G

Sign in to your account (**ma__@m__.ca**) for your personalized experience.

Sign in with Google

Not you? Sign in or create an account

University of Toronto Machine Intelligence Team Follow

Jun 21 · 22 min read · Listen

Save

Wall Street Bots: Building an Automatic Stock Trading Platform based on Artificial Intelligence From Scratch

A UTMIST Project by Jack Cai, Lisa Yu, Younwoo (Ethan) Choi, Dongfang Cui, Alaap Grandhi, Demetre Jouras, Kevin Mi, Yang Qu, Zhenhai (Joel) Quan, Thomas Verth.

The Wall Street Bots project is a 6-month challenge to build a stock trading platform utilizing data analysis and machine learning methods. The final product aims to enable user authentication, perform trading, build portfolios, and use automatic portfolio balancing with one of our tested strategies on our web platform. Readers and developers are welcome to try and contribute to this project on our web platform and Github page.

Creating a stock trading platform based on artificial intelligence (AI) is far more sophisticated than predicting stock prices using numerical methods such as momentum, or machine learning methods with deep learning. While the latter is pure data analysis and machine learning, the former covers a lot greater scopes, from not only price prediction, but also portfolio management strategies, market condition analysis, back-testing, data pipeline, and real-time deployment. The Wall Street Bots project is to do all of the above — what if we built an AI not in the perspective of a data science challenge, but in that of practicality (it makes money)? What if we do something similar to a hedge fund, but individualized and open source? What level of complexity will we achieve, and how does it perform relative to the market? The Wall Street Bots project aims to achieve all of that (or partially in some areas). In addition, we studied the role of natural language processing in stock price/trend prediction and whether stock news and market sentiments can be truly helpful in predicting prices. We built a platform that allows users to manually trade stocks via Alpaca API, or with one of the AI strategies we built.

In this Medium article, we will walk you through a high-level overview of how we built the Wall Street Bots. The following article is organized by:

1. Background

Trading stocks using an automatic stock trading platform is nothing new. In fact, banks, hedge funds, and trading firms have been using similar algorithmic trading methods for ages. Ever since the advent of digitalized market order flow, the idea of algorithmic trading has been becoming more and more relevant. Since these trading entities must constantly execute large quantities of orders accurately, it seems far more reasonable and efficient to leave it to a machine algorithm as opposed to humans. However, when algorithmic trading is used by these mega-corporations in practice, it creates many potential issues. For example, the 1987 stock market crash and the 2010 flash crash are widely speculated to be caused by large-scale orders placed by algorithmic trading machines. In addition, these machine learning algorithms are often not as accessible to the public. If the consumer chooses to put their money in one of these banks or hedge funds, a portion of their profit would often be taken away

from them. Moreover, the investor would often not have control over what kind of trading strategy the algorithm employs. One might ask, “why would anyone choose their own stock trading strategy when they can just leave it to the Wall Street analysts?” The answer to this question is simple. As more and more regular people enter the stock market in the post-Covid-19 era, retail investors now have more power to influence the market than ever before. The influx of investors, and the tension between global powers, compounded with the high growth of tech companies during the pandemic, have created one of the most volatile markets in the past 100 years. Because of this, we believe that an individualized, open-source, automatic trading platform like WallStreetBots could potentially be another tool for common investors and traders to exert their influence in the market and generate profit. In events like the GME short squeeze, these investors and traders have already proved they can punish bad trading decisions made by institutional trading firms. With a platform like WallStreetBots, these traders and investors will have the same tool that institutional firms use to facilitate trading while also making it more consistent and accurate.

2. The Platform

2.1. Alpaca

Before starting to build a stock trading platform, there are a few issues to consider. First, how can we get consistent real-time quote data? How do we take care of different order types, store stock information, and make the simulation as close to real life as possible? Luckily, Alpaca paper trade API takes care of all of those issues so that we can focus on building the strategy and pipeline. Alpaca offers free API keys for paper trading that allow users to place orders, access account information, and retrieve stock information via the Python Alpaca API library. The Wall Street Bots platform is built based on the Alpaca API. Users can retrieve their free Alpaca API key and secret key from Alpaca’s official website and input them into the dashboard page. Whenever a user places an order at

wallstreetbots.org, the same information is updated at Alpaca paper trade and vice versa.

2.2. Tools and Library

The web app was built using the core technologies of docker, Python+Django, and psql. Python was chosen as it is the de facto language for machine learning and keeping the entire codebase in the same language is a huge advantage. Docker is to help with deployment with reproducible builds. Exactly how docker is being used will be elaborated on in the deployment section as well. Psql was chosen for its first-party support with Django. Psql is now the default choice for many web applications unless a specific need can't be met. This project originally might have required lots of time series data for the ML training, but this part of the project was sliced off, and instead, CSVs were used. Keeping this data out of the database kept the complexity down and decreased implementation time. The frontend of the web app is rendered with the Django templating engine — this keeps out most of the complexity that React/Angular/Vue would bring. As this project was mostly time-constrained, being able to work with the smallest amount of setup possible was a priority when choosing all these technologies.

2.3. Web App Structure

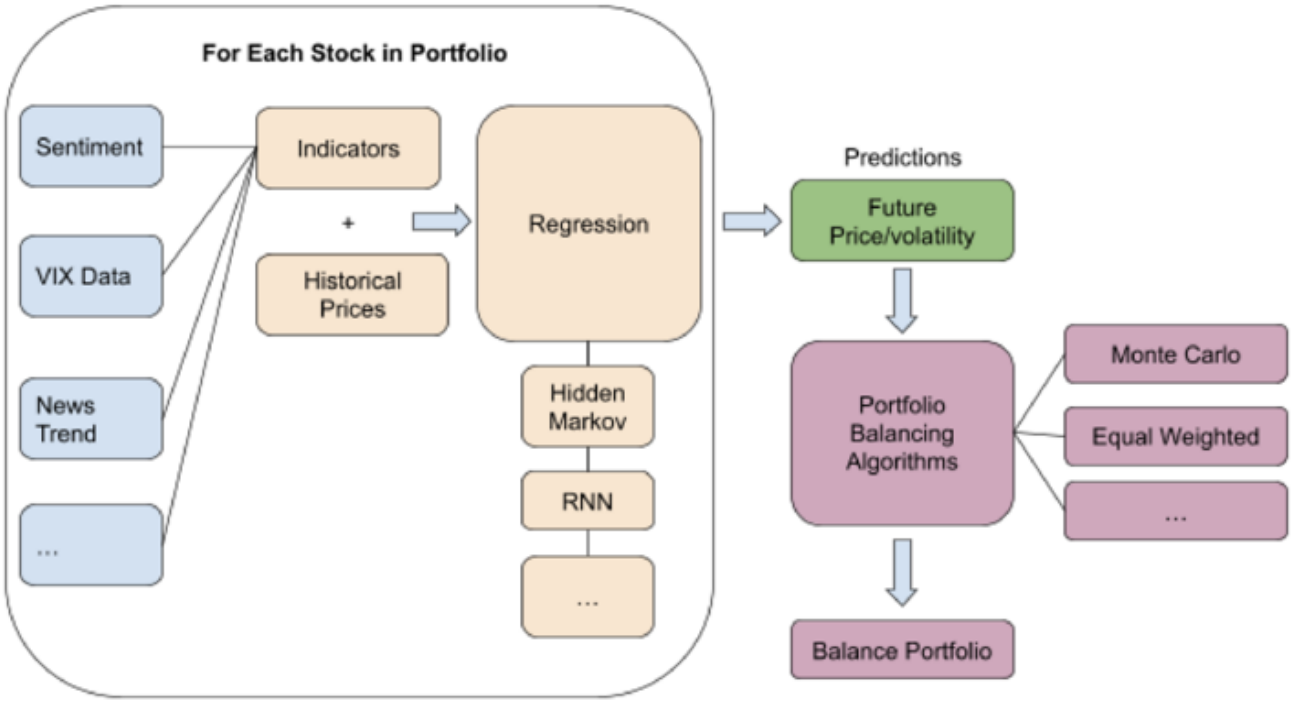
The web app is roughly split into the basic CRUD functionality of the dashboard for showing portfolios and entering orders, and the actual stock trading engine that makes the decisions. This keeps all the Django functionality split from the machine learning parts. The CRUD part is further refined down into different Django apps for further convenience. The homepage, along with all the required models, views, and routes, are split apart from the user data, split apart from how the portfolios and stocks are stored, represented, and the routes that modify them.

