

**GROUP WORK PROJECT # 3:**  
**Group Number: 7636**

MScFE 642: Deep Learning for Finance

FULL LEGAL NAME	LOCATION (COUNTRY)	EMAIL ADDRESS	MARK X FOR ANY NON-CONTRIBUTING MEMBER
CHIEDOZIE AUGUSTINE IKE-OFFIAH	NORTH CYPRUS (TURKEY)	chiexaustine2005@gmail.com	
ARIFIN LIAUW	INDONESIA	james.arifin@gmail.com	
SHIVANSH KUMAR	INDIA	shivansh.business23@gmail.com	

**Statement of integrity:** By typing the names of all group members in the text boxes below, you confirm that the assignment submitted is original work produced by the group (excluding any non-contributing members identified with an "X" above).

Team member 1	CHIEDOZIE AUGUSTINE IKE-OFFIAH
Team member 2	ARIFIN LIAUW
Team member 3	SHIVANSH KUMAR

Use the box below to explain any attempts to reach out to a non-contributing member. Type (N/A) if all members contributed.

**Note:** You may be required to provide proof of your outreach to non-contributing members upon request.

N/A

Step 1

---

- A. For this step, we will gather information on time series of the prices of BBRI.JK as our choice of security which is categorized as equity. We will put any relevant information that we gather for this securities including the analysis that we conducted on it. The time series is that of the closing price for the period from 1st January 2017 to 10th December 2024. The series is as presented in Figure 1.



Figure 1: Time series plot for the closing prices of BBRI.JK

Table 1 briefly summarizes the basic statistical properties of the time series, including its mean, skewness, and kurtosis.

Table 1: Relevant Summary Statistics

Summary Statistics	Value
Count	1971
Mean	3883.60
Standard Deviation	964.17

Figure 2 depicts the histogram of the prices to give an idea of how the closing prices vary. As we can see, the shape of the distribution is not exactly normal. However, it still looks like what we can expect from security prices.

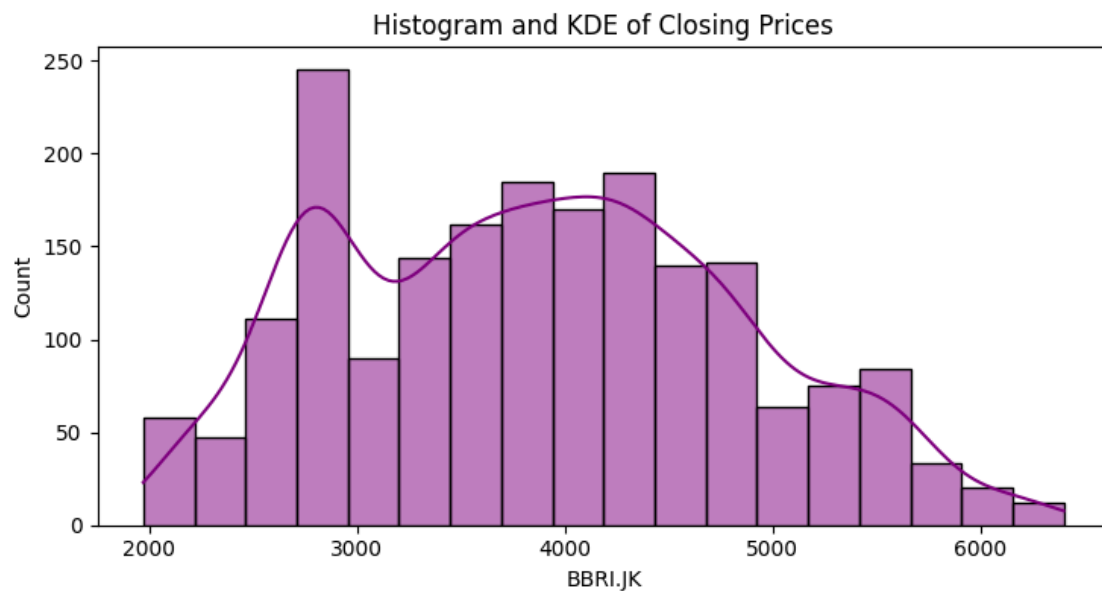


Figure 2: Histogram of the closing prices of BBRI.JK

Still trying to understand the distribution of the series, we can now observe also the QQ plot and autocorrelation of the prices. This is represented together as in Figure 3a. As we can observe from the QQ plot, we see indeed that the series is close to normality as it almost fits on the diagonal of the plot. The outlier region in the QQ plot may have been due to the COVID period when prices of securities sort of behaved erratically. Observing from the ACF plot, we can see that the values in the time series plot are highly correlated (using a lag of 30). To confirm this, we can use the ADF test to see the relevant statistics. In Figure 3, we also incorporate the box plot as our initial analysis.

The Augmented Dickey-Fuller Test Results are as follows:

augmented Dickey-Fuller Test Results:

```
{'ADF Statistic': -1.988175363918835, 'p-value': 0.29178173603060364, 'Critical Values': {'1%':  
-3.4336805486772994, '5%': -2.863011243118318, '10%': -2.567553229292686}}
```

The p-value is greater than 0.05, suggesting that the series is non-stationary. This confirms our initial suspicion.

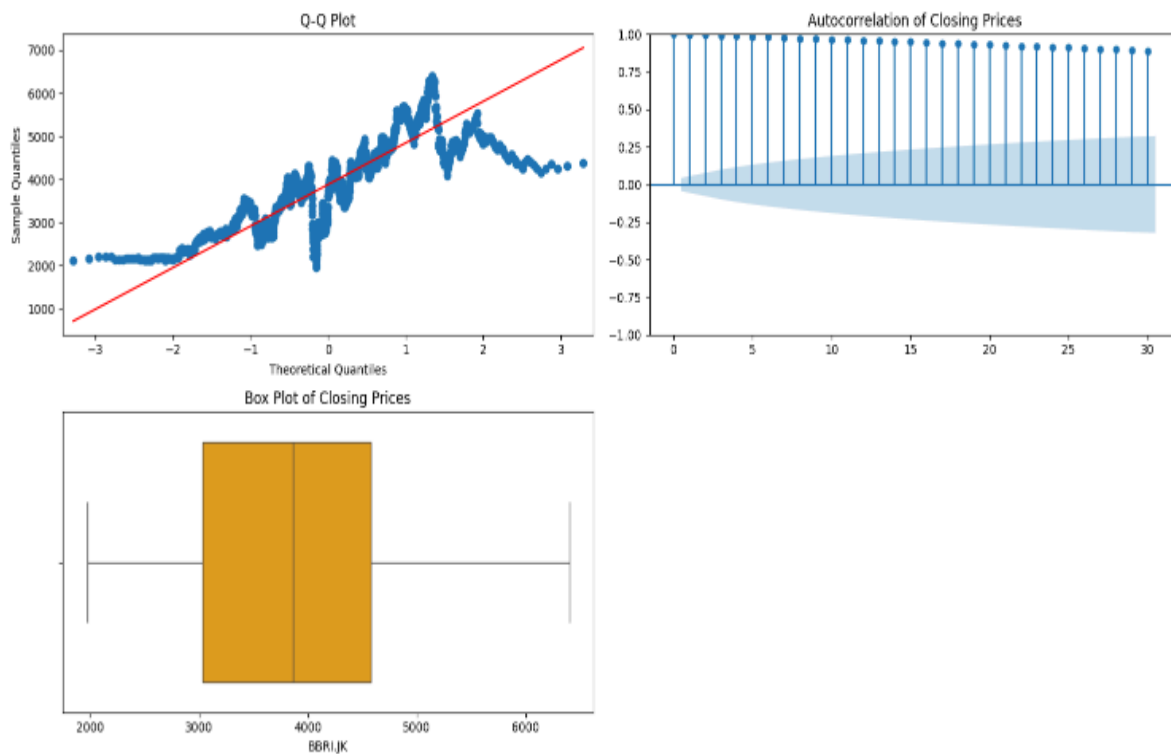


Figure 3: QQ plot and the autocorrelation of the closing prices for the BBRI.JK stock

- B. In this step, we will transform the time series to a logarithmic time series to ensure we build a model based on a stationary, not non-stationary, time series. Upon completing the transformation, we perform statistical analysis to capture the transformed time series' mean, standard deviation, skewness, and kurtosis.

Following is the result of the transform time series for BBRI.JK in figure 4. that we used in this work:

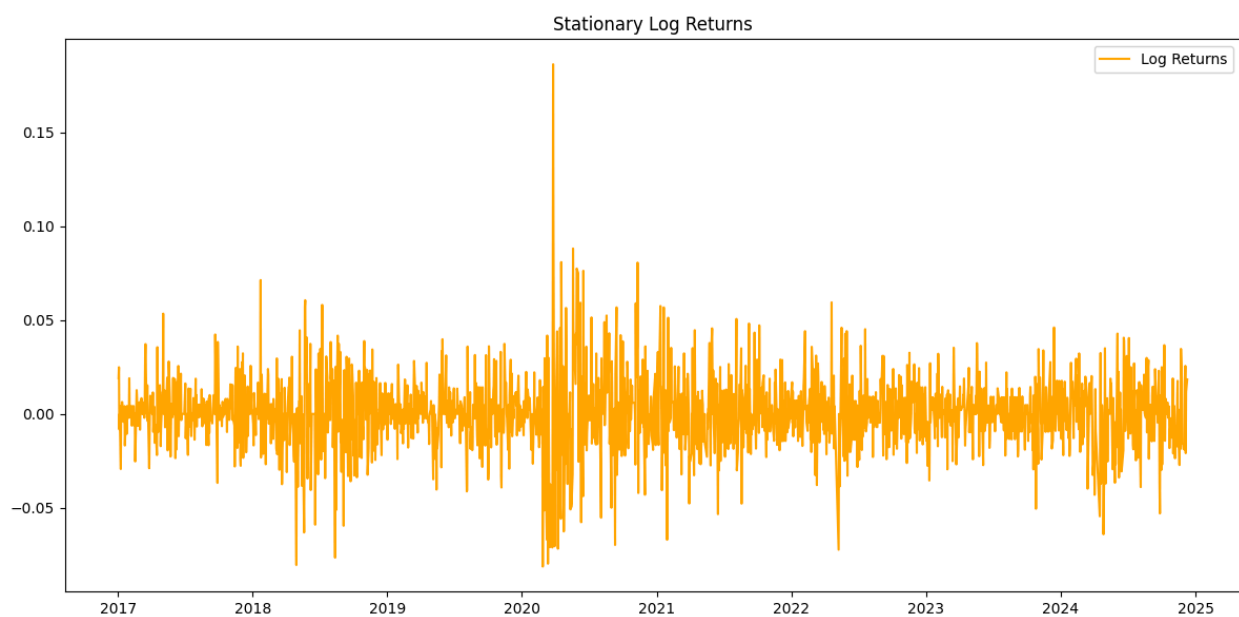


Figure 4: Log Returns plot

Based on this transformed time series, we got the following statistics:

Table 2.: Summary Statistics

<b>Mean</b>	0.00036886
<b>Standard deviation</b>	0.019448
<b>Skewness</b>	0.38523585
<b>Kurtosis</b>	6.60426795

Based on the above statistic summary, the average return for BBRI.JK during the period that we evaluate is around 0.36%, which is very close to zero, indicating that the return is oscillating around that mean with a standard deviation of 1.95%, and the return is slightly positively skewed with high kurtosis (leptokurtic), indicating there is a sharp peak with fat tails.

