Stratoscan: A Natural Language Processing Pipeline for AI-Driven Quantitative Equity Screening

Tanay Chatwani[1]

Received November 18, 2025
Accepted August 7, 2026
Electronic access Decemeber 15, 2026

Background/Objective: Currently, quantitative stock screening systems often involve a graphical user interface or complex understanding of financial variables, which can be difficult for students and non-investors to use. This research introduces and tests Stratoscan, a complete web-based system for converting natural language stock screening descriptions into a quantitative screening system using a fine-tuned NLP system.
Methods: Stratoscan uses a fine-tuned NLP system to convert natural language stock screening statements into a quantitative screening system. The NLP system was built using JavaScript and Python code, utilizing the DeepSeek Coder 1.3B model. A synthetic dataset of 2,076 natural language to code conversions was used to fine-tune the NLP system using supervised learning via cross-entropy loss. Polygon/Alpaca's real-time data API was used for stock price data. A BERT model was used for sentiment analysis of financial news. Performance was tested on 100 new statements using three levels of complexity.Statement parsing was tested on the ability of the generated code to successfully run and return a list of stocks that made semantic sense according to the original statement. Three independent analysts scored these results.
Results: The fine-tuned NLP system converted 96% of statements to code correctly. On average code was generated in 1.8 seconds. When compared to TradingView, multi-factor screening creation and execution time was reduced by 78%.
Conclusions: Stratoscan proves that small domain specific language models can be useful for democratizing stock screening. Extensions to foreign exchange and cryptocurrency screening are planned.

Keywords: natural language processing, FinTech, quantitative finance, stock screening, DeepSeek Coder, sentiment analysis, financial technology

Introduction

Background and Context

The finance industry produces a vast amount of data that is continually increasing. This data includes stock prices in real-time, trading volume data, technical analysis charts, fundamental data for companies, and also data from earnings reports and financial news.

Quantitative screening involves taking a universe of securities and filtering them down to a subset that satisfies a specified set of quantitative conditions. This approach has long been the mainstay for utilizing this data in investment applications. However, current screening tools (like Thinkorswim, Finviz and TradingView) require the user to choose options from a series of dropdown menus, interpret the math behind a technical indicator, and build a filter chain from a graphical user interface that does not easily support complex or multi-condition queries. This approach also favors users with a pre-existing background in technical and financial analysis. Newcomers, students and retail users face barriers in accessing advanced screening tools.

Recent capabilities afforded by large language models (LLMs) have dramatically changed the interaction between natural language and code. LLMs that have been exposed to large quantities of natural language and code have shown an ability to convert natural language into executable code based on user intent. This intent-to-code ability represents a new potential in investment applications. A user could express their investment thesis in natural language and an LLM could automatically create the appropriate screening code and execute it without the user having to possess any technical knowledge.

Problem & Motivation

While there has been a surge in LLMs that can generate code, there has not been much adoption of these models in quantitative screening. Sentiment analysis tools do not integrate well with technical screening tools and the results from these disparate tools often require manual translation. Additionally, general code generation models are not built with the domain

[1] *Stephen F. Austin, US*

© The National High School Journal of Science 2026

and knowledge of finance or the lexicon necessary to correctly interpret natural language relating to investment. There is thus a distinct lack of an integrated one-stop solution that enables natural language processing, technical screening functions and sentiment filtering to be combined into an approachable product.

Research Purpose and Importance

This project aims to create, implement and test Stratoscan, a web-based integrated platform that enables translation between natural language and financial data via intelligence. This project is important for several reasons. Firstly, by showing that a small open-source language model can be tuned to translate intent to code for a domain specific use case (financial screening), a template is provided for future use in the FinTech space. This project also proves that intelligent domain specific AI can bridge the gap between expert use of advanced financial functions and non-expert investors without the need for access to expensive and large closed proprietary AI models.

Research Goals

The research goals for this project are as follows:

i. To create and implement a full-stack web application which can receive natural language stock screening requests and deliver relevant up to date data on screening.

ii. Create a tuning process for the DeepSeek Coder 1.3B language model using domain specific data consisting of natural language to Python/Pandas code translations that allows translation of intent to code for stock screening.

iii. Implement a sentiment analysis system based on BERT that can be used within the screening system to allow the use of both technical filters and sentiment from news items in a single request.

iv. Test the system on parsing correctness, runtime performance and output correctness using 100 new blind test requests.

Scope and Limitations

This project aims to design and evaluate a system, called Stratoscan, that can be used on US stocks, using data from the Alpaca and Polygon APIs for real-time stock data and financial news from the Finnhub API. This system is only for reading and not trading or portfolio management. This project does not consider international stocks, options filtering or high frequency trading. Some limitations to this project include the local inference cost of the LLM used, bias that can be present due to synthetically generated training data and reliance on third party data and API availability.

Theoretical Background

This project uses the concept of Intent-to-Code Translation within natural language processing. Intent-to-Code is the task of translating unstructured intent from natural language to structured code. In this use case, a fine-tuned language model is used as a domain specific compiler. Natural language describing an investment intent is input to the model and an equivalent Python/Pandas script is output that implements that intent as a data filtering function. This can be very relevant for finance as investment intents can be described quite naturally but need to be implemented with exact logic.

