returns of the assets.

Portfolio Risk (Variance)

The portfolio risk is quantified as the variance of portfolio returns, which measures the dispersion of returns due to individual asset risks and their correlations. It is given by the following quadratic form:

$$\sigma_p^2 = \mathbf{w}^T \Sigma \mathbf{w} \tag{2.3}$$

where:

- σ_p^2 is the portfolio variance.
- $\mathbf{w}$ is the vector of portfolio weights.
- Σ is the covariance matrix of asset returns.

The covariance matrix Σ is computed using forecasted asset volatilities and their correlations:

$$\Sigma = \mathbf{D} \cdot \mathbf{C} \cdot \mathbf{D} \tag{2.4}$$

where:

- $\mathbf{D} = \text{diag}(\sigma_1, \sigma_2, ..., \sigma_n)$ is a diagonal matrix of forecasted asset volatilities σ_i.
- $\mathbf{C}$ is the correlation matrix of asset returns.

This formulation captures both the individual volatilities of assets and their pairwise correlations. The quadratic form ensures that each asset's contribution to total portfolio risk is accurately weighted by its allocation and by how it co-varies with other assets.

Adjusted Inputs via Confidence Scores

To improve robustness, predicted returns $\boldsymbol{\mu}$ and volatilities $\boldsymbol{\sigma}$ are adjusted using confidence scores derived from interval forecasting:

$$\tilde{\mu}_i = \mu_i \cdot \frac{1}{\text{ReturnScore}_i + \epsilon} \tag{2.5}$$

$$\tilde{\sigma}_i = \sigma_i \cdot (1 + \gamma \cdot \text{VolScore}_i) \tag{2.6}$$

where:

- ReturnScore_i and VolScore_i denote the uncertainty scores for return and volatility.
- γ is a volatility penalty scaling factor.
- ϵ is a small constant to avoid division by zero.

Optimization Problem with Regularization

The modified optimization problem is solved using convex quadratic programming:

$$\min_{\mathbf{w}} \quad \frac{1}{2}\mathbf{w}^T(\lambda\Sigma + \rho I)\mathbf{w} - \mathbf{w}^T\tilde{\boldsymbol{\mu}} \tag{2.7}$$

subject to:

$$\sum_{i=1}^{n} w_i = 1 \quad \text{(Fully invested)} \tag{2.8}$$

$$w_i \geq 0 \quad \text{(No short-selling)} \tag{2.9}$$

where:

- λ is the risk aversion parameter.
- ρ is the L2 regularization penalty (controls weight magnitude and reduces overfitting).
- I is the identity matrix.

In theory, this framework constructs a forward-looking, risk-aware portfolio that dynamically adapts to new information. By integrating predicted returns and volatilities adjusted for confidence the resulting allocation emphasizes assets with both high expected performance and low estimation uncertainty. Compared to traditional MVO, this approach tends to produce portfolios that are more robust to prediction error and better aligned with anticipated market conditions, rather than purely historical ones. The use of regularization further ensures diversification and guards against overfitting to noisy forecasts, creating a theoretically well-balanced portfolio optimized for risk-adjusted return under uncertainty.

Chapter 3

Methodology and Technical Pre-requisites

This chapter focuses on the foundational aspects of the project, including the methodology used to conduct the study and the technical prerequisites required for its implementation. It outlines the approach taken, the data collection & processing methods and the computational resources necessary to execute the project successfully.

3.1 Methodology

This section outlines the overall approach taken in this project. The methodology was divided into five main phases (Figure 3.1):

1. **Data Collection and Preparation:** Gathering and structuring relevant economic indicators and asset class data from sources such as Bloomberg and Capital IQ.
2. **Model Creation and Hyperparameter Tuning:** Experimenting with different model configurations to identify the optimal set of hyperparameters for each architecture.
3. **Model Training:** Training the LSTM and TFT models on the historical financial dataset, using the tuned hyperparameters.
4. **Model Validation:** Evaluating model performance using various accuracy and error metrics on the validation set.

5. **Theoretical Portfolio Construction and Evaluation:** Constructing a theoretical investment portfolio based on the model predictions and assessing its performance over the defined testing period.

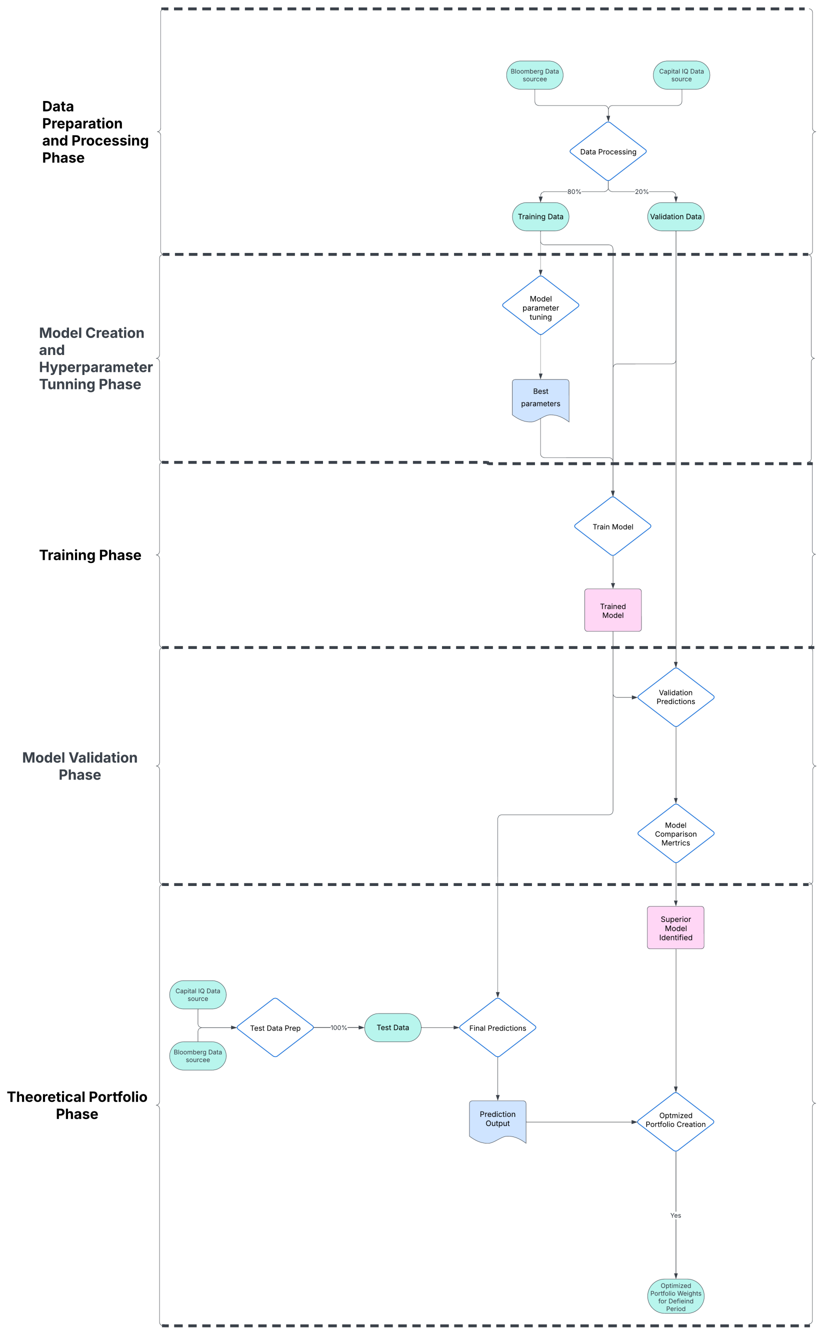

Figure 3.1: High-level methodology overview

3.1.1 Data Collection and Preparation

The first phase involved gathering datasets, which were sourced from **Bloomberg** and **S&P Global**, both of which are accessible to Trinity College undergraduates. The following data points were compiled and used for this project.

Inputs:

- **Date**: Month and year.
- **CPI_actual**: The actual reported Consumer Price Index (CPI) for the current month.
- **Unemployment Rate**: The unemployment rate for the current month.
- **1-year T-Bill Rate**: The 1-year Treasury Bill interest rate for the current month.
- **10-year T-Bill Rate**: The 10-year Treasury Bill interest rate for the current month.
- **PMI_Actual**: The Purchasing Managers' Index (PMI) for the current month.
- **CPI_Forecast_of_this_month**: The forecasted CPI for the current month.
- **Asset_Return_this_month**: The month-on-month return of the asset class.
- **Asset_Vol_this_month**: The dispersion of Returns with in the month.
- **CPI_Forecast_next_month**: The forecasted CPI for the next month.
- **Target_Month**: The month being predicted.

Outputs (Predicted Variables):

- **Target_Return**: The return of the asset class for the next month.
- **Target_Vol**: The volatility of the asset class for the next month.

Data Preparation

The collected data underwent preprocessing to ensure consistency and usability. This phase involved two key steps:

- **Combining all indicators into a master dataset:** The dataset was structured to include both current and forward-looking indicators, ensuring that each data entry corresponded to the current month (or the month just ended) Table 3.1. The dataset includes:
  - Economic indicators available mid-month.
  - Month-end closing values.
  - Forecasted values for the upcoming month (target month for predictions).
- **Computing asset class returns and volatility:** Returns and volatility were calculated to provide meaningful inputs for the predictive models:
  - **Return Calculation:** The return for each asset class was computed as:

$$\text{Return} = \frac{\text{Closing Price} - \text{Opening Price}}{\text{Opening Price}} \tag{3.1}$$

    where the opening price refers to the first trading day of the month, and the closing price refers to the last trading day.

  - **Volatility Calculation:** Monthly volatility was computed as the standard deviation of daily returns within the month:

$$R_t = \frac{P_t - P_{t-1}}{P_{t-1}} \tag{3.2}$$

    where P_t is the price on day t and P_{t-1} is the previous day's price. The monthly volatility is then given by:

$$\sigma_{\text{monthly}} = \sqrt{\frac{1}{N}\sum_{t=1}^{N}(R_t - \bar{R})^2} \tag{3.3}$$

    where $\bar{R}$ is the mean daily return, and N is the number of trading days in the month.

Data Splitting Strategy

To evaluate model performance effectively, the dataset was split as follows:

- **Training Set (80%)** - Used for model learning.
- **Validation Set (20%)** - Used for validating the learning process.
- **Test Set** - Collected separately at the end of the project to prevent data leakage.

Test Data collection:

The final dataset was collected at the end of the project for two reasons:

1. **Maximizing training data:** Using all available historical data for training improved model performance.
2. **Preventing data leakage:** The test dataset was not available during training, ensuring an unbiased evaluation.

Table 3.1: Macroeconomic Indicators and Asset Class Performance Example

Date	CPI	Unemp. Rate	1Y T-Bill	10Y T-Bill	PMI	CPI Forecast	Asset Return	Asset Vol	CPI Forecast (Next)	Target Return	Target Vol	Target Month
Jan-1991	5.65	6.4	6.64	8.09	39.2	5.65	0.0535	0.0122	5.31	0.0700	0.0111	2.0
Feb-1991	5.31	6.6	6.27	7.85	39.4	5.31	0.0700	0.0111	4.82	0.0128	0.0084	3.0

3.1.2 Model Creation

Based on insights from the literature review and performance metric evaluations, this project selected two deep learning models for time series forecasting:

- **Long Short-Term Memory (LSTM):** Chosen for its ability to model sequential dependencies and long-term relationships in time series data.
- **Temporal Fusion Transformer (TFT):** Selected for its interpretability, multi-horizon forecasting capabilities, and ability to capture complex interactions between variables.

Hyperparameter Tuning

To optimize model performance, a hyperparameter tuning process was conducted on a subset of training data. The goal was to identify the optimal model configurations that maximize predictive accuracy while minimizing overfitting.

TFT Hyperparameters For the Temporal Fusion Transformer (TFT), the following hyperparameters were tuned:

- **Maximum Encoder Length** – Determines how much historical data the model considers.
- **Hidden Size** – Defines the number of neurons in each layer.
- **Attention Head Size** – Controls the number of attention heads used in the multi-head attention mechanism.
- **Dropout Rate** – Regulates the probability of randomly dropping units to prevent overfitting.
- **Learning Rate** – Governs the step size of updates in the optimization process.
- **LSTM Layers** – Sets the depth of stacked LSTM layers within the TFT model.
- **Hidden Continuous Size** – Specifies the dimension for continuous feature embeddings.

LSTM Hyperparameters For the Long Short-Term Memory (LSTM) model, the following hyperparameters were tuned:

- **Sequence Length** – Defines the number of time steps considered in each training instance.
- **Number of Layers** – Determines the number of stacked LSTM layers.
- **Hidden Size** – Specifies the number of units per LSTM layer.
- **Dropout Rate** – Controls regularization to mitigate overfitting.
- **Learning Rate** – Adjusts the optimizer's step size for updating weights.

3.1.3 Model Training

The models were trained using the preprocessed dataset, with each model configured to run for a maximum of 1000 epochs (i.e., complete passes through the training data). However, various functions were implemented to improve the model's training performance and robustness:

- **Loss Function:** Quantile loss was used to estimate prediction intervals, allowing the model to capture uncertainty in forecasts.
- **Optimization Strategy:** The Adam optimizer was employed, along with an adaptive learning rate schedule, to facilitate faster and more stable convergence.
- **Early Stopping:** Training was halted when the validation loss stopped improving, helping to prevent overfitting and reduce unnecessary computation.

In practice, most models did not reach the full 1000 epochs due to the implementation of **early stopping**, which terminated training once the validation loss plateaued.

3.1.4 Model Validation Phase

During this phase, the trained models were evaluated for their accuracy using the performance metrics outlined in Section 3.1.6. The superior models were identified based on their relative performance. Comparing LSTM versus TFT—and these top-performing models were then carried forward for use in the later stages, specifically in the construction of the theoretical portfolio.

3.1.5 Theoretical Portfolio

The test data was collected and processed in the same order as described in Section 3.1.1. This data was then loaded into the trained models based on the superior models identified from the validation phase metrics. These superior models were subsequently used to generate predictions, which were used to construct the optimal theoretical portfolio.

3.1.6 Final Evaluation

Following successful training, each model was evaluated based on its predictive performance, as well as its ability to support the construction of a robust theoretical portfolio. The evaluation was conducted according to two main objectives:

Objective 1: Model Comparison Using Validation Data

The predictive performance of the LSTM and TFT models was compared using the validation dataset. This approach was chosen to avoid overfitting to the test data and to ensure that the model selection process remained unbiased. By using the validation set for evaluation, the objective was to identify the model that generalizes best to unseen data. While comparisons on the test dataset are also provided later for additional insight, these results do not influence the model selection for portfolio construction. This methodology is aligned with standard practices in the academic literature on machine learning and time series forecasting.

The following evaluation metrics were used to assess and compare model performance:

- **Coefficient of Determination (R^2):** Indicates the proportion of variance in the actual values that is explained by the model predictions. A higher R^2 (closer to 1) suggests better model fit.
- **Mean Absolute Error (MAE):** Represents the average magnitude of errors between predicted and actual values. It is a straightforward indicator of overall prediction accuracy.
- **Mean Absolute Percentage Error (MAPE):** Expresses prediction accuracy as a percentage. It quantifies the average percentage deviation between predicted and actual values, making it useful for interpretability.
- **Mean Squared Error (MSE):** Measures the average of the squared prediction errors. It penalizes larger errors more severely than MAE.
- **Root Mean Squared Error (RMSE):** The square root of MSE, which maintains the same unit as the original data. Lower RMSE indicates better predictive performance.
- **Normalized Root Mean Squared Error (NRMSE):** RMSE normalized by the range of actual values. This allows for comparison across different scales and datasets.
- **Weighted Absolute Percentage Error (WAPE):** A percentage-based error metric that reduces instability when actual values are near zero. It is considered more robust than MAPE in such situations.
- **Weighted Mean Absolute Percentage Error (W_MAPE):** Often used interchangeably with WAPE, it weights each error based on the actual values and helps

avoid distortion from very small denominators.

- **Prediction Interval Coverage Rate (Coverage):** Measures the proportion of true values that fall within the predicted confidence interval (between lower and upper quantiles). A higher value indicates more reliable interval estimation.
- **Prediction Interval Width:** The average width of the prediction intervals. Narrower intervals are generally preferred, assuming adequate coverage is maintained.
- **Interval Score:** A comprehensive metric that balances interval width and coverage. It penalizes both excessively wide intervals and cases where actual values fall outside the prediction bounds, especially useful for evaluating probabilistic forecasts.

A detailed breakdown of these metrics, including their mathematical formulations, is provided in Appendix B

Objective 2: Portfolio Performance Evaluation

Using the superior model identified in Objective 1, a hypothetical investment portfolio was constructed based on its predicted returns. This portfolio was compared against a traditional 60/40 benchmark—allocated 60% to equities and 40% to bonds, rebalanced monthly (Portfolio Literacy n.d.).

Portfolio performance was assessed under four simplifying assumptions: no transaction costs, full monthly reinvestment, zero slippage or bid-ask spread, and perfect replication of index returns.

While financial indices are not directly investable, instruments like Exchange-Traded Funds (ETFs) allow investors to gain practical exposure by closely tracking index performance. Both portfolios were evaluated using four key financial metrics:

- **Total Return:** Net change in portfolio value over the period.
- **Sharpe Ratio:** Excess return per unit of volatility (Sharpe 1994).
- **Maximum Drawdown:** Largest observed peak-to-trough decline.
- **Annualized Return:** Average yearly return over the evaluation period.

Mathematical definitions of these metrics are provided in Appendix C.

3.2 Technical Pre-requisites

In order to be able to proceed with this project, it is important to understand the prerequisite system and hardware requirements needed to effectively run the state-of-the-art Temporal Fusion Transformer (TFT) model and the well-researched Long Short-Term Memory (LSTM) model. These requirements can be categorized into hardware and software aspects, along with the reasons why they were chosen or necessary adjustments that may be required.

3.2.1 Computational Considerations

The Temporal Fusion Transformer (TFT) and Long Short-Term Memory (LSTM) are deep learning models that use layered neural architectures to capture complex relationships between inputs and outputs. Due to their computational intensity, especially with larger datasets, they require significant processing power.

Training time depends heavily on model complexity and dataset size, making adequate computational resources essential. A review of relevant literature confirmed that both models benefit from high-performance CPUs and GPUs—particularly important given the time-sensitive nature of this project.

To address these needs, two primary computation methods were considered:

1. **Local Machine Hosting**
2. **Cloud Hosting**

To determine the best solution, an analysis of each option was conducted, considering key factors such as **cost, accessibility, GPU performance and storage capacity**.

Local Hosting

Local hosting refers to running the computation on a personal machine. Table 3.2 provides the key specifications of my system deemed important for this project. For full system specifications, please refer to Appendix D.1.

Table 3.2: CPU and GPU Specifications of Local Machine

Component	**Specification**
CPU	Intel64 Family 6 Model 141 Stepping 1 GenuineIntel 2304 MHz
GPU	NVIDIA GeForce RTX 3060
GPU Driver Version	560.94
CUDA Version	12.6
GPU Temperature	39°C
GPU Memory	0MiB / 6144MiB

Cloud Hosting

Cloud-based computation was examined for practical use in this project. Using the accessibility provided by Microsoft through their Azure platform, I explored the available specifications.

The allowable compute clusters were restricted due to the quota allocated for student accounts. Refer to Table 3.3 for the Azure Machine Learning virtual machine specifications.

Table 3.3: Azure Virtual Machine Pricing and Specifications

Name	**Category**	**Workload Types**
Standard_DS11_v2	Memory optimized	Development on Notebooks (or IDE) and lightweight testing
Standard_DS3_v2	General purpose	Classical ML model training on small datasets
Standard_E4ds_v4	**Memory optimized**	**Data manipulation and training on medium-sized datasets (1-10GB)**
Standard_F4s_v2	Compute optimized	Data manipulation and training on large datasets (>10GB)

Comparison of Local vs. Cloud Hosting

To evaluate both options, a direct comparison was made, considering hardware specifications, costs, scalability and accessibility.

Final Decision on Computational Approach

Based on the comparative analysis, it was determined that using the available local machine (Razer laptop) was the most practical and efficient option for this project (see Table 3.5).