Then we constructed the histogram and Q-Q plot to visualize the distribution of the stationary time series which resulted in this graph in figure 5.

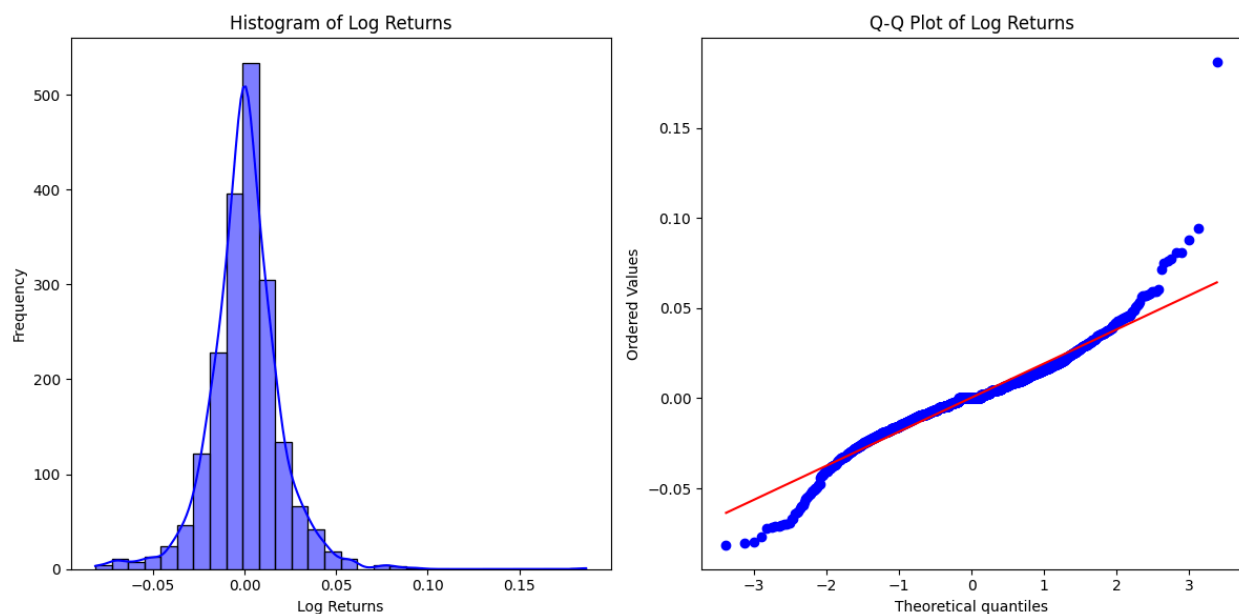


Figure 5: Histogram and Q-Q plot of log returns

The above histogram plotting and Q-Q plot visualize the statistical data above, confirming the positive skewness distribution, and there is an outlier. The outlier based on the log returns plots happened in 2020 in the COVID era.

Based on the same analysis that we have done on GWP1 Deep Learning for Finance, we conclude that the logarithmic transformation resulted in a stationary time series.

Based on this Log time series, we build a predictive model to forecast the return of the BBRI.JK that we have agreed to choose as a security that we will perform our analysis for this GWP. Following is the result of the plot of the predictive model to forecast the result as shown in Figure 6.

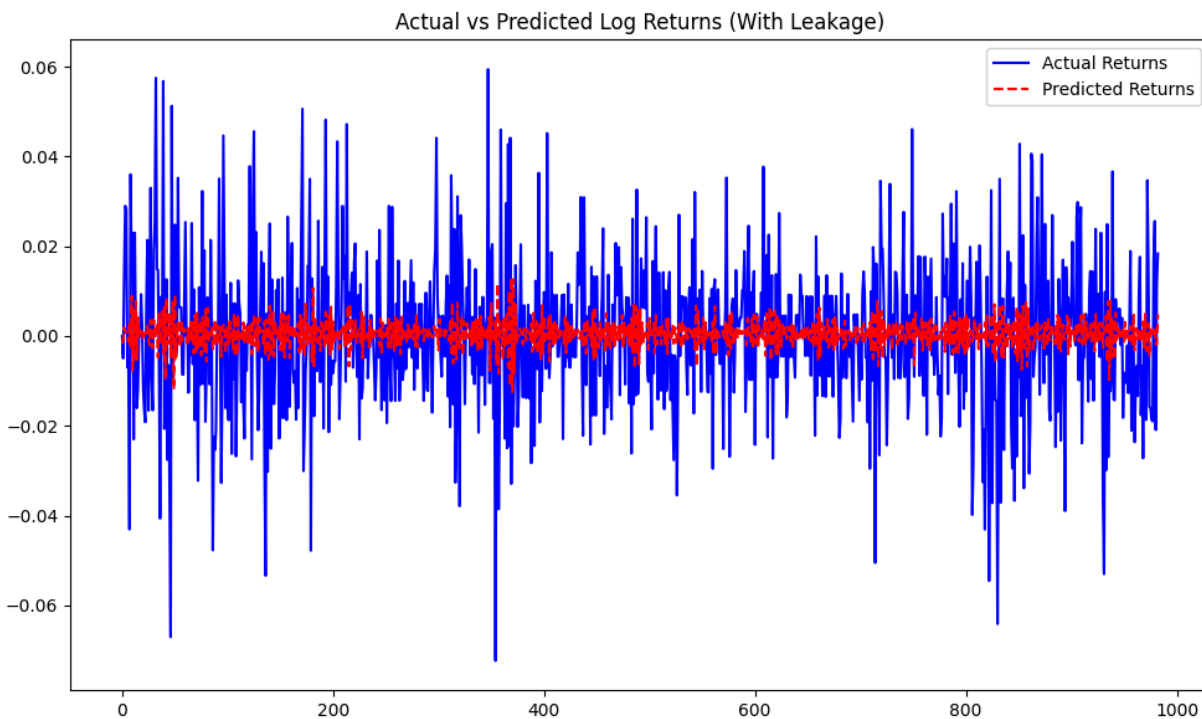


Figure 6.: Actual vs Predicted Log Return with leakage of information

C. For this section we are going to build 3 models and predict the time series, using a single train/test split. The three models are:

- an MLP
- an LSTM
- a CNN based on GAF

**Data processing and feature engineering:**

We will first do some data processing in which we will convert the whole data frame into log values, and after that, we will do some feature engineering in which first we will find the returns and then we will calculate the rolling mean returns of 10, 25, 60, 90, 120, and 240 (Ret10\_i, Ret25\_i, Ret60\_i, Ret90\_i, Ret120\_i, and Ret240\_i) in Figure 7 and delete all the other respective features like Close, Adj Close, Volume, etc. We will use these calculated features as our main features for prediction. As we are using these features, they are providing us information leakage as we are using future data in our rolling window. This type of data can lead to artificially optimistic backtesting results, and this is what we have to check as to how the information leak affected our prediction with different types of train-test splits that we are going to see. But for this section, we are using a single simple train-test split of 80:20.

Now we will understand how models are working:



Price	Ret	Ret10_i	Ret25_i	Ret60_i	Ret90_i	Ret120_i	Ret240_i
Ticker							
Date							
2024-11-25	8.405144	8.393892	8.431227	8.493474	8.490823	8.474690	8.544709
2024-11-26	8.389360	8.391645	8.426921	8.490770	8.489605	8.474503	8.543895
2024-11-28	8.373323	8.386032	8.422302	8.487799	8.488483	8.474464	8.542975
2024-11-29	8.354674	8.380316	8.417268	8.484760	8.487177	8.474175	8.542016
2024-12-02	8.335671	8.373369	8.411891	8.481242	8.485848	8.473842	8.540977
2024-12-03	8.352319	8.370807	8.407264	8.477760	8.484562	8.473629	8.540048
2024-12-04	8.377931	8.370578	8.404168	8.474866	8.483515	8.473706	8.539244
2024-12-05	8.357024	8.368948	8.400151	8.471383	8.482494	8.473864	8.538295
2024-12-06	8.368693	8.370350	8.395844	8.468335	8.481603	8.474280	8.537279
2024-12-09	8.387085	8.370122	8.393115	8.465593	8.480727	8.474511	8.536378

Figure 7: Features calculator using our dataset

As we have to check just the phenomena of the effect of info leakage, we are not going to build very elaborate models but simple toy models.

**MLP Model:** For our MLP model, we have two hidden layers with 64 and 32 neurons, each using the ReLU activation function, followed by a single output neuron for regression. It is trained on scaled input features ( $X_{\text{train\_scaled}}$ ) and the corresponding scaled target variable ( $y_{\text{train\_scaled}}$ ). The MLP is best suited for problems where the relationship between input variables and the target is non-linear but not inherently sequential, such as predicting outcomes from tabular data or engineered features. You can see the model summary in Figure 8.