Inside the machine learning half of the codebase, there are strategies for each of the ways the portfolio can be rebalanced. These strategies are fed into the generic pipeline that is called by the web app every day to automatically rebalance for each user. Each new machine learning strategy inherits from a generic strategy and overrides with a different rebalancing method. This ensures that adding new strategies is as easy as possible. Below is the file tree for the project.

```
WallSreetBots.
              ├── LICENSE
              ├── README.md
              ├── __init__.py
              ├── backend        # This is the Django web app
              │   ├── __init__.py
              │   ├── auth0login # Authentication + Frontend
              │   ├── tradingbot # Backend database + models
              │   ├── ...
              ├── ml             # This is the machine learning pipeline
              │   ├── __init__.py
              │   ├── data_collection # Data collection scripts
              │   ├── tradingbot # Deployed models + strategies
              │   ├── ...
              │
              ...
```

Top-level file tree for the Wall Street Bots web app and trading model deployment.

2.4. Trading Pipeline



Trading strategy pipeline structure for the Wall Street Bots.

The above diagram shows the pipeline of our automatic stock trading system. We use historical prices along with indicators to train regression models to predict future stock prices and volatility. These outputs are fed into portfolio balancing algorithms to rebalance the portfolio. Each component is independent of the others, which means different pipelines can be combined to create new strategies. Strategies implemented by the Wall Street Bots team strictly follow the above structure. This enables upscaling of future strategies and a cleaner codebase.

The Wall Street Bots project implemented Hidden Markov Model (HMM) and various deep neural networks (DNNs) along with natural language processing (NLP) for price and volatility prediction, and two portfolio balancing strategies: naive equally weighted portfolio and Monte Carlo simulation to maximize Sharpe ratio.

2.5. Deployment and Automation

The entire web app is dockerized, which makes deployment very easy. There are only two docker containers used, one with the Django app and one with a prebuilt psql database. The containers are orchestrated with docker-compose. Again, this was chosen for simplicity over Kubernetes or similar tools. It's hosted on DigitalOcean using the smallest droplet size. The project is cloned from GitHub, and the .env files have the secret keys added. The droplet itself has a reverse proxy to redirect HTTP/HTTPS traffic to the web app container, and the firewall rules are changed to allow this outside traffic in and out of those ports.

Importantly, none of the actual machine learning needs to take place on the droplet itself. All the training is done on local, offline machines. This keeps production simple and, most importantly, allows the project to use the most affordable droplet option available.

The automation is handled inside the web app itself. Every fixed time period, on trading days, the rebalancing pipeline is triggered. This resyncs the Alpaca account in case they've changed, for each user, finds the appropriate rebalancing strategy, figures out what needs to be bought or sold to reach the target percentages, and executes those trades using the Alpaca platform.

3. Data Collection

Data is an essential part of any analysis task. The Wall Street Bots project collects market data, including stock prices, indices prices, stock news, investor comments, and fundamentals from a variety of sources. These sources and methods used are listed below.

3.1. Alpaca Market Data

Alpaca offers stock and index quotes and historical data for all US exchanges. In addition, Alpaca offers a news API for historical and real-time news headlines. Although not in large quantities, these news headlines are used in combination

with other sources for sentiment analysis and NLP tasks.

3.2. New York Times News Archive API

The New York Times offers a free news API for parsing NYT historical news headlines.

3.3. Finviz News Headlines

Finviz does not offer any APIs but is a good source for stock headlines. Due to the simplicity of the website, web scraping headlines from Finviz is relatively easy. Scraping news headlines from Finviz is done via the Python Beautiful Soup library. A simple web scraping script that looks for the latest 100 news per stock on FinViz is written. The script is available here.

3.4. r/wallstreetbets Daily Discussion Comments

We access investor comments data via Reddit's r/wallstreetbets discussion thread. Scraping the Daily Discussion Thread comment from r/wallstreetbets is a lot more complicated than scraping Finviz headlines. The reasons are twofold. First, each daily discussion thread contains roughly 10k comments per day. We scraped r/wallstreetbets in the time frame from 2020–01–01 to 2022–05–13, and even if we only take the top 1.5k comments per day, it added up to more than a million comments in total. Second, while the Reddit PRAW API allows one to get all comments of a thread given the thread's identifier, there is no pattern in each daily discussion thread's identifier. To overcome this issue, we adopted an approach similar to [1], in which we programmed Selenium library's chrome driver to automatically search the thread name and look for the specific class tag in the HTML for the unique thread identifier. We then saved those identifiers to a CSV and used Reddit's PRAW API to get the top 1.5k most voted comments for each thread. The full script is available here.

3.5. RSS Feeds

RSS feeds from various news sources were also pulled. Mainly headlines and short bodies were of interest. These sources were often traditional newspapers such as The Economist, or The New York Times. The RSS feed is polled for new additions. In practice, this was not an effective source of data as each news source varied too much from one another to group them together in the same dataset. Articles weren't as frequent as other sources. Historic data was difficult to capture. This source didn't end up being used to train any of the models.