Table 3.4: Comparison of Local System vs. Azure Cloud VM

Feature	**Local System (Razer Blade 15)**	**Azure VM (E4ds_v4)**
CPU	Intel64, 2304 MHz, 6 cores	4 vCPUs
RAM	32GB	32GB
GPU	NVIDIA RTX 3060, 6GB VRAM	No GPU
Storage	1TB SSD (local)	150GB SSD
Cost	Fixed (hardware already owned)	$0.35/hr
Accessibility	Always available (local)	Available when provisioned
Scalability	Limited to hardware specs	Can upgrade to more powerful VMs
Power Consumption	Uses local electricity	No local power usage
Internet Dependency	Not required	Required for access
Performance Variability	Consistent	Performance depends on VM load

While Azure provided scalability and remote accessibility, three key limitations made it less suitable:

- **Lack of GPU Access:** The student quota on Azure restricted access to GPU-enabled virtual machines, which are critical for deep learning models like TFT and LSTM.
- **Additional Cost:** Running models on Azure incurred hourly costs, whereas the local machine involved no additional expenses.
- **Internet Dependency:** Continuous cloud access required a stable internet connection, introducing a potential point of failure.

By contrast, the local system offered consistent performance with an onboard GPU (RTX 3060), ample RAM, and no hourly usage costs. Although a key limitation was the inability to run computations continuously due to resource constraints, this was effectively addressed in Chapter 4 through a stop-and-resume mechanism that improved computational efficiency and ensured training could be conducted incrementally.

Table 3.5: Pros and Cons of Local vs. Cloud Hosting for Training

Aspect	**Local System**	**Cloud Hosting (Azure)**
Pros	- No additional cost (hardware already purchased) - No internet required - Data stays on the device (better security) - Consistent performance	- Easily scalable to larger models - No local power consumption - Accessible from anywhere - Faster for large-scale training (if GPU enabled)
Cons	- Limited by local hardware specs - Consumes electricity - No instant upgrades	- Requires internet connection - Costs per hour of usage - Limited quota for student accounts

In conclusion, the local machine provided the most reliable and cost-effective solution given the resource limitations and time-sensitive nature of the project.

Chapter 4

Implementation and Experimentation

This chapter focuses on the implementation and execution of the code, outlining the steps taken to develop and train the final models for testing.

For this project, a total of 12 machine learning models were trained, utilizing both LSTM and TFT architectures. Each asset class had four models: two `TFT` models and two `LSTM` models, with each model predicting either `Target_Return` or `Target_Volatility`.

This section will not go through the implentation for each invidual model in detail, as they all follow a templated structure. To avoid repetition, the models will be discussed using *Asset_Class* as a general reference rather than specifying each individual asset class. The key differences across models lie in their hyperparameters, while the core structure remains consistent.

4.1 Underlying Architecture

A structured implementation is essential for effectively training predictive models. This section outlines the core setup of the Temporal Fusion Transformer (TFT) architectures and Long Short-Term Memory (LSTM), detailing how they are prepared for training. Rather than focusing on model optimization, the emphasis here is on defining the dataset structure, model configuration and training setup to ensure consistency across implementations.

Each model follows a systematic process consisting of:

- **Dataset Preparation:** Formatting raw data for time series forecasting.
- **Model Initialization:** Defining the required parameters and architecture.
- **Training Setup:** Configuring data loaders and executing the training process.

This standardized approach ensures a consistent and reproducible framework for training both models.

4.1.1 TFT

The TFT model training consists of four key parts: the initialization of the dataset, the initialization of the model, the setup of data loaders and the training configuration.

Dataset Preparation and Processing

The dataset preparation involves multiple steps, from loading raw data to structuring it in a format compatible with the `TimeSeriesDataSet` class. This process ensures that the data is correctly formatted and validated before training.
The first step is to load the raw data from a CSV file and perform preprocessing to ensure it is in the correct structure for training. As shown in Listing 4.1, the `load_data()` function is responsible for loading, formatting and validating the dataset.

The data originates from a master CSV file compiled earlier (Section 3.1.1) and is preprocessed into a usable format. In this stage, a series of validation checks and transformations are performed. The key steps include:

- Reading the CSV file and converting it into a pandas DataFrame.
- Ensuring that the `Date` column is in the correct format, containing only the year and month.
- Converting the `Target_Month` column into a string, as it is treated as a categorical variable rather than a numerical one.
- Assigning a sequential index to `time_idx` to ensure consecutive numbering without gaps.

- Adding a dummy `Group_ID` column, which is necessary for the dataset structure.

Listing 4.1: Python function for loading data

```python
def load_data():
    """
    Load training and validation data.
    """
    train_data = pd.read_csv("asset_class_CSV_file")

    # Convert 'Date' to datetime format
    train_data["Date"] = pd.to_datetime(train_data["Date"])
    train_data['Target_Month'] = train_data['Target_Month'].astype(str)

    # Extract year and month
    train_data["YearMonth"] = train_data["Date"].dt.to_period("M")

    # Remove duplicates
    train_data = train_data.drop_duplicates(subset=["YearMonth"]).
        reset_index(drop=True)

    # Recompute time_idx
    train_data["time_idx"] = range(len(train_data))
    train_data["Group_ID"] = "THIS␣ASSET"

    # Print statements for verification
    print(train_data[["Date", "time_idx"]])
    print(train_data["time_idx"].diff().value_counts())
```

Defining the TimeSeriesDataSet

A crucial step in preparing the data for model training is defining the `TimeSeriesDataSet`. This class structures the dataset in a format suitable for time series forecasting, ensuring that features, time indices and targets are correctly assigned. It enables the Temporal Fusion Transformer (TFT) model to effectively capture patterns and relationships within the data.

Listing 4.2 illustrates the dataset construction process, which serves as a template for training, testing, and validation. Once preprocessed, the data is structured into a format compatible with the `TimeSeriesDataSet` class, comprising 10 key components. The same structure is applied across all asset classes, with modifications to the raw data input as necessary.

- **time_idx**: Captures and recognizes the linear progression of time within the dataset.
- **target_variable**: Represents the feature being predicted (either `Target_Return` or `Target_Volatility`).
- **group_ids**: Uses the dummy placeholder ”Group_ID” to differentiate between different groups/assets. In this case, each asset class has its own CSV file.
- **max_encoder_length**: Defines the lookback window, which determines how many past time steps are used for prediction.
- **max_prediction_length**: Defines the forecast horizon. While TFT can handle multi-time horizon predictions, this project uses a fixed window of 1 month.
- **time_varying_known_reals**: Specifies the features that remain known throughout the time series and are available beforehand.
- **time_varying_unknown_reals**: Represents the features that the model does not know in advance and attempts to predict.
- **time_varying_known_categoricals**: Includes categorical features, such as `Target_Month`, which helps the model identify the month being predicted and potentially recognize any seasonal patterns.
- **target_normalizer**: The `GroupNormalizer()` normalizes the target variable based on the grouping of asset classes.
- **add_encoder_length**: Includes the encoder length as a feature, allowing the model to recognize the number of time steps used in the lookback window.

Listing 4.2: Defining Dataset for Training

```
# Define the training dataset
train_dataset = TimeSeriesDataSet(
    structured,
    time_idx="time_idx",  # Sequential time index
    target=target_variable,  # Target variable for regression
    group_ids=["Group_ID"],  # Grouping by asset class
    max_encoder_length=max_encoder_length,  # Lookback window
    max_prediction_length=max_decoder_length,  # Prediction window
    time_varying_known_reals=[
        "CPI_actual", "Unemployemnt␣Rate", "1-Year␣T-Bill␣Rate",
        "10-Year␣T-Bill␣Rate", "PMI_Actual", "CPI_Forecast_of_this_month",
        "Asset_Class_Return_this_month", "Asset_Class_Vol_this_month"
    ],
    time_varying_unknown_reals=[target_variable],  # Target variable to
       predict
    time_varying_known_categoricals=["Target_Month"],  # Categorical
       variables
    target_normalizer=GroupNormalizer(groups=["Group_ID"]),  #
       Normalization
    add_encoder_length=True,  # Include encoder length feature
)
```

TFT Model Initialization

The initialization of the Temporal Fusion Transformer (TFT) model allows us to specify various underlying parameters. Listing 4.3 provides the basic template for all TFT models. The way the parameters were selected will be discuss in the Section 4.2

- **hidden_size**: Number of hidden units per layer.
- **attention_head_size**: Number of attention heads in the multi-head attention mechanism.
- **lstm_layers**: Number of LSTM layers used in the encoder.
- **dropout**: Dropout rate applied to prevent overfitting.
- **hidden_continuous_size**: Number of hidden units for continuous input embeddings.

In addition to the architectural parameters, the loss function and optimizer were also initialized:

- **Loss**: Defines how the model's performance is measured. In this project, the `QuantileLoss` function with quantiles (0.05, 0.5, 0.95) was used, ensuring the model predicts return intervals within the 5th and 95th percentiles.
- **Optimizer**: An optimization algorithm provided by PyTorch Forecasting to update model parameters efficiently. This project employed the `Ranger` optimizer, which combines RAdam (Rectified Adam), LookAhead, and Gradient Centralization (GC), making it particularly well-suited for handling noisy and sparse gradients often encountered in time series forecasting (Wright 2020).

Listing 4.3: TFT Model Initialization

```python
# Initialize the TFT model
tft = TemporalFusionTransformer.from_dataset(
    TimeSeriesDataSet,
    hidden_size=hidden_size,
    attention_head_size=attention_head_size,
    lstm_layers=lstm_layers,
    dropout=dropout,
    hidden_continuous_size=hidden_continuous_size,
    loss=QuantileLoss(quantiles=[0.05, 0.5, 0.95]),
    optimizer="Ranger",
)
```

Trainer Initialization

A critical aspect of the training process is the initialization and configuration of the trainer. The trainer specifies how the model should be trained and plays a key role in ensuring efficient training execution. Listing 4.4 presents a basic trainer template, which was adjusted as necessary for the training and tuning phases. The following parameters are key components of the trainer configuration:

- **max_epochs**: Defines the maximum number of training epochs. An *epoch* represents a full pass over the training dataset. Each epoch allows the model to see all training samples once and update its weights accordingly. A higher number of epochs can improve learning but may lead to overfitting.

- **callbacks**: Includes functions such as `EarlyStopping`, which halts training before reaching `max_epochs` if a monitored metric (e.g., validation loss) stops improving. This prevents unnecessary computations and reduces overfitting.
- **accelerator**: Specifies the hardware used for training. Setting this to `"gpu"` enables training on an available GPU, significantly improving efficiency compared to CPU-based training.
- **devices**: Determines the number of devices (GPUs or CPUs) to use during training. Setting `devices=1` ensures that only a single GPU is utilized.
- **logger**: Integrates the `TensorBoardLogger`, a mechanism for tracking key training metrics such as loss values, epochs, and validation performance over time.

Listing 4.4: Trainer Initialization

```

# TensorBoard Logger for tracking training metrics
logger = TensorBoardLogger(save_dir="logs/", name="TFT_Training")

# Trainer setup
trainer = Trainer(
    max_epochs=x,  # Maximum number of training epochs
    callbacks=[early_stop_callback],  # Enable early stopping
    enable_checkpointing=False,  # Disable automatic checkpointing
    accelerator="gpu",  # Use GPU for training
    logger=logger,  # Log training progress
    devices=1,  # Use one GPU
)
```

Initializing Training

The final step in training a Temporal Fusion Transformer (TFT) model is to ”fit” the model to the training data using the trainer configuration. At this stage, the system integrates all configurations and initializations to begin the training process.
The `trainer.fit()` function requires three key arguments:

- **model**: The initialized TFT model that will be trained.
- **train_dataloaders**: A PyTorch DataLoader that manages the loading and batching of training data.
- **val_dataloaders**: A PyTorch DataLoader that loads validation data to monitor model performance.

Listing 4.5 illustrates how the time-series dataset is transformed into a DataLoader, which is then fed into the model for training and validation. The flag `train=True` in the DataLoader marks the dataset as training data, while `train=False` designates it as validation data.
Additionally, the **batch size** parameter defines how many samples are processed at a time during training. A batch size of 64 (e.g., `batch_size=64`) means that 64 samples are used for each forward and backward pass before updating the model's parameters.

Listing 4.5: Training Initialization

```
# DataLoaders for training and validation
train_dataloader = TimeSeriesData.to_dataloader(train=True, batch_size=64)
val_dataloader = TimeSeriesData.to_dataloader(train=False, batch_size=64)

# Train the model
trainer.fit(
    model, # The initialized TFT model
    train_dataloaders=train_dataloader, # Training data
    val_dataloaders=val_dataloader, # Validation data
)
```

4.1.2 LSTM

The LSTM model training consists of four key parts: the initialization of the dataset, the initialization of the model, the setup of data loaders and the training configuration.

Dataset Preparation and Processing

Due to the simpler nature of the LSTM models compared to TFT, the dataset preparation follows a slightly different approach. A key difference is that the categorical feature

Target_Month, which was included in the TFT models, is omitted in the LSTM models.

The raw data is first loaded from a CSV file and converted into a pandas DataFrame. The Date column is then transformed into the correct format, ensuring proper time indexing. Following this, feature scaling and reshaping are applied to standardize the input data for the LSTM model.

Since LSTMs are sensitive to varying scales across different features, normalization is performed using MinMaxScaler to transform all input features and target variables into a range between 0 and 1. This prevents features with larger numerical values from dominating the learning process and ensures stable model training. The input features and target variable are scaled separately to prevent data leakage, with the scaler fitted only on the training data and then applied to the validation data.

Additionally, the target variable must be reshaped to ensure compatibility with the LSTM model. Unlike traditional machine learning models, which accept one-dimensional target variables, LSTMs expect input data in a structured two-dimensional format. Therefore, the target variable is reshaped into a column vector before scaling, as shown in Listing 4.6.

Listing 4.6: Feature Scaling and Reshaping for LSTM

```
from sklearn.preprocessing import MinMaxScaler

# Initialize MinMaxScaler for features and target variables
scaler_features = MinMaxScaler(feature_range=(0, 1))
scaler_target = MinMaxScaler(feature_range=(0, 1))

# Normalize the input features
train_data[input_features] = scaler_features.fit_transform(train_data[
    input_features])
val_data[input_features] = scaler_features.transform(val_data[
    input_features])

# Normalize the target variable
train_data[target_column] = scaler_target.fit_transform(train_data[
    target_column].values.reshape(-1, 1))
val_data[target_column] = scaler_target.transform(val_data[target_column].
    values.reshape(-1, 1))
```

This preprocessing step ensures that all input variables are appropriately scaled and formatted for time series forecasting using the LSTM architecture.

Defining the TimeSeriesDataSet

Now that the data has been transformed into a suitable format, it must be converted into sequential data structures compatible with the LSTM model. Unlike standard machine learning models that treat each instance independently, LSTMs models are specifically designed to process and learn from sequences of data requiring inputs in the form of ordered sequences to capture temporal dependencies.

To achieve this, the dataset is divided into overlapping sequences based on a predefined lookback window, known as the sequence length. The `create_sequences` function takes in the formatted dataset along with the lookback period and transforms it into a series of numpy arrays, where each sequence consists of multiple timesteps of input features paired with a corresponding target value.

The implementation of this sequence transformation is shown in Listing 4.7, demonstrating how the raw dataset is structured into time-dependent sequences for model training.

Listing 4.7: Generating Sequential Data for LSTM

```
def create_sequences(input_data, target_data, seq_length):
    sequences = []
    labels = []
    for i in range(len(input_data) - seq_length):
        seq = input_data[i:i + seq_length]  # Sequence of features
        label = target_data[i + seq_length]  # Corresponding target value
        sequences.append(seq)
        labels.append(label)
    return np.array(sequences), np.array(labels)
```

Once the sequences are generated, they are converted into PyTorch tensors to ensure compatibility with the model. These tensor arrays are then combined into a `TensorDataset`, which is subsequently loaded into a `DataLoader` object. The `DataLoader` facilitates efficient batching and to preserve the sequential structure of the data, shuffling is explicitly set to `False` during training and validation.

Listing 4.8 demonstrates the process of preparing the dataset for training and validation, ensuring that the input sequences and target values are correctly structured before being fed into the model.

Listing 4.8: Preparing Data Loaders for Training

```python
# Convert input data to numpy arrays
train_input_data = train_data[input_features].values
train_target_data = train_data[target_column].values
val_input_data = val_data[input_features].values
val_target_data = val_data[target_column].values

# Generate sequences
X_train, y_train = create_sequences(train_input_data, train_target_data,
    seq_length)
X_val, y_val = create_sequences(val_input_data, val_target_data, seq_length
    )

# Convert to PyTorch tensors
X_train = torch.tensor(X_train, dtype=torch.float32)
y_train = torch.tensor(y_train, dtype=torch.float32)
X_val = torch.tensor(X_val, dtype=torch.float32)
y_val = torch.tensor(y_val, dtype=torch.float32)

# Create TensorDataset and DataLoader
train_dataset = TensorDataset(X_train, y_train)
val_dataset = TensorDataset(X_val, y_val)

train_loader = DataLoader(train_dataset, batch_size=64, shuffle=False)
val_loader = DataLoader(val_dataset, batch_size=64, shuffle=False)
```

This process ensures that the dataset is structured appropriately for sequential learning, allowing the LSTM model to effectively capture temporal patterns in the data.

LSTM Model Initialization

The initialization of the Long Short-Term Memory (LSTM) model allows us to specify various underlying architecture features. The base class of the model provided by Beitler (2020) had to be modified and wrapped in a Lightning module to incorporate essential components critical to this project.

The modified "LSTM" model takes in five predefined parameters:

- **input_size:** Specifies the lookback window and the number of features that need to be observed for the model to make predictions.
- **hidden_size:** Defines the number of hidden units per layer, which determines the model's capacity to learn complex patterns in the data.
- **num_layers:** Specifies the number of LSTM layers used in the model, affecting its depth and ability to capture long-term dependencies.
- **dropout:** The dropout rate applied to prevent overfitting by randomly deactivating neurons during training.
- **learning_rate:** Controls the step size during model optimization, influencing how quickly the model converges to an optimal solution.

The modified `LSTMModel` class extends the standard LSTM architecture to predict quantile ranges instead of point estimates. The primary modification involves adjusting the output layer and implementing a custom loss function to accommodate quantile regression. The base class provided by Beitler (2020) was extended and wrapped in a PyTorch Lightning module to facilitate training, evaluation and optimization.

Quantile-Specific Output Layer Unlike a standard LSTM model that predicts a single point estimate, the modified `LSTMModel` produces multiple outputs per sample, each corresponding to a predefined quantile (e.g., 0.05, 0.5 and 0.95). This is achieved by:

- Replacing the final dense layer with a fully connected layer (`fc`) that outputs `len(quantiles)` values, where each output represents a different quantile estimate.
- Allowing the model to estimate prediction intervals by learning separate outputs for different quantiles, rather than a single mean value.

Pinball Loss for Quantile Regression To train the model effectively, a pinball loss function (also known as quantile loss) is implemented in the `quantile_loss()` function. The pinball loss function penalizes over- and under-predictions asymmetrically based on the quantile level, ensuring that the model learns to estimate different quantile intervals correctly.

The implementation of the loss function is detailed in Listing 4.9. This function is critical in enabling the model to predict quantile ranges by optimizing for asymmetric quantile regression loss, commonly known as pinball loss. The key advantages of this loss function include:

- Asymmetric Penalties: The function applies different penalties for over- and under-predictions, ensuring that each quantile prediction appropriately represents the tail behavior of the distribution.
- Multi-Quantile Prediction: By expanding the target variable across multiple quantiles, the model is trained to produce a range of potential values rather than a single point estimate.
- Uncertainty Estimation: The model learns to estimate the distribution of possible outcomes, allowing for the construction of confidence intervals rather than relying on a single deterministic forecast.

By leveraging this loss function, the model generates prediction intervals, where the spread between lower and upper quantiles (e.g., the 5th and 95th percentile) defines a confidence range for the expected return. The function iterates over all specified quantiles, computing the corresponding pinball loss for each. These losses are then averaged to ensure balanced optimization across multiple quantiles. This approach enables the model to predict a full quantile distribution, enhancing its ability to quantify uncertainty and assess risk in financial forecasting.