Model: "sequential\_87"

Layer (type)	Output Shape	Param #
dense_188 (Dense)	(None, 64)	384
dense_189 (Dense)	(None, 32)	2,080
dense_190 (Dense)	(None, 1)	33

Total params: 7,493 (29.27 KB)  
Trainable params: 2,497 (9.75 KB)  
Non-trainable params: 0 (0.00 B)  
Optimizer params: 4,996 (19.52 KB)

Figure 8: MLP Model Summary

**LSTM model:** In our LSTM model, we used two LSTM layers with 64 and 32 units, where the first layer returns sequences to feed into the second layer. The input data (X\_train\_window) is structured as sliding windows of window size = 30 of the time series, ensuring that the temporal context is preserved. The model predicts a single target value at each time step. you can see the model summary in Figure 9.

Model: "sequential\_88"

Layer (type)	Output Shape	Param #
lstm_44 (LSTM)	(None, 30, 64)	17,920
lstm_45 (LSTM)	(None, 32)	12,416
dense_191 (Dense)	(None, 1)	33

Total params: 91,109 (355.90 KB)  
Trainable params: 30,369 (118.63 KB)  
Non-trainable params: 0 (0.00 B)  
Optimizer params: 60,740 (237.27 KB)

Figure 9 : LSTM model Summary

**CNN model based on GAF:** for this a CNN (Convolutional Neural Network) model is applied to time-series data that has been transformed into image-like representations using the Gramian Angular Field (GAF) technique. This transformation encodes the temporal structure of the time series into a 2D matrix by preserving the angular relationships of data points. The resulting GAF images (X\_train\_gaf) are processed through a CNN architecture with two convolutional layers, max-pooling layers, and a fully connected output layer. The CNN extracts spatial patterns from the GAF matrices, allowing it to capture both local and global interactions in the time series. you can see the model summary in Figure 10.

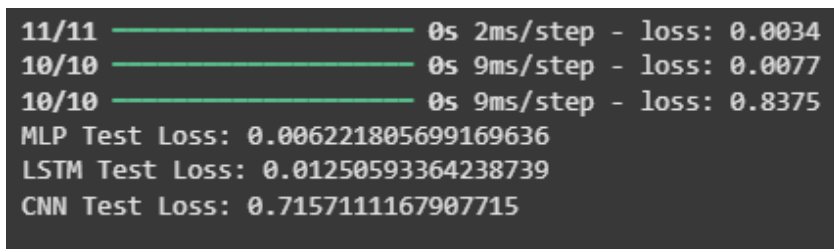
Model: "sequential\_89"

Layer (type)	Output Shape	Param #
conv2d_44 (Conv2D)	(None, 28, 28, 32)	320
max_pooling2d_44 (MaxPooling2D)	(None, 14, 14, 32)	0
conv2d_45 (Conv2D)	(None, 12, 12, 64)	18,496
max_pooling2d_45 (MaxPooling2D)	(None, 6, 6, 64)	0
flatten_29 (Flatten)	(None, 2304)	0
dense_192 (Dense)	(None, 64)	147,520
dense_193 (Dense)	(None, 1)	65

Total params: 499,205 (1.90 MB)  
Trainable params: 166,401 (650.00 KB)  
Non-trainable params: 0 (0.00 B)  
Optimizer params: 332,804 (1.27 MB)

Figure 10 : CNN model Summary

Here we have calculated the losses accross all models to understn which model works best with this train/test split. Our results suggest that MLP model have the lowest loss (0.006) suggesting simpler model working good in this dataset. The LSTM model is the second best perfromer with loss of 0.012 indication that temporal dependencies exist but might not dominate the dataset's structure. And at last the CNN based on GAF model have a very high loss of 0.715 suggesting this model is not suitable and struggling with this preductive task. The compiled model result is in Figure 11 below.



```
11/11 ————— 0s 2ms/step - loss: 0.0034
10/10 ————— 0s 9ms/step - loss: 0.0077
10/10 ————— 0s 9ms/step - loss: 0.8375
MLP Test Loss: 0.006221805699169636
LSTM Test Loss: 0.01250593364238739
CNN Test Loss: 0.7157111167907715
```

Figure 11: Compiled model losses result

#### **D. Summary of the Result for the Backtesting Strategies Implemented for the Three Models**

In this section, we developed and backtested three trading strategies for each model, including the MLP, CNN with GAF, and LSTM models. Throughout this work, we implement three trading strategies, namely:

- a. The buy-and-hold strategy is a passive strategy that does not involve any trading actions. It just helps us obtain the returns results throughout the backtesting period. This serves as a baseline to measure the performance of other strategies we implement.
- b. The long-only strategy takes a long position when the predictions are positive returns; otherwise, it stays out of the market (Sharpe).

- c. Finally, there is the long-and-short strategy, which takes a long position when the model predicts a positive return and takes a short position when the model predicts a negative return (Beaver et al.).

Figure M 1.1 presents the backtested strategy using the predictions that were obtained using the MLP framework. It is important to note that these training and testing sets were designed to allow for possible data leakage.

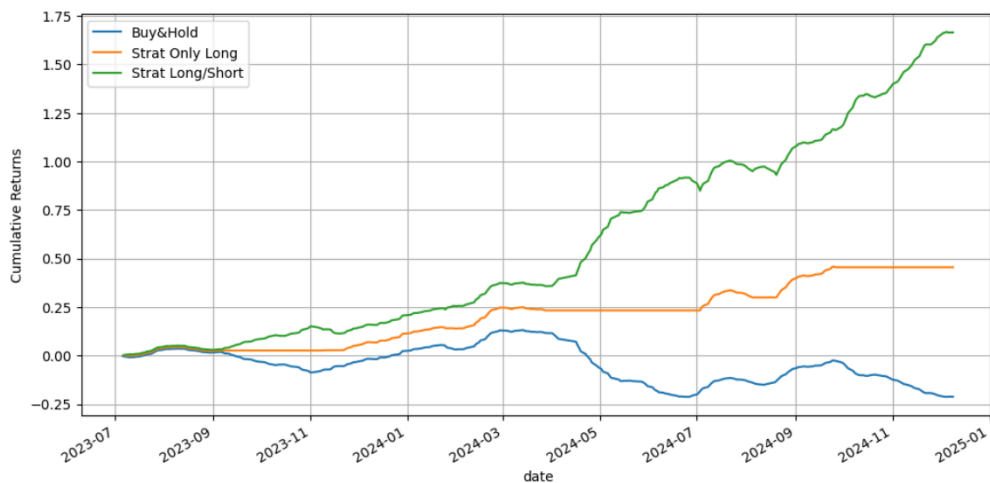


Figure M 1.1: Backtesting the predictions of the MLP.

Table 1.1 summarizes the results of the backtesting using the MLP strategy.

Table 1.1: MLP predictions used to backtest.

Strategy	Cumulative Return (%)	Sharpe Ratio	Conclusion
Buy and Hold	-21.2	-12.8	Performs relatively poorly. This shows that the buy and hold does not adapt to the volatile market. Even having negative cumulative returns.
Long Only	45.51	36.69	These returns are relatively high, and the Sharpe ratio is

			also perfect. The unrealistic cumulative returns could be because of the overfitting issues in the training set or the way the strategy is implemented.
Long and Short	166.65	60.85	Here we see an almost perfect model for the Sharpe ratio, which does not replicate real-world conditions. This could be as a result of data leakage where the model does not actually 'see' new data. The pattern is inherent in the interpretation of the data.

Similarly, figure M 1.2 shows the backtesting strategy when implemented on the predictions of the LSTM model.

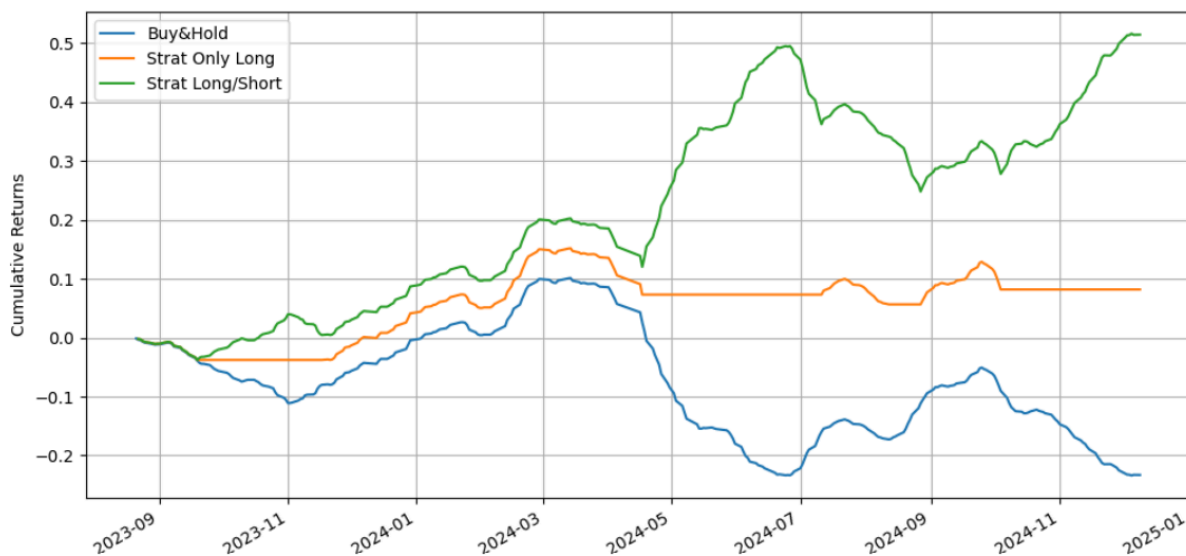


Figure M 1.2: Backtesting the predictions of the LSTM

**GROUP WORK PROJECT # 3:**  
**Group Number: 7636**

Just like in the case of the MLP model, Table M 1.2 summarizes the results of the LSTM prediction while implementing the strategies we discussed.

Strategy	Cumulative Return (%)	Sharpe Ratio	Conclusion
Buy and Hold	-23.30	-14.42	This behavior reflects the nature of the strategy, which does not exactly follow the trends in the market. It represents the cumulative return of the market if we just held on to the stock.
Long Only	8.18	7.67	These returns are relatively moderate, and the Sharpe ratio is also good. This could imply that the model's prediction successfully captured the upward movements of the market.
Long and Short	51.42	23.70	Here we see a good model in terms of the Sharpe ratio. Also, the long and short strategies seem to outperform the long-only strategies. This could imply that the model's predictions successfully capture upward and downward market movement. However, we must be careful of data leakage, where the model does not actually 'see' new data. Generalizability

			might help us confirm our strategy.
--	--	--	-------------------------------------

In the same vein, we backtest using the CNN model's predictions. Figure M 1.3 shows the strategies side by side.

