Automated Trading and Forecasting System Using IBKR API and Deep Learning Models

William Woodruff

June 1, 2025

1 Abstract

This project develops an end-to-end automated trading system that integrates deep learning models with the Interactive Brokers (IBKR) API for real-time execution. Historical market data are collected, preprocessed with technical indicators, and used to train convolutional neural networks (CNN), long-short-term memory (LSTM) networks, and Gated Recurrent Units (GRU). Hyperparameters are optimized via Optuna, and ensemble averaging is applied to enhance predictive robustness.

The system autonomously forecasts short-term price movements and automatically places bracketed trades with dynamically adjusted risk management parameters. Backtesting validates the models' performance, demonstrating improved Sharpe ratios and reduced forecast errors relative to individual models. While the framework shows strong potential for real-world trading, challenges remain in model generalization and execution under market constraints.

GitHub Repository: https://github.com/William-Woodruff/DeepLearningEnsemble

2 Research Questions

The project addresses the following key research questions:

1. **How effective are deep learning models (CNN, LSTM, GRU) in predicting short-term stock price movements when trained on engineered technical indicators derived from historical market data?**

2. **Does an ensemble of deep learning models outperform individual models in terms of prediction accuracy?**

3. **How closely do the results from Backtesting correspond to actual performance under Live Market Conditions?**

3 Resources

- **Machine Specifications:** Apple MacBook Air (M1, 2020), 8-core CPU, 8-core GPU, 16 GB unified memory; Google Colab Pro+ (NVIDIA Tesla T4 GPU, 24 GB RAM).
- **Dataset Details:** 25 months of historical 5-minute interval stock price data extracted via the Interactive Brokers (IBKR) API, including OHLCV features for major U.S. equities.
- **Python Libraries and Versions:** Python 3.11, NumPy 1.24.3, Pandas 2.2.2, Matplotlib 3.8.4, SciPy 1.13.0, Scikit-learn 1.4.2, PyTorch 2.2.2, Optuna 3.5.0, SQLite3 (built-in), Joblib 1.4.2, IBAPI 9.81.1, Pytz 2024.1.

4 Model Justifications

The prediction of short-term stock price movements benefits from capturing both *spatial dependencies* (patterns among technical indicators) and *temporal dependencies* (sequential price behavior). Convolutional neural networks (CNNs), long-short-term memory networks (LSTMs), and gated recurring units (GRUs) were selected due to their complementary strengths:

- **CNNs** are effective in extracting hierarchical characteristic patterns from the multivariate technical indicator space by applying localized convolution operations.
- **LSTMs** model long-term dependencies in sequential data while mitigating the vanishing gradient problem common in traditional Recurrent Neural Networks (RNNs).
- **GRUs** provide a computationally efficient alternative to LSTMs while maintaining comparable performance, making them suitable for faster training and inference.

This ensemble allows the model to capture both *cross-sectional patterns* and *sequential dynamics*, essential for improving prediction accuracy in volatile financial markets.

5 Methodology

5.1 Data Collection

The dataset was acquired using the Interactive Brokers (IBKR) API, retrieving 5-minute interval historical OHLCV (Open, High, Low, Close, Volume) data over a 25-month period. A multithreaded worker system was employed to efficiently manage the data extraction process.

Each worker thread operated by pulling a request for a specific end-time chunk from a shared queue and submitting it to the IBKR API for historical data retrieval. Due to the inherent latency and potential delays of the IBKR historical data API (e.g., data caching or slow return times), if a response was not received within 2 minutes, the request was automatically requeued, and a new request was issued. Currently, the system monitored late-arriving responses; if a previously pending request returned after being requeued, it was captured and integrated into the data set.

The system was designed to comply strictly with IBKR's historical data API rate limits: no more than 6 requests every 2 seconds and no more than 60 requests every 10 minutes. Once data for each chunk were collected, it was added to a central dataset and written to a SQLite database. The entire process took approximately 12 minutes per ticker and was repeated for SPY (as the market proxy), TLT (as the bond proxy) and a comprehensive list of individual stocks.

