TuroMiner : An efficient and advanced process discovery algorithm for complex processes

Submitted to Special Session 5: Advances in Data-Driven Optimization and AI-Enhanced Business Process Mining

Pranav Sahu[1], Avinash Singh[1], Uphar Singh[1,2], and Ranjana Vyas[1]

[1] Department of Information Technology, IIIT Allahabad, Prayagraj, India
prf.pranav@iiita.ac.in, mwc2024005@iiita.ac.in, ranjana@iiita.ac.in
[2] SCSET, Bennett University, Greater Noida, India
upharsingh94@gmail.com

Abstract. Petri Nets are very popular for modeling concurrent and distributed systems but their automata-based nature restricts their expressiveness capabilities in dealing with complex and unbounded behaviors. This paper proposes a novel model that is completely Turing Machine logic-based to address the aforementioned constraints. Unlike conventional Petri Nets, this proposed model utilizes the complete computational capabilities of Turing Machines, which provide more flexibility in dealing with complex system behaviors. The experimental results have confirmed that this proposed model is capable of handling real-world applications like event log analysis more effectively.

Keywords: Process Minning, Process Modle, Petri Nets, Complex Process

1 Introduction

The history of computational theory has been characterized by the development of formal models that facilitate the accurate representation and analysis of algorithmic processes. In this regard, the Turing Machine has been identified as a basic construct that establishes the boundaries of solvability in computational theory. In addition to the Turing Machine, Petri Nets have been established as a powerful model for representing concurrent, distributed and asynchronous systems using a graphical and mathematical model [1].

Although each of these models has its own usefulness, they tend to work under different paradigms Turing Machines are very efficient at modeling sequential computation, while Petri Nets are more suited to modeling structural concurrency and synchronization. Nevertheless, many contemporary computational systems, particularly those that are embedded in cyber-physical settings and software systems, tend to have properties that lie in between these two paradigms. This is what gives rise to the need for a unified modeling framework

that can leverage the algorithmic strength of Turing Machines along with the structural expressiveness of Petri Nets[2].

In view of this requirement, the current study proposes a new computational model called Turo Net, which symbolizes the integration of the Petri Net model structure with the operational semantics of the Turing Machine. The Turo Net is expected to capitalize on the strengths of both models by combining the control and memory capabilities of Turing Machines with the concurrency, synchronization and state transition modeling capabilities of Petri Nets. This offers a more expressive and powerful tool for modeling complex computational processes which can be subjected to both structural analysis and algorithmic execution. The major contributions of our work are as follows:

1. Establishes the theoretical foundations of the Turo Net model.
2. Presents the architecture, formal definition, and operational semantics of Turo Net.
3. Analyzes the computational power of the proposed model and demonstrates applications in process modeling, formal verification, and system analysis.
4. Supports the model's relevance through case studies.
5. Introduces a hybrid modeling framework bridging the gap between sequential and concurrent system modeling.
6. Contributes to the broader field of theoretical computer science through a unified modeling approach.

The rest of the paper are as follows: Section 2 discusses Literature Review. Section 3 presents the Turo-miner algorithm. Section 4 contains experimental results. Section 5 contains conclusion and future work.

2 Literature Review

2.1 Process Mining: Fundamentals and Importance

Process mining is a data-driven technique used for the analysis and optimization of operational processes based on event data. Process models are merged with real execution logs to identify inefficiencies, deviations and ensure compliance. The techniques can be used retrospectively to analyze problems or prospectively to forecast outcomes and propose improvements. The techniques are applicable in different industries like manufacturing, healthcare, logistics and government. [3].

The process mining community is organized around three main tasks: discovery, conformance checking and enhancement. Although the value of process mining has been widely acknowledged, many organizations are not yet aware of its potential. To overcome this problem, the Process Mining Manifesto was launched by a global task force. There are resources such as processmining.org that provide tools, example logs and tutorials to facilitate its adoption [4].