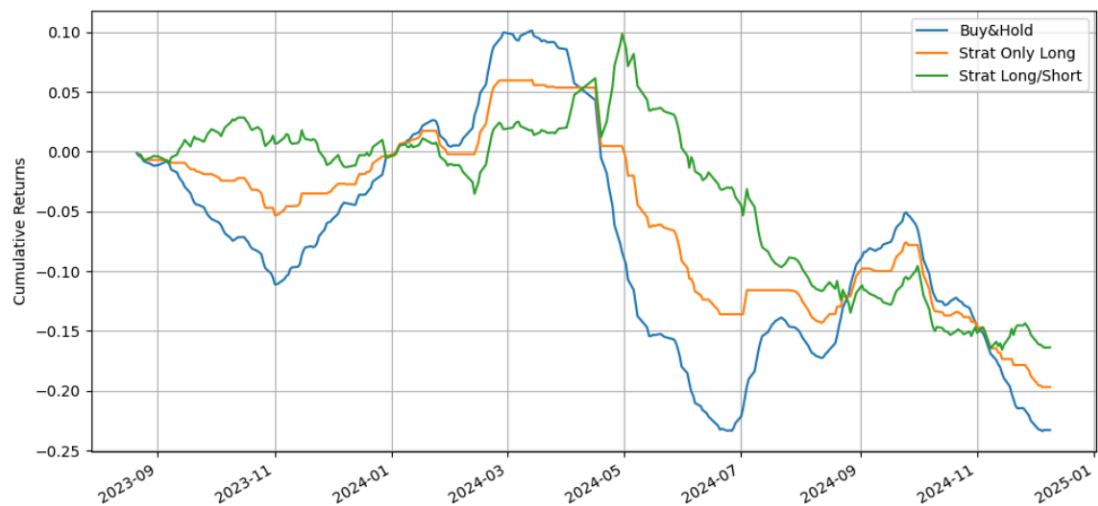


Figure M 1.3: Backtesting using the CNN predictions

Table M 1.3 summarizes the result of this strategy.

Table M 1.3: Summary of the results of the backtest obtained using the CNN predictions.

Strategy	Cumulative Return (%)	Sharpe Ratio	Conclusion
Buy and Hold	-23.30	-14.42	This behavior reflects the nature of the strategy, which does not exactly follow the trends in the market. It is shown by the consistently declining cumulative returns.
Long Only	-19.69	-16.13	These returns are relatively moderate, and the Sharpe ratio



			is also good. This could imply that the model's prediction successfully captured the upward movements of the market. However, it still underperforms when compared to the LSTM. It also fails to capture volatility in the market.
Long and Short	-16.38	-9.59	Here we see a relatively good model in terms of the Sharpe ratio. Also, the long and short strategies seem to outperform the long-only strategies. This could imply that the model's predictions successfully capture upward and downward market movement. Compared to the LSTM, this performs worse. This may be because CNNs are designed and more effective when dealing with image data, as opposed to LSTMs, which are quite effective with sequential data.

Step 2.

A.

Now we will start working on Step 2, In this step our main focus will be on the (Non-anchored) Walk-Forward method that exploits train/test split sets. For 2a. we will use train/test split with 500 observations and for 2b train/test split with 500 observations in each training set and 100 observations in test sets.

Let's First understand the (Non-anchored) Walk-Forward method:

It is a technique used in time-series forecasting and model backtesting in finance world. in this method the training and testing sets are incrementally shifted forward in time, ensuring that predictions are always made using only past data, without any look-ahead bias. The non-anchored method keeps the training window size fixed, discarding older data as the window progresses. This mimics scenarios where only recent data is considered relevant for predictions, often reflecting practical constraints such as memory limitations or the assumption that older data becomes less informative over time.

As we have understood what is our requirement of this step and what method we are using, let's move forward to models, for this step we are again using the same models that we have build for Step 1c. As our goal is to understand the information leakage phenomena and what steps can we take to deal with the leak i.e Walk-forward method in Figure 12.

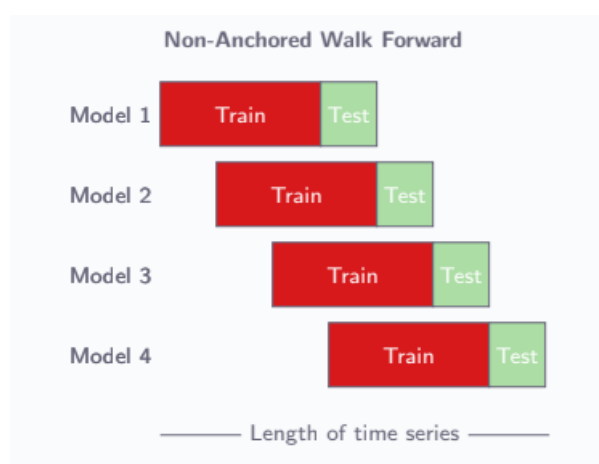


Figure 12: Non-anchored walk-forward method

The results, as you can see in Figure 13 below, highlight distinct performances of the three models, MLP, LSTM, and CNN, on the given task. The MLP achieves the best balance between training and validation losses (0.0172 and 0.0994, respectively), suggesting that it effectively learns patterns in the data while maintaining good generalization to unseen data. The LSTM also performs well, with a low training loss (0.0109) but a slightly higher validation loss (0.1295), reflecting its ability to capture sequential dependencies but hinting at potential overfitting or challenges in generalizing to the validation set. In contrast, the CNN shows the highest losses for both training (0.2465) and validation (0.5067), indicating difficulties in learning and generalizing. This underperformance could be attributed to the complexity of the CNN architecture or the inefficacy of the Gramian Angular Field (GAF) transformation in representing the dataset's temporal patterns. Overall, the MLP emerges as the most effective model for this task, while the LSTM shows promise with improvements, and the CNN approach requires significant re-evaluation.

```
MLP Losses:
[0.01239861 0.13330376]

LSTM Losses:
[0.01090163 0.07674754]

CNN Losses:
[0.20598829 0.52770263]
```

Figure 13: Non-anchored walk-forward method Compiled model losses result.

B.

The results from the non-anchored walk-forward method in Figure 14 demonstrate the strengths and weaknesses of the MLP, LSTM, and CNN models across multiple iterations, each trained on 500 observations and tested on 100. The MLP exhibits consistently low and stable losses, ranging from 0.0008 to 0.0086, making it the most robust and reliable model for this dataset. The LSTM achieves extremely low losses in certain iterations (e.g., 0.0005), highlighting its ability to effectively capture temporal dependencies, but it also shows occasional spikes (e.g., 0.0153), indicating variability in performance that could stem from overfitting or uneven sequential patterns. The CNN, on the other hand, delivers the least reliable performance, with losses ranging from 0.0294 to 0.4488, suggesting that the Gramian Angular Field (GAF) transformation or the CNN architecture is not well-suited to this task. Overall, the MLP stands out as the most consistent model, while the LSTM shows promise with further tuning, and the CNN requires significant adjustments or alternative preprocessing methods to improve its effectiveness.

```
MLP Losses:
[0.00958647 0.010142  0.00252392 0.00170999 0.00121241 0.01324168
 0.00885324 0.00090936 0.00084741 0.00051489 0.00222159 0.00349515]

LSTM Losses:
[0.00906572 0.0050622  0.00241209 0.00162617 0.00143143 0.01161959
 0.00111963 0.00085791 0.00051704 0.00075679 0.00806486 0.00247667]

CNN Losses:
[0.21838503 0.28365371 0.08859477 0.02907122 0.02829161 0.34981591
 0.29996344 0.12945658 0.08293857 0.17451863 0.22784582 0.03943745]
```

Figure 14: Non-anchored walk forward method Compiled model losses result

**C. Backtesting using varying training and testing sets using the non-anchored walk-forward method.**

- a. In this section, first we perform a backtesting strategy using predictions we obtained, first from a walk-forward training and testing set of 500 observations each for the MLP, CNN, and LSTM, respectively. Secondly, we also perform a backtesting strategy using the predictions we obtained from the walk-forward predictions of 500 test observations and 100 train observations. We also discuss the variation of results we obtained from these predictions of each of the three aforementioned models and compare it with the results obtained from the simple models we discussed in section 1. We discuss the results we obtained from this section below.

First, we implement the backtesting of the MLP results. This is shown in Figures M 2.1.1 and M 2.1.2.

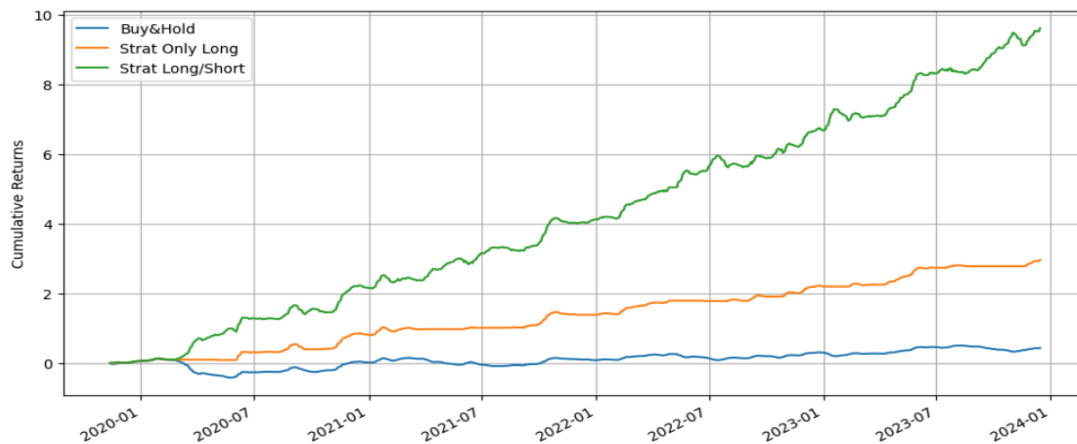


Figure M 2.1.1: Backtesting with the MLP predictions using the non-anchored walk-forward method( $n_{\text{train}}=500, n_{\text{test}}=500$ ).