5.2 Data Preprocessing and Feature Engineering

Historical 5-minute interval data were retrieved from a local SQLite database for each individual stock ticker, together with benchmark datasets for SPY and TLT. For each dataset, 17 technical indicators were engineered, including Simple Moving Averages (SMA), Exponential Moving Averages (EMA), Bollinger Bands, Relative Strength Index (RSI), Moving Average Convergence Divergence (MACD), Average True Range (ATR), Commodity Channel Index (CCI), Rate of Change (ROC), Stochastic Oscillator, and Williams %R, among others.

The three datasets were subsequently merged, resulting in a final feature set of 66 features per sample and approximately 42,000 data points. The target variable was constructed as the next 5-minute close-to-close price difference for the respective stock, representing the short-term return.

Data cleaning involved removing duplicate timestamps and eliminating any rows with missing values to maintain integrity. All features were standardized to zero mean and unit variance using a `StandardScaler`, ensuring compatibility with gradient-based optimization methods.

The dataset was then partitioned into training, validation, and test sets using an 80-10-10 split ratio. Time series were structured into fixed-length sequences of 20 timesteps to capture short-term temporal dependencies. The data was finally transformed into PyTorch tensors, with dimensions adjusted to match model-specific input requirements: 4D tensors for the CNN models, and 3D tensors for the LSTM and GRU models.

5.3 Model Development

Three distinct deep learning architectures were developed to model short-term stock price changes: a Convolutional Neural Network (CNN), a Long Short-Term Memory (LSTM) network, and a Gated Recurrent Unit (GRU) network. Each model was designed to accommodate the sequential nature of financial time series data, with minor adaptations for their specific inductive biases.

5.3.1 Convolutional Neural Network (CNN)

The CNN architecture was composed of multiple convolutional layers with increasing filter depths (e.g., 32 and 64 filters) and kernel sizes of 3. Each convolutional block was optionally followed by batch normalization layers to stabilize training, ReLU activation functions for non-linearity, and max pooling layers to reduce spatial dimensions and extract hierarchical features. After the convolutional layers, the feature maps were flattened and passed through fully connected layers with dropout regularization to mitigate overfitting.

The CNN was initialized using Xavier (Glorot) uniform initialization for all weight matrices, which helps maintain variance stability across layers and accelerates convergence. Input tensors were reshaped to four dimensions: [batch size, channels, sequence length, features] to comply with the convolutional layer requirements.

5.3.2 Long Short-Term Memory (LSTM)

The LSTM network was designed with two recurrent layers, each with 64 or 128 hidden units. Dropout was applied between layers to prevent co-adaptation of neurons. The final hidden state from the last time step was batch-normalized and passed through a fully connected layer to output the predicted return.

The LSTM weights were initialized with Xavier uniform initialization for weight matrices and zeros for biases. Inputs to the LSTM were three-dimensional tensors structured as [batch size, sequence length, features].

5.3.3 Gated Recurrent Unit (GRU)

The GRU architecture followed a similar structure to the LSTM but with a simplified gating mechanism. The GRU used two recurrent layers with dropout, batch normalization after the recurrent layers, and a final fully connected layer. Like the LSTM, GRU models are particularly suited to handling long-range dependencies in time series data without suffering from the vanishing gradient problem.

The GRU parameters were also initialized using Xavier uniform initialization for weights and zeros for biases. Inputs followed the same three-dimensional format as the LSTM.

5.3.4 Input-Output Format

Each model took as input sequences of 20 time steps with 66 features per time step. The output was a single scalar value representing the predicted 5-minute return (Δ Close) for the target stock. Models were trained using Mean Squared Error (MSE) loss and optimized with the Adam optimizer. Early stopping based on validation loss was employed to prevent overfitting.

5.4 Hyperparameter Tuning

To optimize model performance, I conducted hyperparameter tuning using Optuna, an efficient open-source hyperparameter optimization framework. The search space included key parameters such as learning rate, batch size, number of layers, and dropout rate. Optuna's Tree-structured Parzen Estimator (TPE) algorithm was employed to efficiently navigate the search space, balancing exploration and exploitation. The final selected hyperparameters were those that minimized the validation loss, ensuring better generalization to unseen data.

