GPT-Guided Monte Carlo Tree Search for Symbolic Regression in Financial Fraud Detection

Prashank Kadam
prashank.kadam@vesta.io
Vesta Corporation
Lake Oswego, Oregon, USA

Abstract

With the increasing number of financial services available online, the rate of financial fraud has also been increasing. The traffic and transaction rates on the internet have increased considerably, leading to a need for fast decision-making. Financial institutions also have stringent regulations that often require transparency and explainability of the decision-making process. However, most state-of-the-art algorithms currently used in the industry are highly parameterized black-box models that rely on complex computations to generate a score. These algorithms are inherently slow and lack the explainability and speed of traditional rule-based learners. This work introduces SR-MCTS (Symbolic Regression MCTS), which utilizes a foundational GPT model to guide the MCTS, significantly enhancing its convergence speed and the quality of the generated expressions which are further extracted to rules. Our experiments show that SR-MCTS can detect fraud more efficiently than widely used methods in the industry while providing substantial insights into the decision-making process.

1 Introduction

Traditionally, financial fraud detection relied on rules constructed by subject matter experts to approve or deny transactions [15]. While these methods were fast and explainable, they struggled to adapt to evolving fraud patterns. Later, machine learning techniques like Logistic Regression [14], Support Vector Machines [13], Random Forest [12], and XGBoost [11] offered better adaptability but made decision-making less interpretable. Tree-based learners employed algorithms like SHAP [17] for insights, but these were computationally expensive and challenging to interpret as rules. Advanced Graph Neural Network models like Graph Attention Networks [9], Care-GNN [8], and Semi-GNN [10] improved decision-making but remained black-box and costly. In the recent past, Generative Pre-trained Transformers (GPT) [21] have shown path-breaking success in different domains of machine learning. Despite of its success, GPT suffers from issues like hallucinations [22], limiting their use in critical decision-making.

For real-time applications, fraud detection methods must be fast, interpretable, and accurate. Rule-based methods excel in speed and interpretability. This work introduces SR-MCTS, an algorithm that enhances rule-based methods to match state-of-the-art accuracy. We fine-tune a large language model, Symbolic-GPT [18], with financial data and use it to guide Monte Carlo Tree Search (MCTS) [16]. SR-MCTS generates mathematical expressions combining dataset features, operators, and constants to create rule sets for fraud detection. It also minimizes the effect of hallucinations due to limited guidance to the MCTS from Symbolic-GPT.

We evaluate SR-MCTS on our proprietary dataset, showing it outperforms widely used industry methods.

2 Method

In this section, we define the Markov Decision Process (MDP) for SR-MCTS, describe Symbolic GPT's role in guiding the search, and outline the reinforcement learning-style approach for evaluating and refining expressions. Finally, we explain how rule sets are extracted from the generated expressions.

2.1 MDP Formulation

The symbolic regression problem is modeled as an MDP where:

The **state space** S consists of all valid combinations of features and constants which form the operands. Operators consist of unary ($\{\sin, \cos, \log, \exp\}$) and binary ($\{+, -, \times, \div\}$) mathematical operations that can be combined with operands to form expressions. The **action space** $\mathcal{A}_s \subset \mathcal{A}$ for a state $s \in S$ is a conditional set of operators and operands, determined by the current state (e.g., operand followed by operator). The **transition function** $\mathcal{T}(s, a)$ defines the next state s' given s and action a, ensuring mathematically valid expressions. The **reward function** $R(s, a)$ is inversely proportional to the loss $\mathcal{L}(y, \hat{y})$ of the generated expression, with y as the target and $\hat{y}$ as the prediction.

The goal is to find a sequence of states and actions that maximizes cumulative reward, minimizing the loss function in a reinforcement learning style.

2.2 PUCT and Symbolic GPT Guidance

SR-MCTS uses the PUCT (Predictor + Upper Confidence bounds for Trees) [20] strategy to search the expression space:

$$\text{PUCT}(s, a) = Q(s, a) + c \cdot \sqrt{\frac{\log N(s)}{N(s, a)}} \cdot \text{P}_{\text{GPT}}(s, a), \qquad (1)$$

where $Q(s, a)$ is the expected reward (negative loss) for action a in state s, based on previous simulations. $N(s)$ is the number of visits to state s, $N(s, a)$ is the number of times action a was selected in s, and $\text{P}_{\text{GPT}}(s, a)$ is the probability of selecting a in s as predicted by Symbolic GPT. The constant c balances exploration and exploitation.