PM4Py is a popular open-source Python library that offers support for various process mining tasks. The library allows users to perform various process mining tasks such as log management, model discovery and conformance checking. The library is easy to install and has a comprehensive documentation system

with a collaborative GitHub repository where users can contribute and report issues. PM4Py has made process mining more accessible to researchers and practitioners [5].

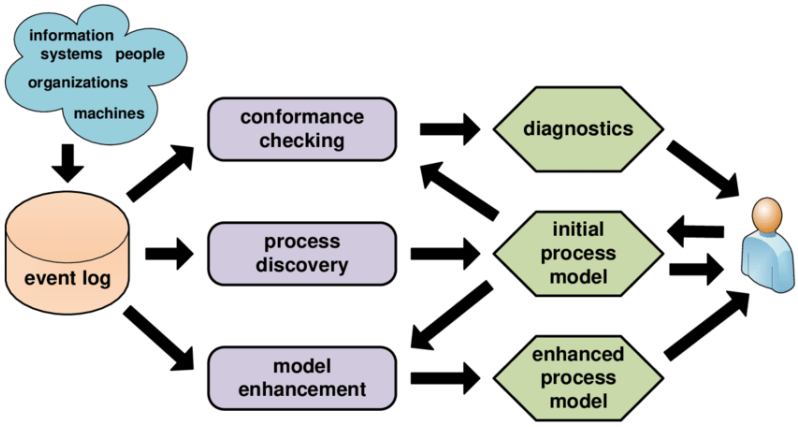

Fig. 1. Overview of Process Mining Techniques

2.2 Role of Petri Nets in Process Modeling

Petri nets are a basic modeling technique broadly used in the field of process discovery because of their ability to model concurrent, distributed and asynchronous systems effectively. A detailed analysis of thirteen major Petri net process discovery algorithms shows that there are major differences in four major areas: assumptions about the completeness of event logs, variety of control-flow constructs supported, level of abstraction between transitions in Petri nets and event classes in event logs and degree to which each algorithm is prone to underfitting or overfitting. This analysis points out that there is a need for improvement in several areas. Future algorithms should be able to handle different levels of abstraction, especially for loosely structured processes, incorporate contextual information such as event attributes and make use of the semantics of event labels to improve the interpretability and robustness of discovered processes. [6].

With the increasing availability of event data, the real-world relevance of process mining is increasing rapidly. However, this also poses a substantial computational problem. Most process discovery algorithms have a linear complexity with respect to the size of the log but an exponential complexity with respect to the number of distinct activities. To overcome this, decomposition approaches have been investigated recently. These approaches divide large process mining problems into smaller sub-problems depending on activity subsets. For conformance checking, both the event log and the process model are decomposed into smaller, manageable pieces, allowing trace validation at the modular level. The same approach can be used for process discovery, where individual sub-logs are

mined independently and their respective models are combined later to form the complete picture. This approach is algorithm-independent and has strong theoretical foundations. Although the current work is focused on trace fitness, future research will aim to integrate other dimensions of quality, such as precision and generalization. These decomposition techniques provide the necessary foundation for improving the scalability of process mining in complex settings [7].

2.3 Traditional Process Discovery Techniques

Alpha Miner: Alpha mining, specifically the α-algorithm, is one of the oldest and most extensively researched methods for process discovery. In a case study examining the "Students Registration" process in a Thai university, the α-algorithm was used to discover the most important routing elements. The event log, containing 299 cases and 569 events, was filtered to retain only "Start" and "Complete" events, allowing the examination of the overall process model. The WF-net was constructed using the α-algorithm plugin in the ProM environment, successfully visualizing the entire set of process activities. Nevertheless, the α-algorithm is known to be noise-sensitive and inefficient when dealing with complex or real-world event logs, suggesting the need for more robust methods in future studies [8].