5.5 Model and Ensemble Evaluation

After identifying the optimal model through hyperparameter tuning, I evaluated its performance using several standard metrics. For regression tasks, the mean squared error (MSE), the mean absolute error (MAE) and the coefficient of determination (R^2) were calculated to assess predictive precision. To further assess directional generalization, the continuous output was discretized into classification labels, and confusion matrices were generated to evaluate classification performance. Additionally, I explored all possible ensemble combinations among the three individual models, resulting in seven different ensemble strategies. Out of these, the ensemble combining all three models demonstrated the best performance on unseen test data, highlighting the benefits of model aggregation in improving robustness and generalization.

Model	**MSE**	**MAE**	R^2	**TP / FP**	**FN / TN**
GRU	0.661	0.469	-0.024	1305 / 637	1296 / 683
LSTM	0.647	0.456	-0.001	1942 / 0	1979 / 0
CNN	0.646	0.456	0.0002	704 / 1238	683 / 1296
GRU + LSTM	0.649	0.460	-0.005	1440 / 502	1465 / 514
GRU + CNN	0.648	0.459	-0.004	1264 / 678	1256 / 723
LSTM + CNN	0.646	0.456	-0.0002	1685 / 257	1740 / 239
GRU + LSTM + CNN	0.645	0.455	0.0004	1545 / 397	1214 / 765

Table 1: Model performance metrics for AMZN: Mean Squared Error (MSE), Mean Absolute Error (MAE), R^2 score, and confusion matrix breakdown.

5.6 Backtesting

To evaluate the real-world applicability of the model's predictions, I developed a custom Backtesting framework. This framework learned two key hyperparameters: the stop-loss level and the entry threshold, while also accounting for slippage and commission costs. Trades were executed at the open of the next candle following a signal, and to avoid overlapping positions, only one active trade per stock was allowed at a time. I performed a hyperparameter search to identify the configuration that maximized total profit while enforcing a minimum number of trades over the testing period to ensure statistical significance.

This approach enabled a more realistic and robust evaluation of the model's trading performance.

Metric	Value
Stock	AMZN
Best Profit ($)	53.71
Stop-Loss Ratio	0.508
Entry Threshold	-0.237
Number of Trades	750
Buy-and-Hold Profit ($)	11.61

Table 2: Optimized trading parameters and performance for 2 shares of AMZN.

6 Live Trading and Results

6.1 Real-Time System Deployment

For live deployment, the system was integrated with Interactive Brokers (IBKR) via their API. Each stock was allocated a uniform share of the portfolio to ensure equal exposure. The system employed 20 worker threads operating on 29 trader objects, covering 28 securities. One dedicated thread fetched SPY and TLT data, which was shared with all stock-specific threads. Each worker merged SPY and TLT with its assigned stock's data, performed feature engineering, normalization, and generated predictions before placing results into a shared queue.

Due to IBKR API rate limits, trading was executed in 5-minute intervals. SPY and TLT updates were fetched first to synchronize market information before stock-specific threads began processing, ensuring efficient and timely trade execution.

6.2 Risk Management and Execution Logic

Trades were managed using bracket orders, setting both stop-loss and take-profit levels automatically to control risk and secure gains. To avoid overlapping exposures, only one open position per security was allowed at any time. A minimum trade count and profit thresholds were enforced to filter out low-quality signals. Portfolio exposure was dynamically adjusted through periodic rebalancing, and strict position sizing rules were applied to maintain diversification and limit risk concentration.

Metric	Value
Ticker	AMZN
Entry Price ($)	204.91
Position	Long
Stop-Loss ($)	204.86
Take-Profit 1 ($)	205.11
Take-Profit 2 ($)	205.31

Table 3: Trade Execution Details for 2 shares of AMZN

7 Automated Cloud-Based Pipeline

To ensure uninterrupted trading and continuous model updates, the system was designed for fully automated operation in a cloud environment. The entire workflow — from historical data collection to model training, Backtesting, and live deployment — was orchestrated within a single execution script.

