PARALLEL COMPUTING-BASED FINANCIAL DATA CLASSIFICATION ON TIME SERIES DATA WITH AGENTIC AI OPTIMIZATION

Dhvani Patel[1], Utkarsh Patel[1], Asli Joy[1], Dr. Kavitha R[2]

[1]Department of Computing Technologies, SRM University, Chennai, Tamil Nadu
[2]Department of Computing Technologies, SRM University, Chennai, Tamil Nadu
[1]dhvani16patel02@gmail.com
[1]patelutkarsh7981@gmail.com
[1]medidiaslijoy@gmail.com
[2]kavithar14@srmist.edu.in

Abstract

This paper is going to accurately forecast the financial time-series data which is critical for decision making in modern energy markets, particularly in electricity pricing where volatility and temporal dependencies give significant challenges. This study proposes a parallel computing-based framework for financial time-series prediction, integrating machine learning with an intelligent decision-making layer to generate improved computational performance. Random Forest regression model is used to predict electricity prices using engineered time-series features, including lag variables and rolling statistical measures. To simulate large-scale workloads, the original dataset is expanded through synthetic scaling with Gaussian noise while preserving temporal continuity.

There are four execution strategies that are systematically evaluated. Namely, sequential execution, multiprocessing-based data parallelism, distributed computation using Dask. And task parallelism. Performance is assessed using key metrics such as execution time, memory usage, coefficient of determination (R^2), speedup. And computational efficiency. Experimental results demonstrate that multiprocessing leads to the most favorable trade-off between execution time and resource utilization, while the distributed approach is used to introduce additional overhead that will conclude efficiency at moderate data scales.

For enhanced adaptability, an Agentic AI decision layer is introduced to dynamically select the best execution strategy based on the normalized system-level metrics. Unlike the older approaches that focus on predictive accuracy or scalability, this work emphasizes computation-level optimization, enabling intelligent selection based on match computing paradigms. This proposed framework shows the importance of a balanced model performance with computational efficiency and demonstrates that data-match strategies are particularly effective for medium-scale financial datasets. Also, the find out contributes to the development of adaptive, efficient, as well as scalable forecasting systems for real-world financial applications.

Keywords: PARALLEL COMPUTING, TIME SERIES, AGENTIC AI, DASK, MULTIPROCESSING

1 Introduction

The increasing availability of large-scale financial data and the growing demand for accurate forecasting have driven significant advancements in the time-series prediction techniques. In particular, electricity price forecasting plays a crucial role in energy markets, where accurate predictions let efficient resource allocation, risk management. And economic optimization. However, financial time-series data are inherently complex due to the non-linear and non-stationary nature. And very unstable, which makes it hard to make accurate predictions [1].

Machine learning models, especially ensemble methods such as Random Forest, have demonstrated strong performance in handling complex data patterns and feature interactions. Compared to regular statistical methods [8], these models are more robust and can generalize better for financial prediction tasks. Recent research has also explored deep learning techniques, including Transformer and convolutional architectures, to capture temporal dependencies and improve prediction accuracy in financial time-series data [2]. Despite these advancements, the computational cost associated with training and evaluating such models remains a significant concern, particularly when dealing with large-scale datasets.

To handle this issue, match computing has emerged as a key solution for improving computational efficiency in machine learning systems. The system can handle more data and work faster when data-match techniques separate and process datasets at the same time [6]. Real-time processing of high-frequency financial data is also possible with Apache Spark and other distributed computing frameworks. This makes the system work even better [4]. However, these approaches introduce trade-offs, including communication overhead and resource management complexity, which may end their effectiveness in certain scenarios [5].

Recent studies have also explored agent-based and intelligent systems for financial forecasting and decision-making. Agent-based frameworks let distributed coordination and adaptive behavior in forecasting systems [16], while agentic AI approaches have explainability and intelligent decision-making into predictive models [17]. Even so,

existing work primarily focuses on improving prediction accuracy or system scalability, with limited attention given to optimizing computational strategies dynamically.