2.3 Evaluation and Loss Calculation

After generating trajectories (expressions) using SR-MCTS, they are evaluated based on their predictive performance relative to the target variable. For a given expression $\hat{y}$ generated from trajectory T, the binary cross-entropy loss is:

$$\mathcal{L}_{\text{BCE}}(y, \hat{y}) = -\left[y\log(\hat{y}) + (1-y)\log(1-\hat{y})\right] \quad (2)$$

where y is the true value and $\hat{y}$ is the predicted value.

The top k trajectories with the lowest loss are selected, and rewards are assigned as:

$$R(T) = \frac{1}{\mathcal{L}(y, \hat{y}) + \epsilon}, \quad (3)$$

where ϵ avoids division by zero.

2.4 Symbolic GPT Model and Retraining

Symbolic GPT, a generative model trained on symbolic regression datasets, is fine-tuned on financial transaction data to guide MCTS. The input to Symbolic GPT is the current state s (a partial expression), and the output is a probability distribution over the next possible actions (operands or operators), effectively serving as the policy $\pi(s)$ for MCTS. This policy guides action selection, enhancing the search process and expression quality.

To further refine Symbolic GPT, the top k trajectories from MCTS are used to retrain the model. The loss function for retraining is cross-entropy with L2 regularization:

$$\mathcal{L}_{\text{CE}} = -\sum_{i=1}^{m} y_i \log(\hat{y}_i) + \lambda \sum_j \theta_j^2, \quad (4)$$

where y_i is the true distribution (one-hot encoded) of the next action, $\hat{y}_i$ is the predicted distribution from Symbolic GPT for the i-th action, θ_j are the model weights, and λ controls the L2 penalty.

2.5 Extracting Rules

After generating the top k expressions using SR-MCTS, we create rules by solving the system of linear equations formed by equating these expressions. By finding the solutions to these equations, we derive interpretable rules that can be used to evaluate transactions as fraudulent or non-fraudulent based on the features present in the dataset.

Algorithm 1 SR-MCTS (Symbolic Regression using MCTS)

Require: Pre-trained Symbolic GPT model, Transaction dataset $\mathcal{T}$

Ensure: Optimized set of symbolic regression expressions

1: Initialize an empty set of expressions $\mathcal{T}_{all} \leftarrow \emptyset$
2: **for** each transaction $t \in \mathcal{T}$ **do**
3:     Initialize an empty set of expressions $\mathcal{T}_t \leftarrow \emptyset$
4:     **for** each iteration i **do**
5:         Generate an expression T_i
6:         Add the expression T_i to $\mathcal{T}_t \leftarrow \mathcal{T}_t \cup \{T_i\}$
7:     **end for**
8:     Evaluate each expression $T_i \in \mathcal{T}_t$ and calculate the loss
9:     Select the top k expressions $\mathcal{T}_{t,\text{top}}$
10:    Update reward $R(T_i)$
11:    Retrain Symbolic GPT with $\mathcal{T}_{t,\text{top}}$
12:    Add the expression T_t to $\mathcal{T}_{all} \leftarrow \mathcal{T}_{all} \cup \{T_t\}$
13: **end for**
14: Repeat the process until convergence
15: Extract rule sets out of the generated expressions

---

3 Experiments

Our dataset, the Proprietary Financial Fraud Dataset (PFFD), sourced from our organization's e-commerce clients, contains hashed customer order details (e.g., Name, Address, Email, Phone, Device, Payment) to ensure privacy. It covers diverse fraud scenarios, demonstrating the robustness of our approach. Our target label, Fraud Score (fs), ranges from 0 to 100, indicating the likelihood of fraud. The dataset includes approximately 1.25 million transactions, with 13,454 fraudulent cases, making up 1.07% of the data.

In the experiments, we set k to 0.2, selecting the top 20% of expressions from each training iteration for fine-tuning Symbolic-GPT. The maximum expression length is 40, with the MDP terminating at the next operand. All categorical variables are one-hot encoded.

We benchmarked SR-MCTS against industry-standard algorithms, including Logistic Regression [2], XGBoost [3], Random Forest [4], LSTM [5], GCN [6], and GAT [7], training and testing each model on the same datasets.

Results show SR-MCTS surpasses these techniques, while also providing interpretable decision-making.