The use of Alpha mining techniques in educational settings has also shown encouraging results. Specifically, the identification of process models from Moodle event logs was successfully carried out using integrated data preprocessing and analysis facilities. These facilities allowed users to perform process mining in an integrated manner, thus underlining the practical usefulness of Alpha mining techniques in formal educational data [9].

Heuristic Miner: Clustering has been proposed to improve educational process mining. Model interpretability is an important aspect in educational institutions to effectively communicate insights to teachers and students. Since Moodle does not have visualization tools to interpret student activity data, these improved models help in real-time analysis and adaptive learning environments for personalized feedback and learning paths [10].

In another research, the Heuristics Miner was used to analyze the low-structured process of student registration in a Thai university, with event logs of 299 cases and 569 events. The default settings of the Heuristics Miner plug-in were used to analyze the process flow, showing its use in analyzing low-structured educational processes [11].

Fuzzy Miner: An enhanced version of the fuzzy miner algorithm was proposed by incorporating time intervals to define binary correlations between activities. This version emphasizes binary significance derived from time intervals, offering more accurate process modeling based solely on this metric. Compared to the original fuzzy miner in ProM, the improved approach achieved better conformance results. However, it lacks the ability to distinguish specific parallel

structures like XOR, OR and AND, which the authors aim to address in future work by introducing threshold-based enhancements [12]. Another study applied fuzzy miner to analyze a purchase-to-pay process using a large dataset from a multinational company in the coatings industry. The analysis revealed low automation usage less than 10% in repetitive tasks and evaluated automation efficiency across three vendors. A return on investment (ROI) of 159% was reported for one vendor, indicating that robotic process automation (RPA) can improve cost savings, speed and reduce human errors. Future research will focus on refining efficiency models and applying RPA bots more extensively [13].

Split Miner: Split Miner is a process discovery algorithm that has been identified for its scalability and robustness in the automated discovery of business process models. Empirical results have demonstrated that Split Miner outperforms all baseline techniques on F-score and generalization on twelve real-life event logs. The models produced by Split Miner have been found to be of similar size and control-flow complexity to those produced by the Inductive Miner and the Evolutionary Tree Miner, which have been shown to be highly effective on these criteria in the past. Split Miner also has better computational efficiency, with execution times at least three times and up to sixteen times faster than the best baseline on all event logs tested [14].

2.4 Turing Machine:

Token Turing Machines (TTMs) are an extension of Neural Turing Machines that enhance the interaction of memory in decision-making tasks through token summarization. TTMs preserve the advantages of the Transformer model and incorporate an external memory component with a constant computational cost that is not affected by the input size, making them applicable to real-time inference and decision-making tasks such as activity detection and vision-guided robotic action learning. [15].

TTMs are flexible general-purpose models capable of handling very long sequences of tokens. Even though the current work is centered on computer vision tasks such as spatio-temporal action detection and robotic decision-making, the model performs well on long sequences, including tasks with over 18,000 tokens, which is similar to the Long Range Arena benchmarks. It is important to note that TTMs are the first models to apply the Neural Turing Machine paradigm to video-based data. [15].

2.5 Challenges in Automata-Based Process Modeling

The reliance on automata-based modeling limits the expressive power of traditional process discovery methods. These models cannot effectively capture unbounded loops, recursive patterns, or intelligent decision-making based on historical data. As a result, many real-world processes with dynamic and adaptive behavior are poorly represented by these models.

2.6 Research Advancements and Hybrid Approaches

Recent studies have attempted to enhance process models using token-based replay, simulation and learning-based techniques. While these methods improve performance in specific cases, they still depend on finite-state abstractions and do not address the core limitations of automata theory.

2.7 Motivation for a Turing Machine-Enhanced Petri Net

This paper proposes a new model that combines the Turing Machine logic with Petri Nets. Turing Machines have unlimited memory and are computationally universal. They are able to simulate any computable process. By combining these capabilities with the structural benefits of Petri Nets, the new model is able to support intelligent, recursive and memory-driven processes. This is because the new model is able to overcome the limitations of the current methods.