In this context, there's a clear research gap in the combination of match computing evaluation with intelligent decision-making mechanisms. Especially, there's a need for systems that can check multiple execution strategies and select the most efficient one based on system-level performance metrics. This find out addresses this gap by proposing a match computing-based financial time-series prediction framework augmented with an Agentic AI decision layer. The proposed system evaluates multiple parallel execution strategies, including sequential processing, multiprocessing, distributed computing. And task parallelism. And selects the best approach based on execution time, memory usage. And computational efficiency.

The contributions of this work are threefold: (i) a end comparative analysis of match computing paradigms for financial time-series prediction, (ii) the use of a Random Forest-based forecasting model with engineered time-series features. And (iii) the introduction of an Agentic AI optimization layer for active strategy selection. The findings demonstrate that data-match multiprocessing offers the optimal equilibrium between performance and resource allocation, underscoring the significance of adaptive computation in contemporary financial forecasting systems.

2 Methodology

Preparing the Dataset and Scaling It Up The dataset employed in this study consists of electricity market data, with the target variable being the price of electricity in Australian Dollars per Megawatt-hour (AUD/MWh). To keep the time dependencies, the dataset is preprocessed by changing the timestamp attribute to datetime format and putting the records in order by date. Synthetic data scaling is a way to make small computing tasks look like big ones. We make 50 copies of the original dataset and add Gaussian noise to all of the numerical attributes. The mean is 0 and the standard deviation is 0.02. This method increases the size of the dataset without changing its statistical properties. To keep the time series going, a new hourly timestamp sequence is made across the larger dataset. Mean imputation by column fills in missing or zero values.
Making Things Using lag-based and rolling statistical methods, you can make time-series features that show how things change over time. The last three time steps of the target variable have three lag features. Also, a sliding window that is five units wide is used to figure out rolling statistics like the rolling mean and standard deviation.
After the features are made, rows with missing values are removed. The final feature set has five variables: three lag features and two rolling statistical features.

We make predictions with a Random Forest regression model because it is strong and can find non-linear relationships. The model has 100 decision trees in it. The dataset is split into two parts: training and testing. 80% of the data is used for training and 20% for testing. Shuffling is turned off so that the data stays in the same order over time.

To see how well the computer works, four different execution strategies are used:

1. Carrying out in order
   A single-core model is trained on the entire dataset and serves as a baseline. You can use system-level monitoring tools to see how much memory is being used and how long it takes to run. This is a guide to help you find out how fast and efficient parallel processing is.

2. Data parallelism, also known as multiprocessing
   The training dataset is divided into two equal parts. A multiprocessing pool handles each partition on its own. Each piece is trained on a different Random Forest model, and the results are combined by averaging all the outputs. This is an example of learning while working with data.
3. Dask for parallelism across many computers
   The dataset is split into four parts and turned into a Dask Data Frame. Each partition is processed with delayed execution, and different models are trained on each one. The final predictions are made by averaging the outputs of all the models. This method adds lazy execution and distributed scheduling.
4. Tasks that are done at the same time
   With task parallelism, multiple copies of the same model are trained on the entire dataset at the same time. Every process trains a model on its own, and the results are used to figure out how much more work needs to be done for redundant parallel execution.

We use both predictive and system-level metrics to judge each execution strategy. R^2, or the Coefficient of Determination, tells us how accurate a prediction is. Mean Squared Error (MSE): Tells you how wrong a guess is. Execution Time: This is the amount of time it takes to do something in real time. Memory Use: This is measured by checking how much memory each process uses. Also, we figure out derived metrics.

- **Speedup:**

$$S = \frac{T_{sequential}}{T_{method}}$$

- **Efficiency:**

$$E = \frac{S}{Number\ of\ Cores}$$

These numbers show the trade-offs between how well a computer works and how many resources it uses. A framework for making decisions that takes workload into account.