Table 1: Fraud detection evaluation

Algorithm	Recall	AUC
SVM	0.536	0.507
Random Forrest	0.629	0.574
XGBoost	0.678	0.634
LSTM	0.621	0.587
GCN	0.725	0.711
GAT	0.784	0.765
SR-MCTS	**0.812**	**0.797**

4 Conclusion

The SR-MCTS approach significantly improves the speed and interpretability of financial fraud detection. Guided by Symbolic GPT, our method achieves faster convergence and generates high-quality expressions that effectively detect fraud. When combined with existing techniques, SR-MCTS enhances performance while maintaining full transparency in decision-making. Future work will refine the model, explore other domains, and experiment with longer expressions, more complex operators, and dynamic detection of k and the terminal state.

References

[1] Hagerup, T., Mehlhorn, K., and Munro, J. I. (1993). Maintaining discrete probability distributions optimally. In *Proceedings of the 20th International Colloquium on Automata, Languages and Programming*, volume 700 of Lecture Notes in Computer Science, pages 253–264. Springer-Verlag, Berlin.

[2] Hosmer, D. W., Lemeshow, S., and Sturdivant, R. X. (2013). *Applied Logistic Regression*. John Wiley & Sons.

[3] Chen, T., and Guestrin, C. (2016). XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794. ACM.

[4] Breiman, L. (2001). Random forests. *Machine Learning*, 45(1), 5–32.

[5] Hochreiter, S., and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8), 1735–1780.

[6] Kipf, T. N., and Welling, M. (2016). Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

[7] Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. (2017). Graph attention networks. *arXiv preprint arXiv:1710.10903*.

[8] Dou, Y., Liu, Z., Sun, L., Deng, Y., Peng, H., and Yu, P. S. (2020). Enhancing graph neural network-based fraud detectors against camouflaged fraudsters. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*.

[9] Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. (2018). Graph attention networks. *International Conference on Learning Representations (ICLR)*.

[10] Wang, D., Qi, Y., Lin, J., Cui, P., Jia, Q., Wang, Z., Fang, Y., Yu, Q., Zhou, J., and Yang, S. (2019). A semi-supervised graph attentive network for financial fraud detection. In *2019 IEEE International Conference on Data Mining (ICDM)*, pages 598–607. IEEE.

[11] Hájek, P., Abedin, M., and Sivarajah, U. (2022). Fraud detection in mobile payment systems using an XGBoost-based framework. *Information Systems Frontiers*, 25, 1–19.

[12] Xuan, S., Liu, G., Li, Z., Zheng, L., Wang, S., and Jiang, C. (2018). Random forest for credit card fraud detection. In *2018 IEEE 15th International Conference on Networking, Sensing and Control (ICNSC)*, pages 1–6.

[13] Gyamfi, N. K., and Abdulai, J.-D. (2018). Bank fraud detection using support vector machine. In *2018 IEEE 9th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, pages 37–41.

[14] Ravisankar, P., Ravi, V., Rao, G. R., and Bose, I. (2011). Detection of financial statement fraud and feature selection using data mining techniques. *Decision Support Systems*, 50(2), 491–500.

[15] West, J., Bhattacharya, M., and Islam, R. (2014). Intelligent financial fraud detection practices: An investigation. In *International Conference on Security and Privacy in Communication Networks - 10th International ICST Conference, SecureComm 2014, Beijing, China, September 24-26, 2014, Revised Selected Papers, Part II*, volume 153 of Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, pages 186–203. Springer.

[16] Coulom, R. (2006). Efficient selectivity and backup operators in Monte-Carlo tree search. In *Proceedings of the 5th International Conference on Computers and Games (CG'06)*, volume 4630 of Lecture Notes in Computer Science, pages 72–83. Springer.

[17] Shapley, L. S. (1953). A value for n-person games. *Contributions to the Theory of Games (AM-28), Volume II*, 28, 307–317.

[18] Valipour, M., You, B., Panju, M., and Ghodsi, A. (2021). SymbolicGPT: A generative transformer model for symbolic regression. *CoRR*, abs/2106.14131.

[19] Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press.

[20] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587), 484–489.

[21] Radford, A., Narasimhan, K., Salimans, T., and Sutskever, I. (2018). Improving language understanding by generative pre-training. *OpenAI Blog*.

[22] Maynez, J., Narayan, S., Bohnet, B., and McDonald, R. (2020). On the faithfulness and factuality of abstractive summarization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1906–1919. Association for Computational Linguistics.