Listing 4.9: Implementation of the Quantile Loss Function

```python
def quantile_loss(self, predictions, targets):
    """
    Pinball loss for quantile regression.
    Computes the loss separately for each quantile and aggregates them.
    """
    # Ensure targets have shape [batch_size, 1]
    if targets.dim() == 1:
        targets = targets.unsqueeze(-1)  # Convert to 2D tensor if
            necessary

    # Expand targets to match the shape of predictions
    targets = targets.expand_as(predictions)  # Shape: [batch_size,
        num_quantiles]

    losses = []
    for i, q in enumerate(self.quantiles):
        errors = targets[:, i] - predictions[:, i]
        losses.append(torch.max((q - 1) * errors, q * errors).mean())

    return torch.stack(losses).mean()
```

Trainer Initiliaztion

The training initialization follows the same principles used for the Temporal Fusion Transformer (TFT) model (see Section 4.1.1), where it specifies how the model is trained and plays a crucial role in ensuring efficient optimization. Building upon the ideas established in the TFT trainer initialization, adjustments were made to accommodate the training and tuning phases of the LSTM model.

A key difference in the LSTM training setup (Listing 4.10) is that it had to be wrapped within a Lightning module, ensuring compatibility with the custom LSTM model. Despite this, the fundamental functionality remains unchanged, allowing for structured training, automated monitoring, and efficient logging.

Listing 4.10: LSTM Trainer Initialization

```
# Initialize PyTorch Lightning Trainer
trainer = pl.Trainer(
    max_epochs=1000,
    accelerator="gpu" if torch.cuda.is_available() else "cpu",
    enable_progress_bar=True,
    logger=pl.loggers.TensorBoardLogger(logger_path, name=model_name),
    callbacks=[EarlyStopping(monitor="val_loss", patience=60)],  # Monitor
        validation loss
)
```

Initializing Training

Following the same Initializing Training implementation described in Section 4.1.1, the `trainer.fit()` function integrates the dataset, model architecture and training specifications to execute the training process.

4.2 Parameter Tuning

A critical aspect of developing machine learning models is defining the underlying architecture, as there is a strong relationship between the model's configuration and its overall performance. However, it is important to note that a more complex model does not necessarily lead to better efficiency. To address this challenge and establish an empirical basis for selecting the optimal architecture and hyperparameters, a series of trials were conducted using the Optuna library, (Akiba et al. 2019).

The Tree-structured Parzen Estimator (TPE), which leverages Bayesian optimization, was used to suggest optimal hyperparameter values based on the results of past trials. In this case, the objective of the trials was to minimize the validation loss (`val_loss`) of the models.

Beyond the advantages offered by Optuna's Bayesian optimization framework, a notable benefit is its study storage functionality. This feature allows previously evaluated hyperparameter configurations to be stored and retrieved, enabling the user to pause and resume optimization trials without losing prior results. Such capability addresses the limitations associated with local machine-based experimentation, as outlined in Section 3.2.1, and illustrated in Figure 4.11. To maintain an organized and efficient optimization process, each study was assigned a unique name corresponding to the specific asset class and target variable under consideration.

By leveraging persistent study storage, Optuna enables efficient trial management, reducing

Listing 4.11: Optuna Study Storage for Hyperparameter Tuning

```python
# Define the storage path for the Optuna study
study_storage_path = "sqlite:///optuna_study.db"

try:
    # Try to load an existing study
    study = optuna.load_study(
        study_name="optuna_study",
        storage=study_storage_path,
    )
    print("Study␣loaded␣successfully.")
except KeyError:
    # If the study does not exist, create a new one
    study = optuna.create_study(
        study_name="optuna_study",
        direction="minimize",
        pruner=optuna.pruners.MedianPruner(),
        storage=study_storage_path,
    )
    print("New␣study␣created.")
```

redundant computations and facilitating structured hyperparameter tuning.

4.2.1 TFT Parameter Tuning

The `objective()` function defines the process of hyperparameter tuning, specifying the target optimization metric. In this case, it was used to optimize the hyperparameters of the Temporal Fusion Transformer (TFT) model for each asset class and for both return and volatility predictions.

Most of the function is dedicated to initializing a TFT model based on the suggested parameters. The model initialization follows the same principles described in Section 4.1.1, with a slight adjustment in the number of training epochs. Instead of fully training the model, the maximum number of epochs was set to 60, as shown in Listing 4.4.

To enhance training efficiency, the model includes early stopping, which halts training if the validation loss plateaus. Additionally, the Optuna framework prunes trials where:

- The validation loss has plateaued and is no longer decreasing.
- The trial shows no improvement over previous guesses in the search space.

This pruning mechanism ensures that computational resources are allocated efficiently, allowing the optimization process to focus on the most promising hyperparameter configurations.

This decision was made because the goal of the trial is not to train a fully optimized model but rather to obtain an estimate of the best hyperparameters The model must still undergo some learning phases to evaluate the effectiveness of different parameter configurations.

Suggested Hyperparameters

While Optuna allows for automated trials, it requires the user to define search ranges for each hyperparameter. The number of trials was set to 1000 for each target variable and asset class.

The suggested parameter ranges, as shown in Listing 4.12, were chosen based on:

- Models analyzed in the literature Review (Section 2).
- Hardware constraints of the available computing resources.
- **Max encoder length**: The minimum was set to 5, excluding shorter lookback windows (1–4) based on Ross (2023), which found them less effective at capturing temporal dependencies. The upper bound was chosen based on prior practical experience, allowing efficient exploration of meaningful longer horizons.

Listing 4.12: Hyperparameter Search Space for TFT

```python
# Define hyperparameter search space
max_encoder_length = trial.suggest_int("max_encoder_length", 5, 24)
max_decoder_length = 1  # Fixed for forecasting the next step
hidden_size = trial.suggest_int("hidden_size", 8, 1500)
attention_head_size = trial.suggest_int("attention_head_size", 1, 1500)
dropout = trial.suggest_float("dropout", 1e-6, 0.5)
learning_rate = trial.suggest_loguniform("learning_rate", 1e-4, 5e-3)
lstm_layers = trial.suggest_int("lstm_layers", 1, 20)
hidden_continuous_size = trial.suggest_int("hidden_continuous_size", 8,
    5000)
```

Optimized Parameters

The potential volume of TensorBoard log files generated during the TFT optimization process could reach approximately 360,000 entries. This estimate is derived as follows:

$$360,000 = 1,000 \times 60 \times 6 \tag{4.1}$$

where:

- 1,000 trials were conducted,
- Each trial included up to 60 epochs,
- A total of 6 separate models were optimized.

Given the sheer scale of log data, manual analysis would be impractical. To address this, a custom function, `tensor_search()`, was developed to automate the extraction of key training metrics from the TensorBoard logs.

Functionality of `tensor_search()` The function assumes each subdirectory corresponds to a unique Optuna trial. It systematically iterates through these directories to identify the trial that achieved the lowest validation loss (`val_loss`). Once identified, the associated set of optimized hyperparameters is extracted for use in final model training and evaluation.

Suggested Optimized Parameters After conducting a series of trials and analyzing the outcomes using the `tensor_search()` function (see Listing E.1), the following optimized hyperparameters were identified (refer to Table 4.1 for details). A breakdown of the individual model architecture components can be found in Appendix H.1.

Table 4.1: Suggested Optimized Parameters for the TFT Model (Trainable parameters are expressed in millions)

Model	**Max Enc. Len.**	**Hidden Size**	**Attn. Head**	**Dropout**	**LSTM Layers**	**Hidden Cont. Size**	**Trainable Params (M)**
Bond Asset Class							
Target Return	7	1372	1334	0.1281	11	2952	568
Target Volatility	9	42	595	0.2649	1	2606	2.6
Equity Asset Class							
Target Return	11	1083	479	0.0006	1	188	66.1
Target Volatility	21	694	892	0.0203	1	169	27.1
Gold Asset Class							
Target Return	16	886	276	0.0464	1	233	48
Target Volatility	14	366	4177	0.0007	1	116	804

4.2.2 LSTM Parameter Tuning

Following the same function and structure outlined in Section 4.2.1, the LSTM hyperparameter tuning also uses an objective function that defines the optimization process.

Suggested Hyperparameters

While the core functionality of the code remains consistent with what was described in Section 4.2.1, the specific parameters being tuned differ and are presented in Listing 4.13. These parameters are chosen based on the relevant features outlined in Section 2.2.2. The selection of parameter ranges is informed by the following considerations:

- LSTM model configurations discussed in the literature review (Section 2).
- Hardware constraints of the available computing environment.

- Limitation of the look-back period, with a maximum of 80 time steps available in the validation set.

Listing 4.13: Hyperparameter Search Space for LSTM

```python
# Define hyperparameter search space
seq_length = trial.suggest_int("seq_length", 1, 80)
num_layers = trial.suggest_int("num_layers", 1, 10)
hidden_size = trial.suggest_int("hidden_size", 32, 1000)
dropout = trial.suggest_float("dropout", 0.1, 0.5)
learning_rate = trial.suggest_float("learning_rate", 1e-5, 1e-2, log=True)
```

Optimized Parameters

The total volume of TensorBoard log files generated during the LSTM trials amounted to approximately 360,000 logs. The same `tensor_search()` function, as shown in Listing E.1, was used to extract the best-performing hyperparameter configurations, following the estimation approach outlined in Section 4.2.1.

Suggested Optimized Parameters After running a series of trials and analyzing the results using the `tensor_search()` function (Listing E.1), the following optimized hyperparameters were identified (see Table 4.2). A detailed overview of the LSTM model architecture is provided in Appendix H.2.

4.3 Training

Following the completion of the tuning phase, the defined parameters for each model were set, and the training process was initiated for the respective model types TFT (see Section 4.1.1) and LSTM (see Section 4.1.2). After initialization, the training process was conducted with three key adjustments:

- The number of epochs (training length) was set to 1000, as previously highlighted in Section 4.1.1. However, an **Early Stopping** function was implemented to halt training if the model's performance plateaued, preventing unnecessary computations.
- **Checkpointing was enabled** during this phase for several key reasons:

Table 4.2: Suggested Optimized Parameters for the LSTM Model (Trainable parameters are expressed in millions)

Model	**Seq Length**	**Hidden Size**	**Num Layers**	**Dropout**	**Learning Rate**	**Trainable Params (M)**
Bond Asset Class						
Target Return	80	130	8	0.4638	1.59×10^{-5}	1.0
Target Volatility	74	534	1	0.4086	3.44×10^{-3}	1.2
Equity Asset Class						
Target Return	80	122	7	0.4920	1.56×10^{-5}	0.785
Target Volatility	80	534	1	0.4920	1.56×10^{-5}	1.2
Gold Asset Class						
Target Return	80	106	7	0.4561	3.85×10^{-5}	0.593
Target Volatility	80	47	2	0.2061	1.89×10^{-5}	0.0289

1. **Mitigating risk of interruption:** With a training cycle extending up to 1000 epochs, there was a heightened risk of system instability, power loss or other unexpected interruptions that could halt the process mid-way. Checkpointing ensured that the training progress could be preserved at regular intervals.
2. **Local machine constraints:** Since the training was conducted on a local machine rather than a dedicated server or cloud instance, it was important to have the flexibility to pause and resume training as needed. Checkpointing enabled this by saving the model's current state, optimizer parameters, and training metrics at specific checkpoints.

This approach allowed for seamless resumption of training from the most recent checkpoint without losing any progress or data. In practice, this meant that even if the training was stopped manually or interrupted unexpectedly, it could be restarted from the saved checkpoint eliminating the need to retrain from scratch and ensuring computational resources were used efficiently.

- A new functionality, called **Learning Rate Monitor**, was introduced. This feature, available in the PyTorch Lightning library, dynamically reduces the learning rate when training begins to plateau, allowing for quicker convergence of the model.

4.3.1 Saving the Trained Model

Upon completion of model training whether by reaching the full 1000 epochs or being halted by early stopping the trained model was saved to the working directory using the PyTorch function:

Listing 4.14: Saving the trained model in PyTorch

```
torch.save(tft.state_dict(), save_path)
```

This command saves the model's **state dictionary** (i.e., learned weights and biases) in a `.pt` file. The `state_dict()` function in PyTorch extracts all trainable parameters of the model, which can then be reloaded into an identical model architecture for inference or further fine-tuning.

Saving in this format provides several advantages:

- **Compact and Efficient Storage**: The `.pt` file stores only the model parameters rather than the entire model architecture, making it lightweight.
- **Flexibility in Deployment**: The saved model can be loaded into different scripts or environments without needing to redefine the full model.
- **Resume Training or Fine-Tuning**: By reloading the `state_dict()`, training can be resumed from a saved checkpoint or the model can be fine-tuned with new data.

4.4 Loading Trained Models

After training, each respective model was saved as a `.pt` file. To use the model for predictions, the original architecture had to be reconstructed, and the saved weights from the `.pt` file were then loaded. This approach effectively recreated the trained model, ensuring that all learned parameters were accurately restored. As a result, the model could be reliably used for evaluation, inference, or deployment without the need for retraining.

4.4.1 Loading the TFT Model

To reload the trained Temporal Fusion Transformer (TFT) model, the same architecture from the training phase was reinitialized, and the model weights were subsequently loaded from the saved `.pt` file. This restored the model to its trained state and enabled it to make predictions on new data without further training. In order to load the model correctly, a new instance of the TFT model must first be initialized (4.1.1) using the exact same parameters as those used during training. This ensures compatibility between the architecture and the stored weights in the `.pt` file.

4.4.2 Loading the LSTM Model

To reload a trained Long Short-Term Memory (LSTM) model, a new instance must be initialized with the same architecture and hyperparameters as during training (see Section 4.1.2). The saved weights are then loaded from the `.pt` file. The same feature and target scalers used during training are also reloaded and applied to the validation data to ensure consistent scaling. These scalers were originally fit on the training data only, to avoid data leakage. Validation sequences are constructed using a rolling window with the same sequence length (80) as used during training. The model is then set to evaluation mode using `model.eval()` to disable dropout and gradient updates (see Listing 4.15).

Listing 4.15: Function to Load LSTM model and set to eval Mode

```

# Drop unused column
train_data = train_data.drop(columns=["Direction"])

# Fit scalers on training data only
scaler_features.fit(train_data[input_features])
scaler_target.fit(train_data[target_column].values.reshape(-1, 1))

# Apply scaling to validation data
```

```
val_data[input_features] = scaler_features.transform(val_data[
    input_features])
val_data[target_column] = scaler_target.transform(val_data[target_column].
    values.reshape(-1, 1))

# Prepare sequences
seq_length = 80
val_input_data = val_data[input_features].values
val_target_data = val_data[target_column].values
X_val, y_val = create_sequences(val_input_data, val_target_data, seq_length
    )

# Convert to tensors and DataLoader
X_val = torch.tensor(X_val, dtype=torch.float32)
y_val = torch.tensor(y_val, dtype=torch.float32)
val_dataset = TensorDataset(X_val, y_val)
val_loader = DataLoader(val_dataset, batch_size=64, shuffle=False)

# Load the trained LSTM model
model = LSTMModel(input_size=len(input_features), hidden_size=130,
                  num_layers=8, dropout=0.46378)
model.load_state_dict(torch.load("Asset_class_Target.pt"))
model.eval()
```

4.5 Creating Predictions

Once the models have been reloaded and initialized, they are ready to process new data and generate predicted outputs. At this stage, the models can be configured to accept unseen input data and produce corresponding forecasts based on the patterns they learned during training.

TFT Model Predictions

Following the loading of the trained Temporal Fusion Transformer (TFT) model, as described in Section 4.4.1, any new data must be formatted identically to the training data. This includes maintaining the same structure, with only the numerical values changing, and ensuring the reference time series remains consistent.

To achieve this, the dataset is prepared using the same preprocessing pipeline described in Section 4.1.1, which converts the input into a `TimeSeriesDataset` object, as outlined in Section 4.2.1. Once properly formatted, the model generates predictions using a rolling window approach that looks back only the required amount defined by each model's look-back period to make single-period forecasts (Appendix E.3).

In this setup, predictions are made sequentially for each month by using the most recent available data, simulating how the model would operate in a real-world, forward-looking setting. This method ensures the model is always making predictions based on an updated context, without needing to retrain.

LSTM Model Predictions

After loading the trained Long Short-Term Memory (LSTM) model, as detailed in Section 4.4.2, the new input data must be prepared in the exact same format as the training data. This involves using identical feature columns, applying the same previously fitted scalers and generating sequences using the approach described in Section 4.1.2. The logic for sequence creation is implemented in Listing 4.7 and further explained in Section 4.1.2.

Once the model has been correctly reloaded and initialized then set to evaluation mode via `model.eval()`, predictions can be generated. This process involves iterating through the validation dataset using a PyTorch `DataLoader`, which batches the data and feeds it to the model in mini-batches. The input features (`X_batch`) are passed into the model, producing a tensor of predictions for each batch.

The predictions are collected, concatenated, and then inverse-transformed back to their original scale using the target scaler. Each prediction consists of three quantiles: the 5%, 50% (median), and 95% confidence bounds, as shown in Listing 4.16.

Listing 4.16: Function to create predictions in LSTM modell

```
all_predictions = []
all_actuals = []

with torch.no_grad():
    for batch in val_loader:
        X_batch, y_batch = batch
        predictions = model(X_batch)  # Shape: [batch_size, 3]
        all_predictions.append(predictions.cpu().numpy())
        all_actuals.append(y_batch.cpu().numpy())

# Combine all batch predictions and actuals
all_predictions = np.concatenate(all_predictions, axis=0)  # Shape: [N, 3]
all_actuals = np.concatenate(all_actuals, axis=0)          # Shape: [N]

# Inverse transform the predictions and actuals
p5 = scaler_target.inverse_transform(all_predictions[:, 0:1]).flatten()
p50 = scaler_target.inverse_transform(all_predictions[:, 1:2]).flatten()
p95 = scaler_target.inverse_transform(all_predictions[:, 2:3]).flatten()
all_actuals_rescaled = scaler_target.inverse_transform(all_actuals.reshape
    (-1, 1)).flatten()
```

4.6 Creating a Theoretical Investment Portfolio

Following the mathematical framework outlined in Section 2.3.4, a theoretical investment portfolio is constructed using the predictions from the most accurate forecasting model based on validation results. Specifically, the median predicted returns and volatilities are combined with the historical correlation matrix computed from asset class returns over the preceding 12 months to form the covariance matrix, which serves as a core input for the optimization procedure.

The covariance matrix is generated using the function `create_covariance_matrix()`, shown in Listing 4.17, by computing the outer product of forecasted volatilities and applying it element wise to the historical correlation matrix. This ensures that the final covariance matrix reflects forward-looking risk (volatility) metrics while preserving the structural relationships observed historically.

Optimization is then carried out using the `cvxopt` quadratic programming solver. The function `optimization()` takes in the adjusted expected returns, the covariance matrix and a specified risk tolerance level and solves for the optimal portfolio weights subject to standard constraints (fully invested and no short-selling). The function returns the optimal weights along with the expected return and risk (standard deviation) of the resulting portfolio.

Listing 4.17: Portfolio Optimization with Covariance Matrix

```python
def create_covariance_matrix(volatilities, correlations):
    vol_matrix = np.outer(volatilities, volatilities)
    covariance_matrix = correlations * vol_matrix
    return covariance_matrix

import cvxopt as opt
from cvxopt import solvers
solvers.options['show_progress'] = False

def optimization(expected_returns, covariance_matrix, risk_tolerance):
    n = len(expected_returns)
    P = opt.matrix(risk_tolerance * covariance_matrix)
    q = opt.matrix(-expected_returns)
    G = opt.matrix(-np.eye(n))
    h = opt.matrix(0.0, (n, 1))
    A = opt.matrix(1.0, (1, n))
    b = opt.matrix(1.0)
    solution = solvers.qp(P, q, G, h, A, b)
    optimal_weights = np.array(solution['x']).flatten()
    portfolio_return = np.dot(optimal_weights, expected_returns)
    portfolio_risk = np.sqrt(np.dot(optimal_weights.T, np.dot(
        covariance_matrix, optimal_weights)))
    return optimal_weights, portfolio_return, portfolio_risk
```

Chapter 5

Results

This chapter reflects on the results and outcomes of the project, evaluated against the Research objectives outlined earlier in Section 1.2.

5.1 Meeting the Project Objectives

Following the successful completion of the project and the generation of a series of final predictions, this section evaluates how each objective, Objective 1 and Objective 2, was achieved.