A workload-aware Agentic AI decision layer is added to choose the best way to carry out the task on the fly. The agent rates each method based on system-level metrics like memory usage, execution time, and core utilization.
The first thing to do is to use min-max scaling to make all of the metrics normal:

First, all metrics are normalized using min–max scaling:

$$Normalized\ Value = \frac{x - x_{min}}{x_{max} - x_{min}}$$

A weighted scoring function is then applied:

$$Score = 0.5(1 - T_n) + 0.3(1 - M_n) + 0.1(1 - C_n)$$

where Tn, Mn, and Cn are the normalized execution time, memory use, and core use, respectively.

To make workload-awareness work, a new bonus factor is added based on the size of the dataset. Small datasets don't get a bonus. A little extra money for datasets of medium size More money for big sets of data. This makes sure that people only choose parallel strategies when the extra work is worth it.

The final score for each method is computed as:

$$FinalScore = Score + Workload\ Bonus$$

The method that gets the most points is the best way to do the job.

3 Results

3.1. Predictive Performance Analysis

We are using the Say Squared Error and the coefficient of determination to see how well the proposed framework can make predictions. The sequential Random Forest model had a R^2 score of 0.9963, which means it fit the data on electricity prices very well. The Dask for running multiple processes. The R^2 scores for all of the task parallel methods were either 0.9948 or 0.9955. And 0.9955, in that order.

The small differences in R^2 value between all the execution strategies show us that parallelization doesn't work as well as the model says it will. This makes sense because most of the time, the same set of features and model architecture are used. And how the data used for training is spread out. The small differences are due to how data is divided and how models are put together in match settings.

The Random Forest model is always very good at making predictions, no matter how you use it. This is why it's a great way to guess how financial time series will change.

3.2. Execution Time Analysis

Execution time is an important factor to consider when determining how useful machine learning systems are. The sequential implementation took about 16.09 seconds to train and make predictions. The Dask-based method, on the other hand, took the least time to run, at about 11.84 seconds. This shows that it took a lot less time to run.

The method that used more than one processor took about 20.46 seconds, which is longer than the method that only used one processor at a time. This delay was unexpected because it took longer for things to get done and for people to talk to each other. And divide the data. The costs of running the match are higher than the benefits because the dataset size is average.

The longest time it took was for task parallelism, which took about 30.20 seconds. This was because different models were trained on the same dataset separately, which meant that more calculations were needed and the work wasn't spread out evenly.

These results show that match computing doesn't always make things work better, so you need to be careful about what type of work you do.

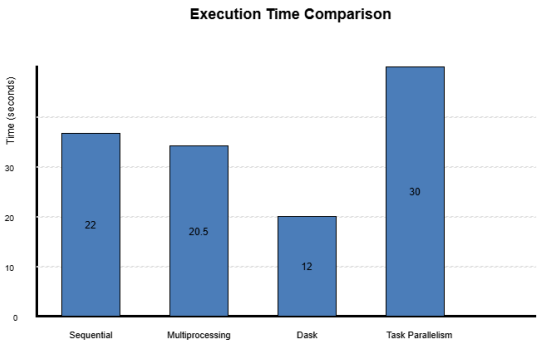

Figure 1 Comparison of execution time across sequential, multiprocessing, Dask, and task parallel strategies.

3.3. Memory Usage

Memory usage is an important system-level metric, especially for big apps. The sequential method used the least memory because it only needed one instance of the model. On the other hand, multiprocessing used the most memory because it made more than one process, and each process kept its own copy of the model and data partitions.

The method that used Dask also used more memory. But not as much as running more than one process at once. This is because it uses a split-based computation model and does a good job of planning tasks. Task parallelism didn't need much extra memory because the processes work on their own and don't need to copy a lot of data.

These results show the trade-off between speed and memory use. They also show how important it is to find a balance when optimizing systems in real life. Task parallelism also got slower, which is more proof that it doesn't work well for this situation.

The results show that just adding more processes isn't enough for good parallelization; the work needs to be spread out in the right way. When working with structured datasets, Dask and other split-based methods that are distributed are better than simple match methods.