3 Methodology

This study proposes a new way of process discovery based on Turing Machine-inspired models, focusing on memory-augmented trace simulation. Two architectures are considered:

- Single-Tape Turing Machine Process Model (ST-TMPM)
- Multi-Tape Turing Machine Process Model (MT-TMPM)

Both approaches rely on the computational paradigm of *Turing Machines (TMs)* to represent sequential decision-making processes in event logs, allowing for *state tracking*, *historical awareness* and *parallelism* properties that are typically not present in conventional process discovery algorithms like Petri nets or α-algorithms.

3.1 Turing Machine-Inspired Single-Tape Process Discovery

In this algorithm, we use the simulation of a Turing Machine with one tape to simulate the way processes change over time. Imagine the tape as a strip where each step of a process trace is written one after another, just like writing steps in a diary. The head of the machine moves ahead with each step of the process trace as it is read and written on the tape. This is how we can track the order of activities in a process. With each new step of the process trace, we mark a transition from the previous step to the current step, just like drawing arrows between steps.

Explation of Single-Tape Process Discovery Algorithm We model the process discovery pipeline using a single-tape Turing Machine abstraction and directed graphs.

- **Step 1: Initialization.**
  - Initialize tape $T = [\,]$ as empty.
  - Head position $h = 0$.
  - Turing Machine state $s = \texttt{START}$.
  - Initialize a directed graph $G = (V, E)$ with $V = \{\texttt{START}, \texttt{END}\}$ and $E = \emptyset$.
  - Initialize frequency map $F : E \to \mathbb{N}$ to store transition counts.
- **Step 2: Event Log Input.**
  - Import log $L = \{T_1, T_2, \ldots, T_n\}$ from XES file using PM4Py.
  - Each trace $T_i = \langle e_1, e_2, \ldots, e_k \rangle$ is a sequence of events.
  - For each event e_j, extract activity name $a_j = e_j[\text{“concept:name”}]$.
- **Step 3: Tape Writing and Transition Recording.**
  - For each trace $T_i \in L$, initialize $\texttt{prev} \leftarrow \texttt{START}$.
  - For each event a_j in T_i:

$$T[h] \leftarrow a_j, \quad h \leftarrow h + 1$$

$$s \leftarrow \texttt{SEEN\_}a_j$$

$$E \leftarrow E \cup \{(\texttt{prev}, a_j)\}, \quad F[(\texttt{prev}, a_j)] \leftarrow F[(\texttt{prev}, a_j)] + 1$$

$$\texttt{prev} \leftarrow a_j$$

  - After the trace ends, insert final transition:

$$E \leftarrow E \cup \{(a_k, \texttt{END})\}, \quad F[(a_k, \texttt{END})] \leftarrow F[(a_k, \texttt{END})] + 1$$

- **Step 4: Layout and Coloring.**
  - Group each node $v \in V$ by prefix, e.g., $v = \texttt{A\_Start}$ belongs to group L_i where $\text{prefix}(v) = A$.
  - Define layers $\mathcal{L} = \{L_1, L_2, \ldots, L_k\}$.
  - Apply shell layout $\text{ShellLayout}(G, \mathcal{L})$.
  - Assign unique color c_i to each layer L_i.
- **Step 5: Visualization.**
  - Use NetworkX to visualize $G = (V, E)$.
  - Color nodes by group and label each edge (u, v) with $F[(u, v)]$.
  - Display arrows, node names and edge frequencies on a 2D layout.

