Portfolio Optimization using Mathematical and Machine Learning Models

Sai Adith Prakash
Introduction to Optimization, Z2007
Prof. Manoj Kumar
30/Jan/2025

Indian Institute of Technology Madras, Zanzibar Campus

Contents

1 Abstract

Portfolio optimization is pivotal in modern finance, aiming to maximize investor returns while managing risks effectively. This project focuses on automating investment management without human intervention by allocating capital to stocks in proportions that maximize returns. The implementation combines classic and modern approaches, beginning with **Markowitz's mean-variance optimization** and extending to **Genetic Algorithms** and more advanced methods like **LSTM**-based deep learning and **NLP** (natural Language Processing) models to predict stock weights dynamically.

The project's performance is compared amongst one another, mainly observing the expected returns sharpe ratios and maximum dropdown amongst other model-specific metrics, ensuring practical applicability. Future directions could include incorporating ESG factors, real-time market analytics, and cross-asset portfolio management to that greatly could boost the allocation ratios and better diversify the portfolio, boosting the model's robustness and relevance.

By leveraging **Modern Portfolio Theory**, **Machine Learning** and **NLP**, this project provides a well-rounded, automated and practical solution to modern investment management challenges that human beings may face and details we may miss out on, making it valuable for both **retail** and **institutional** investors across the board.

2 Deliverables

- **Models**: Implement the models **Mean Variance Optimization (MVO)** a **Genetic Algorithm**, **LSTM** and an **NLP Based Model**
- **Evaluation**: Evaluate model's using backtesting, analyzing portfolio returns for varying periods of time, including one day, one and 2 weeks, 1,3 and 6 months and a year. The varying timeframes make evaluation robust, removing doubts of anomalies. We also analyze **how** the capital is being allocated, whether it is relatively even and diverse or whether it's pinning large percentages of capital on particular stocks. We will consider Max Drawdown, Cumulative Returns and the Sharpe ratio in our evaluation as well.
- **Comparisons**: Compare the results of models amongst one another to identify the best-suited portfolio construction method and their respective trade-offs.
- **Visualizations**: Generate visual representations of the portfolio performance, pie charts of how the capital is allocated and a table of comparisons for varying time periods.

Model evaluation, comparisons and visualizations amongst other details will be present in the report. The source code for all the models will be written in Python and a link to it's Kaggle repository will be provided.

3 Introduction

Portfolio Optimization has become a crucial cornerstone of growing capital, helping investors maximize returns and manage risk effectively. With the rise of AI and in today's rapidly evolving financial market, the integration of Artificial Intelligence in the finance world is imperative. This project aims to automate investment management by leveraging advanced techniques to allocate capital to maximize returns given a particular investment.

3.1 Background and Context

Traditionally, portfolios are handled by specialists in Applied Math and Statistics and they came up with investment strategies using various statistical and computer models. But soon after followed Markowitz with his Mean Variance Optimization method of maximizing returns whilst minimizing risk, laying the foundation for portfolio optimization models - this model is still used to some degree in today's finance world. This model is the first model used in this project. Although the Mean Variance model has been instrumental in shaping Modern Portfolio Theory, this approach often relies on certain static assumptions and may fail to adapt to the dynamic and volatile market. Using machine learning and natural language processing (NLP) , we can enhance portfolio management, enabling dynamic weight predictions and even incorporate sentiment analysis.

This project intends to bridge the gap between traditional finance and modern machine learning by implementing and comparing methods such as the Mean-Variance Optimization model, Genetic Algorithms, LSTM-based models and sentiment analysis to best allocate capital within portfolios. The larger aim is to provide which models provide better return on investment and learn why they do so.

3.2 Structure of the Report

The report is organized as follows:

1. **Literature Review** – A discussion of existing portfolio optimization techniques and their limitations.

2. **Key concepts** - There is a lot of financial keywords and formulae that need to be understood, so we will visit this section to have a grasp of this topic

3. **Methodology** – Detailed descriptions of the approaches used, including Markowitz optimization, Genetic Algorithms, LSTM, and sentiment analysis implementation.

4. **Results & Discussion** – Analysis of the performance of the implemented models using key metrics as discussed in the deliverables.

5. **Conclusion** – A summary of findings and recommendations for future work.

6. Then we will have the **Appendices** with the generated code samples, and the **References** with the texts and videos I referred to.

By integrating traditional finance with modern computational methods, this report aims to contribute to the development of more effective and automated investment strategies.

4 Literature Review

Portfolio optimization has long been a critical area of research in finance, with numerous theories and methodologies developed to balance risk and return. This section reviews the foundational theories and recent advancements that inform the approaches used in this project.

4.1 Markowitz Mean-Variance Optimization

Markowitz's mean-variance optimization, introduced in 1952, remains a cornerstone of modern portfolio theory. In his paper, 'Portfolio Selection', Markowitz proposed the concept of the efficient frontier, where portfolios minimize risk for a given level of return. Despite its groundbreaking impact, critiques such as those in 'Modern Portfolio Theory and Investment Analysis' by Elton et al. (2009) highlight limitations, including assumptions of static correlations and normally distributed returns, which may not hold in real-world financial markets.

4.2 Genetic Algorithms in Portfolio Optimization

Genetic Algorithms (GAs), inspired by natural selection, are increasingly used to tackle complex, non-linear optimization problems in finance. Goldberg's foundational book, 'Genetic Algorithms in Search, Optimization, and Machine Learning' (1989), laid the groundwork for applying evolutionary algorithms. Furthermore, Grinold and Kahn's 'Active Portfolio Management' (2000) explores practical applications of GAs in dynamically rebalancing portfolios.

4.3 Deep Learning for Financial Time Series

Deep learning models, particularly Long Short-Term Memory (LSTM) networks, have proven highly effective for analyzing financial time series. Introduced by Hochreiter and Schmidhuber in their 1997 paper, 'Long Short-Term Memory', LSTMs address the challenge of learning long-term dependencies in sequential data. Fischer and Krauss, in their study 'Deep Learning with Long Short-Term Memory Networks for Financial Market Predictions' (2018), demonstrated how LSTMs outperform traditional models in forecasting stock prices and allocating portfolio weights, establishing their utility in dynamic financial markets.

4.4 Sentiment Analysis for Market Prediction

Sentiment analysis has emerged as a valuable tool for integrating behavioral finance into portfolio optimization. The Financial Sentiment Analysis on News and Reports Using Large Language Models and FinBERT paper discusses various approaches for extracting actionable insights from financial news and social media. Similarly, Bollen et al., in their influential study Twitter Mood Predicts the Stock Market (2011), demonstrated a direct link between social media sentiment and market performance. These findings underscore the potential of combining traditional financial metrics with sentiment analysis for improved portfolio management.