4. The Strategies

4.1. Stock Closing Price Prediction with NLP

2021 is a phenomenal year, not only because the market reached the highest point in history post-pandemic, but also because the rise of meme stock, platforms like r/wallstreetbets, and individual traders that remind us the market is not solely influenced by banks, trading firms, and large hedge funds, but also by individual traders and public sentiments around the market. The idea that the market is correlated with public sentiment is not new. In fact, the famous study in 2011 by Bollen et al. [2] revealed the correlation between Dow Jones Industrial Average (DJIA) and Twitter sentiment from Google's GPOMS model that outputs the sentiment across six categories. Specifically, the "calmness" category revealed the largest correlation, while the other categories revealed almost none. The study demonstrated a striking 86.7% trend prediction accuracy with linear regression combining past price and the calmness score to predict DJIA's next day closing price, while the baseline accuracy based on moving average achieved 73.3%. Despite the strong performance, several things to keep in mind are that the testing period is a short span of time with only 15 trading days, and the period is chosen with the least volatility in DJIA and the absence of large unexpected socio-economic events. In reality, unexpected events frequently occur

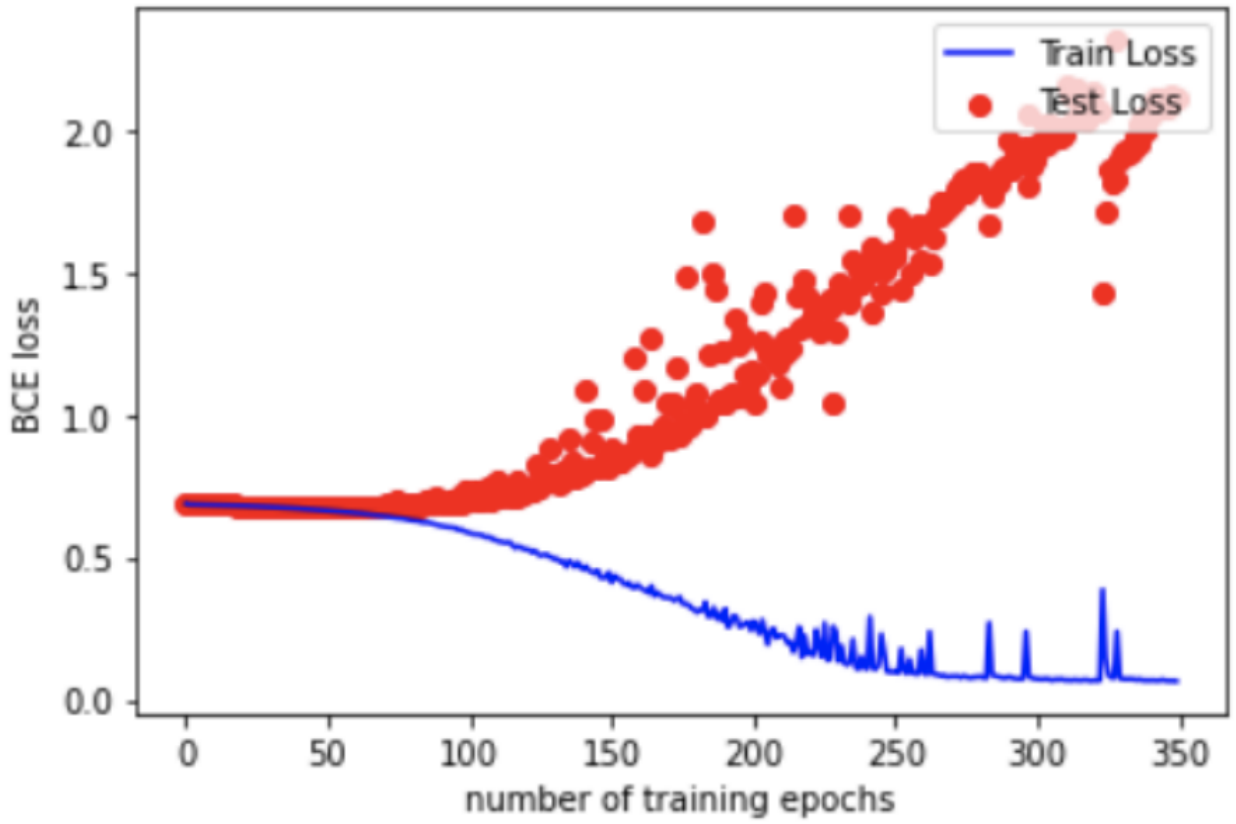
to skew the stock prices from prediction, and moving average trend prediction accuracy is bounded around 50% as opposed to the 73.3% in the study, especially when the market is volatile.

Unfortunately, we could not directly adopt Bollen et al.'s approach to Wall Street Bots due to the GPOMS model used in the study being a closed source model. Instead, we changed our attention to large language models developed in recent years thanks to transformers and attention mechanisms. The specific model we will be using is BERT [3], which stands for Bidirectional Encoder Representations from Transformers. We do not go into details about how BERT works in this article, but one of its key functions is to encode a line of text into a 768-D vector, called a sentence embedding vector. A sentence embedding vector captures the meaning of a sentence, and while it does not tell us directly the calmness of a sentence, a neural network trained based on these embeddings certainly can. We used FinBert [4], a fine-tuned version of BERT trained on financial data. Therefore, instead of using a calmness score from Twitter data to do linear regression, we trained deep neural networks (DNNs) with FinBERT embeddings on a variety of text sources, combined with past prices and indicators such as VIX and QQQ to predict the next day closing price. (Note that FinBERT directly classifies the text into three sentiment scores — positive, neutral, and negative. We extract the FinBERT embedding as the <CLS> token embedding of the last hidden state.)

4.1.1. Trend Prediction Multi-layer Perceptron Based on Stock News Headlines

The first model we tried was a multi-layer perceptron (MLP) to predict the trend of a stock (whether going up or down) solely based on headlines. We did this as the starting point because stock news headlines are the easiest to retrieve via web scraping from FinViz and the Alpaca News API. We gathered 50 growth stocks from NASDAQ with similar fundamentals and gathered their headlines from 2016 to 2022, and we obtained a total of roughly 100,000 headlines. The

train/test sets are the FinBERT embeddings of these headlines, and the labels are whether the stock price goes up or down after the headline is released (1 = up, -1 = down). We tried it for both the hour and the day before and after the news was released. However, no matter what training techniques we apply or what hyperparameter we use, we could not bring the test loss down. The figure below shows a typical loss vs. iteration curve for this model on one of the training sessions.



Train loss and test loss for trend prediction MLP.

The above figure is a typical overfitting problem, but downsizing the model and adding regulations such as dropout did not make it any better — the training loss ended up going up along with the test loss. After more research, we found out that there is something fundamentally wrong with the approach. First, it is almost impossible to predict the price with news. By the efficient market

hypothesis, when news comes out, the stock price reflects it almost immediately. After all, as a side project, we do not have the resource nor the zero latency to react faster than the market. In reality, Wall Street analysts anticipate the news, which is a totally different problem and a much harder one to learn. Furthermore, we did not include the past prices and other indicators in the training set; this is actually a naive assumption because stock trends can not be separated from their prices. For instance, if a stock is already highly overvalued it is likely to go down despite the news being positive. We gave up on this approach and instead looked into r/wallstreetbets comments for price/trend prediction.

4.1.2. GME Price Prediction MLP, CNN, and LSTM with r/wallstreetbets Daily

Discussion Comments Reddit comments, on the other hand, are much better sources for investors' sentiment than news headlines because first, unlike the news headlines that come from a few presses, Reddit comments have diverse sources that represent a greater population. Second, Reddit comments contain more emotions (easier for sentiment analysis) than news headlines that are mainly statements of facts. The r/wallstreetbets daily discussion thread is the place where most investors in r/wallstreetbets express their opinion towards the market, therefore a great source for sentiment analysis. While r/wallstreetbets investors only represent a small portion of the investor population, we believe that they represent a much bigger portion of the meme stock investors. For this reason, we used the Reddit comments in predicting GME price as opposed to indices that represent the general market such as SPY and NASDAQ.