Objective 1 focused on analyzing and comparing the performance of two machine learning models (TFT and LSTM) across multiple asset classes using validation and test datasets. The analysis was supported by a comprehensive set of evaluation metrics, allowing for an evidence-based selection of the best-performing models for each asset class.

Objective 2 involved applying the selected models to construct a simulated investment portfolio spanning from August 2024 to February 2025. Although conducted retrospectively, this simulation demonstrated how model forecasts could be operationalized into a portfolio strategy, showing the practical application of the machine learning techniques explored in this project.

5.2 Objective 1: Model Comparison and Analysis

The first objective of this project is to analyze and compare the relative performance of the two selected machine learning models: the Temporal Fusion Transformer (TFT) and Long Short-Term Memory (LSTM). As outlined in the Final Evaluation Section 3.1.6, this analysis is conducted using two datasets: the validation dataset and the test dataset. The evaluation is separated into these two datasets to ensure that model selection for Objective 2 is not biased by test data performance. This preserves the integrity of the experiment and increases confidence in the generalizability of the chosen model. Refer to Appendix B for a detailed breakdown of the evaluation metrics used in this analysis. A visual representation of the model predictions versus the actual values is provided in Appendix I.

5.2.1 Validation Data Evaluation

The validation dataset spans from November 2017 to July 2024. This timeframe remains consistent across all models. However, it is important to note that historical data from 1990 was utilized to generate predictions. This is because different models required varying look-back periods. Including older historical data ensures that sufficient past context is available for accurate model initialization and prediction from November 2017 onward.

Using the performance metrics outlined in the final evaluation section, the predicted vs. actual values were compared.

Bond Accuracy: TFT vs. LSTM

Table 5.1 demonstrates that the Temporal Fusion Transformer (TFT) model outperforms the LSTM in predicting monthly bond returns across nearly all metrics. TFT achieves lower MAE, RMSE, Normalized RMSE, WAPE and W_MAPE, as well as a higher R^2 value. Although LSTM has a slightly better MAPE and narrower interval width, TFT records a lower interval score and higher coverage rate, suggesting it produces more reliable prediction intervals overall.

In contrast, Table 5.2 shows that TFT also performs better than LSTM in predicting bond volatility on most metrics. It achieves higher R^2, lower RMSE, Normalized RMSE, WAPE, and W_MAPE, along with a better interval score. However, LSTM achieves a better MAPE, higher coverage rate, and narrower interval width. This suggests that while TFT offers improved overall accuracy, LSTM may offer slightly tighter and more consistently covered prediction intervals for volatility forecasts.

Table 5.1: Return Prediction Performance: TFT vs. LSTM (Bond)

Metric	**TFT**	**LSTM**	**Better**
R^2	-0.0807	-0.1513	TFT
MAE	0.0121	0.0127	TFT
MAPE (%)	178.49%	168.06%	LSTM
MSE	0.0003	0.0003	Tie
RMSE	0.0160	0.0165	TFT
Normalized RMSE	0.2203	0.2265	TFT
WAPE (%)	102.83%	107.55%	TFT
W_MAPE (%)	102.83%	107.55%	TFT
Coverage Rate	0.8025	0.7317	TFT
Interval Width	0.0381	0.0314	LSTM
Interval Score	0.0582	0.0889	TFT

Table 5.2: Volatility Prediction Performance: TFT vs. LSTM (Bond)

Metric	**TFT**	**LSTM**	**Better**
R^2	0.5147	0.3819	TFT
MAE	0.0007	0.0007	Tie
MAPE (%)	24.67%	22.50%	LSTM
MSE	0.0000	0.0000	TFT
RMSE	0.0010	0.0012	TFT
Normalized RMSE	0.1391	0.1562	TFT
WAPE (%)	23.00%	23.97%	TFT
W_MAPE (%)	23.00%	23.97%	TFT
Coverage Rate	0.8148	0.8659	LSTM
Interval Width	0.0024	0.0020	LSTM
Interval Score	0.0041	0.0047	TFT

Equity Accuracy: TFT vs. LSTM

Based on the results in Table 5.3, the TFT model clearly outperforms LSTM in predicting equity returns. It achieves better results across all core evaluation metrics, including R^2, MAE, MAPE, MSE, RMSE, Normalized RMSE, WAPE, W_MAPE, interval width, and interval score. While LSTM achieves a slightly higher coverage rate, the overall predictive performance and reliability of TFT are stronger.

Similarly, Table 5.4 shows that TFT also outperforms LSTM in predicting equity volatility. TFT delivers better performance in 9 out of 11 metrics, including R^2, all error measures, and interval quality (interval width and score). LSTM again performs better in coverage rate and MAPE, but TFT provides more accurate and precise forecasts overall.

Table 5.3: Return Prediction Performance: TFT vs. LSTM (Equity)

Metric	**TFT**	**LSTM**	**Better**
R^2	0.1862	-0.077	TFT
MAE	0.0345	0.0386	TFT
MAPE (%)	111.75%	131.45%	TFT
MSE	0.0022	0.0029	TFT
RMSE	0.0471	0.0540	TFT
Normalized RMSE	0.1376	0.1576	TFT
WAPE (%)	82.38%	92.46%	TFT
W_MAPE (%)	82.38%	92.46%	TFT
Coverage Rate	0.8272	0.8902	LSTM
Interval Width	0.1229	0.1507	TFT
Interval Score	0.1937	0.2376	TFT

Table 5.4: Volatility Prediction Performance: TFT vs. LSTM (Equity)

Metric	**TFT**	**LSTM**	**Better**
R^2	0.3933	-0.3584	TFT
MAE	0.0040	0.0047	TFT
MAPE (%)	40.93%	38.76%	LSTM
MSE	0.0000	0.0001	TFT
RMSE	0.0055	0.0082	TFT
Normalized RMSE	0.0997	0.1483	TFT
WAPE (%)	38.44%	45.35%	TFT
W_MAPE (%)	38.44%	45.35%	TFT
Coverage Rate	0.6296	0.7561	LSTM
Interval Width	0.0112	0.0130	TFT
Interval Score	0.0308	0.0466	TFT

Gold Accuracy: TFT vs. LSTM

Table 5.5 shows that the LSTM model outperforms TFT across nearly all evaluation metrics for gold return prediction. LSTM achieves better results in R^2, MAE, MAPE, MSE, RMSE, Normalized RMSE, WAPE, and W_MAPE. It also significantly outperforms TFT in coverage rate (0.9756 vs. 0.7778), indicating more consistent uncertainty calibration. While TFT provides slightly narrower intervals and a marginally better interval score, these advantages are outweighed by LSTM's superior overall predictive accuracy.

Conversely, Table 5.6 highlights that TFT clearly outperforms LSTM in predicting gold volatility. TFT demonstrates stronger performance across nearly all metrics: R^2, MAE, MAPE, RMSE, Normalized RMSE, WAPE, W_MAPE, interval width, and interval score. Although LSTM has a higher coverage rate (0.9268 vs. 0.6667), TFT delivers more accurate and tighter predictive intervals, suggesting a more refined ability to model gold volatility dynamics.

Table 5.5: Return Prediction Performance: TFT vs. LSTM (Gold)

Metric	**TFT**	**LSTM**	**Better**
R^2	-0.2117	0.0062	LSTM
MAE	0.0311	0.0288	LSTM
MAPE (%)	196.94%	155.80%	LSTM
MSE	0.0016	0.0013	LSTM
RMSE	0.0399	0.0359	LSTM
Normalized RMSE	0.2196	0.1979	LSTM
WAPE (%)	109.54%	101.88%	LSTM
W_MAPE (%)	109.54%	101.88%	LSTM
Coverage Rate	0.7778	0.9756	LSTM
Interval Width	0.0917	0.1778	TFT
Interval Score	0.1693	0.1787	TFT

Table 5.6: Volatility Prediction Performance: TFT vs. LSTM (Gold)

Metric	**TFT**	**LSTM**	**Better**
R^2	0.203	-0.5694	TFT
MAE	0.0022	0.0035	TFT
MAPE (%)	24.68%	43.41%	TFT
MSE	0.0000	0.0000	Tie
RMSE	0.0030	0.0042	TFT
Normalized RMSE	0.1384	0.1931	TFT
WAPE (%)	24.53%	38.01%	TFT
W_MAPE (%)	24.53%	38.01%	TFT
Coverage Rate	0.6667	0.9268	LSTM
Interval Width	0.0075	0.0161	TFT
Interval Score	0.0178	0.0181	TFT

Validation data Best performing model per asset class and prediction

Based on the results and conclusions drawn in Section 5.2.1, the second objective of this project the construction of an investment portfolio will utilize a hybrid approach, incorporating both LSTM and TFT models. This decision is informed by the comparative evaluation of each model across key performance and interval based metrics, ensuring that both predictive accuracy and uncertainty quantification are adequately captured for the MVO framework.

The table below summarizes the best-performing models selected for each asset class and prediction task (return and volatility):

Table 5.7: Best Model Selection for Portfolio Construction

Asset Class	**Return Model**	**Volatility Model**
Bond	TFT	LSTM
Equity	TFT	TFT
Gold	LSTM	TFT

5.2.2 Test Data Evaluation

The test dataset covers the period from August 2024 to February 2025 and is applied consistently across all models. While the testing window is fixed, it is important to highlight that historical data dating back to 1990 for the reasons outlined earlier in Section 5.2.1.

Model performance was evaluated by comparing predicted values against actual outcomes using the performance metrics described in the final evaluation section.

Bond Accuracy: TFT vs. LSTM

Table 5.8 shows mixed results between TFT and LSTM for monthly bond return prediction. TFT achieved slightly better R^2, RMSE, normalized RMSE, interval score, and coverage, indicating stronger interval calibration. However, LSTM outperformed in point estimate metrics such as MAE, MAPE, WAPE, and WMAPE, and produced narrower prediction intervals. Overall, TFT offered better interval reliability, while LSTM delivered more accurate point forecasts.
In Table 5.9, LSTM clearly outperforms TFT across nearly all metrics for bond volatility prediction, including lower errors (MAE, MAPE, RMSE, nRMSE, WAPE, WMAPE), higher R^2, and tighter intervals. TFT achieved perfect coverage but at the cost of wider intervals and higher interval score. These results suggest LSTM is better suited for bond volatility forecasting, offering greater accuracy and efficiency.

Table 5.8: Return Prediction Performance: TFT vs. LSTM (Bond)

Metric	**TFT**	**LSTM**	**Better**
R^2	-0.1260	-0.1461	TFT
MAE	0.0158	0.0147	LSTM
MAPE (%)	103.87%	80.99%	LSTM
MSE	0.0004	0.0004	Tie
RMSE	0.0188	0.0190	TFT
Normalized RMSE	0.3845	0.3879	TFT
WAPE (%)	96.20%	89.24%	LSTM
WMAPE (%)	96.20%	89.24%	LSTM
Coverage Rate	0.8333	0.6667	TFT
Interval Width	0.0425	0.0322	LSTM
Interval Score	0.1041	0.1206	TFT

Table 5.9: Volatility Prediction Performance: TFT vs. LSTM (Bond)

Metric	**TFT**	**LSTM**	**Better**
R^2	-1.2738	-0.5283	LSTM
MAE	0.0006	0.0004	LSTM
MAPE (%)	19.79%	12.63%	LSTM
MSE	0.0000	0.0000	Tie
RMSE	0.0007	0.0005	LSTM
Normalized RMSE	0.5478	0.4491	LSTM
WAPE (%)	19.50%	13.85%	LSTM
WMAPE (%)	19.50%	13.85%	LSTM
Coverage Rate	1.0000	0.8333	TFT
Interval Width	0.0022	0.0018	LSTM
Interval Score	0.0022	0.0020	LSTM

Equity Accuracy: TFT vs. LSTM

Table 5.10 demonstrates that the Temporal Fusion Transformer (TFT) model significantly outperforms the LSTM model in predicting monthly equity returns across nearly all metrics. TFT achieves a notably higher R^2, lower MAE, MSE, RMSE, normalized RMSE, WAPE, WMAPE, and interval score. It also produces narrower prediction intervals, as indicated by the lower interval width. Although LSTM shows a marginally better MAPE, this is likely due to the high volatility of equity returns, which can distort percentage-based metrics. Overall, TFT delivers superior performance in both point forecasts and interval reliability for equity return prediction.

Table 5.11 similarly highlights the strong performance of the TFT model in forecasting equity volatility. TFT outperforms LSTM across most metrics, including R^2, MAPE, RMSE, normalized RMSE, WAPE, WMAPE, interval width, and interval score. MAE, MSE, and coverage rate are tied between the two models. The tighter intervals and improved accuracy of TFT indicate its robustness in capturing the dynamics of equity volatility, making it the more effective model for this task.

Table 5.10: Return Prediction Performance: TFT vs. LSTM (Equity)

Metric	**TFT**	**LSTM**	**Better**
R^2	0.3917	-0.1780	TFT
MAE	0.0186	0.0264	TFT
MAPE (%)	1123.64%	946.09%	LSTM
MSE	0.0005	0.0010	TFT
RMSE	0.0223	0.0310	TFT
Normalized RMSE	0.2773	0.3859	TFT
WAPE (%)	70.04%	99.62%	TFT
WMAPE (%)	70.04%	99.62%	TFT
Coverage Rate	1.0000	1.0000	Tie
Interval Width	0.0750	0.1524	TFT
Interval Score	0.0750	0.1524	TFT

Table 5.11: Volatility Prediction Performance: TFT vs. LSTM (Equity)

Metric	**TFT**	**LSTM**	**Better**
R^2	-0.4995	-1.0932	TFT
MAE	0.0008	0.0008	Tie
MAPE (%)	9.69%	10.89%	TFT
MSE	0.0000	0.0000	Tie
RMSE	0.0009	0.0011	TFT
Normalized RMSE	0.4389	0.5186	TFT
WAPE (%)	9.72%	10.16%	TFT
WMAPE (%)	9.72%	10.16%	TFT
Coverage Rate	1.0000	1.0000	Tie
Interval Width	0.0042	0.0110	TFT
Interval Score	0.0042	0.0110	TFT

Gold Accuracy: TFT vs. LSTM

Table 5.12 clearly indicates that the LSTM model outperforms the Temporal Fusion Transformer (TFT) in predicting monthly gold returns. LSTM achieves better performance across nearly all evaluation metrics, including R^2, MAE, MAPE, MSE, RMSE, normalized RMSE, WAPE, WMAPE, and coverage rate. This suggests that LSTM is more effective in capturing the return dynamics of gold during the test period. Although TFT records better results in interval width and interval score, indicating tighter and more accurate prediction intervals, the overall accuracy of its point estimates is inferior to those of the LSTM.

In contrast, Table 5.13 demonstrates that TFT provides superior performance in forecasting gold volatility. It outperforms LSTM in nearly all metrics, including R^2, MAE, MAPE, RMSE, normalized RMSE, WAPE, WMAPE, interval width, and interval score. The only exception is the coverage rate, where LSTM achieves full coverage (1.0000) compared to TFT's 0.6667. These results suggest that TFT offers more precise and efficient

Table 5.12: Return Prediction Performance: TFT vs. LSTM (Gold)

Metric	**TFT**	**LSTM**	**Better**
R^2	-0.8042	-0.1061	LSTM
MAE	0.0390	0.0303	LSTM
MAPE (%)	187.65%	155.89%	LSTM
MSE	0.0021	0.0013	LSTM
RMSE	0.0457	0.0358	LSTM
Normalized RMSE	0.5153	0.4035	LSTM
WAPE (%)	128.23%	99.73%	LSTM
WMAPE (%)	128.23%	99.73%	LSTM
Coverage Rate	0.6667	1.0000	LSTM
Interval Width	0.1129	0.1780	TFT
Interval Score	0.1333	0.1780	TFT

Table 5.13: Volatility Prediction Performance: TFT vs. LSTM (Gold)

Metric	**TFT**	**LSTM**	**Better**
R^2	-0.1167	-0.4785	TFT
MAE	0.0018	0.0026	TFT
MAPE (%)	16.19%	29.28%	TFT
MSE	0.0000	0.0000	Tie
RMSE	0.0024	0.0028	TFT
Normalized RMSE	0.3361	0.3868	TFT
WAPE (%)	18.51%	26.83%	TFT
WMAPE (%)	18.51%	26.83%	TFT
Coverage Rate	0.6667	1.0000	LSTM
Interval Width	0.0042	0.0165	TFT
Interval Score	0.0106	0.0165	TFT

Test Data: Best Performing Model per Asset Class and Prediction

Table 5.14 summarizes the top-performing models for each asset class based on return and volatility forecasts over the test period (August 2024–February 2025).
The Temporal Fusion Transformer (TFT) performed best overall, especially for equity return and volatility, where it consistently outperformed LSTM. For bond returns, results were mixed, with both models excelling in different metrics. However, LSTM outperformed TFT in bond volatility forecasting. Similarly, LSTM was superior for gold return prediction, while TFT showed better performance in gold volatility.
These results underscore the importance of selecting models based on the asset class and prediction type. While TFT excels in complex series like equities, LSTM remains a strong choice for assets with simpler temporal patterns such as bonds and gold.

Table 5.14: Best Performing Models by Asset Class

Asset Class	**Return Model**	**Volatility Model**
Bond	Tie (TFT/LSTM)	LSTM
Equity	TFT	TFT
Gold	LSTM	TFT

5.2.3 Comparison of Test and Validation Dataset Results

Tables 5.14 and 5.7 compare the best-performing models on the test set with those selected for portfolio construction.
In most cases, the selected models matched test outcomes. For equities and gold volatility, the Temporal Fusion Transformer (TFT) consistently delivered the strongest results and was selected accordingly.
The only difference was in bond return prediction. Although test results showed a tie between TFT and LSTM, validation data and performance consistency favored TFT, justifying its selection.
Overall, model choices for the portfolio were strongly aligned with empirical performance, with minor differences supported by validation results and practical modeling considerations such as stability and unified architecture.

5.3 Objective 2: Portfolio Construction Using Forecasted Models

As the validation dataset for this project spans up until July 2024, the proposed period for the simulated investment portfolio runs from August 2024 to February 2025. While the predictions were made retrospectively, the construction of the portfolio is based on the best-performing models identified during Objective 1 Validatation Data set evlaution (see Section 5.2).

These models were used to generate monthly return and volatility forecasts for the selected asset classes. The forecasts were then used to simulate portfolio performance over the specified period. The following prediction table (Table 5.15) summarizes the expected asset allocation weights over this forecasting window.

Based on these weights generated by the MVO function as described in Section 4.6, the system predicted the following recommended holdings, Table 5.15. To compare the relative performance against the traditional portfolio structure of a 60/40 split between equities and bonds, a theoretical simulation was conducted. The graph below (Figure 5.1) shows the theoretical change in value of a €100,000 portfolio.

The following assumptions were made:

- No transaction costs were incurred.
- The securities perfectly replicated the underlying asset class performance.

Following the construction and evaluation of both portfolios using the performance metrics outlined in Section 3.1.6, the MVO portfolio—driven by forecasted returns from the selected deep learning models—consistently outperformed the traditional 60/40 benchmark across all metrics.

Table 5.15 shows the monthly asset allocation weights used in the MVO portfolio, while Table 5.16 presents a direct comparison of portfolio performance. The MVO portfolio achieved a higher total and annualized return, a better Sharpe ratio, and a smaller maximum drawdown. Figure 5.1 illustrates the simulated portfolio value over time, further emphasizing the enhanced performance of the MVO strategy built on model-driven forecasts.