[23] Coulom, R. (2006). Efficient selectivity and backup operators in Monte-Carlo tree search. In *Proceedings of the 5th International Conference on Computers and Games (CG'06)*, volume 4630 of Lecture Notes in Computer Science, pages 72–83. Springer.

[24] Kocsis, L., and Szepesvári, C. (2006). Bandit based Monte-Carlo planning. In *Proceedings of the 17th European Conference on Machine Learning*, pages 282–293. Springer.

[25] Anthony, T., Tian, Z., and Barber, D. (2017). Thinking fast and slow with deep learning and tree search. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 5366–5376.

[26] Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., and Hassabis, D. (2018). A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419), 1140–1144.

A   Neural Networks with MCTS

Monte Carlo Tree Search (MCTS) [23] [24] is widely used for solving combinatorial problems, with its effectiveness enhanced by integrating deep neural networks for value estimation. This approach, known as Neural MCTS, is exemplified in methods like Expert Iteration [25] and AlphaZero [26]. AlphaZero employs a neural network to approximate both policy and value functions. During learning, multiple self-play rounds are conducted, with MCTS simulations estimating a policy at each state. The selected policy guides the next move, and outcomes are propagated back through the states, with trajectories stored in a replay buffer to train the network. In each self-play, MCTS runs a fixed number of simulations to generate an empirical policy. These simulations involve four phases:

(1) **SELECT:** The algorithm begins by selecting a path from the root to a leaf (either a terminal or unvisited state) using an PUCT algorithm [20]. Starting at the root s_0, a sequence of states $\{s_0, s_1, ..., s_l\}$ is determined as follows:

$$a_i = \mathrm{argmax}_a \left[Q(s_i, a) + c\pi_\theta(s_i, a) \sqrt{\frac{N(s_i)}{N(s_i, a)}} \right] \quad (5)$$

$$s_{i+1} = \mathrm{move}(s_i, a_i)$$

Here, Q represents the value of the state-action pair, N is the number of visits to the state-action pair, and π_θ is the policy learned by the network.
MCTS optimizes the output policy to maximize the action value while minimizing deviations from the policy network, assuming accurate value estimates.

(2) **EXPAND:** If the selection phase ends at a previously unvisited state s_l, it is fully expanded and marked as visited. All child nodes are considered leaf nodes in the next iteration.

(3) **ROLL-OUT:** Each child of the expanded leaf node conducts a roll-out, where the network estimates the trajectory's outcome. This value is backpropagated to the previous states.

(4) **BACKUP:** The statistics for each node in the selected states $\{s_0, s_1, ..., s_l\}$ are updated.
For the sequence $\{(s_0, a_0), (s_1, a_1), ...(s_{l-1}, a_{l-1}), (s_l, \_)\}$, the value $V_\theta(s_i)$ for child s_i updates the Q-value iteratively as follows:

$$N(s_t, a_t) \leftarrow N(s_t, a_t) + 1$$

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \frac{V_\theta(s_r) - Q(s_t, a_t)}{N(s_t, a_t)} \quad (6)$$

This process is repeated for all roll-out outcomes from the previous phase.

After the required number of iterations, the algorithm returns the empirical policy $\hat{\pi}(s)$ for the current state s. The next action is sampled from $\hat{\pi}(s)$, and the process continues.

B   Feature Descriptions

In this work, we utilize both base and derived features to construct symbolic expressions for financial fraud detection. The features can be categorized as follows:

B.1   Base Features

The base features consist of direct transactional attributes that are commonly available during online purchases. Some of these features are:

- **Shipping Features:**

– **Shipping Email ($s_{\mathbf{email}}$):** The email address associated with the shipping address of the transaction.
– **Shipping Phone ($s_{\mathbf{phone}}$):** The phone number associated with the shipping address.
– **Shipping Address ($s_{\mathbf{address}}$):** The physical address provided for shipping purposes.

- **Billing Features:**
  – **Billing Email ($b_{\mathbf{email}}$):** The email address associated with the billing address of the transaction.
  – **Billing Phone ($b_{\mathbf{phone}}$):** The phone number associated with the billing address.
  – **Billing Address ($b_{\mathbf{address}}$):** The physical address provided for billing purposes.
- **Other Transactional Features:**
  – **Card Number:** The credit or debit card number used for the transaction.
  – **Bin Number:** The bank identification number (BIN) that identifies the institution issuing the card.
  – **Device ID:** The unique identifier associated with the device used to complete the transaction.
  – **IP Address:** The IP address from which the transaction is conducted.