Methodology

There were four main areas of this project. These were design and implementation of the system, generation of a synthetic dataset and fine tuning of the model, implementation of the sentiment analysis component and finally evaluation using a blinded test set of queries. These are all further detailed in the Methods section.

Literature Review

Natural Language Interfaces to Code and Data

Natural language interfaces to structured data have been researched for some time. Early research on natural language interfaces to databases proposed the idea of enabling non-expert users to query data in a database using limited forms of English. Androutsopoulos et al.[1] laid out some of the early foundations for natural language interfaces to databases. Recently, the rise of the transformer has allowed natural language interfaces to achieve much more. Devlin et al.[2] introduced BERT, a bidirectional encoder representation model that achieved state-of-the-art performance on many tasks in natural language understanding. BERT has since been applied to many domain specific tasks in NLP such as financial sentiment analysis. Brown et al.[3] also introduced generative transformers in their article on GPT-3. These models showed that a sufficiently large language model could generate code given a natural language input with minimal or no fine-tuning, given sufficient few-shot examples. This has allowed for intent-to-code translation to become a viable research problem. Quamar et al.[4] also recently provided an overview of natural language interfaces to structured data showing how fast this line of research has developed.

Intent-to-Code and Code Generation

Intent-to-code has also been the object of a fair bit of research over the past few years. Chen et al.[5] proposed a model, Codex, which is a fine-tuned version of GPT-3 on a massive

© The National High School Journal of Science 2026

amount of publicly available code. Using Codex, they were also able to generate Python functions given a docstring with good performance on the HumanEval data set. Austin et al.[6] also created a programming benchmark, MBPP, consisting of programming problems presented in plain English. They also looked at how language models can solve these programming problems. They found that model size and domain-specific training data were the most important factors in solving these problems.

Fried et al.[7] used infilling tasks on code to create InCoder. Nijkamp et al.[8] presented CodeGen which showed that relatively small models can engage in multi-turn conversations about code. Guo et al.[9] presented DeepSeek-Coder and showed that a 1.3 billion parameter model can perform competitively across code generation tasks when trained on carefully selected code collections. This is very useful for deployment in low-resource settings like in this project. Lahiri et al.[10] presented an interactive code generation system which formalizes user intent through test writing. This increases reliability when the user intent is expressed in ambiguous natural language. Sarhan and Jabr[11] reviewed large language models for code generation and place each of these in context.

Financial NLP and Sentiment Analysis

There is already a large amount of research on using NLP on financial data. Loughran and McDonald[12] created a financial sentiment lexicon that is tuned to the lexicon of securities filings. They showed that off-the-shelf sentiment lexica do not perform well on financial data because some domain specific words exist in securities filings. Malo et al.[13] created the Financial PhraseBank dataset to serve as a resource for sentence level sentiment analysis on financial news. This dataset has been used for both evaluation and fine-tuning. Araci[14] created FinBERT, a BERT model fine-tuned on financial data. FinBERT performed best on all tasks for financial sentiment analysis. This highlights the importance of domain adaptation over off-the-shelf models. Finally, Yang et al.[15] also showed the importance of domain pre-training for financial NLP tasks over off-the-shelf BERT models. Sentiment indicators have also been used in quantitative trading research. Bollen et al.[16] showed that sentiment derived from Twitter was correlated with the next day change in the Dow Jones Industrial Average at a statistically significant level. This serves as an early indication that unstructured text may have predictive power over the stock market. Sousa et al.[17] also showed that BERT-based models can be used on stock market sentiment. This serves to validate the use of sentiment using transformer models in finance.

Retail Access to Quantitative Equity Screening

Quantitative screening of equities is something that is widely used by institutional investors. Fama and French[18] created their three factor model which showed that market and book to market explained a significant amount of cross-sectional variation in stock returns. This set up fundamental screening factors for use by retail investors. Other research has expanded on these three factors to add profitability[19] and momentum[20] which are also widely used screening factors by retail investors today. While there is a large body of research on factor investing, Belak et al.[21] found that retail investors often do not have the knowledge or tools to implement multi-factor models which can lead to suboptimal investment decisions. While APIs like Polygon and Alpaca have helped to open up financial data to retail investors, the complexity of the screening tools available still creates barriers for retail investors.

LLMs and Finance

Another area of interest in using LLMs has appeared in finance outside of sentiment. Lopez-Lira and Tang[22] used ChatGPT to predict stock returns from financial news and found statistically significant return predictions. This shows that LLMs may hold information of use in finance beyond basic sentiment. BloombergGPT[23], a 50-billion-parameter model trained on a large proprietary financial corpus, demonstrates strong performance across financial NLP benchmarks. However, the computational requirements of models at this scale render them inaccessible for lightweight deployment. Such large models, however, are difficult to apply in lightweight systems due to their high compute demands. Instead, Xie et al.[24] have investigated the application of instruction-tuned language models to financial question answering. They found that smaller instruction-tuned language models on financial instruction datasets can achieve performance similar to much larger general purpose models. Zhao et al.[25] have conducted a review of the applications of LLMs to many domains and found financial services to be particularly suited for domain adapted language models. These findings all point to the feasibility and practical applicability of lightweight fine-tuned language models in financial NLP tasks. This study provides evidence for this assertion.