5 Key Concepts

5.1 1. Expected Return

The expected return of a portfolio is the weighted average of the expected returns of its individual assets. The expected return of each individual asset is **the average of it's historical returns.** If $r_1, r_2, \ldots, r_n$ are the expected returns of assets 1 through n, and $w_1, w_2, \ldots, w_n$ are the portfolio weights, the expected return R_p of the portfolio is:

$$R_p = \sum_{i=1}^{n} w_i r_i$$

5.2 2. Risk (Variance)

The risk is assumed to be the the variance of the portfolio returns. The variance σ_p^2 of a portfolio with n assets is calculated as:

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

Where:

- $\mathbf{w}$ is the portfolio weights,
- Σ is the covariance matrix of asset returns.

By definition, **Standard deviation** is then

$$\sigma_p = \sqrt{\mathbf{w}^T \Sigma \mathbf{w}}$$

5.3 3. Covariance

Covariance measures how 2 assets are moving together.

$$\text{Cov}(X, Y) = \frac{1}{T} \sum_{t=1}^{T} (X_t - \bar{X})(Y_t - \bar{Y})$$

Where:

- X_t and Y_t are the returns of assets X and Y at time t,
- $\bar{X}$ and $\bar{Y}$ are the average returns of assets X and Y.

5.4 4. Correlation

Correlation is normalized version of covariance. It's clearer to see the strength and direction eg: positive correlation, negative etc. The correlation $\rho(X, Y)$ between two assets X and Y is:

$$\rho(X, Y) = \frac{\text{Cov}(X, Y)}{\sigma_X \sigma_Y}$$

Where σ_X and σ_Y are the standard deviations of assets X and Y.

5.5 5. Portfolio Weights

Portfolio weights $w_1, w_2, \ldots, w_n$ represents the fraction of the total portfolio for each asset. I have defined it to be like an array, w_n is the ratio of money I will be investing in $asset_n$.

6. Sharpe Ratio

The Sharpe ratio is a measure of the portfolio's risk-adjusted return. It is calculated as:

$$S = \frac{R_p - R_f}{\sigma_p}$$

Where:

- R_p is the expected return,
- R_f is the risk-free rate, normally we observe it to be the return on govt. bonds,
- σ_p standard deviation of the portfolio's returns, what we are taking to be portfolio risk.

6 Methodology

6.1 Mean Variance Optimization

Data collection:
In this project, the data was collected using the `yfinance` library, which allows for downloading historical stock price data directly from Yahoo Finance. The data of interest was the **adjusted closing prices** of the selected stocks Apple (AAPL), Microsoft (MSFT), Google (GOOGL), Amazon (AMZN), Tesla (TSLA), Meta (META), Nvidia (NVDA), Netflix (NFLX), IBM, and Disney (DIS) over a period from **January 1, 2010** to **January 1, 2023**. The reason for using Adjusted Close price instead of just Close Price was the Adjusted Close Price also accounts for the dividends, stock-splits and other corporate actions - making it more accurate.

Optimization Problem

Mean Variance Optimization has an intricate yet simple mechanism: **Maximize the Sharpe ratio**.

$$\text{Maximize } \frac{R_p - R_f}{\sigma_p}$$

Where:

- R_p is the expected return of the portfolio,
- R_f is the risk-free rate,
- σ_p is the standard deviation of the portfolio's returns (risk).

Goal: To find the weights that maximize the Sharpe ratio.

The optimization is performed using **quadratic programming**, a numerical method that is particularly suited for Mean-Variance Optimization due to the quadratic nature of the portfolio variance function. The primary steps involved are:

1. **Rewriting the Sharpe Ratio:** The Sharpe ratio can be rewritten as:

$$S = \frac{w^T \mu - R_f}{\sqrt{w^T \Sigma w}}$$

   This rewriting is done to convert the Sharpe ratio into a form that is more convenient for quadratic programming. Specifically, the portfolio's risk (σ_p) is expressed as the square root of the portfolio variance ($w^T \Sigma w$). μ is the vector of the historial mean returns (for each stock), and $w^T \mu$ simply is the dot product of the weight with the returns vector, essentially giving us the predicted returns.

2. **Objective Function for Quadratic Programming:** To convert this into a quadratic programming problem, we minimize the negative Sharpe ratio. The objective function becomes:

$$\max \left(\frac{w^T \mu - R_f}{\sqrt{w^T \Sigma w}} \right)$$

Alternatively, by squaring both the numerator and the denominator, we eliminate the square root and get:

$$S^2 = \frac{(w^T\mu - R_f)^2}{w^T\Sigma w}$$

which leads to the optimization problem:

$$\max \frac{(w^T\mu - R_f)^2}{w^T\Sigma w}$$

3. **Quadratic Objective Form:** Expanding the objective function to form a purely quadratic function:

$$\max \left[\frac{w^T\mu w^T\mu - 2R_f\mu^T w + R_f^2}{w^T\Sigma w}\right]$$

Thus, the quadratic programming problem is:

$$\max_w \quad \frac{1}{2}w^T\Sigma w - \mu^T w$$

subject to the following constraints:

- Full investment constraint: $\Sigma w = 1$ (the sum of portfolio weights must equal 1),
- Weight bounds: $0 \leq w_i \leq 1$ for all i (non-negativity and upper-bound constraints).

We need to remember that in order to **maximize the Sharpe ratio**, we will need to **minimize** the **negative** of the Sharpe ratio.

The optimization problem is solved using specialized quadratic programming solvers, such as `cvxopt` or `PyPortfolioOpt`, to obtain the optimal portfolio weights that maximize the Sharpe ratio while satisfying the constraints.

Rationale for choosing Mean Variance Optimization:
Fundamentally, Mean Variance optimization is based on Maximizing the Sharpe Ratio. Instead of focusing one just minimizing risk or maximizing returns, we make sure we strike an even balance between both. Furthermore, the integration of the risk-free rate accounts for the investments made with virtually **no risk**. For example, if investing using Mean Variance gives me a 2% return whilst government bonds give me 3% - there is no point in taking risk by investing using MVO. That's why we have to **account for the risk-free rate**, as we would like to perform better than risk-free stocks, taking strategic risks can make more profit than investing in risk-free bonds. Minimizing portfolio variance is inline with traditional practices, so it has been used multiple times - and since variability is low, we can expect more consistent results. In addition to that, it is sensitive to the risk-free rate of the country, so we do take into account of the location and what risk-free investments there are there too. It's a relatively simple computation, relying only on risk and returns, so whilst there are more sophisticated models, this is simple and effective because of it focuses on the core of maximizing returns and minimizing risk. It diversifies the stock, as instead of choosing stocks purely on the basis of its returns, we factor in the covariance of these assets. Later, we will see the Sharpe ratio being used in other models, showing how strong of an indicator it truly is.