Algorithm 1 Single-Tape Process Graph Discovery

Require: Event log $L = \{T_1, T_2, \ldots, T_n\}$ in XES format
Ensure: Directed graph $G = (V, E)$ with transition frequencies
Initialize: $state \leftarrow$ "START", $tape \leftarrow [\,]$, $head \leftarrow 0$
Initialize graph $G \leftarrow (V, E)$ as empty, frequency map $F \leftarrow \{\}$
Prompt user to upload `.xes` file
if no file is uploaded **then**
    Exit
end if
Parse log: $L \leftarrow$ `xes_importer.apply(file)`
for all trace $T \in L$ **do**
    $prev \leftarrow$ "START"
    for all event $e \in T$ **do**
        $a \leftarrow e$["concept:name"]
        Append a to $tape[head]$; $head \leftarrow head + 1$
        $state \leftarrow$ "SEEN_" $+ a$
        Add transition edge $(prev, a)$ to E
        Update frequency: $F[(prev, a)] \leftarrow F[(prev, a)] + 1$
        $prev \leftarrow a$
    end for
    Add terminal edge ($prev$, "END") to E
end for
Draw G using spring layout with labeled nodes and frequency-weighted edges
Print "Process graph generated successfully."

3.2 Turing Machine-Inspired Multi-Tape Process Discovery

In this paper, we propose a method for process analysis based on a conceptual MTTM. The algorithm models the processing of events using two tapes: one for the actual event stream and one for contextual tags, such as human or system activities. The machine will, during the processing of these events, change its state and at the same time build a directed graph representing the process flow. This approach maintains both control flow and semantics and, therefore, enables more expressive process modeling.

Multi-Tape Turing Machine for Process Trace Analysis The algorithm used in the implementation simulates a **Multi-Tape Turing Machine (MTTM)** to analyze event logs and build a process graph. The MTTM is formally defined as:

$$M = (Q, \Gamma, \Sigma, \delta, q_0, B, F, T)$$

where:

- Q is the finite set of machine states
- Γ is the tape alphabet
- $\Sigma \subseteq \Gamma$ is the input alphabet
- $\delta : Q \times \Gamma^T \to Q \times \Gamma^T \times \{L, R\}^T$ is the transition function

- $q_0 \in Q$ is the initial state (`START`)
- $B \in \Gamma$ is the blank symbol
- $F \subseteq Q$ is the set of final states
- T is the number of tapes, here $T = 2$

The algorithm proceeds in the following steps:

Step 1: Initialization Each tape is initialized as an empty list:

$$\text{Tape}_i = [] \quad \text{for } i = 1, 2$$

and the read/write heads are set to 0:

$$\text{head}_i = 0$$

Step 2: Trace Iteration Given an event log $L = \{\tau_1, \tau_2, \ldots, \tau_N\}$, each trace $\tau_i = [e_1, e_2, \ldots, e_n]$ is processed sequentially. Let $a_i = \texttt{concept:name}(e_i)$ represent the activity label.

Step 3: Writing to Tapes Each activity a_i is written to Tape 0. Tape 1 stores semantic labels:

$$\text{Tape}_0[\text{head}_0] \leftarrow a_i$$

$$\text{Tape}_1[\text{head}_1] \leftarrow \begin{cases} \texttt{Parallel:}a_i, & \text{if } a_i \text{ has prefix } W \\ \texttt{System:}a_i, & \text{otherwise} \end{cases}$$

Heads are incremented after writing:

$$\text{head}_j \leftarrow \text{head}_j + 1 \quad \text{for } j = 0, 1$$

Step 4: State Update The machine updates its state after each activity:

$$q \leftarrow \texttt{SEEN\_}a_i$$

Step 5: Graph Construction The process graph $G = (V, E)$ is incrementally built:

$$E \leftarrow E \cup \{(p, a_i)\}$$

where p is the previous event. At the end of a trace:

$$E \leftarrow E \cup \{(a_n, \texttt{END})\}$$

Step 6: Visualization Each edge $(u, v) \in E$ is annotated with frequency:

$$w(u, v) = \text{count of transition from } u \text{ to } v$$

The nodes are classified according to prefixes (such as A_, W_ and so on) and the shell layout is used for visualization. The colors represent activity types and the edges represent frequency.