Gap in the Literature

There have been numerous attempts to apply LLMs to finance, natural language to code, natural language to sentiment in finance, and code generation. There has yet to be a system that combines all four of these applications into one user-friendly system that can be tested on real market data. Other current systems either have a high barrier to use for non-technical users, do not combine sentiment with technical indicators, or use proprietary very large models that are impractical for consumer-facing products. Stratoscan uses a lightweight code generation model that has been fine tuned to financial code, a

sentiment model based on BERT and a live market data API to create a system that is fast, accurate and available to non-technical users.

Methods

Research Design

This study used an experimental and development research design. In the experimental design, the performance of the AI core of the Stratoscan system was evaluated on parsing accuracy, execution time and accuracy of the outputs. In the development design, a complete system that can be implemented into a practical product was created. This implementation thus sought to provide quantitative testing of the AI system as well as testing of the system in an actual screening situation to assess usability.

Participants or Sample

Two datasets were used in this study. The first is the AI Training Dataset. This dataset consists of 2,076 synthesized examples of natural language queries to code that filter financial data. These queries were generated with a generic large language model using a variety of natural language investment queries of varying complexity. From the simplest query containing only a single financial indicator to queries that contain multiple indicators and sentiment indicators the queries were created. For each query a corresponding code in Python/Pandas was created to filter the data according to the query. These queries and codes were then checked by the researcher for correct indicator usage, correct meaning between the query and code, and syntactic correctness. If any of these checks were not met the query/code pair was either corrected or removed from the dataset. The use of synthesized training data brings about some issues regarding how natural language queries are distributed in both realism and variety. However, through the quality checking described above only natural language that was reasonable and could be expected to be used in actual investment screening was included in the dataset. Some potential issues with bias could be over usage of certain indicator combinations and under usage of natural language that is more ambiguous or idiomatic and may be used in actual applications.

The second dataset is the Validation Query Dataset. This dataset consists of 100 queries not created from or in the AI Training Dataset. These queries were created by five people with varying degrees of financial knowledge including high school students with no prior exposure to quantitative finance. This was done in order to create a validation dataset with actual variation in natural language usage and not usage characteristic of the natural language generation tool used to create the training data. All participants gave informed consent and agreed to participate voluntarily.

Data Gathering

We used the Polygon and Alpaca APIs for real-time stock data such as stock prices, volume and company data. We used the Finnhub API to acquire financial news articles used as input to the sentiment analysis component. Data is only acquired on demand when requested and is not stored between runs of filtering operations. No personally identifiable information was gathered in this project.

Variables and Measurements

The main outcome variable in this project is the Query Success Rate. This is the percentage of validation queries for which the system produced valid Python code that ran without runtime exception and produced a non-empty list of stocks semantically matching the query. The following variables were also created and measured.

Parsing Accuracy: We measured each parsed result on two fronts. First we evaluated if the code was syntactically correct. This means that the code would run successfully in Python without exception. We also evaluated if the code was semantically correct. Semantic correctness refers to whether the filtering operations in the code match the meaning of the natural language query. Three independent coders rated each query and generated code as correct or incorrect. Cohen's Kappa was used to measure agreement between the coders and produced a value of 0.91 which represents a high level of agreement among the coders.

Execution Time was the amount of seconds between the time a query was made in the frontend and the time a list of stocks was rendered. This includes time spent on the API data gathering, LLM output, code execution and rendering in the frontend.

Accuracy of the Result refers to the percentage of stocks in the output which meet all the filtering conditions in the user query. For each of the 96 successful queries, the outputted stocks were evaluated against all of the provided filtering conditions in the query. This included conditions on volume, sentiment, RSI values, etc. in the query, which were checked to ensure they were implemented appropriately in the generated code. Additionally, the outputted stocks were checked to verify they met the filtering conditions when applied to the outputted stocks from the obtained stock data.

The size of the test set was set to $N = 100$ based on a power analysis for a one-sample proportion test. With a 95% expected success rate, a 95% confidence level, a 0.05 significance level, and a desired precision of $\pm 4.3\%$, a minimum of 100 queries was calculated to be statistically enough to ob-

© The National High School Journal of Science 2026

serve significant differences from the expected success rate with 95% power.

Procedure

System Overview and Implementation

Stratoscan was implemented as a full stack web-based application that consisted of three main components. The user would interact with a text-based conversational interface built with React.js that would take natural language queries for filtering stocks. The backend was implemented using FastAPI in Python and would be responsible for processing the entire pipeline from the user query, to the LLM inference server, to the execution of the generated code on real-time stock data, to finally returning the filtered stocks to the user interface. An inference server was created for the DeepSeek Coder 1.3B model and was made available to the backend application through a local REST API. The entire system was created and run on the Replit cloud development platform. A virtual machine instance was provided with 2 virtual CPU cores and 4 GB of RAM. Since the model was relatively small in number of parameters, CPU-only inference was sufficient to meet an end-to-end latency of under two seconds.