6.2 Genetic Algorithm

Genetic Algorithms, inspired from natural selection in evolution, is an optimization process that starts with a random set of weights and then evaluates it according to a fitness function. A **fitness function** evaluates how strong a solution is to fulfill the objective.

1. **Initialization:**
   - Start with a population of possible solutions (a random set of weights)
2. **Fitness Function:**
   - Evaluate the quality of each solution using a fitness function, so by how much are we able to maximize returns or minimize risk.
3. **Selection:**
   - Select individuals based on their fitness scores will become the *parents* for the next generation. Parents with better fitness scores have a greater likelihood of getting children that better fulfill these requirements
4. **Crossover:**
   - Combine parent solutions to produce offspring by mixing their parameters, introducing new combination of children and allowing new solutions to be explored.
5. **Mutation:**
   - Introduce random changes to some individuals to maintain diversity.
6. **Elitism:**
   - Retain the best-performing individuals across generations to ensure that the best solutions are kept.

Data collection
The data is the exact same as used for **Mean Variance Optimization (4.1)**. The same data for the same time-frame from the same source, so no changes in the data.

Implementation
The very first step of implementation comes up with a pivotal question: What would be the **fitness function** for portfolio optimization? My first ideas included maximizing the expected return or minimizing the risk, but we can then observe we already know one such function that does both pretty well: the **Sharpe ratio.** To reiterate, the Sharpe ratio is:

$$S = \frac{w^T\mu - R_f}{\sqrt{w^T\Sigma w}}$$

where $w^T\mu$ is the Expected Return, R_f is the Risk-free return and the denominator is the Portfolio standard deviation. However, to make sure we don't just invest all the money into one stock, we added an extra constraint that made sure that no one particular stock has all the investment pinned onto it. We set the upper bound of each weight of the portfolio to 40%.

Constraints:

- Full investment constraint: $\Sigma w = 1$ (the sum of portfolio weights must equal 1),
- Non-negativity: $0 \leq w_i$ for all i (non-negativity and upper-bound constraints).
- Upper bound: $w_i \leq 0.4$ for all i, this makes sure that not all the investment is one particular stock, setting the upper bound to 40%

For the Genetic Algorithm used, the formula for the mutation rate is:

$$\text{mutation\_rate} = \max\left(0.1, \frac{1}{1 + \frac{\text{gen}}{100}}\right)$$

At the beginning, the mutation_rate will be pretty high and the intent behind this is to ensure most of the solution space is explored - more **diversity** is included at the very beginning so we won't be hyper-focused on sub-optimal sub-spaces of the solution space. As we progress, the number of generations increase to a point where the generations exceed 900, and then the mutation rate becomes a constant 10%, because after these many generations the optimal solution is highly likely to be in the span of the existing solution space, so we just need to fine-tune our results.

- **Selection**: The selection process uses rank-based probabilities to choose parents. Weights with higher fitness score (higher Sharpe ratio) are ranked higher and therefore have a higher chance of being selected as parents for the next generations. This ensures the fittest people carry on transferring their advantageous properties.
- **Crossover**: A blending of two parents' genes, where each child is created by combining portions of the two parents' genetic information. A uniform blending coefficient α determines the ratio of influence from each parent.
- **Elitism**: The top individuals, based on fitness, are directly carried over to the next generation to preserve the best solutions. For us, the top 5% are considered most 'Elite'. This means, these weights are carried on directly to the next generations without any modifications, as we would want to preserve them.

To ensure the first 2 constraints are fulfilled, we use a process called **Clipping** - for each generation, it firstly transforms these such that it falls in the range [0,1]. Then, it normalizes the weights, such that they add up to 1. But since we have also implemented the third constraint of no one weight having more than 40% of the investment, the clipping values are readjusted to [0,0.4]. Initially we started with a population of **100** and we stopped at **20,000 generations**. Apart from these concepts, we used **Numpy** for data handling, **Random** library for generating mutations and crossovers, and **SciPy** for optimizing the Sharpe ratio which was the negative of the loss function.

Rationale for choosing Genetic Algorithm:
One major reason for choosing a Genetic Algorithm is that it can explore complex solution spaces - at the beginning we set the mutation rate high so we find more diverse data and test a wide range of solutions, eliminating preconceptions or biases of optimal solutions. Since portfolio optimization is generally non-linear, Genetic Algorithms are suited for the task as they can explore non-linear solution spaces to find a Global Optima instead of getting fixed in a Local minima. Furthermore, our fitness function has the **Sharpe ratio:** this handles the 2 objectives of maximizing returns and minimizing risk. Genetic Algorithms are also inherently parallelizable, so they test multiple solutions simultaneously, improving efficiency. Genetic Algorithms involve stochastic processes by using randomness, further exemplified by the mutations.

6.3 Natural Language Processing - Sentiment Analysis

Data collection:
The first intentions of using sentiment analysis was to create a bot that scrolled through X (formerly known as Twitter) and multiple other websites to 'scrape' data, but due to these websites having restrictions and blocking robots on their website, I was forced to use another method: **NewsAPI**. In a nutshell, NewsAPI returns headlines and news articles from news websites - since this had a decent free tier, I was able to use it to get sentiment data.
Sentiment Analysis Process
Sentiment analysis begins with **preprocessing** the text. This process typically includes tokenization, where we split the text into individual words or tokens. These tokens are then converted to lowercase to ensure uniformity, as text can come in various case formats. Additionally, common stop words—such as "and," "the," or "in" — are removed, as they do not contribute much to the overall sentiment. Lemmatization is also applied to reduce words to their base forms, such as turning "running" into "run" or "better" into "good." In a nutshell, we preprocess the data to remove unnecessary words that don't have significance contribution and lemmatize the words to their most simplex form, from which we can extract meaning.

The next step is to **assign sentiment scores** to individual words in the text. We use a lexicon-based approach, where each word has an associated sentiment score. These scores are typically numerical values representing positive, negative, or neutral sentiments. For example, words like 'happy' might have a high positive score, while words like 'sad' might have a high negative score. Once each word is assigned a sentiment value, these individual scores are combined to form an overall score for the text - I've summed all the scores and normalized it to create what is called a **compound score**.

The **compound score** is crucial. This score is calculated by summing the weighted sentiment values of all the words in the text and normalizing the result to fall within a range of -1 to 1. A score of **1** indicates an overwhelmingly positive sentiment, while **-1** represents a highly negative sentiment. Scores close to **0** indicate neutral sentiment. The compound score serves as a summary measure that reflects the overall emotional tone of the text, simplifying complex sentiment into a single value.