Traditional Velocity Features

The traditional velocity features capture temporal patterns in transactional activity by calculating the count and sum of base features over different time windows. These windows include: 15 minutes, 30 minutes, 1 hour, 4 hours, 12 hours, 1 day, 7 days, 14 days, 30 days, 60 days, and 90 days. The velocity features help detect unusual activity patterns in a short or long time frame. For example:

- **Count of Shipping Emails ($\mathbf{count}_{s_{\mathbf{email}}}$):** The number of times the shipping email appears in transactions during a specified time window.
- **Sum of Transaction Amounts for a Card Number ($\mathbf{sum}_{\mathbf{bin}}$):** The total sum of transaction amounts associated with the same card number over a specified time window.

Relational Velocity Features

Relational velocity features represent aggregations of one base feature relative to another within a specified time window. This helps capture interactions between different transaction attributes that may indicate fraudulent behavior. For example:

- **Shipping Email vs Billing Address ($\mathbf{rv}(s_{\mathbf{email}}, b_{\mathbf{address}})$):** The number of unique shipping emails associated with a particular billing address in a given time window.
- **Device ID vs Card Number ($\mathbf{rv}(\mathbf{device\_id}, \mathbf{card\_number})$):** The number of times the same device ID is used with different card numbers in a specific time window.

These base and velocity features form the core input for constructing the symbolic expressions generated through the SR-MCTS process. By considering both temporal patterns and relationships between features, the model can detect fraudulent activities more effectively and interpretably.

C Generated Expressions

Following is an example of the expression generated by Symbolic MCTS:

$$
\begin{aligned}
\text{Fraud Score} = {}& 0.31 \log\left(1 + \text{count}_{s_{\text{email}}}^{30\ \text{day}}\right) \\
& + 0.54 \log\left(1 + \text{sum}_{b_{\text{address}}}^{15\ \text{min}} + \text{count}_{\text{card\_number}}^{1\ \text{hr}}\right) \\
& + 1.21 \sin\left(\text{sum}_{\text{bin\_number}}^{1\ \text{day}} \cdot \text{count}_{\text{device\_id}}^{4\ \text{hr}}\right) \\
& + \exp\left(-0.77\left(\text{sum}_{s_{\text{phone}}}^{7\ \text{day}} + \text{count}_{\text{ip}}^{90\ \text{day}}\right)\right) \\
& + 0.93\left(\text{count}_{b_{\text{email}}}^{12\ \text{hr}} + \text{sum}_{\text{device\_id}}^{30\ \text{day}}\right) \\
& - 2.53 \log\left(1 + \left(\text{sum}_{s_{\text{email}}}^{60\ \text{day}} + \text{count}_{b_{\text{address}}}^{90\ \text{day}}\right)\right) \\
& + 0.26\left(\text{count}_{s_{\text{phone}}}^{1\ \text{hr}} + \text{count}_{\text{ip}}^{30\ \text{day}}\right)^2 \\
& - 1.84 \log\left(1 + \exp\left(\text{sum}_{b_{\text{email}}}^{4\ \text{hr}} - \text{sum}_{s_{\text{address}}}^{1\ \text{day}}\right)\right) \\
& + 0.41 \log\left(1 + \text{rv}\left(s_{\text{email}}, b_{\text{address}}, 30\ \text{day}\right)\right) \\
& - 0.68 \exp\left(\log\left(1 + \text{sum}_{\text{card\_number}}^{1\ \text{day}} + \text{count}_{\text{device\_id}}^{7\ \text{day}}\right)\right) \\
& + 1.31 \sin\left(\log\left(1 + \text{count}_{s_{\text{phone}}}^{90\ \text{day}}\right) + \log\left(1 + \text{sum}_{\text{ip}}^{30\ \text{day}}\right)\right) \\
& - 0.83\left(\text{count}_{\text{bin\_number}}^{4\ \text{hr}} + \log\left(1 + \text{rv}\left(\text{dev\_id}, \text{bin}, 60d\right)\right)\right) \\
& + 0.25\left(\text{count}_{s_{\text{address}}}^{12\ \text{hr}} + \log\left(1 + \text{sum}_{b_{\text{email}}}^{30\ \text{day}}\right)\right)^2 + 0.27
\end{aligned}
\tag{7}
$$

The coefficient of each term has been rounded-off to the nearest second digit after the decimal. *rv* in the above equation stands for relational velocity. A rule can be further created from the above equation assigning the Fraud Score to be 1 and creating inequalities to trigger the rules.