Period	**Bond Weight**	**Equity Weight**	**Gold Weight**
Feb 2025	0.4029	0.2981	0.2989
Jan 2025	0.4216	0.2825	0.2959
Dec 2024	0.3630	0.3060	0.3310
Nov 2024	0.2957	0.3982	0.3061
Oct 2024	0.3488	0.3570	0.2942
Sep 2024	0.3393	0.3713	0.2894
Aug 2024	0.3335	0.4090	0.2575

Table 5.15: Asset Allocation Weights Over Time

Portfolio	**Total Return**	**Sharpe Ratio**	**Max Drawdown**	**Annualized Return**
MVO Portfolio	13.00%	0.9028	-7.70%	23.33%
60/40 Portfolio	8.08%	0.5236	-9.12%	14.24%

Table 5.16: Performance Comparison Between MVO and Traditional 60/40 Portfolio

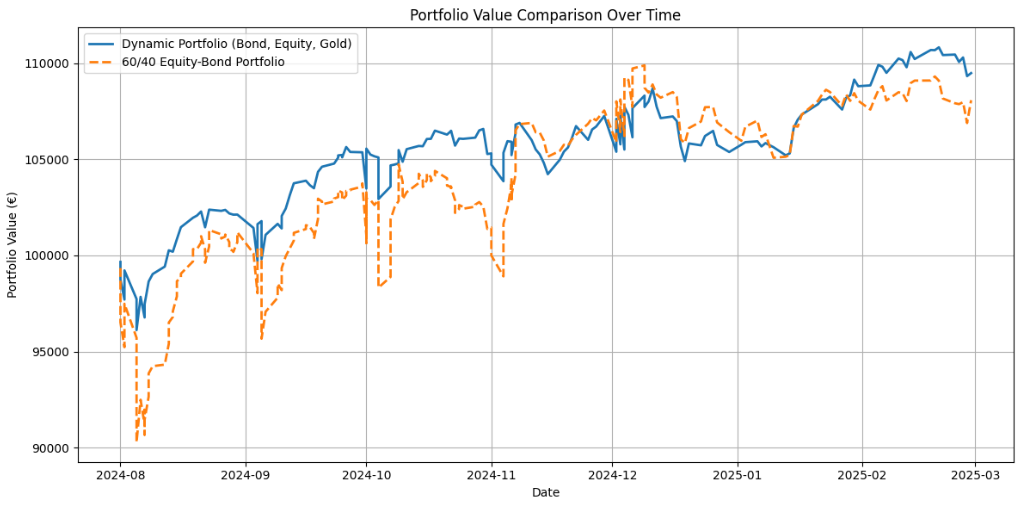

Figure 5.1: Simulated Portfolio Value Over Time Based on Forecasted Weights

Chapter 6

Conclusions and Discussion

This chapter presents a critical reflection on the results and outcomes of the project. It discusses the key insights gained from implementing and evaluating the forecasting models, and outlines the broader implications of the work. In addition, it highlights limitations, proposes directions for future research and considers ethical, environmental, and practical factors related to deploying predictive tools in financial contexts. The aim is to provide a balanced and holistic evaluation of the system's performance and real-world applicability, culminating in final remarks summarising the overall contribution and learning outcomes of the project.

6.1 Conclusion

Following the completion of this project, both LSTM and Temporal Fusion Transformer (TFT) machine learning models were implemented for time series forecasting. A detailed comparison of their relative performance was carried out using multiple evaluation metrics. While the results indicate that the TFT model generally outperformed the LSTM model particularly in terms of interval based accuracy it is important to contextualize the magnitude of this outperformance.

Although the TFT model demonstrated superior results, especially in capturing uncertainty through prediction intervals, the improvement in performance was relatively modest. Given the increased complexity of the TFT architecture and its significantly longer training time, the marginal gains in accuracy must be weighed against computational cost and model interpretability.

In evaluating the theoretical performance of the two constructed portfolios (MVO-based and traditional 60/40), two key conclusions can be drawn. First, machine learning models, even when limited to short-term (30-day) forecasting windows and subject to inherent prediction inaccuracies such as those highlighted by MAPE and WMAPE can still be meaningfully employed to guide asset allocation decisions. Second, the results illustrate that dynamic portfolio construction informed by predictive models can offer a viable alternative to static allocation strategies, such as the conventional 60/40 split.

6.2 Limitations and Future Work

Assumptions on the Underlying Input Features

An important drawback and underlying assumption in the use of predictive models is the relevance of the features selected for making predictions. In this project, all chosen input features were informed by and supported through academic literature, which highlights their historical importance. However, it is essential to acknowledge that correlation does not imply causation, a distinction that must be carefully considered. Marin (2022) demonstrates that even when features show strong historical correlations, this does not necessarily indicate a meaningful or causal relationship. This reinforces the need for a strong theoretical foundation when selecting input features for predictive modeling.

6.2.1 Limitations of the designed system

Time Horizon

The objective of this project was to develop a tool for investment portfolio management rather than short term trading, with a clear distinction between time frames typically used for trading versus investing. The focus was placed on longer time horizons, which aligns with the nature of portfolio construction and strategic asset allocation.

However, one of the inherent challenges of working with longer time periods such as the 30 day prediction horizon used in this model is the increased uncertainty introduced by the passage of time. From the moment a prediction is made to the point when the outcome materializes, there is a window during which unforeseen events or variables many of which are not accounted for in the model may influence market outcomes. This unpredictability must be acknowledged as a natural limitation of any forecasting tool operating over extended time frames.

6.2.2 Identified Underlying Flaws in the Project Models

This project significantly enhanced my understanding of model design, validation and tuning. However, several limitations were identified during the modeling process:

1. **Inconsistent Look-Back Windows:**
   The look-back window—the number of past observations used for prediction—was defined differently for each model. The TFT's window was based on literature and experience, while the LSTM's was restricted by validation set size. This led to different hyperparameter spaces, potentially impacting the fairness of performance comparisons.

2. **Limited Validation with Long Look-Backs:**
   Early training phases used a fixed data split. For LSTM models with long look-back windows, much of the data was consumed as context, reducing the number of validation predictions and possibly skewing performance assessments.

   Resolution: In the final evaluation, the full dataset was used to ensure both models generated predictions across the entire validation period, improving consistency and comparability.

6.2.3 Continuation of Research

This project has provided an empirical analysis of the relative performance of two machine learning models and serves as a strong foundation for further research in the field of data driven portfolio construction. Upon completion, two key extensions and adjustments can be considered to enhance or broaden the scope of this work beyond simply scaling it with more powerful computational methods.

Expanding Asset Classes or Refining Securities

One potential improvement is to split the broad "Equity" component into more refined categories, such as industry sectors or specific market segments. Academic research suggests that different types of companies perform differently across various phases of the economic cycle. Capturing these nuances by refining the classification of asset classes may improve the model's ability to detect relevant patterns and enhance predictive performance.

Inter-Prediction Period Active Management

A key limitation of this project is the fixed monthly prediction interval (approximately 30 days). In reality, markets are highly dynamic and numerous unforeseen variables such as geopolitical events, economic shocks or breaking news can affect asset performance within that time frame.

A valuable extension would be to introduce short-term responsiveness by incorporating real-time sentiment analysis or news data using Natural Language Processing (NLP). These insights could be used to make short-term adjustments to the base portfolio, which is constructed using the machine learning models developed in this project. This approach would allow for both strategic (monthly) and tactical (real-time) portfolio management.

6.2.4 Environmental Impact

While not a primary objective, the environmental impact of this project specifically energy usage during model training warrants consideration.
The total training and tuning time was approximately 1,565 hours (65 days). Power consumption came from both the local machine and an external cooling system used intermittently. Based on ambient temperature data (Figure 6.1), cooling was required for around 54% of the training period.
Assuming a power draw of 230 W for the laptop (Appendix D) and 40 W for the cooling system (Appendix G), total energy consumption is estimated at approximately 389 kWh. With Ireland's 2023 carbon intensity at 283 gCO_2/kWh Statista Research Department 2024, this equates to roughly 100 kg of CO_2 emissions.

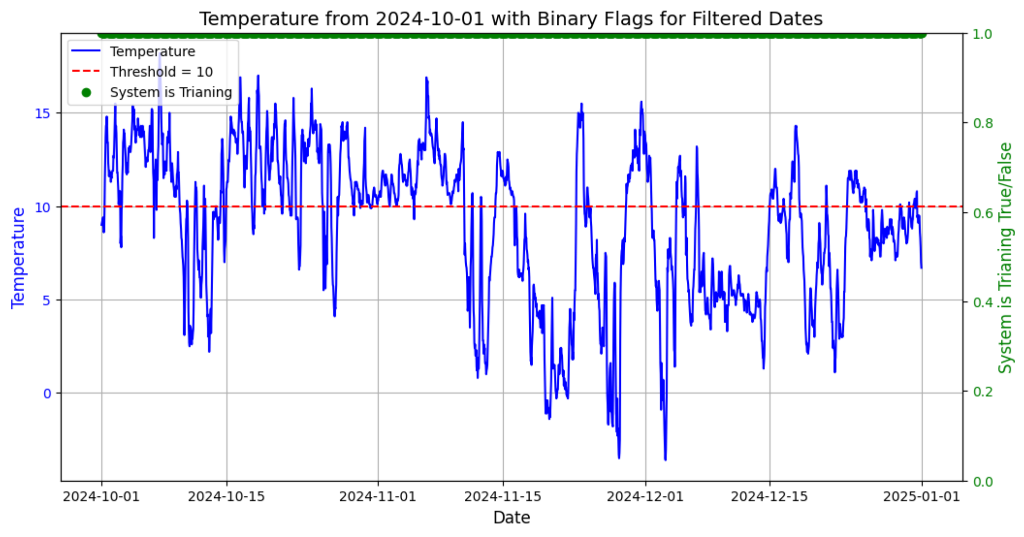

Figure 6.1: Historical Irish temperatures during training period. Source: Met Éireann (2024).

Although the models used had relatively low complexity (total parameters between [0.0289]–[568] million), this estimate highlights the environmental cost of scaling machine learning workloads—an important factor for future large-scale deployments.

6.3 Ethical and Practical Considerations

6.3.1 Risks Associated with Data

Given that the input data used in this project was obtained from financial sources, it is imperative to consider the potential issues and limitations associated with such data. The accuracy and reliability of the input data directly affect the usefulness and validity of the model's predictions. Although the data in this project was sourced from two highly credible providers both recognized as global leaders in financial market data, it remains important to acknowledge potential shortcomings. In the context of financial data, issues such as human error, data entry mistakes, or even deliberate manipulation and false accounting have been documented in the past. These risks, although minimized when using reputable sources, must still be considered in any analysis relying on financial datasets.

6.3.2 Ethical Use of Data and Predictions

Given the domain of these predictive models, it is imperative to acknowledge that their outputs can have tangible real world effects. Predictions related to buying or selling financial assets may influence a wide range of stakeholders beyond just the user of the model. In particular, the financial scale of an investment portfolio means that certain decisions such as the sale of an asset could place downward pressure on its price, resulting in financial repercussions for other market participants.

Furthermore, while the system was designed to improve the efficiency of portfolio management and forecasting, increased automation and efficiency could inadvertently reduce the demand for traditional portfolio analysts. This has potential employment implications, which must be considered when implementing such technologies at scale.

6.3.3 Systemic Bias

Within the scope of this project, there exists inherent systemic bias in the model, primarily due to its focus on the U.S. financial markets. While the rationale for selecting this domain was outlined earlier (see Section 2.3), it is important to recognize that the U.S. market is only one part of the global financial ecosystem. Although it provides transparency, liquidity and abundant data, this market represents just a fraction of the broader global context.

It is important to remain aware of the trade offs involved in focusing on a specific market, particularly when ignoring emerging or developing markets where these techniques could also be applied. Expanding to include other regions in future iterations could yield more inclusive, globally applicable insights.

6.3.4 Accessibility of This Technology

While at the beinging of this project outlined the goal of making this system more broadly accessible Section 1, it is worth further clarifying what that entails. The aim was to build a practical and actionable system that could be used by investment professionals particularly those in smaller firms and demonstrate the potential of machine learning in portfolio construction and prediction at a realistic, achievable scale.

Although such a system may not be directly applicable by the general public, due to limitations in financial knowledge and data interpretation skills, this project illustrates that smaller players in the financial field such as boutique investment managers or small pension funds could implement or adapt similar systems. This would help democratize access to advanced predictive tools, reducing the concentration of such technology within a select group of large institutions.

6.3.5 Use of Predictive Tools

It is important to highlight that the models developed during this project are purely predictive tools. To understand their use and application in any scenario, one must be aware of the limitations and potential issues associated with forecasting tools.

To Be Used in Conjunction with Other Tools

Beyond the relative accuracy or inaccuracy of the models, it is crucial to recognize that these tools are meant to support rather than replace more complex decision making processes. They should be used in conjunction with other analytical methods, expert judgment and domain specific knowledge to form a comprehensive decision making framework.

Feedback Loop

In any form of prediction, there exists a significant possibility of a feedback loop phenomenon. Essentially, even if the predictive forecasts are entirely incorrect, the predictions themselves can influence outcomes particularly in financial markets. Within the context of this project, if sufficient weight and financial backing are placed on the selected asset class based on these predictions, the effectiveness of the model can become skewed. This raises the possibility that incorrect predictions may, through market participation and investment flows, manifest as correct outcomes resulting in a self fulfilling prophecy.

6.4 Final Remarks

This project set out to explore the potential of machine learning for time-series forecasting in the context of portfolio management. By implementing and comparing Long Short-Term Memory (LSTM) networks and the Temporal Fusion Transformer (TFT), the research provided a data-driven assessment of how modern deep learning architectures can inform asset allocation decisions.

Throughout the process, significant insights were gained into the strengths and trade-offs of each model, the challenges of applying forecasting tools to real-world financial data and the importance of integrating these predictions within a broader investment strategy. While the TFT showed stronger performance in capturing uncertainty, the LSTM offered practical advantages in terms of efficiency and simplicity.

More broadly, this project demonstrated that even in resource-constrained settings, machine learning models can be meaningfully applied to financial problems when designed thoughtfully and evaluated rigorously. With continued development, these tools have the potential to enhance decision-making in asset management, particularly for smaller firms seeking to adopt scalable, intelligent forecasting systems.
Ultimately, this project has deepened the understanding of machine learning in financial contexts and laid the groundwork for future research in the intersection of Machine Learning and investment strategy.

Appendix A

Chosen Input Variables Tables

This appendix contains a summary of the referenced research papers used to justify the inclusion of various input features.

A.1 Justification for Choosing Input Features

A.1.1 Inflation Rate as an Input Feature

Inflation is a key macroeconomic indicator with direct implications for asset pricing and investor behavior. It affects purchasing power, corporate profitability, interest rate expectations and overall market sentiment. Its inclusion in forecasting models for asset returns and volatility is therefore well-supported in the literature.

Sathyanarayana and Gargesa (2018) conducted a multi-country study using monthly data from 2000 to 2017 and found a predominantly negative correlation between inflation and stock market returns. Countries such as Austria, Belgium and Canada exhibited statistically significant inverse relationships, confirming the classic view that higher inflation erodes real investment returns. Even in emerging markets like India, Johansen's cointegration test showed a long-run relationship between inflation and returns, suggesting inflation can systematically influence asset performance across economies.

In addition, inflation expectations often shape monetary policy. Central banks may respond to rising inflation by increasing interest rates, which in turn affects equity valuations, bond yields and investor risk appetite. As such, incorporating both actual and expected inflation into predictive models enhances the ability to anticipate market shifts and optimize portfolio decisions.

These findings justify the inclusion of inflation as a core input feature for forecasting asset returns and volatility in this project.

A.1.2 Unemployment Rate as an Input Feature

Table A.1: OLS Regression of Unemployment Surprise (SUER) on Market Volatility and Returns

Variable	**(1) CIV (Change in VIX)**	**(2) CIS (Change in SPY)**
SUER	0.0379***	-0.00491***
	(0.00859)	(0.00145)
MIR	0.00639***	-0.00113***
	(0.00220)	(0.000371)
MP	-0.000711	-5.12e-05
	(0.000666)	(0.000113)
Constant	0.0484	0.00724
	(0.0664)	(0.0112)
Observations	240	240
R-squared	0.107	0.079

Note: Robust standard errors in parentheses. *** $p < 0.01$. SUER = Surprise in Unemployment Rate. MIR = Monetary Interaction Rate. MP = Monetary Policy dummy. Adapted from Chi (2020).

Summary

This regression uses SPY and VIX as proxies for stock returns and market volatility, respectively. The SUER variable represents the surprise component of the unemployment rate, calculated as the deviation between actual and forecasted values. Chi's (2020) findings indicate a statistically significant positive effect on market volatility, with smaller but significant effects on stock returns. The analysis includes robustness checks with 30-minute intraday data and volume controls.

Further supporting evidence is provided by Gonzalo and Taamouti (2017), who analyze U.S. data from 1967 to 2015 using nonparametric Granger causality and quantile regression. Their findings distinguish between anticipated and unanticipated components of unemployment:

- **Anticipated Unemployment:** Granger-causes stock returns, particularly in the upper quantiles of the return distribution, implying stronger market reactions during bullish conditions.
- **Unanticipated Unemployment:** Shows limited causality, mostly confined to extreme return events.
- **Transmission Mechanism:** The effect of anticipated unemployment on returns is largely driven by expectations of monetary policy responses (e.g., interest rate cuts).

These findings strengthen the case for incorporating unemployment forecasts into financial return prediction models, especially when accounting for nonlinear effects and monetary policy channels.

A.1.3 Interest Rate as an Input Feature

Interest rates are a central component in macro-financial analysis and are widely considered a key determinant of equity market movements. To examine the relationship between interest rates and stock prices, regression analyses were performed using the Dhaka Stock Exchange (DSE) Index as the dependent variable.

Regression Results Using All Observations

Table A.2: OLS Regression Results Between Interest Rates and DSE Index (All Observations)

Equation	**Model**	**Coefficient**	**t-value**	**p-value**	**R^2**
Eq-2	Interest Rate	-19.847	-3.78	0.000	0.090
Eq-3	Growth of Interest Rate	4.708	2.02	0.045	0.027
Eq-4	Interest Rate (Δ Index)	-0.976	-0.98	0.328	0.006
Eq-5	Growth of Interest Rate (Δ Index)	-0.797	-1.97	0.051	0.026

The results show that while interest rate changes (Eq-5) are marginally significant at the 5% level, the explanatory power (R^2) remains low (2.6%), indicating limited overall predictive capacity. However, the negative coefficient suggests a potential inverse relationship between growth in interest rate and share price growth.

Regression Results Excluding Outliers

Table A.3: OLS Regression Results Between Interest Rates and DSE Index (Outliers Removed)

Equation	**Model**	**Coefficient**	**t-value**	**p-value**	$\mathbf{R^2}$
Eq-2	Interest Rate	-17.905	-7.18	0.000	0.271
Eq-3	Growth of Interest Rate	2.184	1.77	0.080	0.022
Eq-4	Interest Rate (Δ Index)	-0.822	-1.41	0.160	0.046
Eq-5	Growth of Interest Rate (Δ Index)	-0.615	-2.55	0.012	0.046

After excluding outliers, the significance of the relationship strengthens, particularly in Eq-2 and Eq-5 . In Eq-5 , the coefficient of -0.615 ($p = 0.012$) confirms a statistically significant negative relationship between growth of interest rate, growth of share price and the explanatory power improves to 4.6

Summary

The regression analyses above highlight that while the predictive power of interest rate variables remains modest (low R^2), the direction of the relationship is consistently negative, especially after removing outliers. This supports the inclusion of both the *interest rate* and its *monthly growth rate* as relevant features in modeling asset returns and volatility, consistent with the findings of Uddin and Alam (2010).

A.1.4 Stock Return Correlation Analysis

Understanding the persistence of stock return correlations is essential for portfolio construction and risk management. Ross (2023) investigates whether historical average pairwise stock return correlations can predict future correlations in a portfolio, particularly from the perspective of a risk-averse investor.

Table A.4: Prior Stock Return Correlation Predicting Future Correlation Ross 2023

Prior Return Correlation (m months)	**Coefficient (β_1)**	**T-Statistic**	**R^2**	**F-Statistic**
12	0.383	3.496	14.02%	149.14
24	0.376	2.819	10.84%	144.01
36	0.496	3.936	15.57%	151.23
48	0.548	4.415	17.46%	159.27
60	0.587	5.164	19.52%	179.75

Summary

The regression results in Table A.4 indicate that prior average stock return correlation is a strong and statistically significant predictor of future average correlation. The explanatory power (R^2) increases with longer historical windows, peaking at 60 months with an R^2 of 19.52% and a highly significant t-statistic.

These findings suggest that average historical excess return correlation contains useful information for forecasting future correlation, particularly in portfolios composed of small-cap stocks. Moreover, Ross shows that U.S. household equity ownership growth is negatively correlated with future average correlation. This macroeconomic factor, while relevant, does not eliminate the strong predictive power of prior correlations.