In our project, a compound score greater than or equal to **0.05** is considered positive, **-0.05** or less is negative, and values between **-0.05** and **0.05** indicate neutral sentiment. This threshold allows for easy classification of the sentiment and is helpful when processing large amounts of text, such as news articles. In your application, the compound scores of individual articles are averaged to determine the overall sentiment for a stock ticker, giving you a numerical representation of public sentiment toward that stock. This approach helps distill complex, subjective opinions from multiple sources into a clear, actionable metric.

After these scores are received, we normalize the score. The raw compound data gives us a range of [-1,1], but we want a weight - stocks cannot have a negative weight, so we normalize the code to the range [0,1] - the transformation is:

$$\textbf{normalized\_sentiment} = \frac{\textbf{compound\_score+1}}{2}$$

This allows the non-negativity constraint to be satisfied. We then sum up the weights and then divide each weight by this sum, so they add up to 1. This satisfies the constraint $\Sigma\mathbf{w}=\mathbf{1}$.

We used:

1. **requests**: A Python library for sending HTTP requests. It is used to fetch news articles from the NewsAPI based on the stock tickers.

2. **SentimentIntensityAnalyzer**: Part of the Natural Language Toolkit (NLTK) library, I used it to perform sentiment analysis on news headlines. It computes a compound sentiment score as previously discussed, indicating the overall sentiment of a piece of text. We got the **polarity score** form this library that we ended up evaluating the texts with. Once each article was evaluated, we averaged the score for all articles, giving us the overall sentiment indicator.

3. **API (NewsAPI)**: A third-party tool used to fetch news articles. The API is queried by sending HTTP requests to retrieve stock-related news based on my stock tickers.

Rationale for choosing sentiment analysis:
Portfolio decisions are normally made by observing the market. This information is made available through the **news** and other media outlets - so to be able to analyze gives us a good insight on the quality of the stock. Furthermore, **successful traders** and financial enthusiasts also express their **opinions** on outlets, so we can factor their analysis as well. Use of **simple** financial data may be **outdated** - stock prices change by the second and maybe just because a company appreciated in the past doesn't mean their stock always does well in the present. Stock prices are based off of **demand** - the higher the demand for the stocks, the more likely it is at appreciating. When we perform sentiment analysis for a wide range of articles, we can observe how the demand is for that particular stock and therefore gauge how good of an investment it is. This also helps us in risk-analysis - a continued negative sentiment is an indicator of high risk of depreciation. All these reasons compelled me to test Sentiment Analysis on portfolio optimization.

6.4 LSTM

Data Collection & Preprocessing

Data collection involved downloading historical stock data for ten major companies from Yahoo Finance, including OHLC (Open, High, Low, Close) and **Adjusted close** prices from 2015 to 2023. Preparation began with calculating **log returns** to normalize price changes, followed by the integration of technical indicators: Simple Moving Average (SMA), Exponential Moving Average (EMA), Relative Strength Index (RSI), Moving Average Convergence Divergence (MACD), Average True Range (ATR), Average Directional Index (ADX), and Bollinger Bands (BB) to analyze trends, momentum, volatility and general market strength and calculated with pandas_ta. Missing values were imputed (replaced with 0), and the data was scaled using MinMaxScaler for uniformity. The main reason is to make it easy for the neural network so all indicators contribute equally to the learning process.

Preprocessing included creating sequences of fixed lengths for input into LSTMs, allowing the model to capture temporal dependencies effectively.

Model Architecture
LSTM works by using gates to control the flow of information. The forget gate decides which parts of the previous memory should be 'forgotten. The way it does that is by comparing the current input and previous hidden state to output a value between 0 and 1(0 to forget and closer to 1 to not forget). The input gate determines what new information should be added, with one component deciding which values to update and another adding values. The cell state is then updated by combining the old memory (after some parts are forgotten) with the new values. This process repeats, and what we can observe is that the computer can essentially 'forget' information whilst capturing long-term dependencies.

Specific Model Analysis
Let us now deep-dive into our model.
In this step, the prepared data (returns and technical indicators) is transformed into sequences. We split the data into chunks of size 10 (in this case). Each sequence contains the returns and technical indicators for the past 10 **time steps** (this is the point in the sequence where an update happens, with the goal of predicting the return at the next time step. This means that for each sequence, the model will take in the returns and technical indicators for the previous 10 time steps as input, and predict the return for the subsequent time step as output. The inputs are prepared by concatenating the scaled returns and technical indicators into a single array, ensuring that both sources of data are available to the model for each time step.

The model has multiple LSTM layers and additional layers like Dropout and BatchNormalization for regularization and stability. The input layer basically accepts sequences of data with a shape corresponding to the number of features and of sequence length. The first LSTM layer is Bidirectional, meaning it processes the input data in both forward and backward directions, which allows the model to capture patterns that depend on both past and future data. Dropout layers are included to prevent overfitting by randomly deactivating a percentage of neurons during training. BatchNormalization layers follow each LSTM layer to normalize the activations and ensure faster and stable training. BatchNormalization normalizes the outputs of the previous layer before feeding them to the next layer. This results in improved training efficiency and stability. The model also includes two more LSTM layers that return sequences of data, followed by a dense layer with ReLU activation to produce the final output. Why use ReLU? Well, after your linear transformation, applying activation like ReLU to a dense layer introduces non-linearity, enabling the model to learn more complex relationships within the data. Without this, we are as good as just linear models. Our case has multiple LSTM's working together, to catch even more long-term data dependencies - the output of the first layer is passed onto the input as the next. This is known as **Stacking** an LSTM. `NumPy` is used for computations and creating sequences from time-series data, while `Pandas TA` (`pandas_ta`) is used for Technical Analysis helped in integrating technical indicators like SMA, RSI, and MACD. `Scikit-learn` aids in preprocessing data through normalization (`MinMaxScaler`) and splitting it into training and testing sets. The deep learning model is built using `TensorFlow`, employing layers like `Bidirectional LSTM` for capturing time-series dependencies, `Dropout` for overfitting prevention, and `BatchNormalization` for training stability. A custom Sharpe ratio-based loss function optimizes the model for risk-adjusted returns, with training enhanced by callbacks like `EarlyStopping` and learning rate reduction.

The Loss Function:
In order to optimize the models, we changed the loss function to prefer sequences maximizing risks whilst minimizing variance. And in comes the Sharpe Ratio Loss Function . This custom loss function computes the negative Sharpe ratio, which the model attempts to minimize during training. (If we are minimizing the negative of the Sharpe ratio, we are basically choosing sequences with the highest Sharpe ratio). A small constant ($1e-7$) is added to the denominator to avoid division by zero. This approach ensures that the model is optimizing the portfolio for maximum risk-adjusted returns, rather than just maximizing returns without considering the risk associated with the investment.
We finally then normalize the weights by dividing each weight by the sum of the total weights so they all add upto 1.