Fine-Tuning

The base DeepSeek Coder 1.3B model was fine-tuned using a supervised learning approach with causal language modelling on the synthetic data used for training. Given a natural language input it is expected to generate the target code given a context. The model was trained for 10 epochs using the AdamW optimiser at a learning rate of $2e{-}5$, with a linear warmup across the first 10% of the training iterations. A weight decay of 0.01 was also used. The batch size used for training was 8 query-code pairs. The model was trained to minimise the cross entropy loss for the target code given the natural language input. After each epoch a 10% hold out set of queries was used to calculate the validation loss. If the validation loss does not improve over 2 consecutive epochs then training is stopped to help avoid overfitting. The final model used is the checkpoint achieving the minimum validation loss.

Incorporating Sentiment on Screening

Sentiment analysis for financial news is introduced into the screening system using a fine-tuned BERT based sentiment analysis model on the Financial PhraseBank dataset. For queries that incorporate a sentiment filter, for each of the potential stocks in the query the 10 most recent financial news items related to that stock are retrieved using the Finnhub API. A mean sentiment score is calculated for these news items and then a user defined sentiment filter is also applied. This sentiment score is then used in conjunction with the technical filters provided by the output of the DeepSeek Coder model to provide a combined hybrid screening result.

Testing

The 100 validation queries were run through the live Stratoscan system in sequence with all queries being treated in the same way. The Python code produced for each query was captured automatically. The processing time for each query was also recorded automatically using server timestamps. For each successful query, the generated stock list was scored by three experts using the criteria from the natural language query.

Data Analysis

Query parsing accuracy was determined by the number of queries successfully parsed divided by the number of validation queries. This value was expressed as a percentage and a 95% confidence interval was generated using the Wilson score interval. Execution time was reported as mean, standard deviation and 90th percentile. Agreement for semantic correctness was determined using Cohen's Kappa. Accuracy of the generated results was determined through expert agreement. A stock list was deemed accurate if all three experts determined that the list met both explicit and implicit criteria from the original query. All numerical data analysis was performed using Python with the NumPy and SciPy packages.

Technical Indicator Explanations

The following technical indicators were available through the Stratoscan screening system. Explanations for each indicator are as follows.

Relative Strength Index (RSI)

The RSI is a momentum oscillator that measures the speed and direction of price changes over a period of time using a value from 0 to 100. It is calculated as:

$$\mathrm{RSI} = 100 - \left[\frac{100}{1+\mathrm{RS}}\right]$$

where RS is the average gain divided by the average loss over a specified number of periods (usually 14 days). An RSI value above 70 is often interpreted as overbought while a value less than 30 is often interpreted as oversold.

Moving Average Convergence Divergence (MACD)

The MACD is a trend following momentum indicator that is calculated as the difference between the 12-period EMA and the 26-period EMA of the closing price:

$$\mathrm{MACD} = \mathrm{EMA}(12) - \mathrm{EMA}(26)$$

© The National High School Journal of Science 2026

A 9-period EMA of the MACD is used as a signal line to generate trading signals based on crossovers. When the MACD is positive the price is trending up and when the MACD is negative the price is trending down.

Ethics

Publicly available data from stock markets were obtained using professional APIs that are publicly available. The data were obtained using the APIs terms of service. No personally identifiable information was obtained or stored at any point in this work. All participants in the authoring of the validation queries gave informed consent to participate and all participation in this was voluntary. The Stratoscan system does not suggest, simulate or execute trades of any kind. The system only queries data.

Results

Accuracy of query to code translation

The fine-tuned DeepSeek Coder 1.3B model performed well at translating investment queries into screening code in Python/Pandas for all three levels of query complexity present in the validation set. 96 out of 100 new queries in the validation set were translated successfully. This gives an overall query to code translation accuracy of 96% (95% CI: 90.1%–98.9%, Wilson score interval). The accuracy for each query type is provided in Table 1.S

Simple single indicator queries were translated with 100% success which indicates that simple screening conditions are processed well. Multi-factor queries were translated with 96% success. Sentiment queries required both the technical screening code from the LLM as well as the integration of the sentiment prediction code from the BERT model to be processed correctly. For these queries translation success was 92%.

Failure case analysis

There were four queries in the validation set that were not translated successfully. For each of these failure cases, the reasons for the failure and suggestions for future model improvements were determined by examining each failure case.

Failure Case 1 - Spikes in volume on different timeframes and conflicting RSI conditions. The query "Find stocks where RSI is above 70 on the 1-hour chart and volume is spiking on the daily chart" required two different calls to the API to be parameterized with different candle data intervals. One call should use hourly data while the other should use daily data. One request that failed was the request for an RSI condition on daily data using only the daily interval. The system attempted to evaluate the RSI on daily data instead of hourly data. This is an example of the system failing to split a multi-timeframe request into several requests for data with different intervals.