The implication for portfolio construction is clear: investors seeking to minimize idiosyncratic risk while maintaining reasonable exposure to systematic risk can benefit from incorporating historical return correlation into their risk models.

A.1.5 Inflation Expectations as an Input Feature

Inflation is widely regarded as one of the key macroeconomic indicators influencing investor sentiment and equity market volatility. Chiang Chiang (2023) provides robust empirical evidence across 20 major economies demonstrating a consistent negative relationship between expected inflation and real stock returns.

GED-APARCH Model Estimates

Using a Generalized Error Distribution–Asymmetric Power ARCH (GED-APARCH(1,1)) model, Chiang estimates the impact of domestic and U.S. expected inflation on real stock returns, controlling for major macroeconomic shocks such as the 2008–2009 global financial crisis and the COVID-19 pandemic.

Table A.5 summarizes key regression estimates, showing that expected inflation significantly and negatively affects real returns in most countries, with consistent evidence of volatility clustering and asymmetry in financial markets.

Table A.5: GED-APARCH(1,1) Estimates of U.S. Real Stock Returns

Variable	**C**	$\ln\sigma$	π_d	π_u	I_d	I_u	ω	$\lvert\epsilon\rvert$	ϵ	σ^I	R^2
rtUS (coefficients)	-6.808	3.000	-0.655	-4.765	-5.852	-2.920	9.088	0.100	-0.154	0.562	0.11
rtUS (z-statistics)	(-7.49)	(15.58)	(-4.44)	(-7.97)	(-26.58)	(-8.67)	(9.22)	(5.28)	(-5.32)	(6.59)	(17.88)

Note: This table is based on the period January 1990–June 2022. The lowercase r denotes the real stock return. Variables π_d and π_u denote domestic and U.S. expected inflation, respectively. UK inflation was used to estimate the U.S. market due to shared macroeconomic characteristics. The first row reports estimated coefficients; the second row reports z-statistics. The critical values for the t-distribution at the 1%, 5%, and 10% significance levels are 2.58, 1.96, and 1.65, respectively. R^2 is the adjusted coefficient of determination. Source: Chiang (2023).

Summary

Chiang's findings strongly support the hypothesis that expected inflation negatively impacts stock returns. Specifically:

- 18 out of 20 countries show a statistically significant negative correlation between expected inflation and real stock returns.
- The analysis controls for global crises using dummy variables, ensuring robustness against extreme observations.
- U.S. inflation and monetary policy uncertainty spill over to global markets, amplifying volatility and reducing returns.

- The GED-APARCH(1,1) model accounts for key features in financial time series such as volatility clustering, asymmetric effects, and fat tails.

A.1.6 PMI as an Input Feature

The Purchasing Managers' Index (PMI) is frequently used as a leading economic indicator, reflecting trends in business activity, production, and sentiment in the manufacturing sector. To evaluate its predictive power for financial markets, Özkan and Kiriş Özkan and Kiriş (2020) investigated whether PMI can be used as a leading indicator for Turkey's stock, bond, and foreign exchange markets.

Causality Analysis and Key Results

Using monthly data from December 2012 to August 2018 and applying both the standard Toda–Yamamoto causality test and the Fourier Toda–Yamamoto approach (which accounts for structural breaks), the study found:

- No causality from PMI to stock indices when structural breaks are ignored.
- One-way causality from PMI to the BIST-100 index and BIST-Metal Goods Sector index when structural breaks are considered.
- Additional evidence of causality from PMI to the bond and foreign exchange markets under structural breaks.

These results suggest that PMI becomes a statistically significant leading indicator for Turkish financial markets in the presence of macroeconomic shocks or regime changes.

Summary

The study provides evidence that PMI has predictive power over financial market performance when structural dynamics are accounted for. This justifies its inclusion as an input variable in models attempting to forecast asset returns and volatility. In particular, it reflects business sentiment and production expectations, which may feed into broader market pricing mechanisms.

Appendix B

Chosen Metrics Mathematical Formulae

The following metrics were used to evaluate the model performance in predicting asset class returns and volatilities. Let y_t denote the true value at time t, $\hat{y}_t$ the predicted value, and n the total number of observations.

- **$\mathbf{R^2}$ (Coefficient of Determination)**

  Measures the proportion of variance in the target variable explained by the model. Higher values (closer to 1) indicate better fit.

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

  where $\bar{y}$ is the mean of the actual values.

- **MAE (Mean Absolute Error)**

  The average of the absolute differences between predictions and actual values. Lower values indicate better accuracy.

$$\text{MAE} = \frac{1}{n}\sum_{t=1}^{n}|y_t - \hat{y}_t|$$

- **MAPE (Mean Absolute Percentage Error)**

Expresses prediction accuracy as a percentage. Lower values are better.

$$\text{MAPE} = \frac{100}{n} \sum_{t=1}^{n} \left| \frac{y_t - \hat{y}_t}{y_t} \right|$$

- **MSE (Mean Squared Error)**

  Penalizes larger errors more than MAE due to squaring the error terms.

$$\text{MSE} = \frac{1}{n} \sum_{t=1}^{n} (y_t - \hat{y}_t)^2$$

- **RMSE (Root Mean Squared Error)**

  The square root of MSE. Represents error in the same units as the target variable.

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

- **Normalized RMSE (NRMSE)**

  RMSE normalized by the range or mean of actual values. One common form:

$$\text{NRMSE} = \frac{\text{RMSE}}{\max(y_t) - \min(y_t)}$$

- **WAPE (Weighted Absolute Percentage Error)**

  An alternative to MAPE that avoids instability when actual values are near zero.

$$\text{WAPE} = \frac{\sum_{t=1}^{n} |y_t - \hat{y}_t|}{\sum_{t=1}^{n} |y_t|} \times 100$$

- **W_MAPE (Weighted MAPE)**

  A weighted version of MAPE, where errors are weighted by actual values.

$$\text{W\_MAPE} = \frac{\sum_{t=1}^{n} \left(\frac{|y_t - \hat{y}_t|}{|y_t|} \cdot |y_t| \right)}{\sum_{t=1}^{n} |y_t|} \times 100$$

- **Coverage Rate (Prediction Interval Coverage)**

  Measures the proportion of true values that fall within the predicted interval $[l_t, u_t]$, where l_t and u_t are the lower and upper quantiles at time t.

$$\text{Coverage Rate} = \frac{1}{n}\sum_{t=1}^{n} ⊮[l_t \le y_t \le u_t]$$

where $⊮[\cdot]$ is the indicator function.

- **Prediction Interval Width**

  The average width of the prediction interval over all timesteps.

$$\text{Interval Width} = \frac{1}{n}\sum_{t=1}^{n}(u_t - l_t)$$

- **Interval Score**

  A comprehensive scoring rule for evaluating the quality of prediction intervals. It penalizes both wider intervals and predictions that fall outside the interval.

$$\text{Interval Score}_t = (u_t - l_t) + \frac{2}{\alpha}(l_t - y_t) \cdot ⊮[y_t < l_t] + \frac{2}{\alpha}(y_t - u_t) \cdot ⊮[y_t > u_t]$$

$$\text{Interval Score} = \frac{1}{n}\sum_{t=1}^{n}\text{Interval Score}_t$$

where α represents the significance level (e.g., 0.1 for 90% intervals).

Appendix C

Portfolio Evaluation Metrics

The following mathematical formulas were used to evaluate portfolio performance in this project:

- **Total Return**:
$$\text{Total Return} = \frac{V_{\text{final}} - V_{\text{initial}}}{V_{\text{initial}}} \tag{C.1}$$
where V_{final} and V_{initial} are the portfolio values at the end and start of the evaluation period, respectively.

- **Sharpe Ratio**:
$$\text{Sharpe Ratio} = \frac{E(R_p) - R_f}{\sigma_p} \tag{C.2}$$
where $E(R_p)$ is the average portfolio return, R_f is the risk-free rate, and σ_p is the standard deviation of portfolio returns.

- **Maximum Drawdown (MDD)**:
$$\text{MDD} = \max_{t\in[0,T]} \left(\frac{\max_{\tau\in[0,t]} V(\tau) - V(t)}{\max_{\tau\in[0,t]} V(\tau)} \right) \tag{C.3}$$
This represents the largest observed percentage drop from a historical peak in portfolio value over time.

- **Annualized Return**:
$$\text{Annualized Return} = (1 + \text{Total Return})^{\frac{1}{N}} - 1 \tag{C.4}$$

where N is the number of years (or fraction of years) in the evaluation period.

Appendix D

System Specifications

This appendix contains the full specifications of the local system used for this project.

Table D.1: System Specifications

Specification	**Details**
OS Name	Microsoft Windows 11 Pro
OS Version	10.0.26100 N/A Build 26100
OS Manufacturer	Microsoft Corporation
System Manufacturer	Razer
System Model	Blade 15 Base Model (Mid 2021) - RZ09-0410
Processor	Intel64 Family 6 Model 141 Stepping 1 GenuineIntel 2304 MHz
BIOS Version	Razer 1.02, 13/07/2021
Total Physical Memory	32,515 MB
Virtual Memory Max Size	51,847 MB
Maximum Power Consumption	230W
GPU Specifications	
GPU Name	NVIDIA GeForce RTX 3060
Driver Version	560.94
CUDA Version	12.6
Memory Usage	0MiB / 6144MiB

Appendix E

Code Snippets of Key Functions Mentioned in the Report

E.1 Tensor Search Function

The following function iterates through TensorBoard event files to identify the trial with the lowest validation loss and retrieves the corresponding hyperparameters.

Listing E.1: Tensor Search Function for Extracting Best Hyperparameters

```python
# Iterate through all subdirectories (each representing a trial)
for root, dirs, files in os.walk(log_dir):
    trial_name = os.path.basename(root)  # Each trial has its own subdirectory
    for file in files:
        if file.startswith("events.out.tfevents"):  # Match TensorBoard log files
            log_file_path = os.path.join(root, file)
            print(f"Processing␣file:␣{log_file_path}")
            # Iterate through events in the log file
            for event in tf.compat.v1.train.summary_iterator(log_file_path):
                for value in event.summary.value:
                    if "val_loss" in value.tag:  # Check for validation loss
                        current_val_loss = value.simple_value
                        if current_val_loss < smallest_val_loss:
                            # Update smallest validation loss and
```

```
                                corresponding trial details
                            smallest_val_loss = current_val_loss
                            smallest_val_loss_trial = trial_name
                            smallest_val_loss_step = event.step

# If a trial with the smallest validation loss is found, retrieve its hyperparameters
if smallest_val_loss_trial is not None:
    hparams_file = os.path.join(log_dir, smallest_val_loss_trial, "hparams.yaml")
    if os.path.exists(hparams_file):
        with open(hparams_file, "r") as f:
            try:
                smallest_val_loss_params = yaml.load(f, Loader=yaml.FullLoader)
            except yaml.YAMLError as e:
                print(f"Error reading hparams.yaml: {e}")
```

E.2 Model Loading Functions

E.2.1 Function to Load a Trained TFT Model

The following function reconstructs the Temporal Fusion Transformer (TFT) model using the same architecture and hyperparameters from training, and then loads the saved weights from a `.pt` file. This ensures the model is ready for inference without retraining.

Listing E.2: Function to load a trained TFT model

```
def load_model(model_path, train_data, target_variable):
    """
    Load the Temporal Fusion Transformer (TFT) model with the given architecture and saved weights.
    """
    train_dataset = TimeSeriesDataSet(
        train_data,
        time_idx="time_idx",
        target=target_variable,
        group_ids=["Group_ID"],
        max_encoder_length=11,
        max_prediction_length=1,
        time_varying_known_reals=[
            "CPI_actual", "Unemployemnt␣Rate", "1-Year␣T-Bill␣Rate",
            "10-Year␣T-Bill␣Rate", "PMI_Actual", "
                CPI_Forecast_of_this_month",
            "Equity_Return_this_month", "Equity_Vol_this_month"
        ],
        time_varying_unknown_reals=[target_variable],
        time_varying_known_categoricals=["Target_Month"],
        target_normalizer=GroupNormalizer(groups=["Group_ID"]),
        add_encoder_length=True,
    )

    tft = TemporalFusionTransformer.from_dataset(
        train_dataset,
        hidden_size=1083,
        attention_head_size=479,
        lstm_layers=1,
        dropout=0.0006301076661392908,
        hidden_continuous_size=188,
        loss=QuantileLoss(quantiles=[0.05, 0.5, 0.95]),
        optimizer="Ranger",
        reduce_on_plateau_patience=10,
    )

    # Load saved weights
    tft.load_state_dict(torch.load(model_path))
    tft.eval()  # Set the model to evaluation mode
    return tft
```

E.2.2 Function to Load a Trained LSTM Model

The following code initializes the LSTM model with the same architecture and hyperparameters used during training, then loads the saved weights from a `.pt` file. This prepares the model for inference by setting it to evaluation mode.

Listing E.3: LSTM model loading for prediction

```
# === Load the trained model weights from .pt ===
```

```python
input_size = len(input_features)
num_layers = 8
hidden_size = 130
dropout = 0.4638

model = LSTMModel(
    input_size=input_size,
    hidden_size=hidden_size,
    num_layers=num_layers,
    dropout=dropout
)

model.load_state_dict(
    torch.load(r"Model.pt")
)

model.eval()  # Set the model to evaluation mode
```

E.3 Rolling Prediction Function for the TFT Model

The following function is used to generate rolling quantile predictions from the trained Temporal Fusion Transformer (TFT) model. It simulates monthly forecasting by updating the context window and predicting a single time step at each iteration.

Listing E.4: Function to generate rolling quantile predictions and match with actuals

```python
def rolling_predict(model, full_data, target_variable, asset, max_encoder=
   MAX_ENCODER, start_month="2025-03-01", end_month="2017-11-01"):
    full_data["Date"] = pd.to_datetime(full_data["Date"])
    full_data["Target_Month"] = full_data["Target_Month"].astype(str)
    full_data["Group_ID"] = "THIS␣ASSET"

    start_date = pd.to_datetime(start_month)
    end_date = pd.to_datetime(end_month)

    results = []

    current_date = start_date
    while current_date >= end_date:
        encoder_start = current_date - pd.DateOffset(months=max_encoder)
        encoder_data = full_data[(full_data["Date"] >= encoder_start) & (full_data["Date"] < current_date)].copy()

        if len(encoder_data) < max_encoder:
            print(f"Skipping␣{current_date.strftime('%Y-%m')}␣due␣to␣insufficient␣encoder␣history.")
            current_date -= pd.DateOffset(months=1)
            continue

        predict_row = full_data[full_data["Date"] == current_date].copy()
        if predict_row.empty:
            print(f"Skipping␣{current_date.strftime('%Y-%m')}␣because␣target␣row␣is␣missing.")
            current_date -= pd.DateOffset(months=1)
            continue

        combined = pd.concat([encoder_data, predict_row], ignore_index=True)
        combined["time_idx"] = range(len(combined))

        return_this_month = f"{asset}_Return_this_month"
        vol_this_month = f"{asset}_Vol_this_month"

        dataset = TimeSeriesDataSet(
            combined,
            time_idx="time_idx",
            target=target_variable,
            group_ids=["Group_ID"],
            max_encoder_length=max_encoder,
            max_prediction_length=1,
            time_varying_known_reals=[
                "CPI_actual", "Unemployemnt␣Rate", "1-Year␣T-Bill␣Rate",
                "10-Year␣T-Bill␣Rate", "PMI_Actual", "CPI_Forecast_of_this_month",
                return_this_month, vol_this_month
            ],
            time_varying_unknown_reals=[target_variable],
            time_varying_known_categoricals=["Target_Month"],
            target_normalizer=GroupNormalizer(groups=["Group_ID"]),
            add_encoder_length=True,
        )

        dl = dataset.to_dataloader(train=False, batch_size=1, num_workers=8)

        predictions = model.predict(dl, mode="quantiles", trainer_kwargs=dict(accelerator="gpu")).cpu().numpy()
        predictions = predictions[:, 0, :]  # remove batch dimension
        last_step_preds = predictions[-1]

        results.append({
            "Date": current_date,
            "Predicted_5%": last_step_preds[0],
            "Predicted_50%": last_step_preds[1],
            "Predicted_95%": last_step_preds[2],
            "Actual": predict_row[target_variable].values[0]
        })

        current_date -= pd.DateOffset(months=1)

    return pd.DataFrame(results)
```

Appendix F

Required Libraries

F.1 Chosen Language

Given the nature of this project working extensively with time series data, particularly financial time series **Python** was selected as the programming language of choice. This decision was guided by several key factors:

- **Rich Ecosystem of Data Analysis Libraries:** Python offers powerful libraries such as `pandas`, `numpy`, and `scipy` which are indispensable for processing and analyzing time series data.
- **Widespread Industry Adoption:** Python is extensively used in the financial services sector, making it a natural fit for tasks involving financial modeling, analysis, and machine learning.
- **Robust Community and Documentation:** A vast community and extensive documentation facilitate faster troubleshooting and access to best practices.
- **Integration with ML Frameworks:** Python supports seamless integration with cutting-edge machine learning libraries such as `TensorFlow`, `PyTorch` and `scikit-learn`, enabling efficient development and experimentation.

These strengths made Python the optimal choice for implementing this project efficiently and effectively.

F.2 Utilized Libraries

A wide range of Python libraries were leveraged throughout this project to support data ingestion, preprocessing, modeling, optimization, and visualization. Below is an overview of the most essential ones:

Time Series Forecasting

- **PyTorch Forecasting:** Built on top of `PyTorch Lightning`, this library simplifies the development of deep learning models for time series forecasting. It was pivotal in implementing the Temporal Fusion Transformer (TFT) and Long Short-Term Memory (LSTM) models tailored to financial data. Originally developed by Jan Beitner, it has since evolved through contributions from the open-source community.
- **PyTorch Lightning:** A high-level wrapper around PyTorch that abstracts away boilerplate training code. It enabled cleaner and more modular model training pipelines, including custom adjustments for the project's specific needs.

Optimization and Hyperparameter Tuning

- **Optuna:** Used for automated hyperparameter optimization, allowing for efficient model tuning and performance improvements.
- **CVXPY:** Applied for convex optimization tasks, particularly in the context of asset allocation.

Monitoring and Visualization

- **TensorFlow & TensorBoard:** Although PyTorch was the primary deep learning framework, TensorBoard was employed for its effective visual tracking of training metrics.
- **Matplotlib:** Utilized for producing visualizations of model performance, forecast accuracy, and exploratory data analysis.

Data Manipulation and Machine Learning

- **Pandas & NumPy:** Core libraries for data wrangling, manipulation, and numerical operations.
- **Scikit-learn:** Used for preprocessing tasks (e.g., scaling) and evaluation metrics for model assessment.
- **SciPy:** Provided advanced mathematical and statistical tools needed during modeling and analysis.

Resource Management and Utility Libraries

- **GPUtil, psutil, multiprocessing:** Employed for efficient GPU/CPU resource monitoring and enabling parallel processing.
- **OS, Logging, Time, JSON, YAML, GC, Traceback:** Core Python libraries used for configuration parsing, file I/O, logging, memory management, and exception handling.

Appendix G

Dyson External Cooling system Specs

Table G.1: Product Specifications for Dyson Hot+Cool Purifying Fan Heater (Model HP09)

Feature	**Specification**
Batteries Included	No
Battery Size & Number	Remote control battery
Bladeless	Yes
Brand	Dyson
Cable Length	1.80 m
Country of Origin	Malaysia
Dimensions (H × W × D)	76.4 × 20.5 × 13 cm
Filtration	HEPA
Guarantee	2 year guarantee included
Manufacturer Part Number (MPN)	381387-01
Maximum Room Size	27.0 m^2
Model Name / Number	HP09
Power	40.00 W
Power Supply	Mains
Weight	5.70 kg

Source: `https://www.johnlewis.com/dyson-hot-cool-purifying-fan-heater/p5729624`

Appendix H

Model Architecture Specifications

This appendix provides a comprehensive breakdown of the initial architecture configurations for each Temporal Fusion Transformer (TFT) and Long Short Term Memory (LSTM) model used in this study.