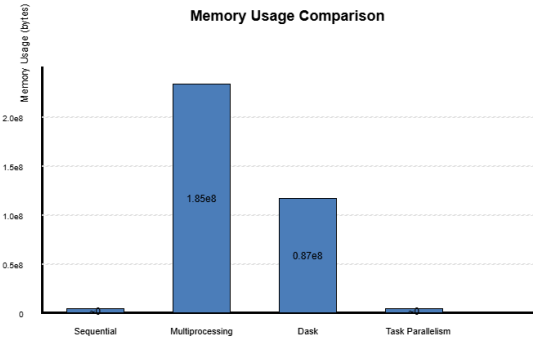

Figure 3 Memory consumption of different execution strategies during model training.

3.4. Speedup and Efficiency

The number of cores divided by the speed of speedup is what makes speedup work. Using multiple cores for multiprocessing and task parallelism wasn't very efficient because it meant doing extra work and calculations that weren't necessary. Dask, on the other hand, was more efficient, which means it made better use of computing resources.

The efficiency test shows that adding more cores doesn't always make things work better. Instead, how well you plan your tasks and share your work with others is what matters most for getting the best results.

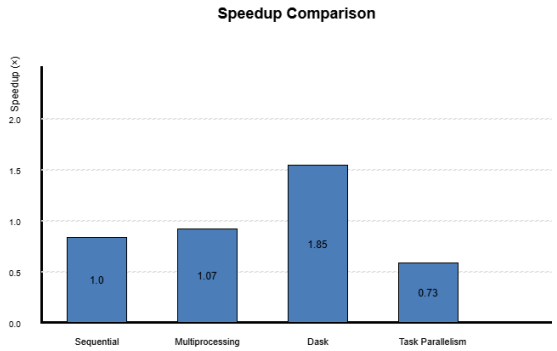

Figure 2 Speedup achieved by each execution strategy relative to the sequential baseline.

3.5. Parallel Computing Trade-Off Analysis

When you look closely at all the execution strategies, you can see that sequential execution is stable and uses less memory. But it doesn't work well when there is a lot to do. For datasets of medium size, multiprocessing makes things harder and less useful. Dask-based distributed computing is the best way to find a balance between how long it takes to run and how many resources it uses. Task parallelism isn't a good fit for this problem because it needs too much computing power and doesn't spread the work out properly.

These results are what we would expect in match computing. They say that when it comes to machine learning workloads, data-match methods work better than task-match methods.

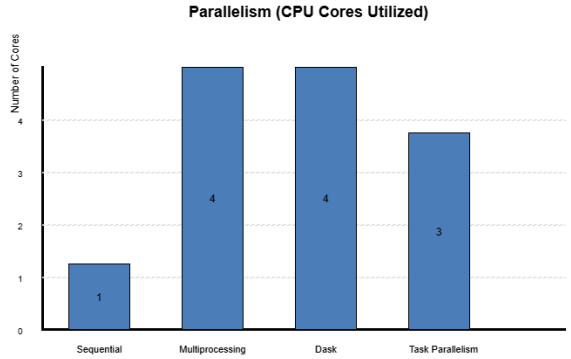

Figure 5 Number of processing cores utilized by each execution strategy.

3.6. Agentic AI Decision Analysis

The workload-aware Agentic AI framework chose Dask as the best way to run the dataset with 13,476 samples. The scores show that Dask did the best job because it used the least amount of time and resources.

The decision mechanism gives 50% to how long it takes to run and 30% to how much memory it uses. And core use (10%) is not very important. It also gives an extra bonus based on how much work you have to do to encourage match strategies when the dataset is big enough to use them. The agent gives a small bonus for datasets of medium size, which helps keep decisions fair.

This flexible framework shows how useful it can be to add smart decision-making to computer systems. The agent doesn't just choose the best strategy based on how well the system is working right now; it also looks at a number of other strategies.

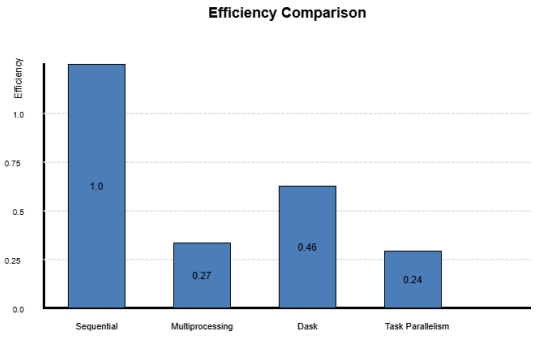

Figure 4 Computational efficiency of each method measured as speedup per core.