Implicit timeframe use and cross-indicator request Failure Case 2: "Show me tech stocks that had a MACD crossover this week and were oversold last month." This request involves two conditions that are at different times. One condition is a recent MACD crossover and the other condition is an RSI value from last month. Each condition uses a different number of periods in the back calculation for the request. The system did implement the crossover for the MACD correctly, but the system used a 14-period RSI instead of a condition based on last month. This led to a semantically incorrect response.

Ambiguous natural language modifier on a technical condition Failure Case 3: "Find undervalued stocks with really positive news lately." In this request "really positive" is being used as a modifier to express a particular level of positive news. The system interpreted this to be a sentiment value and used a sentiment value of 0.5 (where 1 is positive and 0 is negative). The expected value was determined by the reviewers to be somewhere at 0.75 or above. The condition worked but it was determined to be semantically incorrect based on the difference between the interpretation and the expected meaning of the request.

Multi-condition request with implicit fundamental and sector request Failure Case 4: "Give me profitable small cap healthcare stocks with low debt and improving margins." This request involves a sector request, a market cap request, a profitability request, a margin improvement request over multiple reporting periods, and a low debt request. The profitability, market cap, sector and margin trend conditions are all encoded properly.

However, the logic for margin trend is not implemented. Margin trend uses gross margin from one quarter of earnings data and compares it to another quarter of earnings data. This is a multi-period fundamental analysis and is more complex than any of the multi-period analysis used in the training data.

These four error types imply that the biggest shortcomings in the current implementation are in multi-period and temporally-related condition analysis, multi-period fundamental analysis, ambiguous sentiment analysis, and multi-timeframe co-ordination. These error types can be used to guide the augmentation of the training data in a future iteration.

Execution Time

Execution time for the complete solution was measured for all 100 validation requests. This execution time includes the time from when the request is submitted until when the solution is rendered on the frontend. This includes time for the API to retrieve data from Polygon and Alpaca, the LLM inference time, the Python code execution time and all other processing

© The National High School Journal of Science 2026

Table 1 Query-to-code translation accuracy by query type.

Query Type	**Validation Count** ($N = 100$)	**Correctly Parsed**	**Success Rate**
Simple (e.g., "RSI over 70")	25	25	100%
Complex Multi-Factor (Tech + Indicator)	50	48	96%
Sentiment-Integrated (e.g., "Low P/E and bullish news")	25	23	92%
Overall	100	96	96%

Table 2 Execution latency summary.

Metric	**Value**
Mean Execution Latency	1.80 s
Standard Deviation	0.35 s
Median Latency	1.74 s
90th Percentile Latency	2.21 s
Minimum Latency	1.12 s
Maximum Latency	2.89 s

times. These values are shown in Table 2.

The average execution time is 1.80 seconds and the 90th percentile execution time is 2.21 seconds. This shows that the solution falls into the range of acceptable execution times for an interactive website (under 3 seconds). The standard deviation of 0.35 seconds shows that the execution time does not vary significantly between different requests. The variance in execution time is likely driven more by the variability in the execution time of the Polygon and Alpaca APIs rather than the variance in the LLM execution time. The worst performing request (with an execution time of 2.89 seconds) involved sentiment analysis and required news data to be collected and scored for a large number of candidate stocks. This is also the most compute intensive data-processing pipeline.

Usability and Screening Accuracy

For each of the 96 successfully processed queries, three quality checkers manually compared the output stock list and generated screening code to the stated and implied requirements in the natural language input query. A stock list that met every requirement according to all three quality checkers was deemed an accurate result. For the successful queries, 100% of the generated output stock lists were deemed accurate. This resulted in an accuracy rate of 100% for successful queries. The following are examples of a query and the resulting screening code that resulted in accurate output.

Example 1 - Simple query. Query: "Find tech stocks with RSI above 70." Screening Code: RSI calculated on closing prices from the Polygon API using a 14-day period. Filtered for IT sector stocks. Filtered further for RSI values above 70. 7 stocks in output list. All stocks in the output list were verified by all three quality checkers as matching both the RSI and sector filters.

Example 2 - Multi-factor complex query. Query: "Show me large cap stocks with a bullish cross in MACD and above average volume for the past 5 days." Screening Code: Filtered for market caps above $10 billion. Calculated MACD with default settings (12, 26, 9) and filtered for bullish crosses. Calculated volume as a ratio to a 20-day simple moving average with a threshold of 1.0. 12 stocks in output list. All stocks in the output list were verified by all three quality checkers as matching all three filters.

Example 3 - Sentiment-aware query. Query: "Identify energy stocks with a low P/E and positive news sentiment." Execution logic: Energy sector selected, low P/E selected (P/E < 15), sentiment determined by the average output of BERT sentiment classification over the 10 most recent Finnhub news items for each stock, with a minimum average sentiment score of > 0.6 applied. 4 stocks were returned and all were verified as meeting the requested criteria.