H.1 Temporal Fusion Transformers Models

H.1.1 Bond Asset Class Models

Table H.1: Bond Return Model Architecture

Name	**Type**	**Params**	**Mode**
loss	QuantileLoss	0	train
logging_metrics	ModuleList	0	train
input_embeddings	MultiEmbedding	72	train
prescalers	ModuleDict	59.0K	train
static_variable_selection	VariableSelectionNetwork	9.7M	train
encoder_variable_selection	VariableSelectionNetwork	87.7M	train
decoder_variable_selection	VariableSelectionNetwork	78.0M	train
static_context_variable_selection	GatedResidualNetwork	7.5M	train
static_context_initial_hidden_lstm	GatedResidualNetwork	7.5M	train
static_context_initial_cell_lstm	GatedResidualNetwork	7.5M	train
static_context_enrichment	GatedResidualNetwork	7.5M	train
lstm_encoder	LSTM	165M	train
lstm_decoder	LSTM	165M	train
post_lstm_gate_encoder	GatedLinearUnit	3.8M	train
post_lstm_add_norm_encoder	AddNorm	2.7K	train
static_enrichment	GatedResidualNetwork	9.4M	train
multihead_attn	InterpretableMultiHeadAttention	3.7M	train
post_attn_gate_norm	GateAddNorm	3.8M	train
pos_wise_ff	GatedResidualNetwork	7.5M	train
pre_output_gate_norm	GateAddNorm	3.8M	train
output_layer	Linear	4.1K	train
Trainable params		568M	
Non-trainable params		0	
Total params		568M	
Model size (MB)		2,275.91	
Modules in train mode		3,078	
Modules in eval mode		0	

Table H.2: Bond Volatility Model Architecture

Name	**Type**	**Params**	**Mode**
loss	QuantileLoss	0	train
logging_metrics	ModuleList	0	train
input_embeddings	MultiEmbedding	72	train
prescalers	ModuleDict	52.1K	train
static_variable_selection	VariableSelectionNetwork	120K	train
encoder_variable_selection	VariableSelectionNetwork	1.3M	train
decoder_variable_selection	VariableSelectionNetwork	1.2M	train
static_context_variable_selection	GatedResidualNetwork	7.3K	train
static_context_initial_hidden_lstm	GatedResidualNetwork	7.3K	train
static_context_initial_cell_lstm	GatedResidualNetwork	7.3K	train
static_context_enrichment	GatedResidualNetwork	7.3K	train
lstm_encoder	LSTM	14.4K	train
lstm_decoder	LSTM	14.4K	train
post_lstm_gate_encoder	GatedLinearUnit	3.6K	train
post_lstm_add_norm_encoder	AddNorm	84	train
static_enrichment	GatedResidualNetwork	9.1K	train
multihead_attn	InterpretableMultiHeadAttention	0	train
post_attn_gate_norm	GateAddNorm	3.7K	train
pos_wise_ff	GatedResidualNetwork	7.3K	train
pre_output_gate_norm	GateAddNorm	3.7K	train
output_layer	Linear	129	train
Trainable params		2.6M	
Non-trainable params		0	
Total params		2.6M	
Model size (MB)		10.54	
Modules in train mode		1,600	
Modules in eval mode		0	

H.1.2 Equity Asset Class Model

Table H.3: Equity Return Model Architecture

Name	**Type**	**Params**	**Mode**
loss	QuantileLoss	0	train
logging_metrics	ModuleList	0	train
input_embeddings	MultiEmbedding	72	train
prescalers	ModuleDict	3.8K	train
static_variable_selection	VariableSelectionNetwork	486K	train
encoder_variable_selection	VariableSelectionNetwork	4.4M	train
decoder_variable_selection	VariableSelectionNetwork	3.9M	train
static_context_variable_selection	GatedResidualNetwork	4.7M	train
static_context_initial_hidden_lstm	GatedResidualNetwork	4.7M	train
static_context_initial_cell_lstm	GatedResidualNetwork	4.7M	train
static_context_enrichment	GatedResidualNetwork	4.7M	train
lstm_encoder	LSTM	9.4M	train
lstm_decoder	LSTM	9.4M	train
post_lstm_gate_encoder	GatedLinearUnit	2.3M	train
post_lstm_add_norm_encoder	AddNorm	2.2K	train
static_enrichment	GatedResidualNetwork	5.9M	train
multihead_attn	InterpretableMultiHeadAttention	2.1M	train
post_attn_gate_norm	GateAddNorm	2.4M	train
pos_wise_ff	GatedResidualNetwork	4.7M	train
pre_output_gate_norm	GateAddNorm	2.4M	train
output_layer	Linear	3.3K	train
Trainable params		66.1M	
Non-trainable params		0	
Total params		66.1M	
Model size (MB)		264.35	
Modules in train mode		1368	
Modules in eval mode		0	

Table H.4: Equity Volatility Model Architecture

Name	**Type**	**Params**	**Mode**
loss	QuantileLoss	0	train
logging_metrics	ModuleList	0	train
input_embeddings	MultiEmbedding	72	train
prescalers	ModuleDict	3.4K	train
static_variable_selection	VariableSelectionNetwork	297K	train
encoder_variable_selection	VariableSelectionNetwork	2.7M	train
decoder_variable_selection	VariableSelectionNetwork	2.4M	train
static_context_variable_selection	GatedResidualNetwork	1.9M	train
static_context_initial_hidden_lstm	GatedResidualNetwork	1.9M	train
static_context_initial_cell_lstm	GatedResidualNetwork	1.9M	train
static_context_enrichment	GatedResidualNetwork	1.9M	train
lstm_encoder	LSTM	3.9M	train
lstm_decoder	LSTM	3.9M	train
post_lstm_gate_encoder	GatedLinearUnit	964K	train
post_lstm_add_norm_encoder	AddNorm	1.4K	train
static_enrichment	GatedResidualNetwork	2.4M	train
multihead_attn	InterpretableMultiHeadAttention	0	train
post_attn_gate_norm	GateAddNorm	966K	train
pos_wise_ff	GatedResidualNetwork	1.9M	train
pre_output_gate_norm	GateAddNorm	966K	train
output_layer	Linear	2.1K	train
Trainable params		28.1M	
Non-trainable params		0	
Total params		28.1M	
Model size (MB)		112.31	
Modules in train mode		2194	
Modules in eval mode		0	

H.1.3 Gold Asset Class Model

Table H.5: Gold Return Model Architecture

Name	**Type**	**Params**	**Mode**
loss	QuantileLoss	0	train
logging_metrics	ModuleList	0	train
input_embeddings	MultiEmbedding	72	train
prescalers	ModuleDict	4.7K	train
static_variable_selection	VariableSelectionNetwork	528K	train
encoder_variable_selection	VariableSelectionNetwork	4.8M	train
decoder_variable_selection	VariableSelectionNetwork	4.3M	train
static_context_variable_selection	GatedResidualNetwork	3.1M	train
static_context_initial_hidden_lstm	GatedResidualNetwork	3.1M	train
static_context_initial_cell_lstm	GatedResidualNetwork	3.1M	train
static_context_enrichment	GatedResidualNetwork	3.1M	train
lstm_encoder	LSTM	6.3M	train
lstm_decoder	LSTM	6.3M	train
post_lstm_gate_encoder	GatedLinearUnit	1.6M	train
post_lstm_add_norm_encoder	AddNorm	1.8K	train
static_enrichment	GatedResidualNetwork	3.9M	train
multihead_attn	InterpretableMultiHeadAttention	1.5M	train
post_attn_gate_norm	GateAddNorm	1.6M	train
pos_wise_ff	GatedResidualNetwork	3.1M	train
pre_output_gate_norm	GateAddNorm	1.6M	train
output_layer	Linear	2.7K	train
Trainable params		48.0M	
Non-trainable params		0	
Total params		48.0M	
Model size (MB)		192.00	
Modules in train mode		962	
Modules in eval mode		0	

Table H.6: Gold Volatility Model Architecture

Name	**Type**	**Params**	**Mode**
loss	QuantileLoss	0	train
logging_metrics	ModuleList	0	train
input_embeddings	MultiEmbedding	72	train
prescalers	ModuleDict	2.3K	train
static_variable_selection	VariableSelectionNetwork	114K	train
encoder_variable_selection	VariableSelectionNetwork	1.0M	train
decoder_variable_selection	VariableSelectionNetwork	931K	train
static_context_variable_selection	GatedResidualNetwork	538K	train
static_context_initial_hidden_lstm	GatedResidualNetwork	538K	train
static_context_initial_cell_lstm	GatedResidualNetwork	538K	train
static_context_enrichment	GatedResidualNetwork	538K	train
lstm_encoder	LSTM	1.1M	train
lstm_decoder	LSTM	1.1M	train
post_lstm_gate_encoder	GatedLinearUnit	268K	train
post_lstm_add_norm_encoder	AddNorm	732	train
static_enrichment	GatedResidualNetwork	671K	train
multihead_attn	InterpretableMultiHeadAttention	0	train
post_attn_gate_norm	GateAddNorm	269K	train
pos_wise_ff	GatedResidualNetwork	538K	train
pre_output_gate_norm	GateAddNorm	269K	train
output_layer	Linear	1.1K	train
Trainable params		8.4M	
Non-trainable params		0	
Total params		8.4M	
Model size (MB)		33.66	
Modules in train mode		1244	
Modules in eval mode		0	

H.2 Long Short Term Memory

H.2.1 Bond Asset Class Models

Table H.7: BOOND Return LSTM Model Architecture

Name	Type	Params	Mode
lstm	LSTM	1.0M	train
fc	Linear	393	train
Trainable params		1.0M	
Non-trainable params		0	
Total params		1.0M	
Model size (MB)		4.107	
Modules in train mode		2	
Modules in eval mode		0	

Table H.8: BOND VOL LSTM Model Architecture

Name	Type	Params	Mode
lstm	LSTM	1.2M	train
fc	Linear	1.6K	train
Trainable params		1.2M	
Non-trainable params		0	
Total params		1.2M	
Model size (MB)		4.654	
Modules in train mode		2	
Modules in eval mode		0	

H.2.2 Equity Asset Class Models

Table H.9: Equity Return LSTM Model Architecture

Name	**Type**	**Params**	**Mode**
lstm	LSTM	784K	train
fc	Linear	369	train
Trainable params		785K	
Non-trainable params		0	
Total params		785K	
Model size (MB)		3.140	
Modules in train mode		2	
Modules in eval mode		0	

Table H.10: Equity Volatility LSTM Model Architecture

Name	**Type**	**Params**	**Mode**
lstm	LSTM	1.2M	train
fc	Linear	1.6K	train
Trainable params		1.2M	
Non-trainable params		0	
Total params		1.2M	
Model size (MB)		4.654	
Modules in train mode		2	
Modules in eval mode		0	

H.2.3 Gold Asset Class Models

Table H.11: Gold Return LSTM Model Architecture

Name	**Type**	**Params**	**Mode**
lstm	LSTM	593K	train
fc	Linear	321	train
Trainable params		593K	
Non-trainable params		0	
Total params		593K	
Model size (MB)		2.376	
Modules in train mode		2	
Modules in eval mode		0	

Table H.12: Gold Volatility LSTM Model Architecture

Name	**Type**	**Params**	**Mode**
lstm	LSTM	28.8K	train
fc	Linear	144	train
Trainable params		28.9K	
Non-trainable params		0	
Total params		28.9K	
Model size (MB)		0.116	
Modules in train mode		2	
Modules in eval mode		0	

Appendix I

Prediction Plots for Asset Classes

This appendix presents the prediction plots generated using LSTM and TFT models for each asset class (Bonds, Equities, Gold) across two prediction targets: Returns and Volatility.