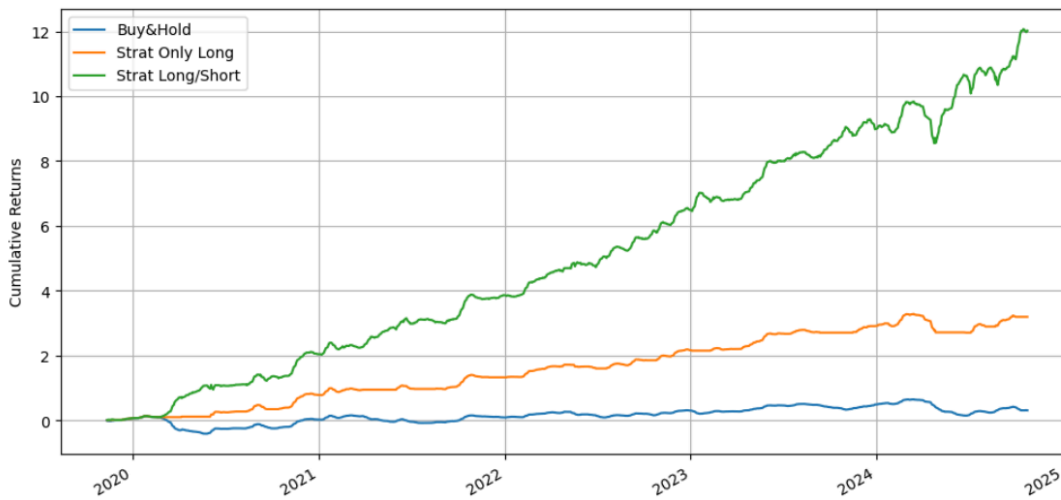


Figure M 2.1.2: Backtesting with the MLP predictions using the non-anchored walk-forward method ( $n_{\text{train}}=500, n_{\text{test}}=100$ ).

Table M 2.1: Summarizing the results from the MLP together with the MLP non-anchored walk-forward method.

Metric	MLP (Single Test-train set)	Non-Anchored Walk Forward MLP ( $n_{\text{test}}=500, n_{\text{train}}=500$ )	Non-Anchored Walk Forward MLP ( $n_{\text{test}}=500, n_{\text{train}}=100$ )
Buy & Hold Return (%)	-21.20	43.89	30.59
Buy & Hold Sharpe	-12.80	5.49	3.59
Strat Long Only Return (%)	45.51	295.93	319.02
Strat Long Only Sharpe	36.69	29.26	25.98
Strat Long/Short Return (%)	166.65	960.09	1201.8
Strat Long/Short Sharpe	60.85	36.38	32.89

As we can see from Table M 2.1, there is an increase in the return for the non-anchored walk-forward method for the buy-and-hold, long-only strategy and long and short strategies. While there is an increase in the Buy and Hold Sharpe, there seems to be a decrease in the Sharpe ratio for both the long-only strategy. This could be because, in the non-anchored walk-forward method, there is potentially more risk involved. For the MLP, we could also see a relatively small Sharpe ratio, which indicated limited risk-adjusted returns. In the long-only strategy, we see that there is an increase in the return when  $n_{\text{test}}$  is decreased to 100. This shows that the strategy is more profitable but riskier in shorter training windows.

There is also an increase in returns in the long and short strategy, which indicates the model's ability to capture both upward and downward trends in the market (Bieganski and Ślepaczuk). Comparing the returns of both MLPs in the long-short strategy for the non-anchored walk-forward method, we immediately can understand the trade-off that occurs between profitability and risk adjustment, as this increase in returns is accompanied by a decrease in the Sharpe ratio. We see that with the smaller test window, the model is responsive to short-term changes in the market, which could and does eventually lead to overfitting, as we can understand from the results of the returns. We can generally see the effect of leakage in our backtesting results, as we have seemingly inflated results of returns.

Next, we observe the results obtained using the simple LSTM predictions and the LSTM predictions in the non-anchored walk-forward method. Figures M 2.2.1 and 2.2.2 show the plots for the different trading strategies.

The results are compared with those of a single test-train split and are presented in Table 2.2.

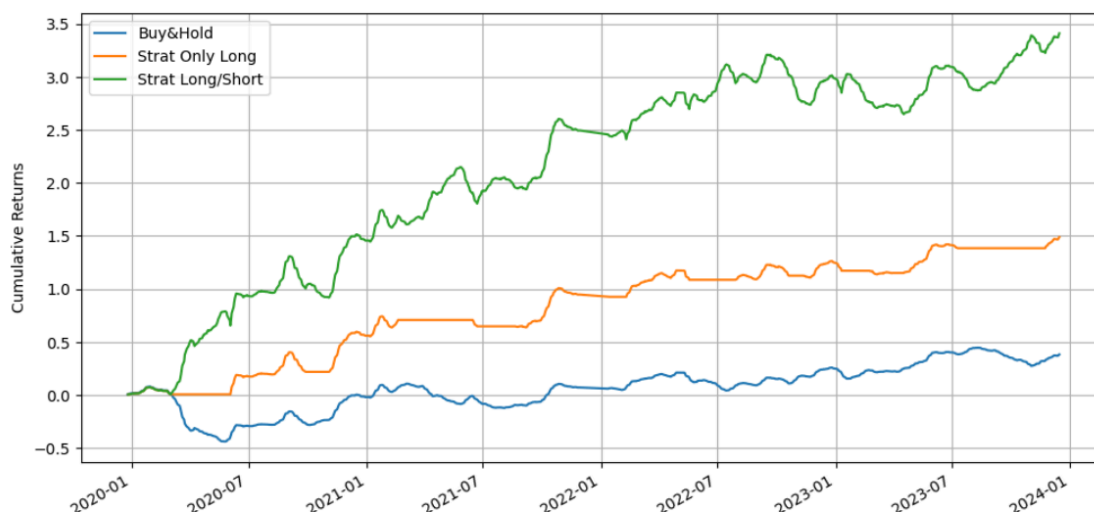


Figure M 2.2.1: Backtesting with the LSTM predictions using the non-anchored walk-forward method  
( $n_{\text{test}}=500$ ,  $n_{\text{train}}=500$ ).

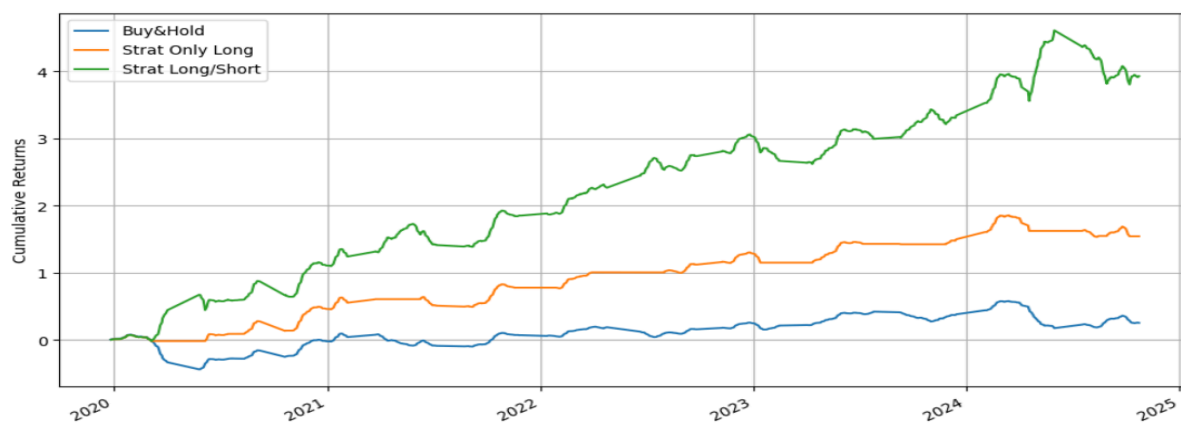


Figure M 2.2.2: Backtesting with the LSTM predictions using the non-anchored walk-forward method  
( $n_{\text{test}}=500$ ,  $n_{\text{train}}=100$ ).



Table M 2.2: Summarizing the results from the LSTM together with the LSTM non-anchored walk-forward method.

Metric	LSTM (Single Test-train set)	Non-Anchored Walk Forward LSTM (n_test=500, n_train=500)	Non-Anchored Walk Forward LSTM (n_test=500, n_train=100)
Buy & Hold Return (%)	-21.20	43.89	25.52
Buy & Hold Sharpe	-12.80	5.49	3.01
Strat Long Only Return (%)	45.51	295.93	154.45
Strat Long Only Sharpe	36.69	29.26	16.68
Strat Long/Short Return (%)	166.65	960.09	393.06
Strat Long/Short Sharpe	60.85	36.38	18.45

Again, just like the results we obtained for the MLP framework, we see that the buy-and-hold strategy performs very poorly relatively. This points out inadequate model performance. Nevertheless, we see that the long-only and the long-and-short strategies perform better than the benchmark buy-and-hold strategy. We see that the LSTM is relatively able to capture temporal dependencies. When we compare the n\_tests of 500 and 100, we understand that there is a decrease in the returns for the long-only strategy. This points out that the LSTM model with the larger training set might capture more robust market patterns. Again, the returns for the long-short trading strategy decrease with a decrease in the n\_test; we understand that even though these results perform well with returns (having high returns),

the decrease in the Sharpe ratio also suggests that there is a trade-off between profitability and risk. We see that the larger training set might have exposed the data to more future data, which may worsen leakage already present. Conclusively, just like with the MLP results, we see the effect of leakage from the high, almost unrealistic returns present in the data.