New ticker data can be fetched and processed dynamically without stopping or interrupting the system. For stocks without existing models, data is automatically collected, features are generated, and new models (CNN, LSTM, GRU) are trained and optimized using Optuna-based hyperparameter tuning. These models are Backtested, and their performance is evaluated against existing stocks. Only tickers whose models meet or exceed profitability thresholds are added to the live trading pool, maintaining a cap of 28 stocks for active trading based on the top-performing models.

The architecture leverages multi-threading and hardware acceleration (CUDA, MPS, or CPU) for efficient data processing and model inference. This design ensures that the system can adapt to new opportunities and evolving market conditions in near real-time without requiring manual intervention or downtime.

8 Discussion of Live vs Backtest Results

While Backtesting showed promising results, live trading consistently resulted in negative daily profits.

Trades | Summary | Enter symbol filter | More options

Fin Instr	Buys	Sells	Net	Avg (BT)	Avg (SLD)	Total (BOT)	Total (SLD)	Net Total	Comm	Nt Incl. Cmm
WFC	27,...	27...	0	73.87335	73.88461	2,007,212....	2,007,518.76	305.92	278.13	27.79
MSFT	1,200	1,...	0	458.12...	458.17800	549,747.84	549,813.60	65.76	50.29	15.47
JNJ	4,234	4,...	0	152.70...	152.69509	646,562.76	646,511.00	-51.76	63.01	-114.77
CVX	2,576	2,...	0	136.65...	136.61890	352,016.84	351,930.29	-86.55	33.61	-120.16
HD	2,400	2,...	0	366.65...	366.53942	879,974.20	879,694.60	-279.60	80.57	-360.17
GS	1,836	1,...	0	603.19...	603.05235	1,107,465....	1,107,204.12	-261.00	102.43	-363.43
NVDA	15,...	15...	0	140.02...	140.01044	2,181,672....	2,181,502.67	-169.82	202.69	-372.51
GOOGL	14,...	14...	0	171.56...	171.55519	2,463,164....	2,463,017.84	-146.51	228.41	-374.92
BA	6,572	6,...	0	207.63...	207.59442	1,364,560....	1,364,310.50	-249.59	125.57	-375.16
AAPL	12,...	12...	0	199.72...	199.71279	2,483,601....	2,483,428.49	-172.80	229.95	-402.75
AMD	21,...	21...	0	113.13...	113.11764	2,402,922....	2,402,618.72	-303.46	228.05	-531.51
JPM	6,911	6,...	0	263.64...	263.58810	1,822,038....	1,821,657.34	-380.77	169.64	-550.41
AMZN	11,...	11...	0	205.96...	205.93021	2,374,537....	2,374,169.44	-368.05	221.74	-589.79
TSLA	6,733	6,...	0	360.68...	360.62573	2,428,507....	2,428,093.03	-414.45	223.62	-638.07
META	2,380	2,...	0	645.18...	644.93900	1,535,532....	1,534,954.82	-578.00	140.57	-718.57
MRNA	36,...	36...	0	26.90411	26.89267	973,848.07	973,433.83	-414.24	371.62	-785.86
NFLX	1,224	1,...	0	1,188.3...	1,187.75941	1,454,586....	1,453,817.52	-768.60	136.29	-904.89
COST	1,848	1,...	0	1,009.8...	1,009.14852	1,866,184....	1,864,906.47	-1,278.27	176.44	-1,454.71

Figure 1: Screenshot of Live Daily Trade Summary

The figure above depicts a typical daily trade summary. Although a few stocks, such as WFC and MSFT, closed with marginal positive gains, the majority of securities incurred losses, often substantial. A key contributor to the poor performance was high commission costs, which accumulated rapidly due to the high-frequency nature of the trading strategy — thousands of shares were traded every few seconds across multiple stocks.

The discrepancy between the Backtesting and live trading results are mainly due to a few factors. First, trading hyperparameters tuned during Backtesting may have been overfit to historical data, reducing their effectiveness in live conditions. Second, Backtesting was conducted using 5-minute interval data, whereas live trading operated on continuous tick data, making it difficult to determine whether stop-loss or take-profit levels were hit first within a candle. This mismatch introduced bias into Backtest results. Future Backtesting must leverage tick-level data for more accurate simulation, though access to such data remains limited and costly.