I.1 LSTM Model Prediction Plots

I.1.1 Bonds

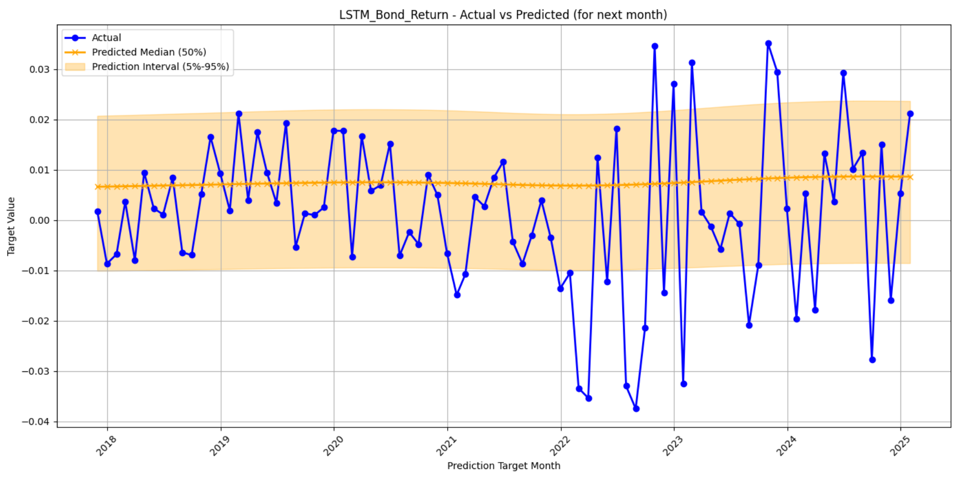

Figure I.1: LSTM - Bond Return Prediction Plot

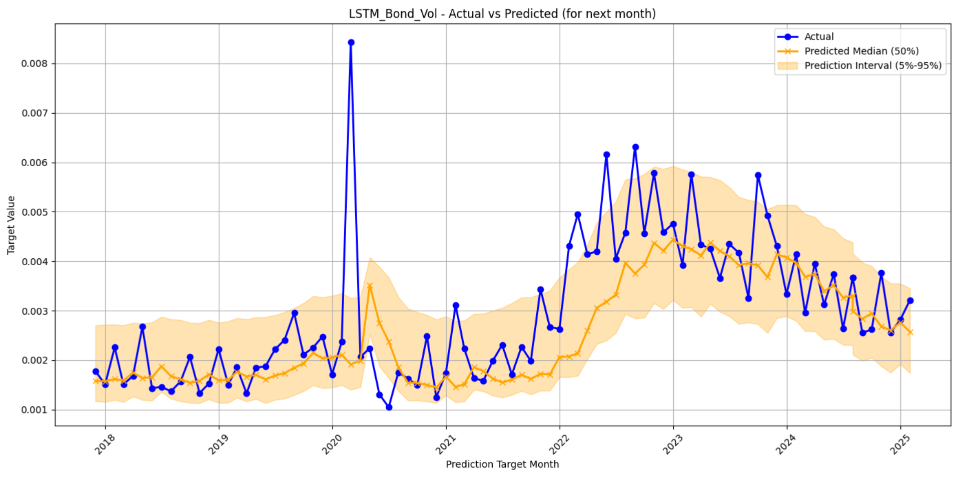

Figure I.2: LSTM - Bond Volatility Prediction Plot

I.1.2 Equities

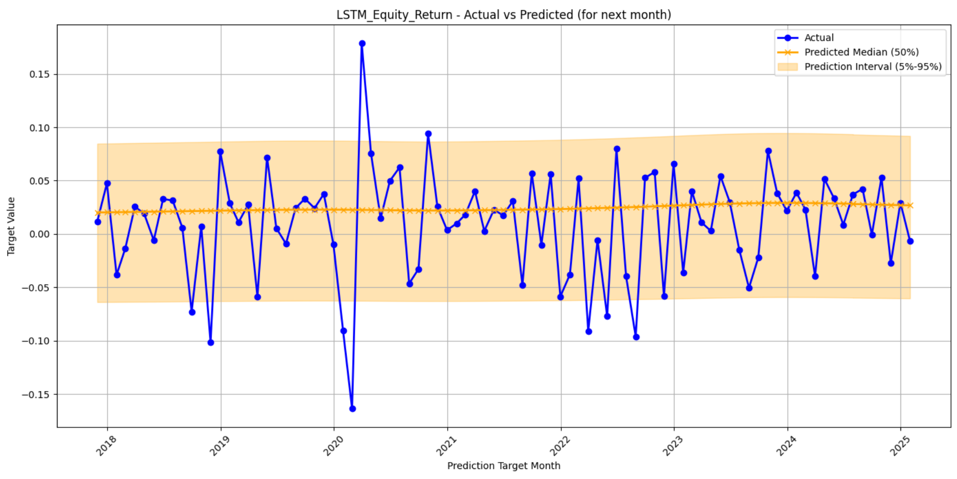

Figure I.3: LSTM - Equity Return Prediction Plot

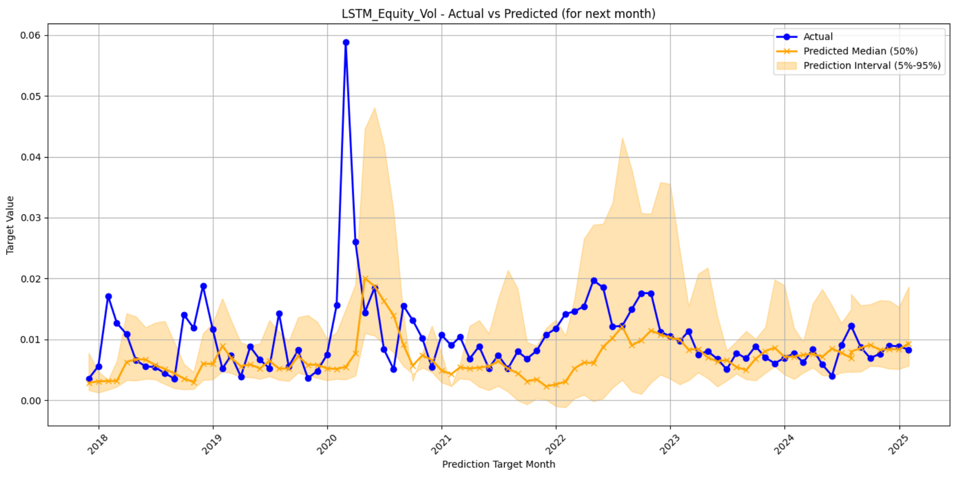

Figure I.4: LSTM - Equity Volatility Prediction Plot

I.1.3 Gold

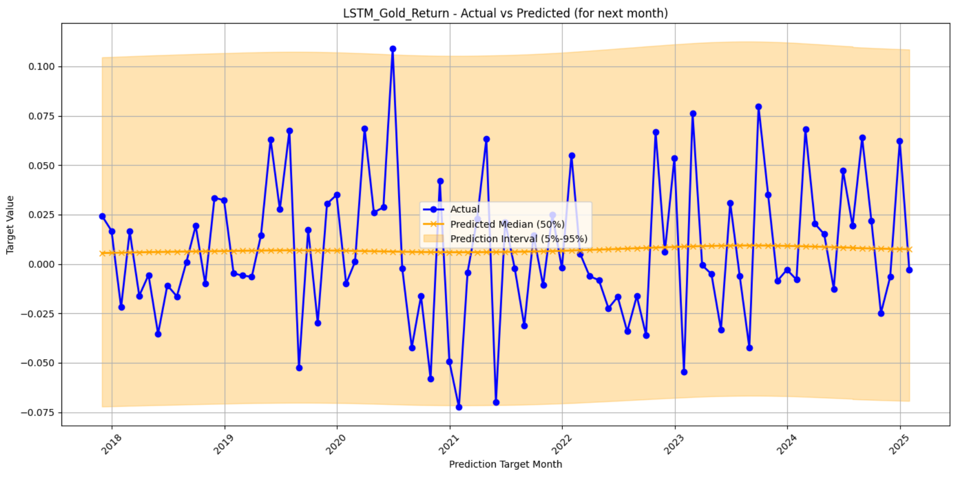

Figure I.5: LSTM - Gold Return Prediction Plot

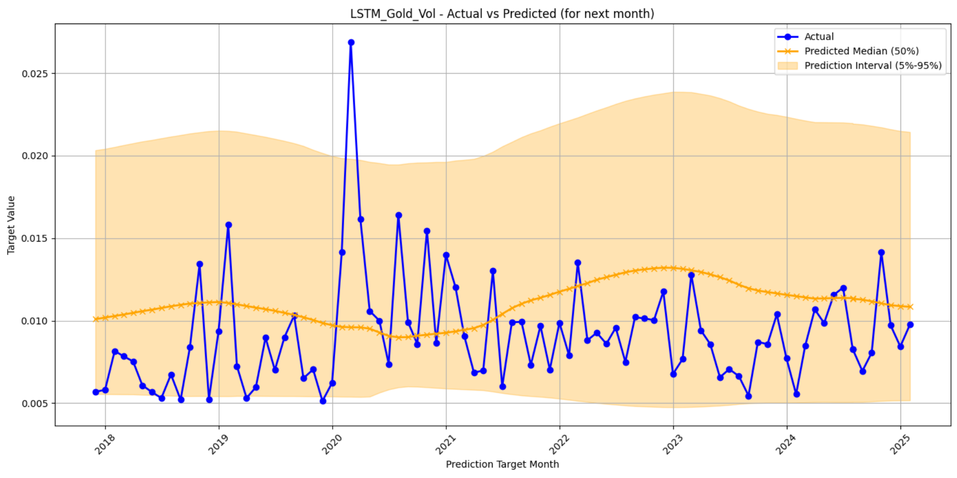

Figure I.6: LSTM - Gold Volatility Prediction Plot

I.2 TFT Model Prediction Plots

I.2.1 Bonds

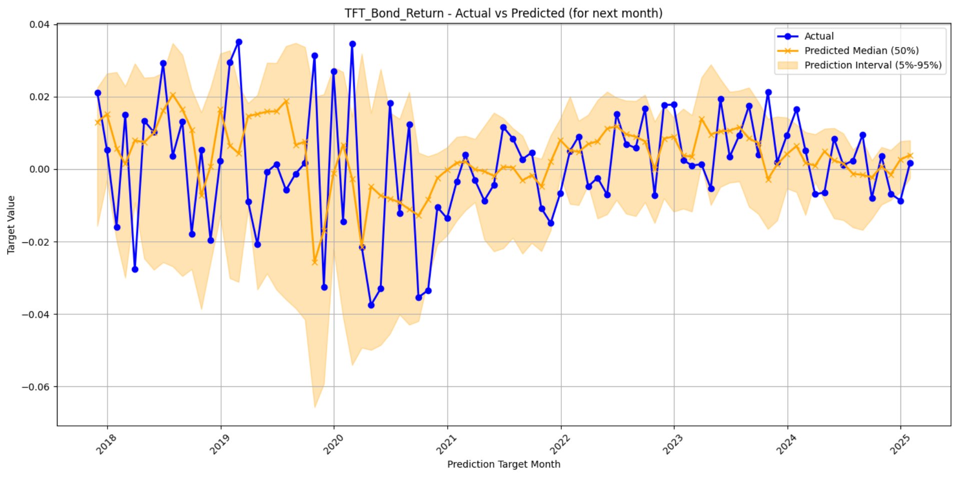

Figure I.7: TFT - Bond Return Prediction Plot

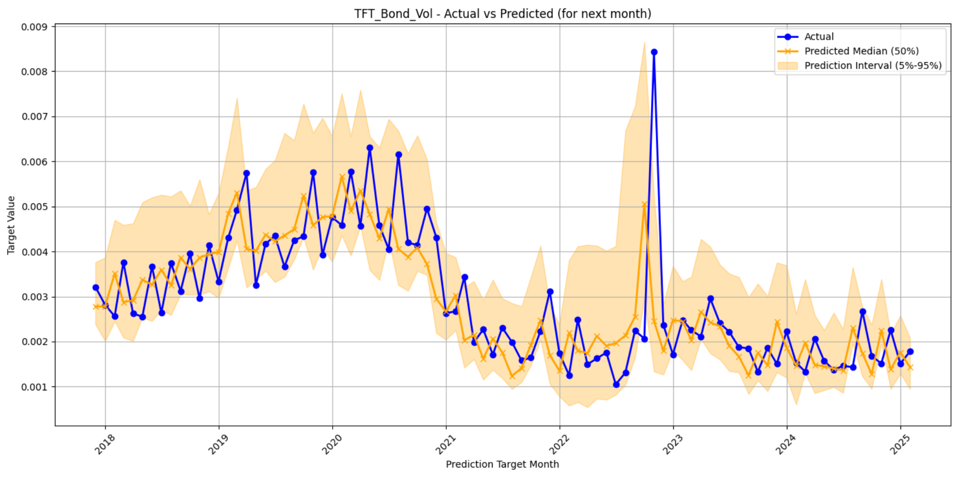

Figure I.8: TFT - Bond Volatility Prediction Plot

I.2.2 Equities

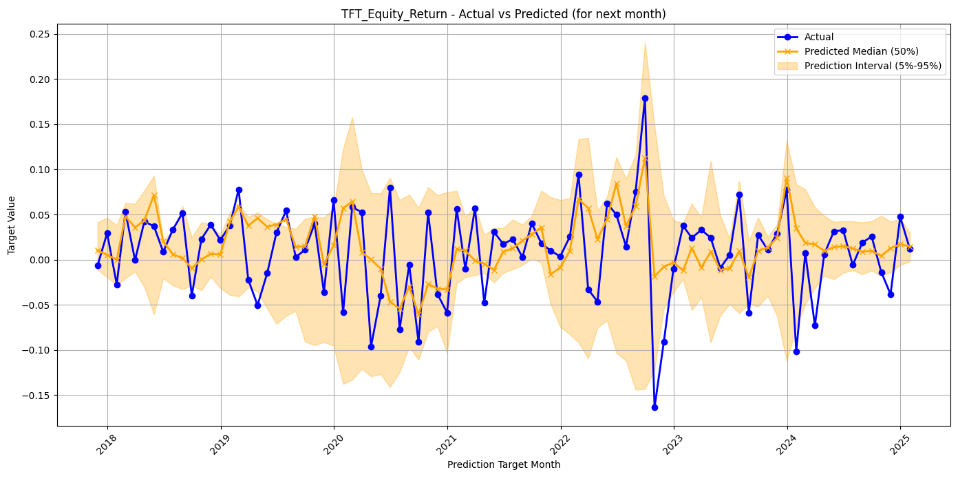

Figure I.9: TFT - Equity Return Prediction Plot

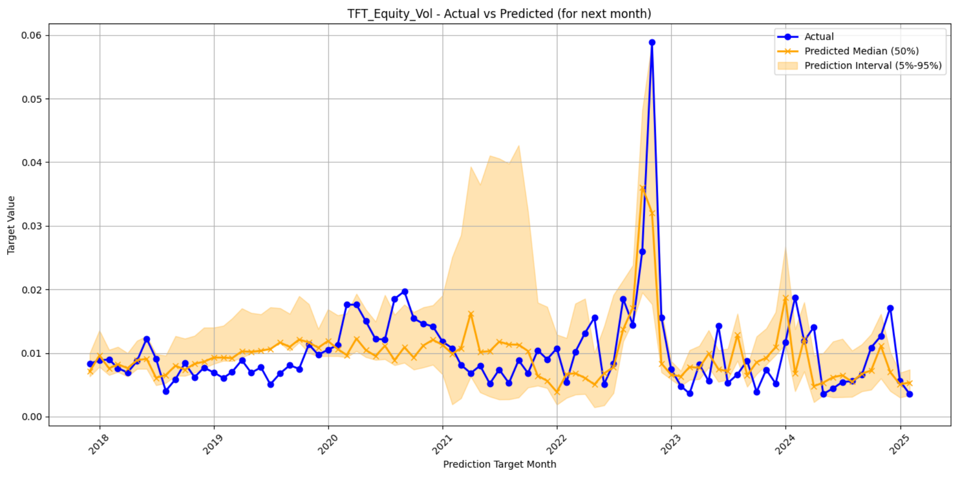

Figure I.10: TFT - Equity Volatility Prediction Plot

I.2.3 Gold

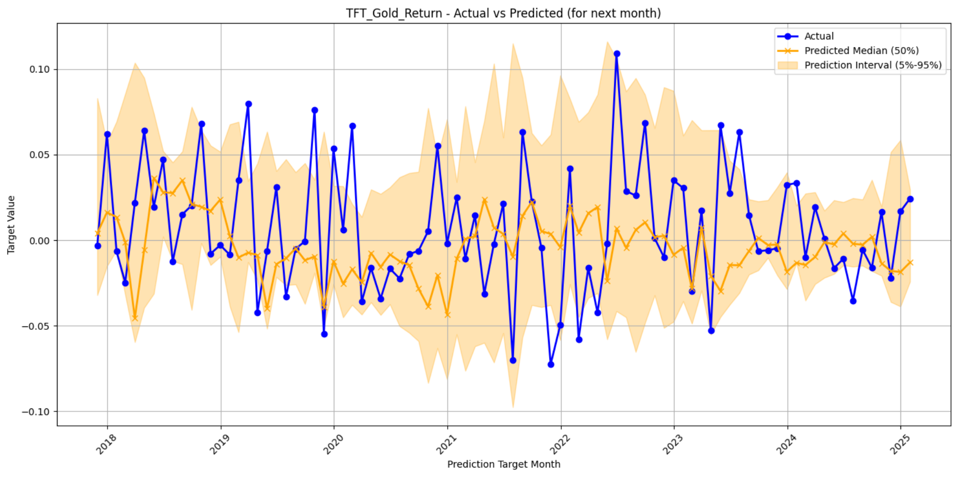

Figure I.11: TFT - Gold Return Prediction Plot

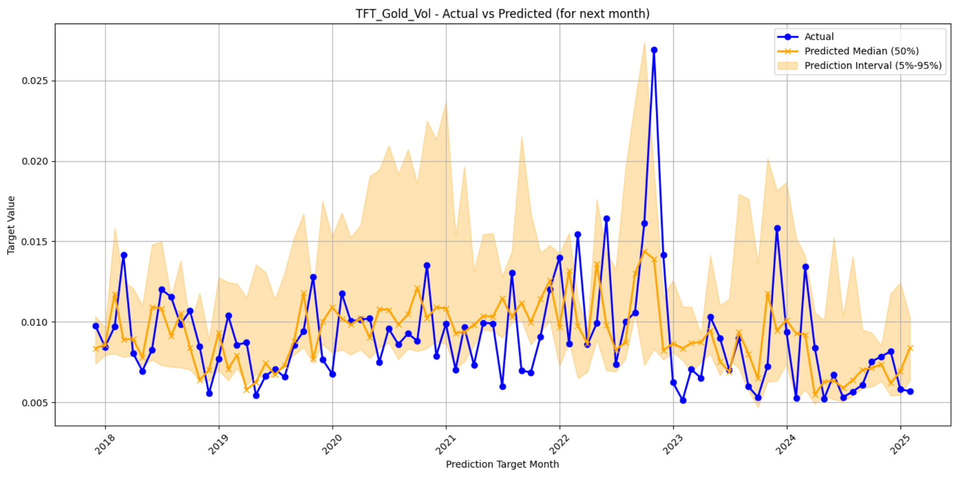

Figure I.12: TFT - Gold Volatility Prediction Plot

Bibliography

Ahmed, Sabeen et al. (July 2023). “Transformers in Time-Series Analysis: A Tutorial”. In: *Circuits, Systems, and Signal Processing* 42.12, pp. 7433–7466. ISSN: 1531-5878. DOI: 10.1007/s00034-023-02454-8. URL: http://dx.doi.org/10.1007/s00034-023-02454-8.

Akiba, Takuya et al. (2019). “Optuna: A Next-generation Hyperparameter Optimization Framework”. In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.*

Beitler, Jan (2020). *PyTorch Forecasting: Making time series forecasting with PyTorch accessible.* https://github.com/jdb78/pytorch-forecasting.

Black, F. and R. Litterman (1992). “Global Portfolio Optimization”. In: *Financial Analysts Journal* 48.5, pp. 28–43.

BlackRock (2025). *What are equity investments?* Accessed: 2025-02-10. URL: https://www.blackrock.com/us/individual/education/equities.

Chen, N. (2024). “Exploring the development and application of LSTM variants”. In: *Applied and Computational Engineering.*

Chi, David (2020). “Unanticipated Unemployment Rate News on the Stock Market”. In: *Econ Honor Thesis.*

Chiang, Thomas C. (2023). “Stock Returns and Inflation Expectations: Evidence from 20 Major Countries”. In: *Quantitative Finance and Economics* 7.4, pp. 538–568. DOI: 10.3934/QFE.2023027. URL: http://www.aimspress.com/journal/QFE.

CME Group (2025). *Gold Futures Contract Specifications.* Accessed February 28, 2025. URL: https://www.cmegroup.com/markets/metals/precious/gold.contractSpecs.html.

Dubey, Ashutosh Kumar et al. (2021). “Study and Analysis of SARIMA and LSTM in Forecasting Time Series Data”. In: *Sustainable Energy Technologies and Assessments*

47, p. 101474. ISSN: 2213-1388. DOI: 10.1016/j.seta.2021.101474. URL: https://www.sciencedirect.com/science/article/pii/S2213138821004847.

Fernando, J. (2024). *Bonds: How They Work and How To Invest.* Accessed: 2025-02-10. URL: https://www.investopedia.com/terms/b/bond.asp.

Gonzalo, Jesús and Abderrahim Taamouti (2017). "The Reaction of Stock Market Returns to Unemployment". In: *Studies in Nonlinear Dynamics and Econometrics* 21.4, p. 20150078. DOI: 10.1515/snde-2015-0078. URL: https://www.researchgate.net/publication/318161001_The_reaction_of_stock_market_returns_to_unemployment.

Hochreiter, S. (1997). "Long Short-term Memory". In: *Neural Computation.*

Kenton, W. (2024). *S&P 500 Index: What It's for and Why It's Important in Investing.* Accessed: 2025-02-10. URL: https://www.investopedia.com/terms/s/sp500.asp.

Kong, Yaxuan et al. (2024). "Unlocking the Power of LSTM for Long Term Time Series Forecasting". In: *arXiv preprint arXiv:2408.10006.* Accepted at the 39th AAAI Conference on Artificial Intelligence (AAAI 2025). URL: https://doi.org/10.48550/arXiv.2408.10006.

Kontopoulou, Vaia et al. (2023). "A Review of ARIMA vs. Machine Learning Approaches for Time Series Forecasting in Data Driven Networks". In: *Future Internet* 15.8, p. 255. DOI: 10.3390/fi15080255. URL: https://doi.org/10.3390/fi15080255.

Lai, Joseph et al. (Sept. 2024). *Beyond the Balance Sheet: North American Asset Management 2024.* McKinsey & Company. URL: https://www.mckinsey.com/industries/financial-services/our-insights/beyond-the-balance-sheet-north-american-asset-management-2024.

Lim, Bryan et al. (2021). "Temporal Fusion Transformers for Interpretable Multi-Horizon Time Series Forecasting". In: *International Journal of Forecasting* 37.4, pp. 1748–1764. DOI: 10.1016/j.ijforecast.2021.03.012.

Machine Learning Models (2023). *The Evolution of Machine Learning: A Brief History and Timeline.* Accessed: 2025-03-31. URL: https://machinelearningmodels.org/the-evolution-of-machine-learning-a-brief-history-and-timeline/.

Marin, A. (2022). *Correlation does not imply causation: Examples.* https://www.statology.org/correlation-does-not-imply-causation-examples/. Accessed:

2025-03-24. URL: https://www.statology.org/correlation-does-not-imply-causation-examples/.

Markowitz, H. M. (1991). "Foundations of Portfolio Theory". In: *The Journal of Finance* 46.2, pp. 469–477.

Markowitz, Harry (1952). "Portfolio Selection". In: *The Journal of Finance* 7.1, pp. 77–91.

Masood, S. (2023). "Neural Networks and Deep Learning: A Comprehensive Overview of Modern Techniques and Applications". In: *Journal Environmental Sciences and Technology* 2.2, pp. 8–20.

Met Éireann (2024). *Historical Data - Climate.* https://www.met.ie/climate/available-data/historical-data. Accessed: 2025-03-24. URL: https://www.met.ie/climate/available-data/historical-data.

Mortezapour Shiri, Farhad et al. (2024). "A Comprehensive Overview and Comparative Analysis on Deep Learning Models: CNN, RNN, LSTM, GRU". In: *Journal on Artificial Intelligence* 6.1. Originally submitted as arXiv:2305.17473, pp. 301–360. DOI: 10.32604/jai.2024.054314. URL: https://doi.org/10.32604/jai.2024.054314.

Özkan, Funda and Mustafa Kiriş (2020). "Is Purchasing Managers' Index (PMI) a Leading Indicator for Stock, Bond and Foreign Exchange Markets in Turkey?" In: *Çukurova Üniversitesi Sosyal Bilimler Enstitüsü Dergisi* 29.1. Accessed April 2025, pp. 103–120. ISSN: 1304-7194. URL: https://dergipark.org.tr/en/pub/cusosbil/issue/52314/674826.

Pattnaik, Debidutta et al. (2023). "Investment in gold: A bibliometric review and agenda for future research". In: *Research in International Business and Finance* 64, p. 101854. DOI: 10.1016/j.ribaf.2022.101854. URL: https://doi.org/10.1016/j.ribaf.2022.101854.

Perold, André F. (2004). "The Capital Asset Pricing Model". In: *Journal of Economic Perspectives* 18.3, pp. 3–24. DOI: 10.1257/0895330042162340. URL: https://www.aeaweb.org/articles?id=10.1257/0895330042162340.

Portfolio Literacy (n.d.). *What Is the 60/40 Portfolio Allocation?* Accessed: 2025-03-26. URL: https://portfolioliteracy.com/what-is-the-60-40-portfolio-allocation/.

Ross, Jonathan (2023). “Does prior stock return correlation predict future stock return correlation?” In: *SN Business & Economics* 3.176. DOI: 10.1007/s43546-023-00551-z. URL: https://doi.org/10.1007/s43546-023-00551-z.

Salinas, David, Valentin Flunkert, and Jan Gasthaus (2020). “DeepAR: Probabilistic Forecasting with Autoregressive Recurrent Networks”. In: *International Journal of Forecasting (IJF)*. Preprint available at arXiv:1704.04110. Elsevier. URL: https://arxiv.org/abs/1704.04110.

Sathyanarayana, C.H. and Anushree M. Gargesa (2018). “An Analytical Study of the Effect of Inflation on Stock Market Returns”. In: *International Journal of Economics and Financial Issues* 8.4, pp. 274–282. ISSN: 2146-4138. URL: https://www.econjournals.com/index.php/ijefi/article/view/6725.

Sharpe, W. F. (1994). “The Sharpe Ratio”. In: *Journal of Portfolio Management* 21.1, pp. 49–58.

Statista Research Department (2024). *Carbon intensity of the power sector in Ireland from 2010 to 2023*. https://www.statista.com/statistics/1290237/carbon-intensity-power-sector-ireland/. Accessed: 2025-03-24. URL: https://www.statista.com/statistics/1290237/carbon-intensity-power-sector-ireland/.

Staudemeyer, R. C. and E. R. Morris (2019). “Understanding LSTM–a tutorial into long short-term memory recurrent neural networks”. In: *arXiv preprint* arXiv:1909.09586.

Uddin, Md. Gazi Salah and Md. Mahmudul Alam (2010). “The Impacts of Interest Rate on Stock Market: Empirical Evidence from Dhaka Stock Exchange”. In: *South Asian Journal of Management Sciences* 4.1, pp. 21–30. ISSN: 2074-2967. URL: https://ssrn.com/abstract=2941287.

Wallaroo AI (2024). *Machine Learning Models and the Black Box Problem.* Accessed: March 17, 2025. URL: https://wallaroo.ai/machine-learning-models-and-the-black-box-problem/.

Wright, Less (2020). *Ranger-Deep-Learning-Optimizer.* https://github.com/lessw2020/Ranger-Deep-Learning-Optimizer. Accessed: 2025-03-27.

Wynne, Mark A. and Fiona D. Sigalla (1994). “The Consumer Price Index”. In: *Economic Review, Federal Reserve Bank of Dallas* Second Quarter 1994. URL: https://www.researchgate.net/publication/5029912_The_consumer_price_index.

Yu, Hsiang-Fu, Nikhil Rao, and Inderjit S. Dhillon (2016). “Temporal Regularized Matrix Factorization for High-dimensional Time Series Prediction”. In: *Advances in Neural Information Processing Systems* 29. URL: `https://proceedings.neurips.cc/paper_files/paper/2016/file/85422afb467e9456013a2a51d4dff702-Paper.pdf`.

Zhang, Yuanyuan, Xiang Li, and Sini Guo (2018). “Portfolio selection problems with Markowitz’s mean–variance framework: a review of literature”. In: *Fuzzy Optimization and Decision Making* 17.2, pp. 125–158. DOI: `10.1007/s10700-017-9266-z`.