Finally, we observe the results obtained using the simple CNN predictions and the CNN predictions in the non-anchored walk-forward method. Figures M 2.3.1 and 2.3.2 show the plots for the different trading strategies. Table 2.3 presents the results of the single test-train CNN, compared with the CNN  $n_{\text{train}}$  of 500 and CNN  $n_{\text{train}}$  of 100. Much of the conclusions we derived from both the LSTM and the MLP models are also applicable to the CNN framework. The difference, though, is that we observe negative values for cumulative returns and Sharpe ratios, respectively. This indicates that the baseline prediction for performance is highly susceptible to risks. A decrease in the returns accompanied by a decrease in the Sharpe ratio for the long-only strategy across the decrease in  $n_{\text{test}}$  might be a result of an increase in volatility for the shorter testing window. Due to leakage, we might cause the model not to be applicable in unseen data. Worth mentioning is the fact that GAF might have led to more errors present in the prediction capability, which could lead to poor model performance.

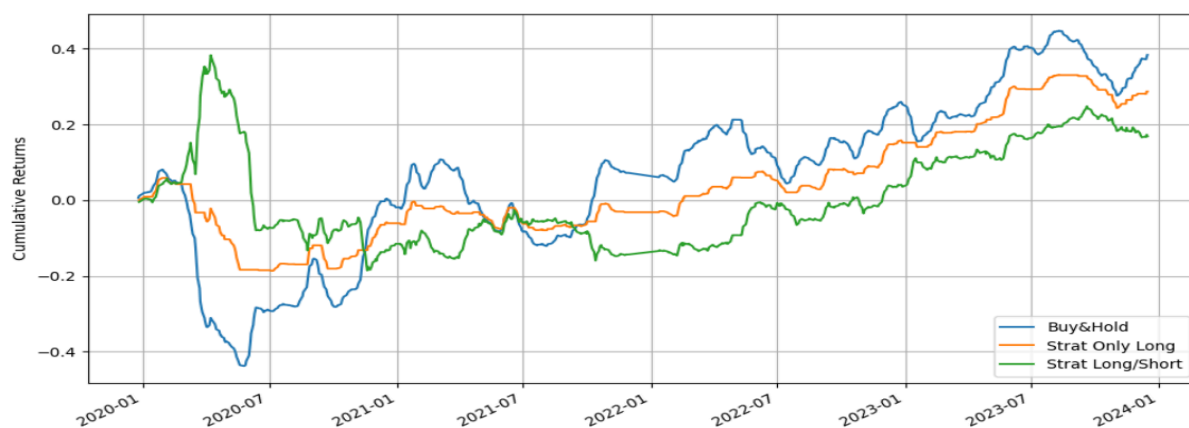


Figure M 2.3.1: Backtesting with the CNN predictions using the non-anchored walk-forward method ( $n_{\text{test}}=500$ ,  $n_{\text{train}}=500$ ).

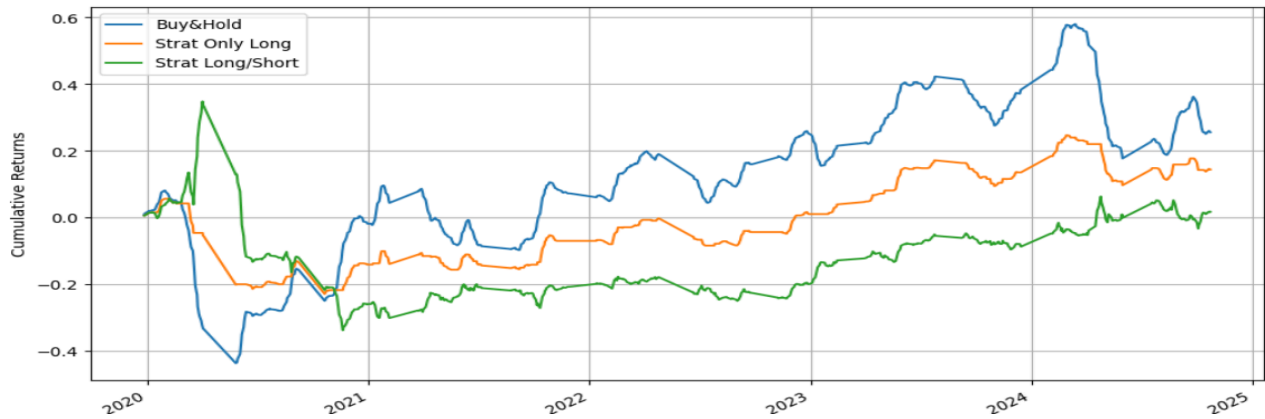


Figure M 2.3.2: Backtesting with the CNN predictions using the non-anchored walk-forward method ( $n_{\text{test}}=500$ ,  $n_{\text{train}}=100$ ).

Table M 2.3: Summarizing the results from the CNN together with the CNN non-anchored walk-forward method.

Metric	CNN (Single Test-train set)	Non-Anchored Walk Forward CNN ( $n_{\text{test}} = 500$ , $n_{\text{train}} = 500$ )	Non-Anchored Walk Forward CNN ( $n_{\text{test}} = 500$ , $n_{\text{train}} = 100$ )
Buy & Hold Return (%)	-23.30	38.30	25.52
Buy & Hold Sharpe	-14.42	5.11	3.01
Strat Long Only Return (%)	-19.69	28.65	14.32
Strat Long Only Sharpe	-16.13	6.9	2.24
Strat Long/Short Return (%)	-16.38	16.82	1.66
Strat Long/Short Sharpe	-9.59	2.61	0.68

Step 3.

---

A.

For this step, we have to work on alleviating the leakage of information between training and test sets. Then we will again use the same 3 models and the same train/test split, i.e., Step 3.b, exploiting a train/test split with 500 observations in each, and Step 3.c exploits a train/test split with 500 observations in each training set and 100 observations in the test sets that we are using in the previous steps. This time the technique that we are using to reduce the information leakage is by using CPCV (Combinatorial Purged Cross-Validation”) with a mechanism. We will also write a custom loss function that we will discuss below.

But before that, let's understand the CPCV method with the purging mechanism:

The CPCV (“Combinatorial Purged Cross-Validation”) method is a technique to solve the issues of information leakage and overlapping dependencies, which are common in time series and sequential data (Arian et al.). Unlike traditional cross-validation techniques, CPCV systematically generates all possible combinations of train-test splits while ensuring that the test sets are mutually exclusive. It introduces a purging mechanism to prevent leakage by removing data points adjacent to the boundaries of the train and test sets. This ensures that the training set does not include information that might influence the test set, maintaining the integrity of the validation process. CPCV with purging is particularly useful in financial time series or other domains where temporal dependencies can lead to misleading performance estimates, providing robust and unbiased model evaluation.

In our implementation of the Combinatorial Purged Cross-Validation (CPCV) method with the purging mechanism, we use two key functions: the `walk_forward_split` function and the `purged_indices` function you can see in Figure 15. The `walk_forward_split` function creates train-test splits by sequentially

generating indices for a rolling window of fixed sizes, with required observations each for the training and testing sets. This ensures a required number of observations in each split while enabling a walk-forward approach. To address potential information leakage, we have applied the `purged_indices` function, which removes a proportion of the most recent observations from the training set adjacent to the test set, based on a defined `purge_ratio`. By purging this overlap, we ensure that no future information from the test set influences the training process. Together, these functions enable an effective and unbiased evaluation framework for time-series models, maintaining the robustness of the validation process.

```
# Purging function to prevent data leakage
def purged_indices(train_idx, test_idx, purge_ratio=0.1):
    train_size = len(train_idx)
    purge_count = int(train_size * purge_ratio)
    purged_train_idx = train_idx[:-purge_count] # Remove the last 'purge_count' observations
    return purged_train_idx, test_idx
```

Figure 15: CPCV function with purging mechanism

Now let's understand the custom loss function that we have built in Figure 16:

A custom loss function is a user-defined metric that is designed to measure the error or divergence between predicted and actual values, tailored to the specific requirements of a problem.

In our problem we have designed a custom loss function to enhance model performance by combining the standard Mean Squared Error (MSE) with a penalty for large deviations, ensuring robustness against outliers. The MSE term computes the average squared difference between true and predicted values, penalizing larger errors more heavily and serving as the primary optimization objective. To address outliers, the loss includes a penalty term that activates only when the absolute difference between true and predicted values exceeds a predefined threshold (e.g., 0.05). This penalty is computed as the mean

of the squared excess deviations beyond the threshold, encouraging the model to reduce significant prediction errors. The total loss is the sum of the MSE and a weighted penalty term, where the weighting factor (e.g., 0.1) controls the balance between minimizing average error and mitigating outliers. This combination ensures the model is both accurate and robust, making it suitable for tasks where large deviations are particularly undesirable.

```
from tensorflow.keras import backend as K

# Define the custom loss function
def custom_loss(y_true, y_pred):
    """
    Custom loss function that combines MSE with a penalty for large deviations.
    """
    # Mean Squared Error (MSE)
    mse_loss = K.mean(K.square(y_true - y_pred))

    # Penalty for large deviations (outliers)
    penalty = K.mean(K.square(K.maximum(0., K.abs(y_true - y_pred) - 0.05)))

    # Combine both: the MSE + the penalty for large deviations
    total_loss = mse_loss + 0.1 * penalty # You can tune the penalty weight (0.1 here)

    return total_loss
```

Figure 16: Custom Loss Function

B.