Like the previous approach, we run FinBERT on the Reddit comment and extract the embeddings. But this time, we consider past prices, FinBERT sentiment outputs, SPY index, VIX index, and NASDAQ index. The overall train/test set features and label is shown below.

Features (X)									Label (Y)
Last GME Close Price	FinBERT Embeddings (768 D)	FinBERT Sentiment Score (3D)	Last SPY Close Price	Last VXX Close Price	Last QQQ Close Price	Hour of the Day	Score (# of upvotes)		Next GME Close Price

We collected ~ 1.1 million comments across 598 trading days, from January 1st, 2020 to May 13th, 2022. The input features include the previous close price, the positive/neutral/negative sentiment scores of the Reddit comments given by the FinBert model, the previous VXX close (as a substitute for VIX), the previous SPY close, the previous QQQ close (as a substitute for NASDAQ), the number of upvotes of the Reddit comment, the hour of the day that the comment was posted, and the 768 dimension FinBert embedding of the comment. The target we are trying to predict is the next closing price of the stocks. We tested the following models (as listed below) and some with data augmentation from other stock prices such as AMC, TSLA, AAPL, and KOSS, which are tickers frequently mentioned in r/wallstreetbets or have a high correlation with GME.

N-day Moving Average (Benchmark): We compute the past N-day moving average to be the next predicted price for GME, and we selected the N that yields the highest trend prediction accuracy (50.0%) and the lowest Mean Absolute Percentage Error (MAPE) (6.24%) to be the benchmark model.

Multi-layer Perceptron (MLP): Using the full features ($N \times 777$) as inputs, the 3-layer MLP model with ReLU activations tries to predict the normalized daily return based on the next closing price. To avoid look-ahead bias, we split the data for training and validation sets chronologically (80% and 20%). The MLP model reached a validation trend prediction accuracy of 54.3% and MAPE of 10.87%. These results are worse than the moving average benchmark strategy. This may be due to the fact that each individual comment is treated as a single data point, leading to multiple comments with different sentiments within one day all having

the same target return for the model to predict. This input/target relationship only makes sense if all the comments each day have very similar sentiments.

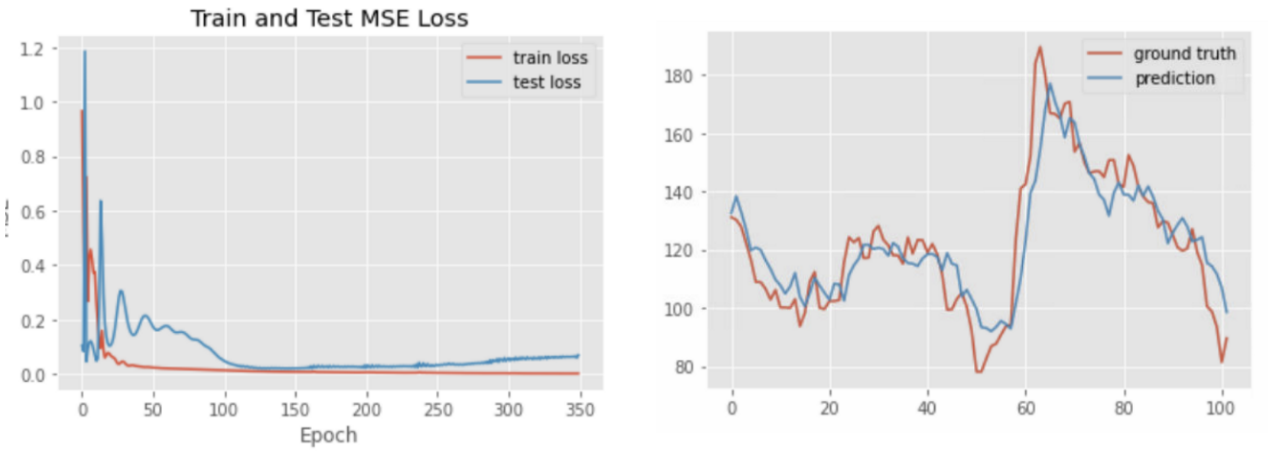
Convolutional Neural Network (CNN): We first consider GME price data only. To address the issue raised in the previous MLP model section, we concatenated the input features of all the comments on the same day to a matrix of dimensions 500×777 (since we scraped 500 comments each day) and treated each of these "daily" matrices as an individual data point with one corresponding target return. The final dataset has dimensions $606 \times 500 \times 777$ after dropping the days with less than 500 comments. We split the data randomly to get the training and testing sets (80% and 20%). The final CNN network with two convolution layers and two dense layers with max-pooling and ReLU activations achieved validation trend prediction accuracy of 49.5% and MAPE of 14.49%. Again, this model underperformed the moving average benchmark. This is likely because after grouping the data points based on the comment date, the new dataset for GME is only of size 606, which is too small for the model to train on.

CNN with Data Augmentation: We try to solve the problem of insufficient data by augmenting the GME data with price data from different stocks whose returns have a high correlation with GME returns. This leads us to develop the CNN model with data augmentation. We picked AMC, KOSS, TSLA, and AAPL, and performed the train-test split chronologically (80%, 20%). With the same CNN model, we achieved validation trend prediction accuracy of 58.1% and MAPE of 8.7%. The trend prediction accuracy greatly outperformed the moving average benchmark.

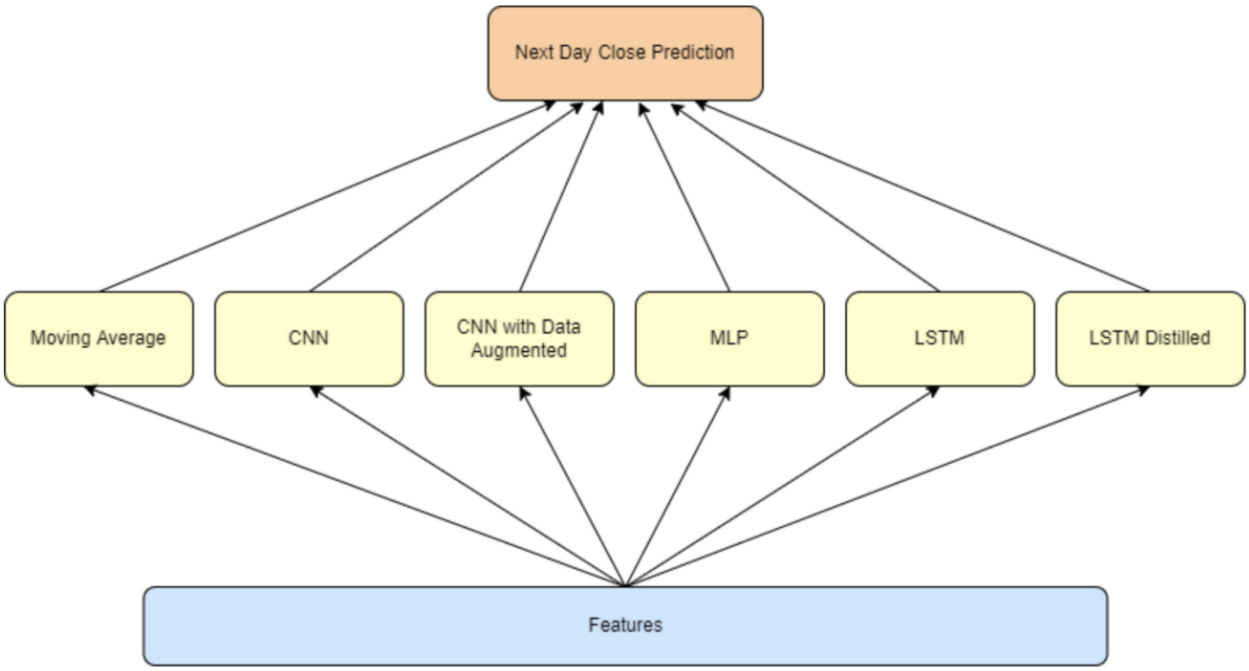
Long Short Term Memory (LSTM): A LSTM model is created with full features ($N \times 777$) in the training set X, and we averaged each day's comment embeddings and sentiment scores, creating a total of 611 data points. Due to the lack of data points and high dimensional features, Principal Component Analysis (PCA) was used to reduce the FinBert embedding to 20 dimensions that kept