The result is a semantically annotated graph with frequency weights that represents the process flow simulated using the multi-tape Turing machine abstraction.

Algorithm 2 Multi-Tape Turing Machine-Based Process Graph Construction

Require: Event log L in XES format, number of tapes T (default: 2)
Ensure: Directed process graph G with annotated transition frequencies
Initialize Turing Machine with state `START`, T empty tapes and tape heads at position 0
Initialize an empty directed graph G
for each trace $\tau \in L$ **do**
  Set previous event $p \leftarrow$ `START`
  for each event $e \in \tau$ **do**
    Extract event name $n \leftarrow e$["concept:name"]
    Write n to Tape 0
    if n contains prefix `"W"` **then**
      Write `"Parallel:`n`"` to Tape 1
    else
      Write `"System:`n`"` to Tape 1
    end if
    Update state $\leftarrow$ `SEEN_`n
    Log the transition execution
    Add edge (p, n) to graph G
    Update $p \leftarrow n$
  end for
  Add edge $(p,$ `END`$)$ to graph G
end for
Group nodes in G by prefix (e.g., A, W) to structure layout
Assign colors to nodes based on categories (human, system, etc.)
Annotate edges in G with frequency counts
Render the process graph G

4 Result

4.1 Using Single-Tape Turing Machine Simulation

In this implementation, a simplified form of the Turing Machine with **single tape** is employed to simulate the processing of event traces and to build a

corresponding process flow graph. The system parses standard XES files and builds a visual representation of how events move through the business process.

The architecture comprises three core classes:

- **Transition:** Represents the processing of a single event (transition) in the trace. It logs its execution for debugging and simulation traceability.
- **TuringMachine:** Simulates a single-tape Turing machine that:
  - Maintains a tape to store a linear sequence of events.
  - Advances a tape head after each event is processed.
  - Updates its internal state based on the most recently seen event.
- **ProcessGraph:** Uses `networkx` to build a directed graph:
  - Each edge represents a transition from one event to the next.
  - Edge frequencies are tracked and displayed using `matplotlib`.

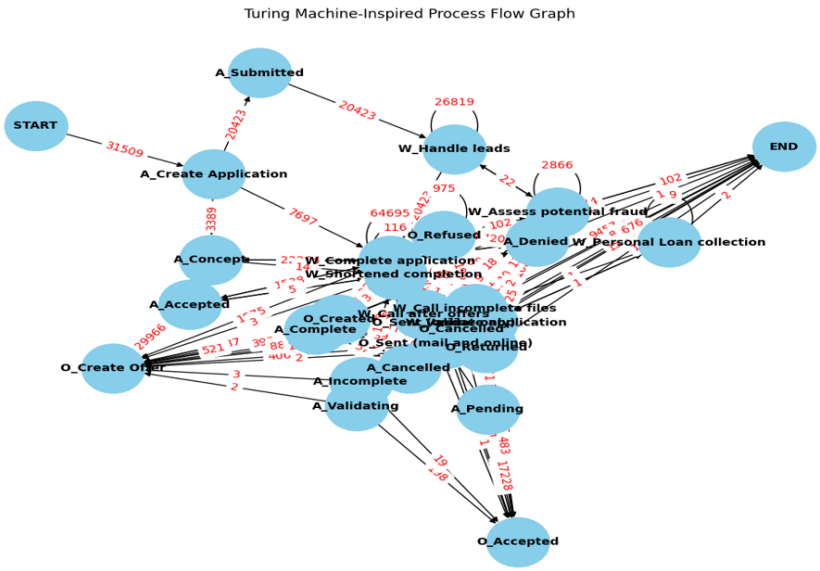

Fig. 2. Turing Machine-Inspired Process Flow Graph using Single Tape

Execution Workflow

1. The user uploads a `.xes` file with event logs via the Google Colab environment.
2. The log file is imported using `pm4py`.
3. For each trace in the log:

- The trace is processed one by one.
- Events are appended to the Turing Machine's tape.
- Transitions are performed and recorded.
- Transitions are added to the process graph.