From the results, we got in Figure 17 that highlight the performance of three models, MLP, LSTM, and CNN, based on GAF evaluated using a custom loss function over two iterations. The MLP model demonstrates moderate performance, with losses decreasing from 0.0041 to 0.00077, indicating its ability to improve as training progresses. The LSTM model, designed for sequential data, exhibits better overall performance with lower losses of 0.0023 and 0.00031, showcasing its strength in capturing temporal dependencies effectively. The CNN, employing Gramian Angular Field (GAF) transformations for image-like feature representation, achieves the lowest losses among the three, with values of 0.00084 and 0.00040, suggesting superior accuracy in this specific task. These results emphasize the importance of model selection based on data characteristics and the task at hand, with CNN showing the highest potential for minimizing error in this context.

```
All Model Losses:  
MLP Losses: [0.004124121740460396, 0.0007665659650228918]  
LSTM Losses: [0.0023035563062876463, 0.0003087349468842149]  
CNN Losses: [0.0008416344644501805, 0.00039531098445877433]
```

Figure 17: Models Compiled Losses Result

C. We again find the result with different training/test observation sets in Figure 18. MLP, LSTM, and CNN—trained and evaluated using the CPCV method with a purging mechanism, ensuring robust testing by mitigating data leakage. The MLP model shows varying losses across the 12 iterations, starting at 0.0041 and fluctuating, with some higher losses like 0.0315 and lower ones such as 0.00044. This indicates sensitivity to the training-test splits and potential challenges in capturing patterns consistently. The LSTM model, designed for sequential data, exhibits a more stable performance with generally lower losses, reaching a minimum of 0.00024, reflecting its strength in learning temporal dependencies. The CNN, leveraging Gramian Angular Field (GAF) transformations, achieves the most consistent and lowest losses overall, ranging between 0.0013 and 0.00031, showcasing its robustness and precision in handling the task. These results highlight the effectiveness of the CPCV framework with purging, as it provides detailed insights into model reliability across varying data splits, with CNN emerging as the most promising model in this scenario.

```
All Model Losses:  
MLP Losses: [0.0041276924312114716, 0.01464879047125578, 0.003779561724513769, 0.0030489086639136076, 0.  
LSTM Losses: [0.002752939471974969, 0.007064779754728079, 0.0009332036133855581, 0.0016810477245599031,  
CNN Losses: [0.0013059132033959031, 0.0013026644010096788, 0.0007233563810586929, 0.0004548292199615389,
```

Figure 18: Models Compiled Losses Result

**D. Backtesting using varying training and testing sets using the non-anchored walk-forward method with reduced leakage.**

In this section, first we perform a backtesting strategy using predictions we obtained, first from a walk-forward training and testing set of 500 observations each for the MLP, CNN, and LSTM, respectively. Secondly, we also perform a backtesting strategy using the predictions we obtained from the walk-forward predictions of 500 test observations and 100 train observations. We do this whilst applying the purging function described in section 3.B. We also discuss the variation of results we obtained from these predictions of each of the three aforementioned models and compare it with the results obtained from the simple models we discussed in section 1. We discuss the results we obtained from this section below.

First, we implement the backtesting of the MLP results. This is shown in Figures M 3.1.1 and M 3.1.2.

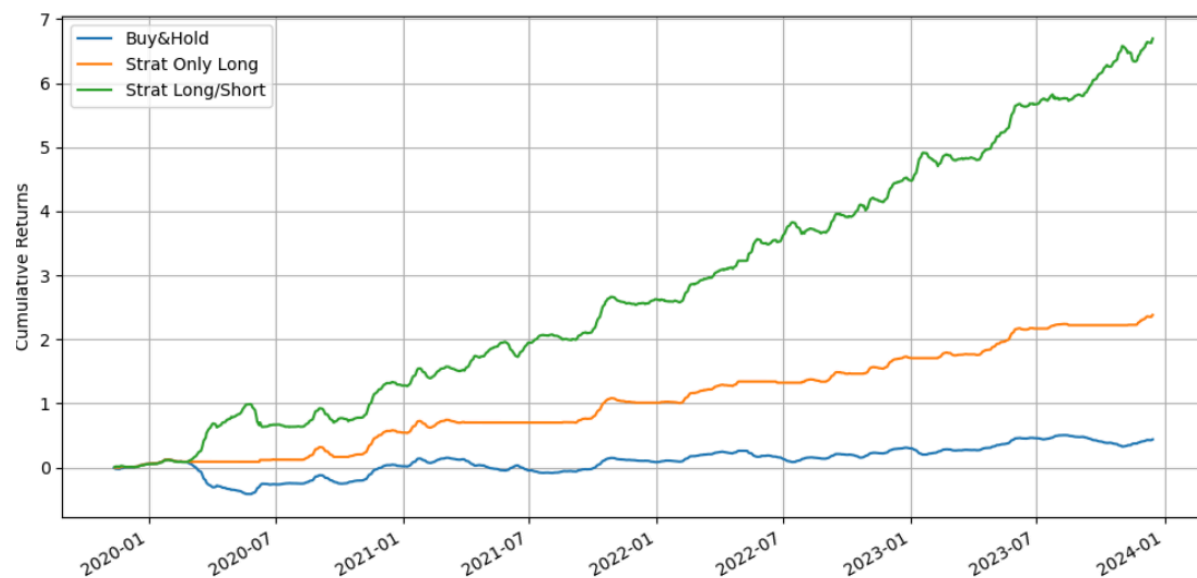


Figure M 3.1.1: Backtesting with the MLP predictions using the non-anchored walk-forward method ( $n_{\text{test}}=500$ ,  $n_{\text{train}}=500$ ) with purging.



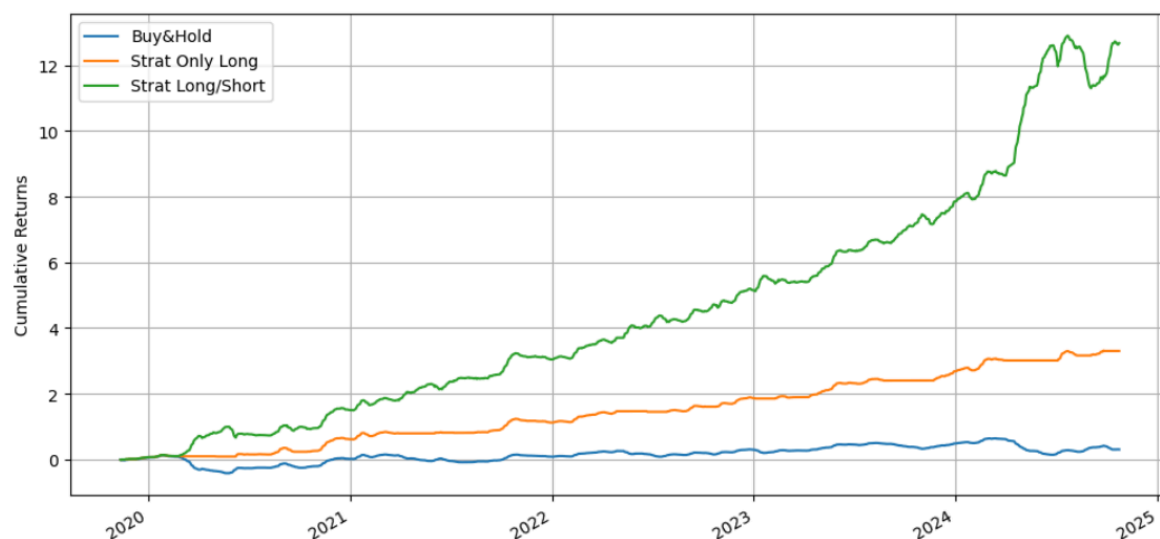


Figure M 3.1.2: Backtesting with the MLP predictions using the non-anchored walk-forward method ( $n_{\text{test}}=500$ ,  $n_{\text{train}}=100$ ) with purging.

Table M 3.1: Summarizing the results from the MLP together with the MLP non-anchored walk-forward methods implementing purging.

Metric	Non-Anchored Walk Forward MLP ( $n_{\text{test}}=500$ , $n_{\text{train}}=500$ )	Non-Anchored Walk Forward MLP ( $n_{\text{test}}=500$ , $n_{\text{train}}=100$ )
Buy & Hold Return (%)	43.89	30.59
Buy & Hold Sharpe	5.49	3.59
Strat Long Only Return (%)	238.17	330.39
Strat Long Only Sharpe	29.34	29.62
Strat Long/Short Return (%)	669.64	1267.17
Strat Long/Short Sharpe	30.91	33.59

From Table M 3.1, we immediately observe that there is more or less no change in the results obtained for the buy-and-hold strategy when compared with Table 2.1. This suggests that the strategy was indeed less sensitive to the impact of data leakage. Also for the long-only strategy, when we compare the results for the cumulative returns, we notice that purging might have helped bring the results to a lower, more realistic value. For the long-short strategy, when compared to the unpurged data in Table 2.1, we notice that the returns dropped for the  $n_{\text{train}}$  of 500 but increased for the  $n_{\text{train}}$  of 100. Sharpe ratios are reduced also. It is still high, but it might also suggest that a long-short strategy is still susceptible to model assumptions (Bailey et al.). Even within the purged result we see a doubling of returns for the long-short strategy. Overall, this might suggest that we implement a different purging function. It might also indicate the strategy is unaffected by the purging we applied.

Next, we observe the results of backtesting using predictions from the LSTM framework. This is shown in Figures M 3.2.1 and M 3.2.2.

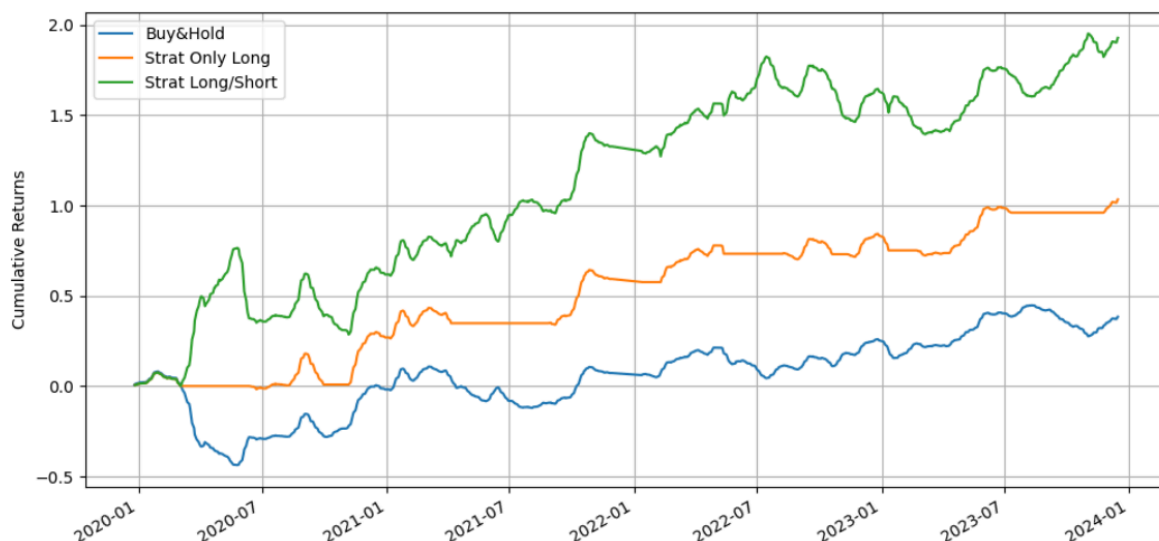


Figure M 3.2.1: Backtesting with the LSTM predictions using the non-anchored walk-forward method ( $n_{\text{test}}=500$ ,  $n_{\text{train}}=500$ ) with purging.