63.2% variance. Despite the dimension reduction, the test loss was not able to converge. The overall best validation trend accuracy is 51.0% and 29.52% for MAPE.

LSTM Distilled:
A distilled version of LSTM is created without the FinBERT embeddings, with only 8 features in each time step. The test loss was able to converge, and we found the best model with 256 hidden dimensions using past N=10 data points to predict the next closing price. The overall best validation trend accuracy is 61.8% and 6.20% for MAPE.



Train and Test Loss (left) and Ground Truth and Predicted Price (Right) for LSTM Distilled.



Flow chart for models tested.

Comparison of the above models is shown below.

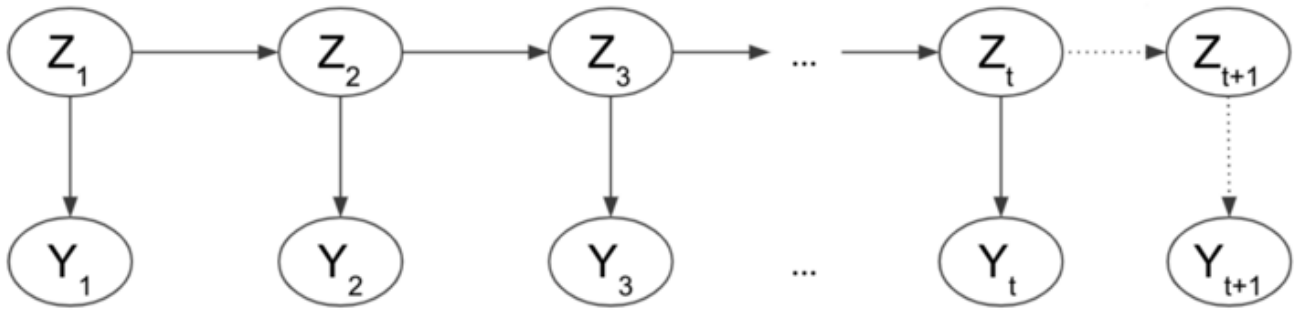
Model	Validation Trend Accuracy (%)	Validation MAPE (%)
N-Days Moving Average	50.0 (N=5)	6.24 (N=2)
MLP	54.3	10.87
CNN	49.5	14.49
CNN + Data Augmentation	58.1	8.65
LSTM	51.0	29.52
LSTM Distilled	**61.8**	**6.20**

Overall, our best model (LSTM Distilled) achieved significantly greater validation trend accuracy (61.8% vs. 50.0%) than the benchmark, with a slight improvement in MAPE (6.20% vs. 6.24%). Indeed, GME is a volatile stock, therefore predicting prices close to the target is challenging, which explains why

it is so hard to decrease MAPE. Still, a trend accuracy of 61.8% may be enough for building a portfolio management algorithm that gains a positive expected return in the long run. Note that compared to Bollen et al's, our validation period is significantly longer (120 days vs. 15 days) and our benchmark accuracy is lower (50.0% vs. 73.3%) for the stock and time period we picked.

4.2. Stock daily opening price prediction with HMM

Hidden Markov model (HMM) is a sequential model with the assumption of the statistical Markov process and is broadly applied in time series modeling, e.g., disease progression and text segmentation. The Markov process assumes that each observation is only dependent on a short history, and each observation is only dependent on the current hidden state. The interpretation of the hidden state depends on the context of the problem, and here it can be interpreted as underlying stock market conditions. In the Wall Street Bots project, we use each day's intraday minute price to predict the next day's opening price.



Hidden Markov model structure.

The above figure illustrates a standard HMM (in the context of stock opening price prediction):

- Z_(1) …, Z_(t+1) represents the hidden state, which is a representation of hidden stock market conditions.

- Y_(1), …, Y_(t) represents the current day's closing price on a minute basis, and Y_(t+1) represents the next day's opening price.
- Arrow represents dependency.

Implementation: Here, we assume that the stock price at each point is only dependent on its current hidden state and each hidden state is only dependent on the previous hidden state. We use the hmmlearn implementation for the model architecture. To build our training data, we standardized the same-day closing price on a minute basis by subtracting the first closing price. We construct one time series as the current day's closing price on a minute basis, followed by the next day's open price. The ultimate goal is to predict the next day's opening price using the previous day's closing price on a minute basis. The rationale behind it is that we assume that the pattern stays the same during market time and aftermarket. As the market opens, the stock price stays in the same hidden state as the last closing price. Note that it is not possible to trade aftermarket, so during deployment, we run HMM right before the market closes and assume that in the remaining time the stock price remains stable and within the same hidden states.

There are several reasons to use HMM here: one is because HMM has shown its strong inference and prediction power in many other research and use cases, we believe that HMM can also have a promising performance in the stock price prediction scenario; another reason is that HMM outputs a distribution of predicted values. Under certain model configurations, it outputs a normal distribution with a mean equal to the expected value of the stock and variance proportional to its volatility. Thus, we are able to learn the market fluctuation from the model and directly use it for portfolio balancing algorithms such as Monte Carlo search for maximizing the Sharpe ratio.