4. A transition from the last event to "

Error Handling Effective error handling mechanisms are put in place to ensure that:
- Missing or empty uploads are properly identified.
- Corrupted or invalid XES files are handled.
- Unexpected data formats or I/O errors result in graceful failure.

Graphical Output The system produces a process graph, which has the following characteristics:
- The graph contains nodes that represent events found in the logs.
- The graph contains directed edges that represent the order of events.
- The graph contains red edge labels that represent the transition frequency of events.

Significance This simulation is a learning tool for understanding how a simple automaton can model process execution. It is also a link between:
- **Theoretical models of computation** (Turing machines).
- **Practical process mining** (log analysis and process simulation).

Although it is a simple simulation, the single-tape simulation is useful for:
- Illustrating linear event progression.
- Showing the importance of high-frequency transitions.
- Allowing initial insights into business process definition.

4.2 Using Multi-Tape Turing Machine Simulation

For the purpose of event log analysis and process simulation, a new framework has been designed that uses the **Multi-Tape Turing Machine (MTTM)** paradigm to represent the behavior of business processes. The framework has been developed in Python and utilizes process mining techniques through the `pm4py` library for event log import in the `.xes` format, as well as `networkx` and `matplotlib` libraries for dynamic process graph visualization.
The simulation process has the following main components:

- **MultiTapeTuringMachine Class:** This class models a multi-tape Turing machine with two tapes:
  - Tape 0 holds the actual sequence of events.
  - Tape 1 marks events as system events or human events, depending on naming conventions (for example, events named with `"W_"` prefixes are treated as human tasks).

  The Turing machine's state transitions are dynamically determined by the sequence of events being processed.

– **Transition Class:** Models individual state transitions. Each transition records its execution and simulates movement through the event sequence in the trace.
– **ProcessGraph Class:** Builds a directed graph with `networkx` to model the control flow of events. Each edge is frequency-annotated to show how often transitions happen. Events are organized by prefix for easier understanding (for example, "A_", "W_" and so on).
– **Visualization:** The graph is drawn with `matplotlib`, using color-coded nodes for different event types. The graph is organized to group similar tasks together and offers a clear, layered view of the process.

Fig. 3. Turing Machine-Inspired Process Flow Graph using Multi-Tape

Execution Flow The following is the sequence of execution of the tool:
(a) The user uploads a `.xes` event log file through the Google Colab environment.
(b) The log file is parsed by `pm4py`.
(c) Each trace is then processed as follows:
  – Events are written to both tapes of the MTTM.
  – Transitions are recorded and applied.
  – Graph edges and nodes are updated accordingly.

(d) A final process graph is then plotted and shown, indicating the frequency and direction of event transitions.

Error Handling Error handling has been implemented throughout the system to handle the following:
- Incomplete or incorrect event attribute data.
- Tape write errors (such as out-of-bound writes).
- File I/O errors.
- Graph rendering problems.

Significance This hybrid simulation model integrates ideas from:
- **Automata Theory:** by simulating event behavior using a multi-tape Turing machine.
- **Process Mining:** by interpreting execution traces as annotated control-flow graphs.

This simulation model is especially relevant for analyzing logs with:
- A combination of human and system processes.
- Partial or semi-structured event information.
- A requirement for traceability between logical execution and process structure.

Sample Output The system produces a directed process graph with:
- Each node symbolizing a unique event.
- Edges symbolizing transition routes with annotated frequency information.

This graphical representation enables a clear understanding of process behavior, most prominent routes and possible parallelism in process activities.