Rationale for choosing an LSTM:

1. **Modeling Time Series Data:** In the code, historical stock data, including 'Open', 'High', 'Low', 'Close', and 'Adj Close' prices, is utilized. LSTMs are particularly suited to learn from sequence based data

2. **Captures Long-Term Dependencies:** With stacked LSTM layers, the model is can capture long-term dependencies within the stock market environment. This is imperative, as the model shouldn't misunderstand short-term shocks and changes, as stock price movements are often influenced by long-term trends. LSTMs effectively capture these complex, long-term relationships.

3. **Handling Multiple Features:** This model uses multiple technical indicators (SMA, EMA, RSI, MACD, ATR and so on) alongside the adjusted close prices. Increasing relevant features may allow the model to capture more intricate details in the data. LSTMs can handle multi-dimensional time-series data, making them ideal for data of stock prices and technical indicators.

4. **Stacked LSTM:** The stacking shows that the output of the first layer of the LSTM is then fed into the second, and then that is fed into the third one (I used 3 layers). This allows us to get a better approximation of the long-term dependencies.

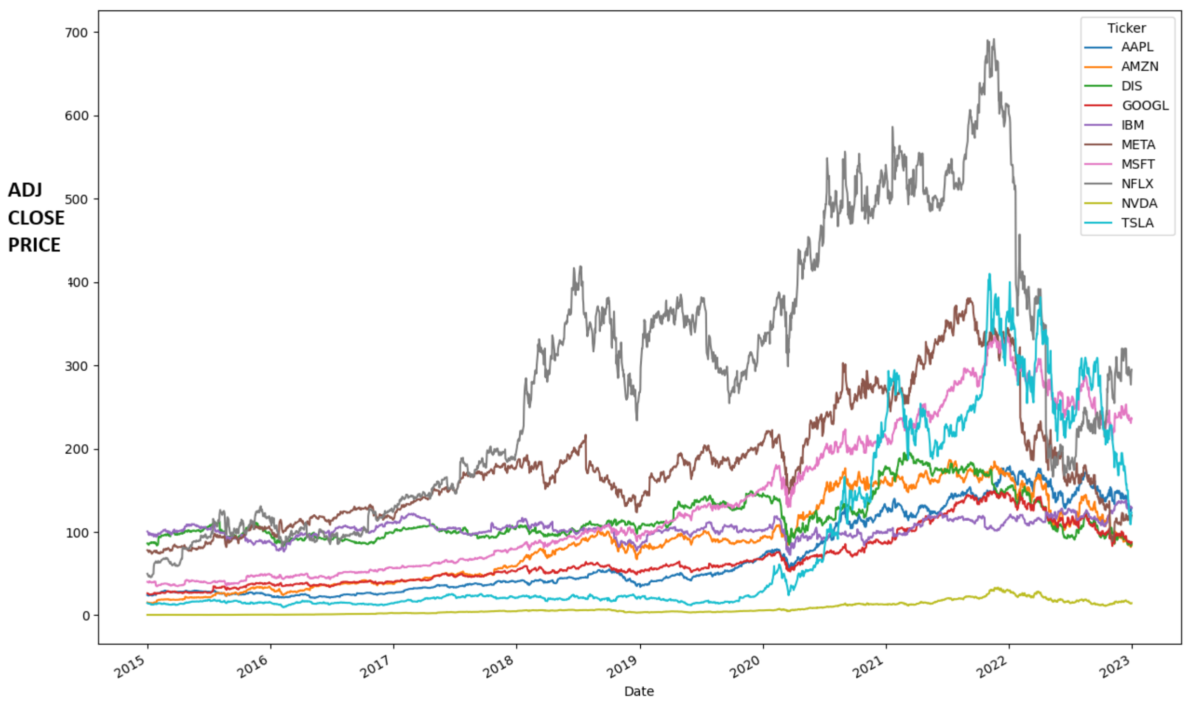
5. **Overcoming the Vanishing Gradient Problem:** LSTMs are designed to avoid the vanishing gradient problem(this is a problem where the gradient's value keeps

decreasing during the backpropogation phase - backpropogation is when the error is calculated at the output layer and it uses the gradient to update the weights. But since the gradients are becoming smaller and smaller, it doesn't have a major impact on the weights.LSTM that don't have this issue, can retain long-term dependencies.

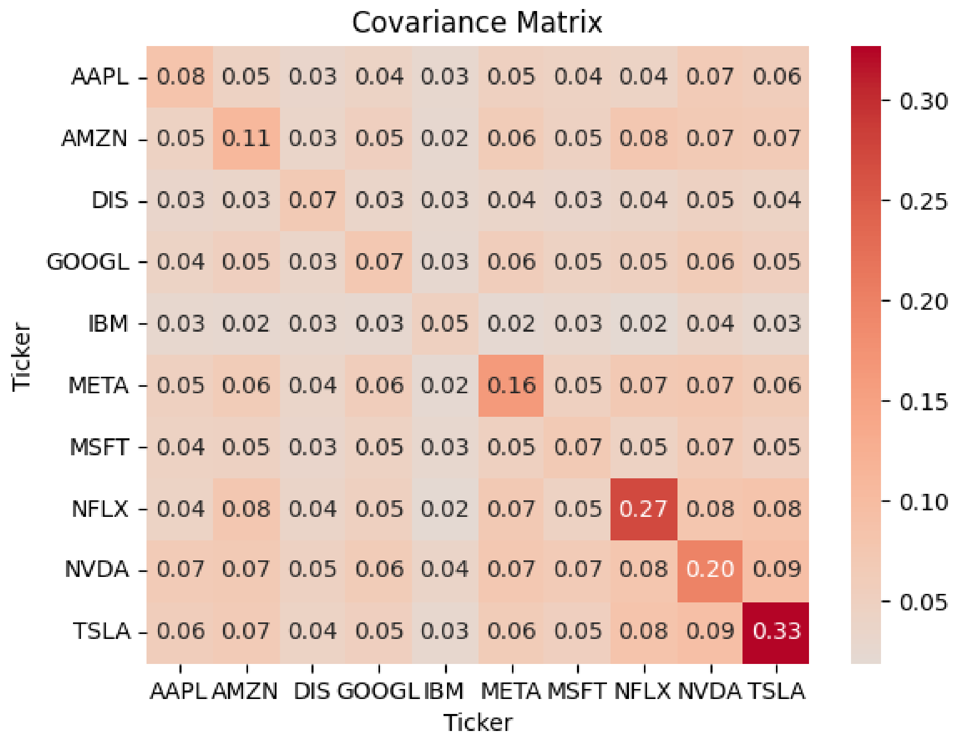
6. **Handling Noise and Volatility:** The stock market is inherently noisy and volatile. However, LSTMs are capable of filtering out irrelevant noise and focusing on significant patterns, enhancing the model's ability to make accurate predictions despite market volatility.