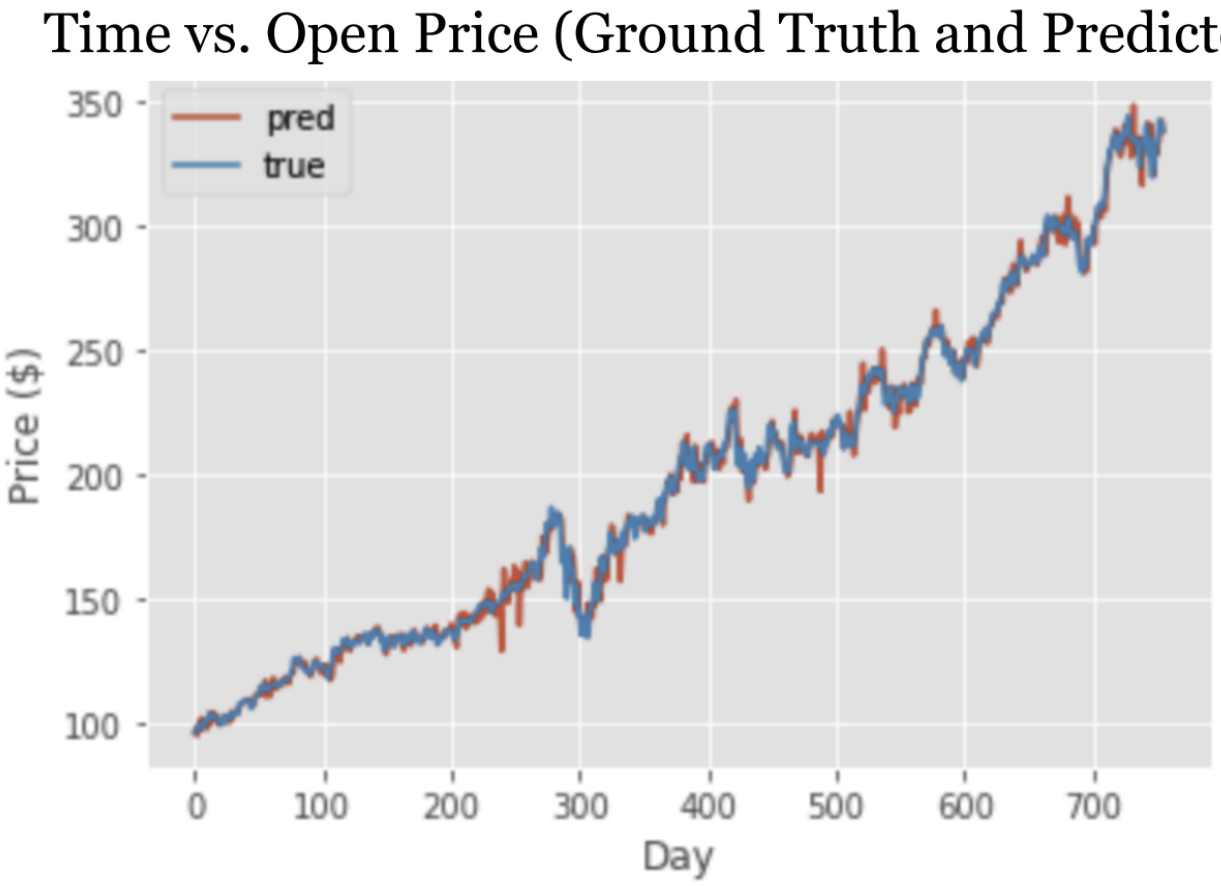
Result: We evaluate the model performance by two metrics: Mean Square Error (MSE) and trend prediction accuracy. We define the trend prediction accuracy as

the following: the proportion of the correct prediction on the next day's stock trend (going up or down) and MSE as usual:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y})^2$$

y_i : Ground Truth

$\hat{y}$: Prediction



Opening price prediction by HMM on MSFT from 2019 to 2021.

During evaluation, we used Microsoft data from 2019 to 2021 and trained an HMM with 15 hidden states. The blue plot represents the true price, and the red plot represents the prediction outputs from the model. HMM is able to make predictions very close to the true values with an MSE of 21.0 compared to 3623.6 from the benchmark moving average model. With this setting, we're able to achieve a 66.2% accuracy in trend prediction. To show that the prediction is meaningful, we compare it with the proportion of days where there is an uptrend (53.2%), showing that the HMM trend prediction greatly outperforms random guessing.

	MSE	Trend Accuracy (%)
Moving Average (Baseline) (N=5)	3623.6	53.1
HMM	**21.0**	**66.2**
Total Percent of Uptrends (Random Guessing) (%)	53.2	

Remarks: One limitation of our current design is that only stock price is considered as features. To fully utilize the data we have, the next step is to implement the HMM with multivariate features where each observation is dependent on both the current hidden state and the multivariate features.

While it seems like HMM outperforms all the DNN models with a trend accuracy of 66.2%, it is important to keep in mind that HMM is only predicting the next day's open price, while the DNNs are predicting the closing prices. These two types of models are not comparable as predicting the next day's closing price is a harder problem with more randomness. Still, HMM remains a powerful algorithm, as our result has proven.

4.3. Portfolio Balancing with Predicted Stock Prices

Given our predicted next-day closing and opening price, and assuming we only take long positions, we built our trading strategy as follows:

4.3.1. Equally Weighted Portfolio

An equally weighted portfolio can be viewed as a baseline portfolio allocation strategy. The desired portfolio weights $\widehat{w}$ at any given day are $\widehat{w} = \frac{[w_1, w_2, \ldots w_n]}{\Sigma w_i}$ where $w_i = 1$ if the predicted return is positive, and $w_i = 0$ otherwise. Note that we assume zero transaction fees in our models. In other words, we invest equally in all the positive-return stocks.

Our resulting portfolio under this strategy has an expected return equal to the average return of all the positive-return stocks. $E[R_p] = \frac{1}{\Sigma \widehat{w}_i} r_1 + \ldots + \frac{1}{\Sigma \widehat{w}_i} r_5 + \ldots + \frac{1}{\Sigma \widehat{w}_i} r_9 + \ldots$

The resulting portfolio has a variance equal to (1/n) times the average variance plus$(n^2 - n)/n^2$ times the average covariance of the positive-return stocks. (We can use historical price data to estimate the future variance/covariance of stocks.)

$$Var[R_p] = \sum_{i=1}^{n} \sum_{j=1}^{n} \widehat{w}_i \widehat{w}_j \sigma_{ij} = (\frac{1}{n})(average\ variance) + (\frac{n^2-n}{n^2})(average\ covariance)$$

4.3.2. Monte Carlo Simulation for Maximum Sharpe Ratio Portfolio

Sharpe ratio is a measure of a portfolio's risk-adjusted returns. It has the formula

$$Sharpe\ Ratio = \frac{portfolio\ return - risk\ free\ rate}{standard\ deviation\ of\ portfolio's\ excess\ return}$$