5 Conclusion and Future Scope

This research presents a new way of process discovery through a Multi-Tape Turing Machine (MT-TM) model, which has its own tapes for system and human behavior in events to aid trace simulation and process analysis. Unlike conventional approaches such as Petri nets, which are associated with issues of noise and parallelism, the MT-TM model boasts of linear awareness of its history, contextual memory based on its tape, and state traceability. Additionally, customization is achievable through its MT-TM module, which enables easy tracking of execution, processing, and graph representation. Therefore, it is ideal for application in tracing complex systems that include both system and human behavior.

The Multi-Tape Turing Machine (MT-TM) paradigm can be extended by incorporating additional tapes, better semantic analysis through NLP and noise modeling and temporal reasoning. Future research will also involve combining the output of MT-TM with machine learning techniques for prediction and simplifying the complexity of graphs for large logs. Efficient history representation based on Token Turing Machines can also be explored.

References

1. Vardakis, C., Dimolitsas, I., Spatharakis, D., Dechouniotis, D., Zafeiropoulos, A., & Papavassiliou, S. (2025). A Petri Net-based framework for modeling and simulation of resource scheduling policies in Edge Cloud Continuum. *Simulation Modelling Practice And Theory, 141*, 103098.
2. Zaitsev, D. (2023). Strong Sleptsov nets are Turing complete. *Information Sciences, 621*, 172-182.
3. Van der Aalst, W.M.P. (2022). Process mining: a 360 degree overview. In *Process Mining Handbook* (pp. 3-34). Cham: Springer International Publishing.
4. Van Der Aalst, W.M.P. (2012). Process mining: Overview and opportunities. *ACM Transactions on Management Information Systems (TMIS), 3*(2), 1-17.
5. Berti, A., Van Zelst, S.J., & Van der Aalst, W. (2019). Process mining for python (PM4Py): bridging the gap between process-and data science. *arXiv preprint arXiv:1905.06169*.
6. Van Dongen, B.F., De Medeiros, A.K.A., & Wen, L. (2009). Process mining: Overview and outlook of petri net discovery algorithms. *Transactions on Petri Nets and Other Models of Concurrency II: Special Issue on Concurrency in Process-Aware Information Systems*, 225-242.
7. Van der Aalst, W.M.P. (2013). Decomposing Petri nets for process mining: A generic approach. *Distributed and Parallel Databases, 31*, 471-507.
8. Weerapong, S., Porouhan, P., & Premchaiswadi, W. (2012). Process Mining Using α-Algorithm as a Tool (A Case Study of Student Registration). *Proceedings of the 2012 Tenth International Conference on ICT and Knowledge Engineering*, IEEE, 213-220.
9. Nafasa, P., et al. (2019). Implementation of alpha miner algorithm in process mining application development for online learning activities based on moodle event log data. *2019 3rd International Conference on Informatics and Computational Sciences (ICICoS)*. IEEE.
10. Bogarín, A., et al. (2014). Clustering for improving educational process mining. *Proceedings of the fourth international conference on learning analytics and knowledge*, 11-15.
11. Ayutaya, N.S.N., Palungsuntikul, P., & Premchaiswadi, W. (2012). Heuristic mining: Adaptive process simplification in education. *2012 tenth international conference on ict and knowledge engineering*, IEEE, 221-227.
12. Effendi, Y.A., Sarno, R., & Marsha, D.V. (2021). Improved fuzzy miner algorithm for business process discovery. *Telkomnika (Telecommunication Computing Electronics and Control), 19*(6), 1830-1839.
13. Bahaweres, R.B., Amna, H., & Nurnaningsih, D. (2022). Improving purchase to pay process efficiency with RPA using fuzzy miner algorithm in process mining. *2022 International Conference on Decision Aid Sciences and Applications (DASA)*. IEEE.
14. Augusto, A., Conforti, R., Dumas, M., La Rosa, M., & Polyvyanyy, A. (2019). Split miner: automated discovery of accurate and simple business process models from event logs. *Knowledge and Information Systems, 59*, 251-284.
15. Ryoo, M.S., et al. (2023). Token turing machines. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*.