Stratoscan was compared to the stock screener in TradingView. This tool was chosen because of its popularity among retail traders and an example of how user selects parameters for a screen. Ten multi-factor queries were created by the same users who created the test queries and executed on both platforms. For TradingView users the filters needed to be identified and then each query condition was set using the slider and dropdown menu available in the tool and then the screen was run by the user. Stratoscan users inputted the same queries in natural language. The time to complete a screen was defined as the time from when a user started creating a screen until a valid list of stocks was first returned. For all ten queries and all five test users, Stratoscan was 78% faster at completing the screen than TradingView. Between 71% and 84% faster for each individual query was achieved. These improvements can be attributed to the not having to navigate through filters and not requiring the user to understand the meaning and the specific titles of the metrics they want to use.

Discussion

Summary of Key Results

Stratoscan was built to be a full-stack web application that can convert English-language questions about stocks into quantitative filters for screening stocks using a lightweight fine-tuned language model. In this validation study, Stratoscan was shown to successfully perform this task with high reliability. Using a fine-tuned version of the DeepSeek Coder 1.3B model, 96% of 100 new questions were successfully converted to stock filter code with a 95% confidence interval of 90.1% to 98.9%. This suggests that the 96% performance is unlikely to be due to either luck or the specific content of the validation questions. The system also performs with sufficient speed for an interactive application. The system had an average end-to-end execution time of 1.80 seconds. Three independent judges also evaluated the correctness of the generated results for successfully converted questions and found 100% accuracy with excellent inter-judge agreement (Cohen's Kappa $= 0.91$). This suggests that the generated code is not only syntactically correct but also accurately reflects the meaning of the original natural language question. Finally, user testing compared to TradingView also showed a 78% decrease in average time to complete a task, suggesting an actual reduction in the barrier to advanced stock screening for non-investors.

Implications and Importance

This research has a number of implications in both financial technology and natural language processing. Perhaps most importantly, this research shows that a 1.3B parameter language model can reach near-optimal performance on a specific and well-defined task (code generation for stock filtering) when fine-tuned on an appropriate synthetic dataset. This 1.3B model is several orders of magnitude smaller than cutting edge models such as BloombergGPT or GPT-4. This work can have a direct application to the FinTech community due to the high compute costs that arise when rolling out large in-house models at scale. However, we have shown that using a small open-source base language model, a well-curated domain training set and a modular system made up of useful NLP components can result in a solution that is efficient, fast, highly accurate and highly rollable on commodity cloud infrastructure using Stratoscan's system architecture.

This work also has potential applications to financial user interfaces. The 78% improvement in task completion in our usability study shows that current parameter selection user interfaces for financial screening tools place a high cognitive burden on users. Natural language interfaces show themselves to be a significantly higher quality interface for users with financial knowledge but little experience with screening tools. Instead of needing to convert their investment thesis into the parameter names, parameter definitions and parameter ranges of a visual interface, Stratoscan allows users to communicate their investment thesis in the users' native language. This new interface has the potential to allow quantitative investment to be accessible to a wider group of users beyond those users who are technically adept and currently receive the advantages of sophisticated screening tools.

Finally, this work makes a contribution to the research of hybrid financial NLP as we have successfully connected an LLM-based technical screening method to BERT-based sentiment analysis. Previous research into sentiment analysis in finance has typically focused on sentiment classification alone. Previous technical screening methods have also typically been disconnected from unstructured text. Using Stratoscan we have combined both pieces of financial NLP into a natural language interface that supports queries not currently available from any commercial screening tool.

Relation to Research Objectives

Each of the four main research objectives has been completed. A complete web application has been implemented and put into production. The DeepSeek Coder 1.3B model was fine-tuned and shown to accurately translate natural language into financial screening code. The sentiment analysis model using BERT has also been added to the system and has played a useful role in the system's ability to handle qualitative and technical queries. Performance metrics for translation accuracy, execution time and screening accuracy were shown to be high and this supports the research objective that lightweight domain adapted language models can act as a bridge between an individual's intent regarding investments and financial data.

The 4 parsing failures that were observed in the validation study also can be used to give an indication of the current system's limitations. These failures indicate that the current system performs best with certain types of queries and has some difficulty performing with queries that require multi timeframe data, temporal based conditionality, or multi timeframe fundamental analysis. These limitations are not widespread across the types of queries but are limited to this specific type of query. This also indicates that improving performance on these query types with an expanded training set will likely result in an improvement to the system.

Recommendations

From this research a few recommendations for future work have been made. A major expansion of the training set to include a large amount of multi timeframe queries with the ability to explicitly model the data retrieval across multiple candle lengths for conditions that need to be met across multiple timeframes would be beneficial. This would likely help

© The National High School Journal of Science 2026

to solve the most frequently occurring type of parsing failure observed during the validation study. Second, the system can be expanded to support foreign exchange and cryptocurrency as well as US equities. These assets also follow the same basic paradigm of the screening system but bring new challenges to data sourcing and the creation of indicators. Third, a system that automatically executes trades using a rule-based system could be added to automatically take actions based on the screening results. These actions would be translated into API calls to a broker and would require user approval before execution. This would move Stratoscan from a screening system to an end-to-end investment solution. Fourth, the validation approach used in this project could be expanded to use a frozen point in time archive of data as a ground truth. This would allow for the results of the screening system to be validated based on results rather than the live data used to create the results, helping to alleviate the circularity problem that occurs when screening results are validated on the same data used to act on those results.