7 Results & Discussion

Let's compare the **Adjusted Closing prices** of our stocks for the period 2015-2023.

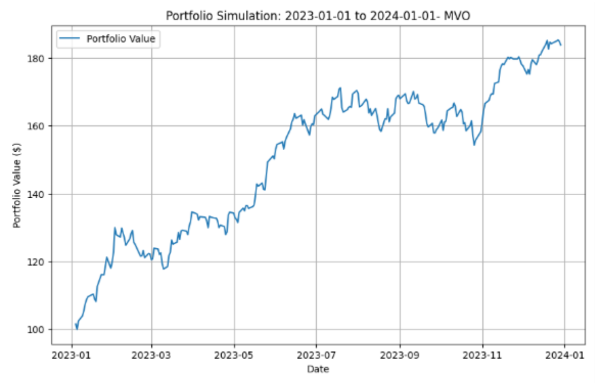


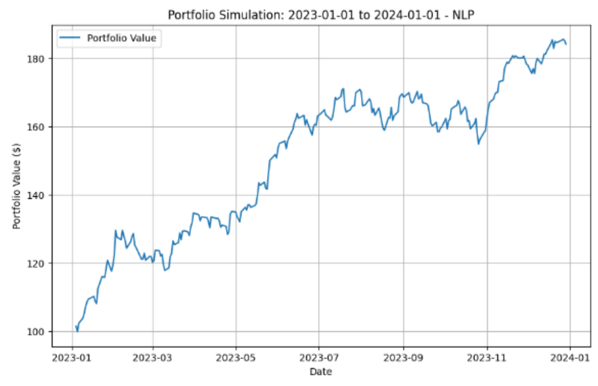
We can observe the variation - we can see that NVIDIA's stock price stayed very steady whilst Netflix and Tesla's stock prices have had big variations throughout their graphs. We would like to observe the covariance or the relationship between any two stocks - this is shown on the covariance matrix. As we can observe from the covariance matrix, most

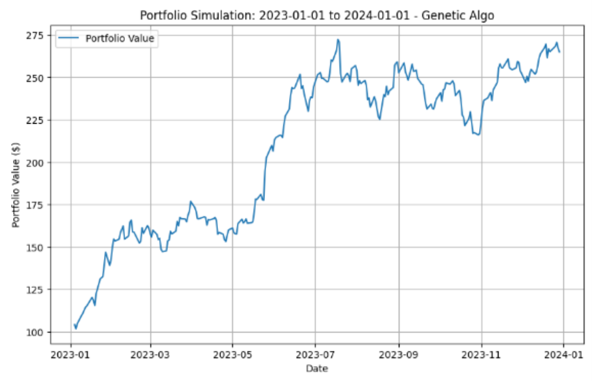


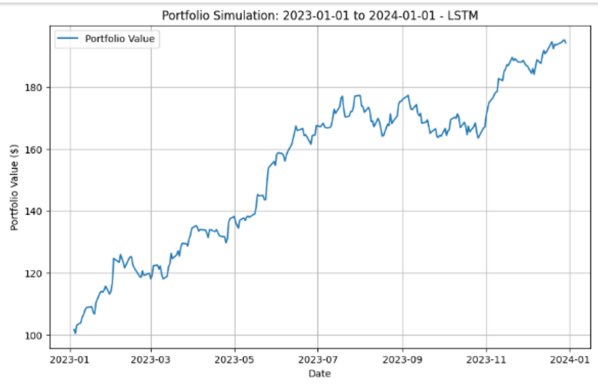
Ticker	AAPL	AMZN	DIS	GOOGL	IBM	META	MSFT	NFLX	NVDA	TSLA
AAPL	0.08	0.05	0.03	0.04	0.03	0.05	0.04	0.04	0.07	0.06
AMZN	0.05	0.11	0.03	0.05	0.02	0.06	0.05	0.08	0.07	0.07
DIS	0.03	0.03	0.07	0.03	0.03	0.04	0.03	0.04	0.05	0.04
GOOGL	0.04	0.05	0.03	0.07	0.03	0.06	0.05	0.05	0.06	0.05
IBM	0.03	0.02	0.03	0.03	0.05	0.02	0.03	0.02	0.04	0.03
META	0.05	0.06	0.04	0.06	0.02	0.16	0.05	0.07	0.07	0.06
MSFT	0.04	0.05	0.03	0.05	0.03	0.05	0.07	0.05	0.07	0.05
NFLX	0.04	0.08	0.04	0.05	0.02	0.07	0.05	0.27	0.08	0.08
NVDA	0.07	0.07	0.05	0.06	0.04	0.07	0.07	0.08	0.20	0.09
TSLA	0.06	0.07	0.04	0.05	0.03	0.06	0.05	0.08	0.09	0.33

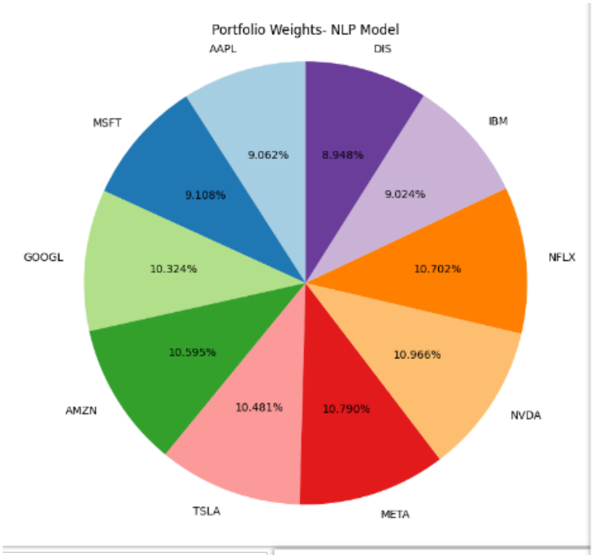
assets are unrelated. This is intuitive - a change in the price of Apple shouldn't change the price of Amazon. We would like to check if there is some relation between any assets and broadly, we can conclude that there's no relation since the covariances are pretty close to zero. On the diagonal, we can observe the **variance** of the stock - generally the lower this value is, the higher the probability of values to be near the mean. In simpler terms, a higher variance is pretty volatile - and we can take this as *risk*.