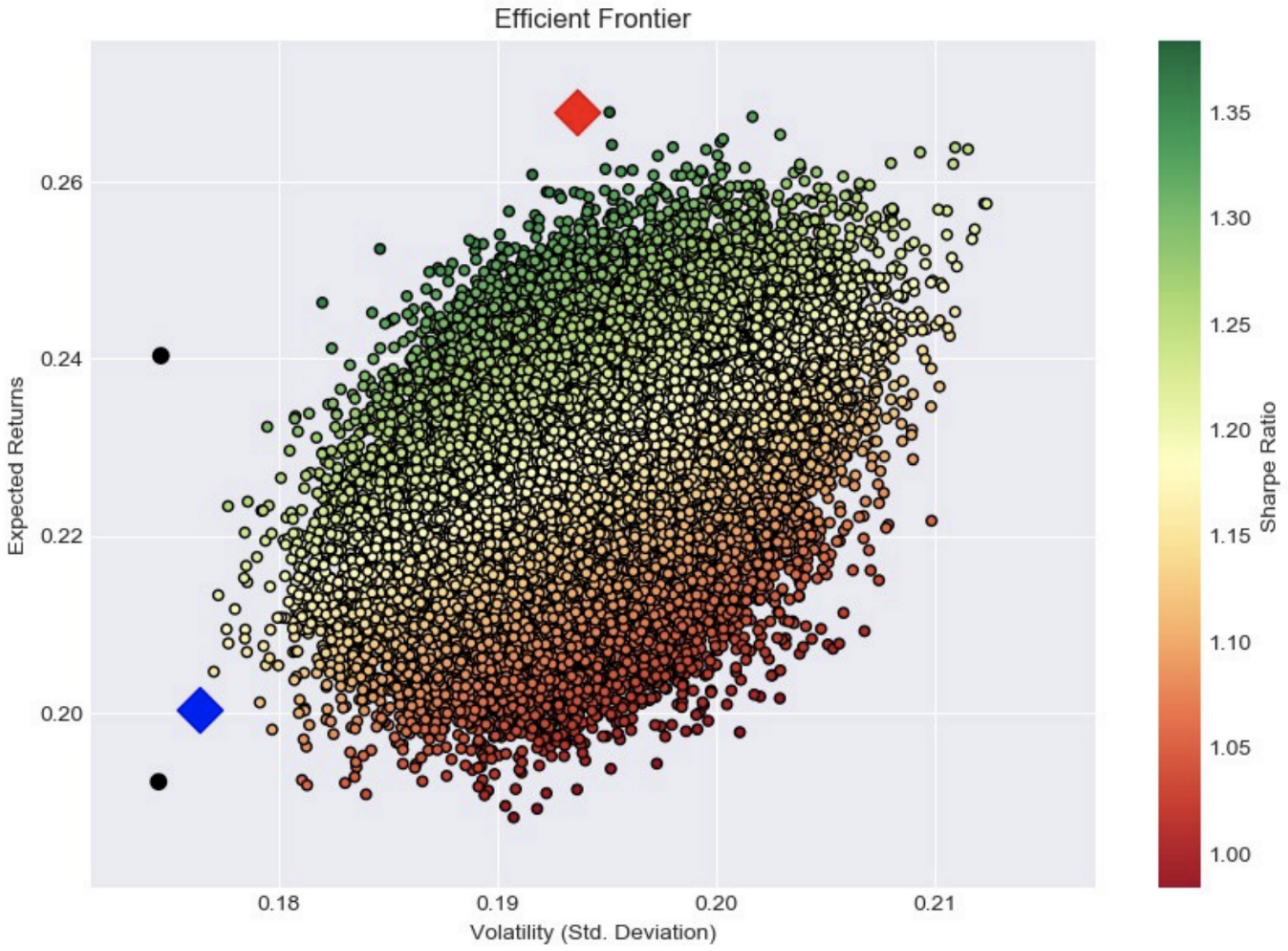
Monte Carlo is a common method used for portfolio optimization. We approximate the max Sharpe portfolio by randomly assigning portfolio weights with each weight between 0 and 1 such that the total weights of all the stocks sum up to 1. Then we calculate the expected return and variance of the portfolio.

$$E[R_p] = \sum w_i r_i$$

$$Var[R_p] = \sum_{i=1}^{n} \sum_{j=1}^{n} \hat{w}_i \hat{w}_j \sigma_{ij}$$

Again, we use historical price data to estimate the future variance/covariance matrix.

We then calculate the portfolio's Sharpe Ratio with the above formula assuming a risk-free rate of 2%. We repeat this process thousands of times to get an optimal portfolio weighting that maximizes the Sharpe ratio. See the below graph for an example Monte Carlo analysis — the max Sharpe portfolio is labeled red and the min variance portfolio is labeled blue.



Example graph for Monte Carlo analysis from [5]

After getting the optimal portfolio weights for the stocks, we can place trade orders equivalent to the difference between the optimal weights and our current portfolio weights to balance our portfolio each day.