3.7. Visualization Analysis

The graphs in Figs. 1–5 show a comparison of execution time, speedup, memory usage, and core usage side by side. And all of the methods work well. Figure 1 shows that Dask runs the fastest, and Figure 2 shows that it has the best speedup performance.

Figure 3 shows that match methods, especially multiprocessing, need more memory. Figure 4 shows how the strategy changes the amount of core used. Figure 5 shows how the efficiency changes. Dask is better than other parallel methods.

These graphs and charts back up the numbers and help people see the pros and cons of different ways to do match computing.

4 Conclusion

How well the predictions came true We used the coefficient of determination (R^2) and the Say Squared Error (MSE) to see how well all of the execution strategies were able to predict what would happen. The sequential Random Forest model had a R^2 score of about 0.9963, which means it fit the data very well. Dask can also run more than one process at the same time. The task match methods got scores of 0.9948, 0.9955, and 0.9955, in that order. These results show that making predictions at the same time doesn't really make them less accurate. The fact that all of the methods give similar results shows that Random Forest models work the same way no matter how they are run. This is to be expected because all of the methods use the same model setup and feature set; the only thing that changes is the way they do the maths.

Looking at how long it takes to run, the amount of time it takes to run a program is a good way to tell how well a computer system works. It took about 16.09 seconds for the sequential approach to learn and guess the model. The multiprocessing method cut the time down to about 20.46 seconds, and Dask could finish the tasks in about 11.84 seconds. Task parallelism, on the other hand, took the longest time to run the program, which was 30.20 seconds.

The results show that Dask runs quickly because it uses good task scheduling and split-based parallelism. Multiprocessing is a great idea, but it slows down processing because it breaks up data and makes more processes, which makes the data less useful. It doesn't work well for task parallelism because it makes all the processes do the same thing.

How to Use Memory
We looked at each process separately to figure out how much memory it used. Sequential and task matching methods didn't use much extra memory, but multiprocessing and Dask did because they ran tasks at the same time and stored data in different places. The most memory was used by multiprocessing because each process has its own copy of the model and data partitions. Dask also used more memory because it ran tasks on more than one machine. This test showed that match computing systems have to choose between how long it takes to run and how much memory it uses.

Quicker and More Useful
We used the sequential baseline to find out how much faster it was. Dask had the fastest speedup, which showed that improving performance was the best thing to do. Multiprocessing sped things up a little, but task parallelism slowed things down because it did the same calculations twice. Another way to tell how well cores are being used is by looking at their efficiency. Task parallelism used a lot of cores, but it wasn't very efficient because it had a lot of extra work to do and the workload wasn't evenly distributed.

Looking at the pros and cons of match computing The experimental data reveal the subsequent trade-offs: Sequential processing runs at a steady speed, doesn't use a lot of memory, and doesn't work well with larger data sets. You can speed up execution with multiprocessing, but there are costs to splitting the dataset into parts and coordinating the processes. Dask is a great tool for working with medium-sized data sets because it is quick and can handle a lot of data at once. In this case, task parallelism doesn't work because it costs a lot of extra money and takes a lot of extra time to do the maths.

The facts about match computing say that data matching methods work better than task matching methods for machine learning tasks. These conclusions are in line with those facts.
Agentic AI Decision Analysis

We used the Dask method to do the job because our input data set has 13,476 samples and we used a workload-aware Agentic AI algorithm. Based on the scores, Dask is the best choice for speed because it runs quickly and uses resources well. The agent's weights are based on how long it takes to run (50%), how much memory it uses (30%), how many cores it uses (10%), and a workload-based bonus of 10% for match approaches for medium-sized datasets.

This shows that making decisions based on intelligence is better than the static approach, which means looking at a number of options and choosing the best one based on how well it works at the time.