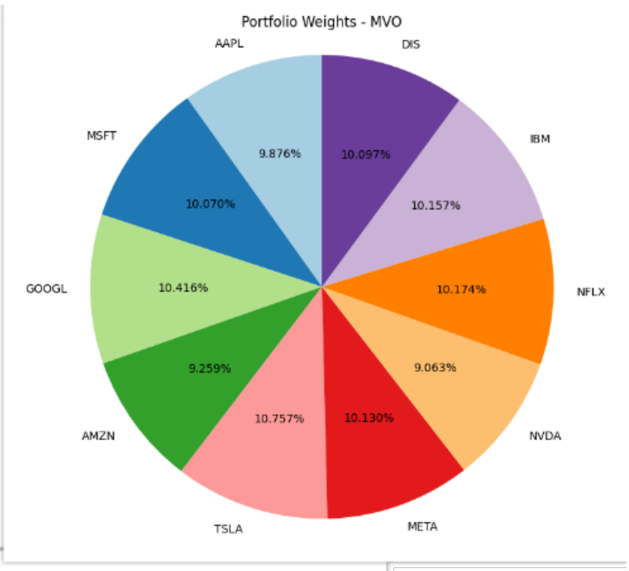


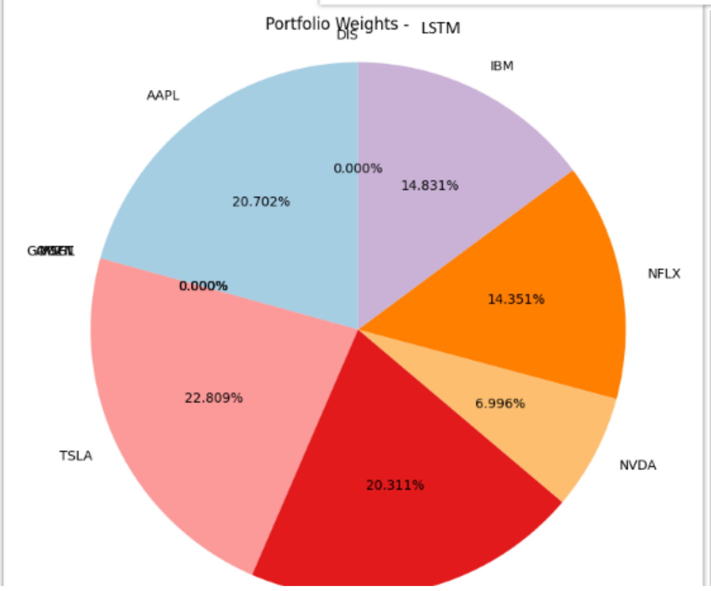




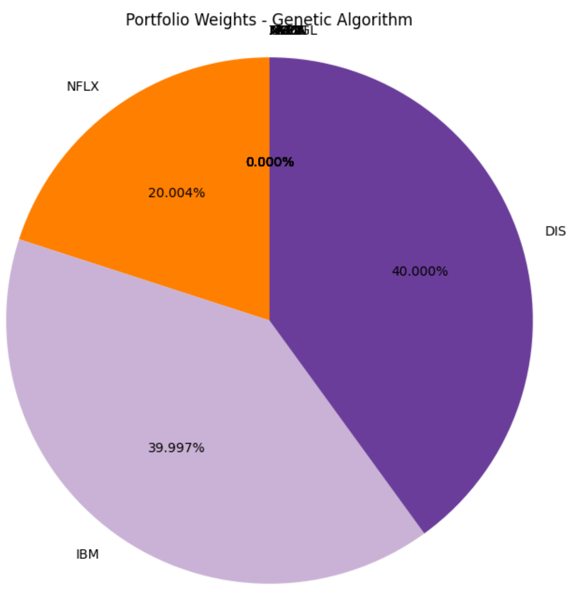








Furthermore, let's observe the stock allocation. Let's observe how much the MVO, NLP, LSTM and Genetic Algorithm has allocated capital per stock from the pie chart. Here we observe that the capital allocation is very even in both the MVO and NLP models. As we shall see later, their Max Drawdown will be relatively low, as they have diversified by minimizing variance. LSTM's, whilst still pretty even, have certain stocks at virtually no investment - one of the reasons is because certain stocks may have not performed well when being screened by the loss function, the Sharpe ratio. Since unlike MVO we are passing the sequence of data with other technical features, these features also have a weight and they also contribute to capital allocation which causes this slightly unbalanced allocation. Looking at the Genetic Algorithm, we observe that it's allocation is radical. It really only has 3 of the 10 companies contributing to the portfolio. So instead of diversifying, this GA model suggests a high-risk, high-reward investment. Despite having a penalty of no more weight having more than 40%, 2 weights have 40% and one has 20%. The 2 stocks with the 40%, IBM and Disney both have extremely low variances (and covariance in general) which satisfies the fitness function. The 20% is with Netflix, although with high variance, it also has some of the highest returns (check the first figure) and therefore has a pretty high return per unit of risk.



Evaluation
The evaluation is done by backtesting - this is the process whereby the weights that the model assigned for each stock is used in new, more recent data to see the profit this model would produce. We simulate an investment of $100 20 times for each model, and find the average weights of each model and use those weights in analysis.

1. **Final Portfolio Value:** Total value of the portfolio at the end of the backtest period, starting from an initial investment of $100. We can see how much the initial investment has grown (or shrunk) over the backtest period.

2. **Cumulative Returns:** This metric shows the overall percentage change in the portfolio's value over backtest period, incorporating both gains and losses.

3. **Sharpe Ratio:** The Sharpe ratio measures the risk-adjusted return of the portfolio, as shown in the Key Concept section below. It indicates how much return the portfolio has earned for per unit risk. A higher Sharpe ratio implies better risk-adjusted performance - what we aim to maximize.

4. **Max Drawdown:** Max drawdown represents the largest peak-to-trough decline in the portfolio's value over the backtest period. The higher this is, the more risky the portfolio.

Let's now compare these 3 over One week, One month, 6 months and One Year - including Final Portfolio is pretty much the same as including Cumulative returns, so to avoid redundancy, I'm just using Cumulative Returns.

Table 1: Cumulative Returns

Model	Week	Month	6 Months	Year
MVO	1.54	6.88	41.42	84.01
GA	3.31	13.69	86.34	164.89
NLP	1.64	6.94	41.11	84.10
LSTM	1.84	7.66	49.27	94.26

The cumulative returns across the four models reveal distinct differences in their ability to generate wealth. GA (164.89%) outperforms the other models, it's a literal 1.6 time increase! LSTM (94.26%) follows second, leveraging its deep learning architecture to capture long-term trends. MVO (84.01%) and NLP (84.10%) provide similar returns, with MVO relying on static optimization techniques and NLP focusing on sentiment analysis from news/text data. The outperformance of GA can be attributed to its ability to span the whole of the solution space using dynamic mutation to evolve and diversify portfolios effectively and not getting stuck in local minima, retaining the best of the population. Although LSTM is a full-blown neural network (and we have stacked it), stock markets can be volatile and comparing only long-term may not necessarily work optimally if changes are rapid. There is a high possibility that a part of the solution space hasn't been found by the LSTM that the GA optimized on, as it covers an expanse of the solution space. In conclusion, GA is the most successful model due while LSTM offers solid returns too, and both MVO and NLP provide the same value pretty much. This behaviour is consistent throughout the time periods, whether as short as a week or long as a year.