Limitations

There are a number of limitations to this project. Perhaps the largest methodological limitation is the approach used for result validation. Since the screening results were validated using the same Alpaca and Polygon data source used to create the results, the validation only verifies that the code created properly applies its filtering conditions to the data provided during screening and does not independently verify that the natural language meaning is properly applied to a known ground truth. The manual validation of the semantic similarity between the query and code to an independent reviewer helps mitigate this but would benefit from a frozen dataset with known ground truth results for independent verification in future work.

A second source of potential issues is the use of a synthetically-created training set. All queries and responses in the training set were verified to be semantically and syntactically correct, but the variation of query types and phrasing in the synthetic training set do not completely capture the full variety of natural language that users might actually use in practice. The four parsing errors that were found in the validation tests indicate that some types of more complicated queries were underrepresented in the training set. There is also likely to be more modes of failure with a larger and more varied validation set. There are also implementation issues that affect the applicability of the latency results presented. The implementation is using a 2 virtual CPU core and 4 GB of RAM Replit virtual machine. Inference is running on CPU and no GPU acceleration is being used. This level of compute is enough to process the current model at less than two seconds, but would not be enough to scale to multiple concurrent users or to run a larger model version. Also, the use of external news and market data providers creates a reliability issue that is beyond the scope of the system itself. Data feed latency, provider outages, rate limits on the provider APIs, and other such issues are potential issues that were not systematically tested in this project but should be looked at in future implementations.

Lastly, the accuracy of the BERT-based sentiment analysis module will affect the accuracy of any query that uses sentiment analysis. The fine-tuned model for financial sentiment used in this project has been well tested for financial sentiment analysis but is not perfect. If the sentiment of a piece of news related to a specific stock was misclassified then this would result in an error in screening that would not be caught using the method for judging the accuracy of the results used in this project.

Closing Thoughts

We believe Stratoscan is more than a proof-of-concept for one specific use case. Rather, we hope to show that the gap between high level finance and the retail investor is not an inherent aspect of today's financial technology, but instead a design issue that can be addressed by leveraging today's advances in natural language processing. As domain specific small language models get better and AI based interfaces get cheaper to implement the analytic tools previously only available to quantitative analysts at large financial institutions will start to become available to the individual who wants to ask a question in natural language. Beyond this, we hope to show that AI used correctly and purposefully for specific domains can do more than automate existing processes; they can also change who is able to participate in those processes.

Acknowledgments

The author would like to thank Replit for providing the cloud development environment used throughout this project, and the open-source community behind the DeepSeek Coder 1.3B model, which served as the foundation for the AI engine. The author also thanks the five volunteer participants who contributed validation queries and usability testing time, and acknowledges the Financial PhraseBank dataset contributors whose work enabled the sentiment classification module.

References

1 I. Androutsopoulos, G. D. Ritchie, P. Thanisch. Natural language interfaces to databases — an introduction. *Natural Language Engineering*. Vol. *1*, pg. 29–81, 1995. https://doi.org/10.1017/S1351324900000005X.

2 J. Devlin, M. W. Chang, K. Lee, K. Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. *Proceedings of the 2019 Conference of the North American Chapter of the Association*

for Computational Linguistics. Vol. *1*, pg. 4171–4186, 2019. https://doi.org/10.18653/v1/N19-1423.

3 T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, ... D. Amodei. Language models are few-shot learners. *Advances in Neural Information Processing Systems*. Vol. *33*, pg. 1877–1901, 2020. https://doi.org/10.5555/3495724.3495883.

4 A. Quamar, V. Efthymiou, C. Lei, F. Özcan. Natural language interfaces to data. *arXiv preprint arXiv:2212.13074*. 2022. https://arxiv.org/abs/2212.13074.

5 M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, A. Ray, R. Puri, G. Krueger, M. Petrov, H. Khlaaf, G. Sastry, P. Mishkin, B. Chan, S. Gray, ... W. Zaremba. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*. 2021. https://arxiv.org/abs/2107.03374.

6 J. Austin, A. Odena, M. Nye, M. Bosma, H. Michalewski, D. Dohan, E. Jiang, C. Cai, M. Terry, Q. Le, C. Sutton. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*. 2021. https://arxiv.org/abs/2108.07732.

7 D. Fried, A. Aghajanyan, J. Lin, S. Wang, E. Wallace, F. Shi, R. Rubinstein, L. Zettlemoyer, N. Muennighoff, M. Lewis. InCoder: A generative model for code infilling and synthesis. *Proceedings of the International Conference on Learning Representations*. 2023. https://arxiv.org/abs/2204.05999.