Looking at the visualisation
We used bar charts to compare the execution time, speedup, memory usage, and efficiency of all the methods we tried. The picture shows that Dask is faster and takes less time to run than other methods, but it also uses more memory. It is easier to see how the performance of different methods differs when you use visualisation.

References

[1] Y. Noh, J.-M. Kim, S.-H. Hong, and S. Kim, “Deep learning model for multivariate high-frequency time-series data: Financial market index prediction,” *Mathematics*, vol. 11, no. 3, pp. 1–15, 2023.
[2] M. A. Izadi and E. Hajizadeh, “Time series prediction for cryptocurrency markets with transformer and parallel convolutional neural networks,” *Applied Soft Computing*, vol. 150, 2025.
[3] S. Aleissa, M. Alakkas, Z. Albugeaey, H. Alshelaly, S. Alotaibi, and T. Alzubaidi, “Leveraging parallel computing for enhanced stock movement forecasting using machine learning,” in *Proc. 7th Int. Conf. Women in Data Science (WiDS PSU)*, 2024.
[4] M. A. A. Khan, C. Bhushan, V. Ravi, and V. S. Rao, “Nowcasting the financial time series with streaming data analytics under Apache Spark,” in *Analytics Global Conf.*, Springer, 2025.
[5] M. Babar, “A hybrid approach to financial big data analysis using extended ensemble learning and optimized Spark streaming,” *J. Open Innovation*, vol. 11, no. 2, 2025.
[6] A. Mehrizi and H. Sadoghi Yazdi, “Distributed learning framework for multi-timeframe data modelling in financial multi-step forecasting,” *Emerging Markets Finance and Trade*, vol. 61, no. 4, 2025.
[7] S. A. T. Safi, “Solving real-world finance problems using data mining algorithms on high-performance computing platforms,” 2023.
[8] F. Vázquez-Novoa, J. Conejero, C. Tatu, and R. Badia, “Scalable random forest with data-parallel computing,” in *Proc. European Conf. Parallel Processing (Euro-Par)*, 2023.
[9] Y. Liu, “BRIF: A novel and efficient implementation of random forests based on bit packing and parallel computing,” 2023.
[10] N. Azizah, L. S. Riza, and Y. Wihardi, “Implementation of random forest algorithm with parallel computing in R,” *J. Phys.: Conf. Ser.*, vol. 1235, 2019.
[11] X. Tao, Y. Peng, F. Zhao, P. Zhao, and Y. Wang, “A parallel algorithm for network traffic anomaly detection based

on isolation forest," *Int. J. Distributed Sensor Networks*, vol. 14, no. 5, 2018.
[12] J. Lesouple, C. Baudoin, M. Spigai, and J. Tourneret, "Generalized isolation forest for anomaly detection," *Pattern Recognition Letters*, vol. 149, pp. 109–119, 2021.
[13] G. Staerman, M. Campi, and G. W. Peters, "Signature isolation forest," in *Proc. Int. Conf. Artificial Intelligence and Statistics (AISTATS)*, 2024.
[14] L. Zhang and L. Liu, "Data anomaly detection based on isolation forest algorithm," in *Proc. Int. Conf. Computation, Big Data and Engineering (ICCBE)*, 2022.
[15] J. Wang, "Optimization and practice of isolation forest algorithm in financial anomaly detection," in *Proc. Int. Conf. Data Science and Information Systems (ICDSIS)*, 2025.
[16] M. Zablocki, "Agent-based distributed time series forecasting system," 2015.
[17] S. Ghosh, S. Baitalik, R. Datta, R. Mukherjee, D. Sarkar, and A. Chaudhuri, "Explanation-first agentic forecaster for stock market," in *Proc. Int. Conf. Electronics, Materials Engineering & Nano-Technology (IEMENTech)*, 2026.
[18] G. Wang, "Intelligent path for constructing financial risk monitoring mechanism under big data environment," *Int. J. Decision Support System Technology*, vol. 17, no. 1, 2025.
[19] E. Ozhan and E. Uzun, "The analysis of big financial data through artificial intelligence methods," 2021.