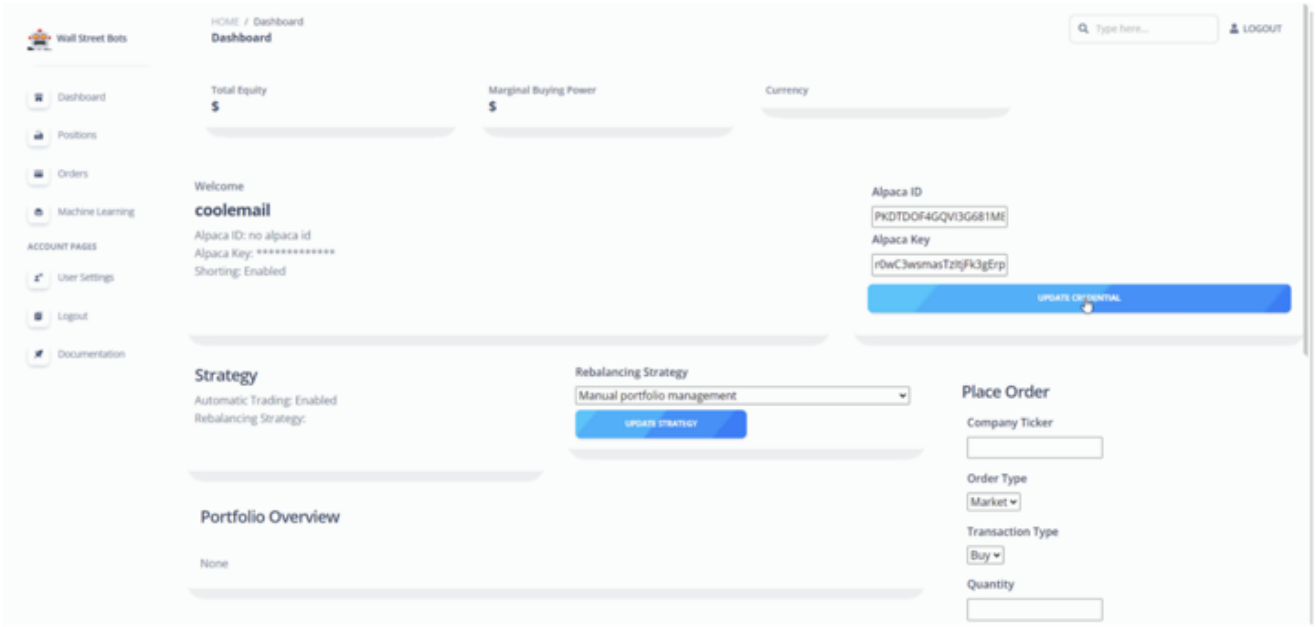
5. Demonstration

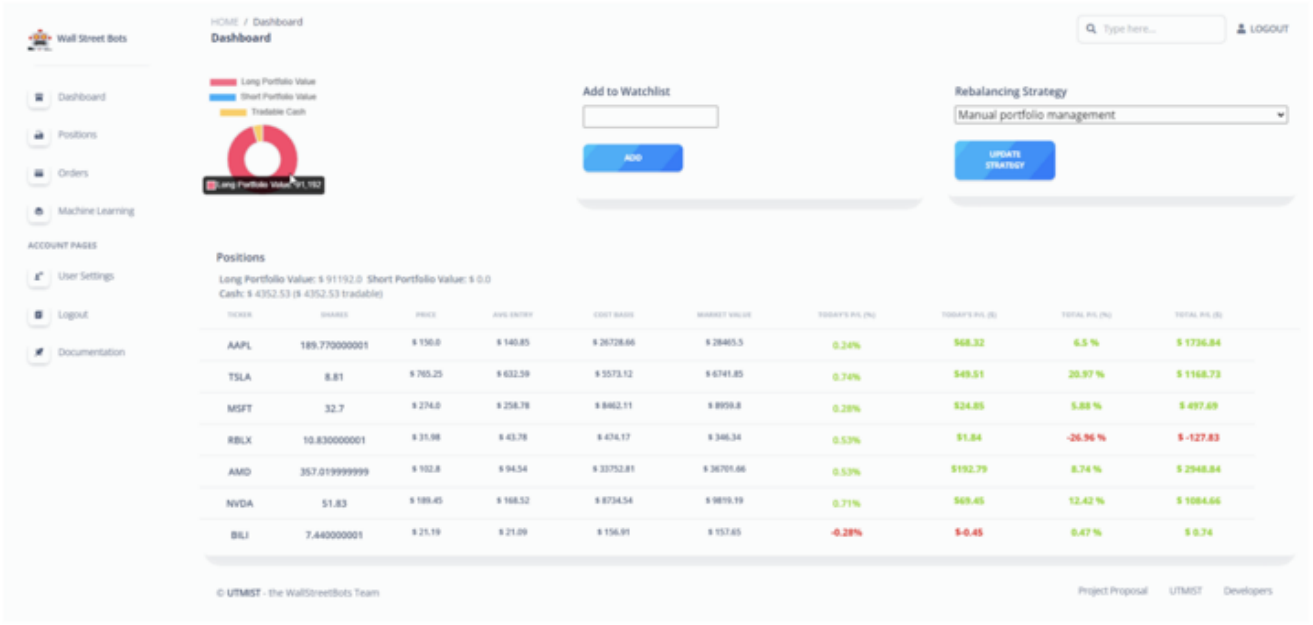
This section demonstrates all components in Wall Street Bots working together as a whole. It is also a step-by-step guide on how to use the web app to try out the strategies.

1. First, you will need to create a free account at https://alpaca.markets/
2. After you create your free account, navigate to the paper trade dashboard at

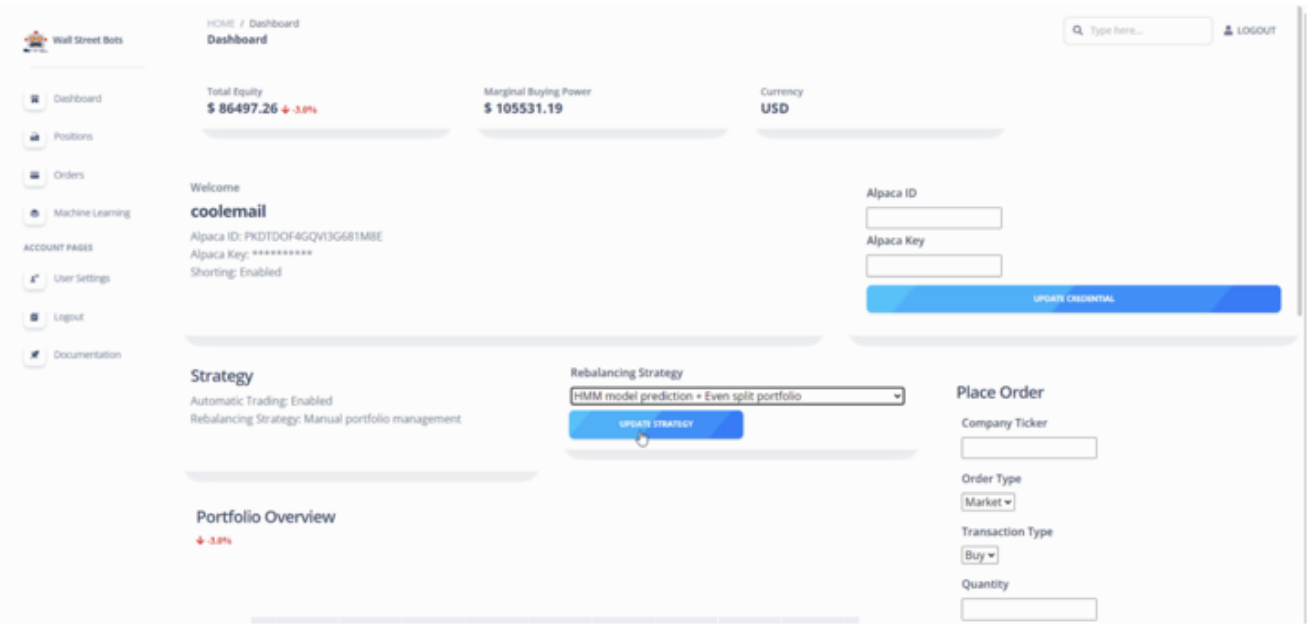
https://app.alpaca.markets/paper/dashboard/overview

3. Beside **Your API Keys**, click the **View** button and then click **Regenerate Key**.
4. Copy the **Key ID** and **Secret Key**.
5. Now navigate to http://wallstreetbots.org/. Follow the prompt and create an account. Login in and you will be directed to the dashboard page.
6. Paste your **Key ID** and **Secret Key** in the **Alpaca ID** and **Alpaca Key fields** respectively, then click **UPDATE CREDENTIAL.** (Note: by doing this step, you agree to share your Alpaca API credential to Wall Street Bots database)
7. You are all set! Now you can place orders, view stock information/portfolio history, and build your own portfolio on the Position page.





8. Of course, don't forget to choose one of our pre-built strategies that automatically balance the portfolio for you.



6. Closing Remarks

In conclusion, predicting stock movement is hard, and building a platform that integrates various models and strategies and executes orders on behalf of the

users is even harder. In this project, we went through all the steps to build a “mini hedgefund” — from selecting and verifying strategies to software development and deployment. We hope this article inspires you to do something similar or even consider contributing to this open source project. Wall Street Bots is a continuous effort and new features and strategies will constantly be developed in the future. Stay tuned!

Once again, you are welcome to check our codebase in our GitHub repository here or try out our website at wallstreetbots.org. Collaboration is also welcomed in the form of pull requests.

Thanks for reading!

References

[1] Smith, A. Reddit Wallstreetbets — Web Scraping Tutorial. https://algotrading101.com/learn/reddit-wallstreetbets-web-scraping/

[2] Bollen, J, Mao, H, Zeng, X. Twitter mood predicts the stock market. https://arxiv.org/pdf/1010.3003.pdf

[3] Devlin, J, Chang, M, Lee, K, Toutanova, K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.

Open in app Get started

[4] Araci, D. FinBERT: Financial Sentiment Analysis with Pre-trained Language Models. https://arxiv.org/pdf/1908.10063.pdf

[5] Ahmed, S. How to Construct an Efficient Portfolio Using The Modern Portfolio Theory in Python? https://towardsdatascience.com/how-to-construct-an-efficient-portfolio-using-the-modern-portfolio-theory-in-python-5c5ba2b0cff4

42 | 3

Medium

About Help Terms Privacy

Get the Medium app

Download on the App Store

GET IT ON Google Play