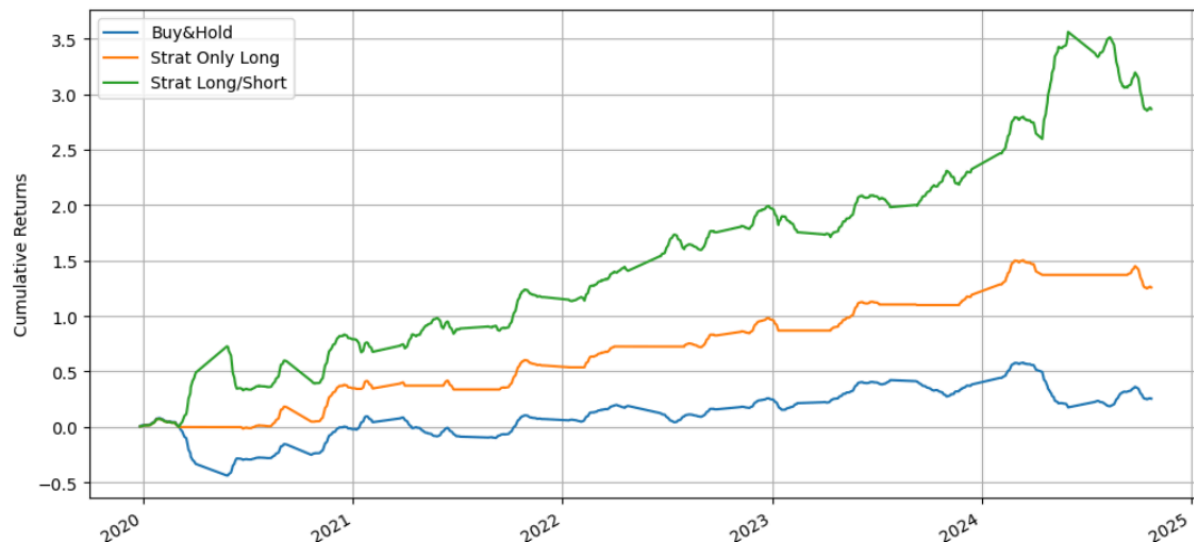


Figure M 3.2.2: Backtesting with the LSTM predictions using the non-anchored walk-forward method ( $n_{\text{test}}=500$ ,  $n_{\text{train}}=100$ ) with purging.

Table 3.2 summarizes the results of this backtesting briefly. Looking at Table 3.2, we see that for the buy-and-hold strategy, there is a decrease in the returns. As with other models, we know that the buy-and-hold strategy does not depend on the model. For the long-only strategy, while there is an increase in the returns, there is a decrease in the Sharpe ratio, which again suggests an increase in volatility. The results across the  $n_{\text{test}}$  from 500 to 100 seem consistent, though, which may be due to purging that reduces exaggerated results. The long/short strategy is also similar to the long strategy as we see an increase in returns and a decrease in the Sharpe ratio. Again, we see that the results are not exaggerated. For the unpurged backtesting, the results seem significantly inflated, particularly for the 500 test observations. This might indicate overfitting caused by leakage. After purging, returns and Sharpe ratios stabilize, as we can see in the purge results. This may suggest improved generalization and realistic performance. More importantly, though, we see that when we compare the results from step 2 to the result from step 3, we see a significant decrease in returns for the  $n_{\text{test}}=500$ , from 960.09% to 196.69%. We also see a decrease in the returns for the  $n_{\text{test}}=100$ . This shows that purging works effectively in this strategy. Considering that it is an LSTM framework, there might be some complexity

involved due to the nature of the framework. It is also possible that some noise is present even after purging.

Table M 3.2: Summarizing the results from the LSTM together with the LSTM non-anchored walk-forward methods implementing purging.

<b>Metric</b>	<b>Non-Anchored Walk Forward LSTM (n_test = 500, n_train = 500)</b>	<b>Non-Anchored Walk Forward LSTM (n_test = 500, n_train = 100)</b>
Buy & Hold Return (%)	38.30	25.52
Buy & Hold Sharpe	5.11	3.01
Strat Long Only Return (%)	103.25	111.57
Strat Long Only Sharpe	17.12	13.24
Strat Long/Short Return (%)	192.69	241.12
Strat Long/Short Sharpe	16.42	14.19

Finally, we observe the results of backtesting using predictions from the CNN framework. This is shown in Figures M 3.3.1 and M 3.3.2.

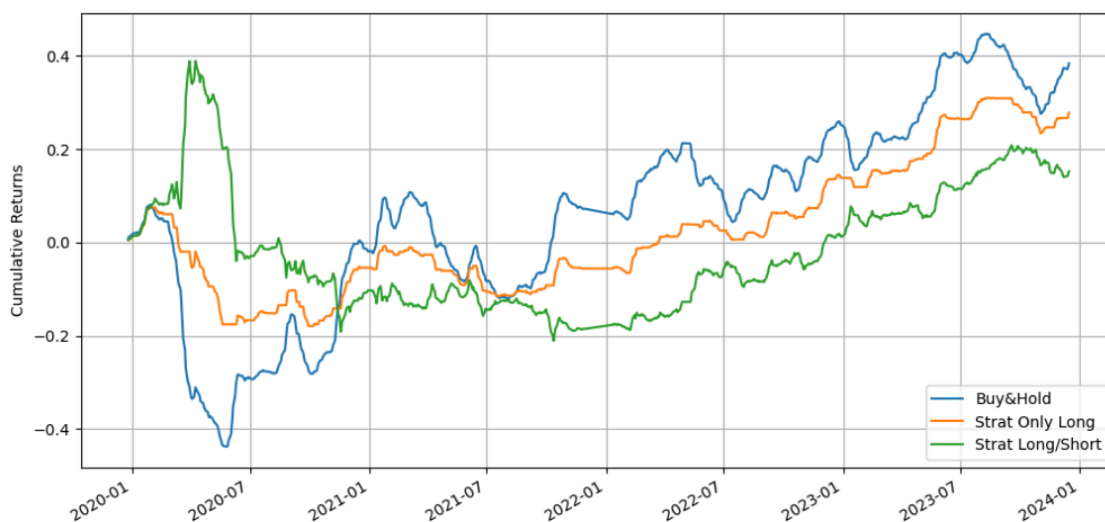


Figure M 3.3.1: Backtesting with the CNN predictions using the non-anchored walk-forward method ( $n_{\text{test}}=500$ ,  $n_{\text{train}}=500$ ) with purging.



Figure M 3.3.2: Backtesting with the CNN predictions using the non-anchored walk-forward method ( $n_{\text{test}}=500$ ,  $n_{\text{train}}=100$ ) with purging.

Table M 3.3 briefly summarizes the result of the backtesting.

Table M 3.3: Summarizing the results from the CNN together with the CNN non-anchored walk-forward methods implementing purging.

<b>Metric</b>	<b>Non-Anchored Walk Forward CNN (n_test = 500, n_train = 500)</b>	<b>Non-Anchored Walk Forward CNN (n_test = 500, n_train = 100)</b>
Buy & Hold Return (%)	38.30	25.52
Buy & Hold Sharpe	5.11	3.01
Strat Long Only Return (%)	27.72	27.72
Strat Long Only Sharpe	6.57	3.74
Strat Long/Short Return (%)	15.20	26.96
Strat Long/Short Sharpe	2.41	3.01

As we can observe across Table M.3, we see that the buy-and-hold returns and Sharpe Ratio decrease with a decrease in the  $n_{\text{test}}$  (from 500 to 100), which may indeed reflect the difficulty of capturing market trends within a shorter time frame. Interestingly, we notice a constant return in the long-only strategy, but a decrease in the Sharpe ratio, which might suggest an increase in the volatility with a shorter test window. The stable returns might have been as a result of the purging process. Surprisingly, we notice an increase in the performance of the long/short strategy as the testing period decreases. This shows the model's ability to capture upward and downward trends in the market, combined with purging, which ensures that future data is not leaked.

When compared with the results in step 2, as expected, the results of the buy-hold strategy do not really change, since the model is not really an active one. Hence, leakage did not affect these results in the buy and hold strategy. For the long strategy, purging reduces returns slightly for 500 test observations and improves returns significantly for 100 test observations. The Sharpe ratios are also generally stabilized. This may indicate that purging eliminated leakage, which had inflated the unpurged results. When looking at the unpurged results in the CNN framework, we notice a very low return (1.66% for 100 test observations) and Sharpe ratios for the long/short strategy, which improve significantly in the purged results. Hence in this strategy, we may conclude that purging did help deal with the leakage problem. Generally, even though purging was applied and some leakage was dealt with, we may still need some other form of regularization to reduce overfitting.

### References

- Sharpe, William F. "Capital Asset Prices: A Theory of Market Equilibrium under Conditions of Risk." The Journal of Finance, vol. 19, no. 3, Sept. 1964, p. 425.  
<https://doi.org/10.2307/2977928>.
- Beaver, William, et al. "The costs and benefits of long-short investing: A perspective on the market efficiency literature ☆." Journal of Accounting Literature, vol. 37, no. 1, Aug. 2016, pp. 1–18. <https://doi.org/10.1016/j.acclit.2016.07.001>.
- Arian, Hamid, et al. "Backtest overfitting in the machine learning era: A comparison of out-of-sample testing methods in a synthetic controlled environment." Knowledge-Based Systems, Oct. 2024, p. 112477. <https://doi.org/10.1016/j.knosys.2024.112477>.
- Bieganski, Bartosz, and Robert Ślepaczuk. "Supervised Autoencoder Mlp for Financial Time Series Forecasting." SSRN Electronic Journal, Jan. 2024, <https://doi.org/10.2139/ssrn.4781472>.
- Bailey, David, et al. "The probability of backtest overfitting." The Journal of Computational Finance, Sept. 2016, <https://doi.org/10.21314/jcf.2016.322>.