Additionally, live order execution faced challenges due to tight stop-loss and take-profit thresholds. Normal bid-ask spreads were often wide enough to immediately trigger both entry and exit orders, resulting in instant losses. Future improvements could include using wider trade thresholds, selecting stocks with tighter bid-ask spreads, or implementing slippage-adjusted execution models to mitigate these issues.

9 Conclusion

This project demonstrated the feasibility of building an end-to-end automated trading system integrating deep learning models with the Interactive Brokers (IBKR) API for real-time market execution. Convolutional Neural Networks (CNNs), Long Short-Term Memory (LSTM) networks, and Gated Recurrent Units (GRUs) were developed and optimized using advanced hyperparameter

tuning strategies. Ensemble models consistently outperformed individual architectures in backtesting, validating the benefits of model aggregation.

While historical simulations showed strong predictive performance and promising risk-adjusted returns, live trading revealed key challenges, including execution latency and model generalization under real-time market conditions. In particular, the simplicity of the model architectures and the limited set of input features contributed to the system's inability to generate positive gains during live trading. Furthermore, due to the inherent noise present in short-term stock price data and the restricted feature set, the ensemble model combining CNN, LSTM, and GRU architectures was not able to profitably trade in live markets and thus proved ineffective at reliably predicting short-term price movements. These limitations rendered the models suboptimal for real-world deployment in managing actual trading capital. The discrepancies between backtesting and live results highlight the critical importance of using high-fidelity backtesting methods and robust live execution pipelines in financial machine learning applications.

Future improvements, such as incorporating tick-level data, dynamic model retraining, and latency-aware execution models, are essential for bridging the gap between historical performance and live trading outcomes. Overall, this work lays the foundation for more sophisticated and adaptive algorithmic trading systems, demonstrating both the potential and challenges of applying deep learning in financial markets.

10 Limitations and Future Improvements

10.1 Methodological Limitations

Several constraints affected the development and evaluation of the models. First, the availability of historical high-frequency data was limited; acquiring comprehensive tick-level or minute-level data is both costly and time-intensive, restricting the breadth of training and testing datasets. Hyperparameter optimization was also constrained — each model's Optuna search was limited to 10 trials due to the significant computational expense associated with training deep learning models on large datasets.

Moreover, hardware limitations posed additional challenges. Access to CUDA-enabled GPUs was restricted to one hour per day via Google Colab Pro+, severely limiting the time available for model training and hyperparameter tuning. While Apple's Metal Performance Shaders (MPS) backend provided an alternative for local training on an M1 MacBook Air, it was substantially slower than CUDA-based training, further elongating experimentation cycles.

Additionally, it was observed that for certain stocks, individual models failed to identify meaningful patterns within the data. In such cases, the model would default to a degenerate solution, predicting the same output for every instance — either consistently forecasting all upward (long) or all downward (short) movements — rather than learning nuanced predictive relationships. This be-

havior highlights challenges in training robust models under noisy or weakly structured financial datasets.

10.2 Live Trading Limitations

In live trading, additional operational constraints surfaced that were not visible during Backtesting. Each cycle involved fetching live data, preprocessing, feature generation, model inference, and order submission, introducing several seconds of latency per stock. This cumulative delay can have a significant impact on trade execution prices, particularly in volatile market conditions where rapid price movements are common.

Backtesting on 5-minute interval data did not simulate such latencies, leading to an optimistic bias in performance evaluation. Real-world market frictions, including network delays, API response times, and exchange matching delays, were not captured, highlighting the gap between historical simulation and live trading environments.

11 Future Related Research Inspiration

This work opens several avenues for future research. Incorporating reinforcement learning techniques, such as Deep Q-Learning or Proximal Policy Optimization (PPO), could enable the model to dynamically learn optimal trading policies instead of relying on fixed thresholds.

Further, integrating alternative data sources such as news sentiment analysis, earnings reports, and macroeconomic indicators may enhance model accuracy and robustness. Finally, exploring high-frequency trading strategies using millisecond-level data and improving execution algorithms to minimize slippage remain compelling areas for further investigation.