Table 2: Max Drawdown

Model	Week	Month	6 Months	Year
MVO	29.06	133.67	835.78	1609.67
GA	103.49	489.87	2952.29	5627.63
NLP	30.24	132.99	840.51	1620.24
LSTM	26.30	119.80	698.17	1378.20

GA has the highest risk by a large margin, with a **yearly** drawdown of **5627.63%**. Fundamentally, genetic algorithms explore a large range of the solution space and may converge on highly risky strategies - this high risk may not be a choice for more safer investors. **NLP** follows with a **yearly** drawdown of **1620.24%**, indicating moderate risk compared to the GA, but it's definitely going to keep the investors in doubt. Its higher risk may stem from the complex nature of the model, that do not always correspond well to financial data or that the majority of the news and articles may not be portraying the stocks in it's actual form. Also, the fact that maybe contextual sentiment analysis could have done a better job could also be a reason for this risk. **MVO** falls next with a **yearly** drawdown of **1609.67%**, showing not a major change from **NLP**. This could be attributed to the model's reliance on mean-variance optimization, which may overestimate the stability of asset returns, leading to declines. Finally, **LSTM** has the lowest drawdown at **1378.20%** over the **year**, suggesting that it is the best at managing risk, despite showing notable volatility. This could be because of the model's capacity to learn long-term dependencies in data and adjust its predictions, helping it adapt and avoid some of the risky behaviors seen in other models. This pattern is observed consistently over time, indicating that the trends are genuine and no coincidence.

Table 3: Sharpe Ratio

Model	Week	Month	6 Months	Year
MVO	0.40	0.19	0.19	0.18
GA	0.61	0.32	0.29	0.24
NLP	0.50	0.21	0.20	0.18
LSTM	0.32	0.25	0.21	0.21

GA has the highest Sharpe ratio, indicating its ability to achieve higher returns but with more risk. However, its Sharpe ratio declines over time, from 0.29 at six months to 0.24 at one year, suggesting that as time goes on, the return per unit of risk might steadily decrease. **LSTM**, on the other hand, exhibits a stable Sharpe ratio of 0.21 over both six months and one year, indicating that it manages risk more effectively and offers more consistent returns, albeit at a lower level compared to GA. **MVO** and **NLP** have similar Sharpe ratios, with **MVO** showing a slightly lower value than **NLP** across all time periods. However, **NLP** seems to perform similarly to **MVO**, with similar declines in the Sharpe ratio over time, suggesting both models offer relatively comparable risk-return profiles. While **GA** provides high short-term returns at the expense of higher risk, **LSTM** offers more stable, albeit lower, returns. **MVO** and **NLP** lie in between, offering moderate risk-adjusted returns with slight differences in performance.

Challenges:

When dealing with Genetic Algorithms, I experimented with loss functions that punishes high variance or rewards high returns, but it took experimentation to force the use of Sharpe ratio, as it can do both profit maximization whilst minimizing returns. Furthermore, the initial few runs gave worse results than the Mean Variance Optimization model. This was because the Genetic Algorithm had no mutation, so I had to come up with a dynamic mutation rate that initially introduces a lot of variability, and then we stick to a standard 10% mutation.

With the use of LSTM's, I had to ensure all the data was into a sequence of equal lengths, which wasn't needed for any other models. Also, feature engineering to create Simple and Exponential Moving Averages, Relative Strength Index and other such variables had to be made because the initial model outputs were not satisfactory. In addition to all of this, the primary runs of the LSTM model wasn't providing satisfactory results and so we needed to stack models, adding to more of the time complexity.

For sentiment analysis, I wanted to analyze tweets but their API's had to be paid for (for getting information) and when making my own web scraper I realized that there are restraints that stop robots from crawling websites. As a final resort, we ended up using NewsAPI's free tier API that gave us access to top sites and their headlines and articles.

8 Conclusion

The backtesting results highlight the unique performance traits of each model. **Genetic Algorithms** achieved the highest cumulative returns of 164.89% for a one-year period and short-term Sharpe ratios up to 0.61, emphasizing its potential for high profitability. However, this came with significant risk, as shown by its max drawdown of 5627.63% - which is truly a deterrent for any safe investor. In contrast, **LSTM** displayed the most stable performance, with a relatively low max drawdown of 1378.20% and a consistent Sharpe ratio of 0.21 for the year, making it more suitable for risk-averse, long-term strategies. It made returns 94.26% which is almost doubling the investment. **MVO** and **NLP** demonstrated balanced outcomes, both achieving moderate returns of 84.01% and 84.10%, respectively, with max drawdowns of 1690.67% and 1620.24%. Their Sharpe ratios of 0.18 for the year indicate steady risk-adjusted returns. Ultimately, **GA** is best for aggressive, short-term strategies, while **LSTM** is ideal for stable, long-term investment goals, with **MVO** and **NLP** offering a middle ground for moderate risk and return preferences, although both sub-optimal. The agressive trader will seek to go to use GA's while more balanced traders may choose to go for LSTM for it's long term advantages.
In the future we can integrate several ESG indicators that measure the sustainability of the company, as more demand is increasing for eco-friendly products. Furthermore we can integrate marco-economic indicators like *inflation* and *employment stats* to further improve our results. Use of Reinforcement Learning, where the agent learns based on the stock market may also be more adaptable and may have learned the volatility of the market.

9 References

1. Markowitz, H. (1952). *Portfolio Selection.* Link
2. Elton, E., Gruber, M., Brown, S., & Goetzmann, W. (2009). *Modern Portfolio Theory and Investment Analysis.* Link
3. Goldberg, D. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning.* Link
4. Grinold, R., & Kahn, R. (2000). *Active Portfolio Management.* Link
5. Hochreiter, S., & Schmidhuber, J. (1997). *Long Short-Term Memory.* Link
6. Fischer, T., & Krauss, C. (2018). *Deep Learning with Long Short-Term Memory Networks for Financial Market Predictions.* Link
7. Bollen, J., Mao, H., & Zeng, X. (2011). *Twitter Mood Predicts the Stock Market.* Link
8. *Financial Sentiment Analysis on News and Reports Using Large Language Models and FinBERT.* Link
9. *Long Short-Term Memory (LSTM), Clearly Explained.* (2021). Link
10. *Genetic Algorithms Explained By Example.* (2020). Link

10 Appendices

(a) *Mean Variance Optimization Code*Link

(b) *LSTM Code*Link

(c) *Genetic Algorithm Code*Link

(d) *Sentiment Analysis NLP Code*Link

(e) *Portfolio Optimization by LSTM with a Selection of Six Stocks* Reference