8 E. Nijkamp, B. Pang, H. Hayashi, L. Tu, H. Wang, Y. Zhou, S. Savarese, C. Xiong. CodeGen: An open large language model for code with multi-turn program synthesis. *Proceedings of the International Conference on Learning Representations*. 2023. https://arxiv.org/abs/2203.13474.

9 D. Guo, Q. Zhu, D. Yang, Z. Xie, K. Dong, W. Zhang, G. Chen, X. Bi, Y. Wu, Y. K. Li, F. Luo, Y. Xiong, W. Liang. DeepSeek-Coder: When the large language model meets programming — the rise of code intelligence. *arXiv preprint arXiv:2401.14196*. 2024. https://arxiv.org/abs/2401.14196.

10 S. Lahiri, S. Fakhoury, A. Naik, G. Sakkas, S. Chakraborty, P. Choudhury, C. von Veh, M. Musuvathi, J. P. Inala, C. Wang, J. Gao. Interactive code generation via test-driven user-intent formalization. *arXiv preprint arXiv:2208.05950*. 2022. https://arxiv.org/abs/2208.05950.

11 Q. I. Sarhan, S. K. Jabr. A systematic survey on large language models for code generation. *Proceedings of the 4th International Conference on Emerging Smart Technologies and Applications (eSmarTA)*. 2024. https://doi.org/10.1109/eSmarTA62850.2024.10638040.

12 T. Loughran, B. McDonald. When is a liability not a liability? Textual analysis, dictionaries, and 10-Ks. *The Journal of Finance*. Vol. *66*, pg. 35–65, 2011. https://doi.org/10.1111/j.1540-6261.2010.01625.x.

13 P. Malo, A. Sinha, P. Korhonen, J. Wallenius, P. Takala. Good debt or bad debt: Detecting semantic orientations in economic texts. *Journal of the American Society for Information Science and Technology*. Vol. *65*, pg. 782–796, 2014. https://doi.org/10.1002/asi.23062.

14 D. Araci. FinBERT: Financial sentiment analysis with pre-trained language models. *arXiv preprint arXiv:1908.10063*. 2019. https://arxiv.org/abs/1908.10063.

15 Y. Yang, M. C. S. Uy, A. Huang. FinBERT: A pretrained language model for financial communications. *arXiv preprint arXiv:2006.08097*. 2020. https://arxiv.org/abs/2006.08097.

16 J. Bollen, H. Mao, X. Zeng. Twitter mood predicts the stock market. *Journal of Computational Science*. Vol. *2*, pg. 1–8, 2011. https://doi.org/10.1016/j.jocs.2010.12.007.

17 M. G. Sousa, K. Sakiyama, L. S. Rodrigues, P. H. de Moraes. BERT for stock market sentiment analysis. *Proceedings of the IEEE 32nd International Conference on Tools with Artificial Intelligence (ICTAI)*. pg. 1069–1076, 2020. https://doi.org/10.1109/ICTAI50040.2020.00162.

18 E. F. Fama, K. R. French. The cross-section of expected stock returns. *The Journal of Finance*. Vol. *47*, pg. 427–465, 1992. https://doi.org/10.1111/j.1540-6261.1992.tb04398.x.

19 R. Novy-Marx. The other side of value: The gross profitability premium. *Journal of Financial Economics*. Vol. *108*, pg. 1–28, 2013. https://doi.org/10.1016/j.jfineco.2013.01.003.

20 N. Jegadeesh, S. Titman. Returns to buying winners and selling losers: Implications for stock market efficiency. *The Journal of Finance*. Vol. *48*, pg. 65–91, 1993. https://doi.org/10.1111/j.1540-6261.1993.tb04702.x.

21 C. Belak, L. Mich, F. T. Seifried. Optimal investment for retail investors. *Mathematical Finance*. Vol. *32*, pg. 555–594, 2021. https://doi.org/10.1111/mafi.12332.

22 A. Lopez-Lira, Y. Tang. Can ChatGPT forecast stock price movements? Return predictability and large language models. *arXiv preprint arXiv:2304.07619*. 2023. https://arxiv.org/abs/2304.07619.

23 S. Wu, O. Irsoy, S. Lu, V. Dabravolski, M. Dredze, S. Gehrmann, P. Kambadur, D. Rosenberg, G. Mann. BloombergGPT: A large language model for finance. *arXiv preprint arXiv:2303.17564*. 2023. https://arxiv.org/abs/2303.17564.

24 Q. Xie, W. Han, X. Zhang, Y. Lai, M. Peng, A. Lopez-Lira, J. Huang. PIXIU: A large language model, instruction data and evaluation benchmark for finance. *arXiv preprint arXiv:2306.05443*. 2023. https://arxiv.org/abs/2306.05443.

25 W. X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Dong, Y. Du, C. Yang, Y. Chen, Z. Chen, J. Jiang, R. Ren, Y. Li, X. Tang, Z. Liu, ... J. R. Wen. A survey of large language models. *arXiv preprint arXiv:2303.18223*. 2023. https://arxiv.org/abs/2303.18223.

© The National High School Journal of Science 2026