Evaluating Nested and Non-Nested Software Transactional Memory Using Machine Learning Classifiers

Meenu
Department of CSE, M. M. M. U. T., Gorakhpur, India
myself_meenu@yahoo.co.in

Keywords: machine learning, software transactional memory, performance analysis, transactional systems, parallel computing, concurrency control

Received: February 20, 2025

Software Transactional Memory (STM) provides a robust solution for addressing concurrency challenges in software systems. This paper explores the performance evaluation of nested and non-nested STM configurations using a machine learning-based framework. The dataset used for model training was generated through detailed heap profiling of nested and non-nested STM configurations, capturing memory allocation and usage patterns as key indicators of STM behaviour. The framework leverages profiling datasets to analyse STM operations through four machine learning models: Naive Bayes, Decision Tree, K-Nearest Neighbours (KNN), and Random Forest. The methodology includes data preprocessing, model training, and visualization using MATLAB R2020b, with a focus on 20 profiling metrics that encapsulate key STM operations. Computational experiments reveal that Naive Bayes, KNN, and Random Forest achieved 100% accuracy, precision, recall, and F1-score, while Decision Tree showed lower performance. These results demonstrate the potential of machine learning to evaluate STM behaviour through data-driven analysis.

Povzetek: Predstavljeno je podatkovno ogrodje za upravljanje programsko transakcijskega pomnilnika in strojno učenje za ocenjevanje konkurenčnih konfiguracij. Iz profiliranja kupov izlušči dvajset metrik ter z modeli Naive Bayes, odločitvenim drevesom, KNN in naključnim gozdom razvršča ugnezdene in neugnezdene STM, z vizualizacijami, poročanjem rezultatov ter postopki za povečanje robustnosti analize.

1 Introduction

This section discusses Software Transactional Memory (STM) as a solution for concurrency challenges, comparing the benefits and trade-offs of nested and non-nested STM. This paper proposes a machine learning-based framework to evaluate STM configurations and support future optimization. Lastly, the Introduction outlines the paper's structure, covering STM evolution, the proposed machine learning-based framework, experimental setup, results, conclusions, and future directions, ensuring a logical flow from concepts to insights.

Software Transactional Memory (STM) is a well-established technique for managing shared memory in parallel and distributed systems, offering a flexible alternative to traditional lock-based synchronization methods in concurrent programming. Unlike traditional lock-based synchronization mechanisms, STM provides a higher level of abstraction, ensuring the properties of atomicity, consistency, and isolation (ACI) for transactional operations. This abstraction not only simplifies the development process but also mitigates common concurrency issues such as deadlocks, race conditions, and priority inversion. By allowing transactions to proceed speculatively and resolving conflicts only when they occur, STM fosters improved performance and scalability in modern multicore and distributed environments. Among the various STM configurations, nested and non-nested STM have gained prominence due to their distinct features and capabilities. Nested STM introduces a hierarchical structure, allowing transactions to contain subtransactions. This structure offers fine-grained control over transactional operations, isolating conflicts at the subtransaction level to enhance concurrency. For instance, in scenarios where a parent transaction encounters a conflict, unaffected subtransactions can continue execution independently. This modularity and flexibility make nested STM particularly suitable for complex applications requiring high levels of concurrency and adaptability. However, these benefits come with trade-offs, including increased memory overhead and computational complexity, which can pose challenges in resource-constrained systems. In contrast**,** non-nested STM adopts a flat transaction structure, focusing on simplicity and lower overhead. By avoiding the hierarchical organization of transactions, non-nested STM reduces the computational burden associated with managing nested dependencies and conflicts. This simplicity translates to faster execution times and more efficient resource utilization, making it an

attractive choice for applications where performance and straightforward conflict resolution are critical. Nonetheless, the lack of hierarchical isolation in non-nested STM can lead to limitations in concurrency management, especially in scenarios involving highly interdependent operations. Given the trade-offs inherent in these configurations, a systematic evaluation of their performance is crucial to guide their adoption in specific application domains. Key metrics such as memory usage, execution time, resource allocation efficiency, and concurrency levels serve as benchmarks for understanding their strengths and limitations. Traditional performance analysis methods often rely on manual profiling and ad-hoc comparisons, which may not fully capture the intricate patterns and dependencies within STM systems. To address this gap, we propose a novel approach that leverages supervised machine learning techniques to analyse and compare nested and non-nested STM configurations. By employing a diverse set of classifiers, including Naive Bayes, Decision Trees, K-Nearest Neighbours (KNN), and Random Forests**,** we aim to classify STM configurations based on their performance profiles [1]. These classifiers, known for their robustness and interpretability, enable us to uncover patterns and insights within STM profiling datasets. Furthermore, we enhance the analysis through data augmentation techniques to ensure robust model training and visualization tools to provide intuitive representations of the results. The proposed methodology provides a structured framework for evaluating STM configurations, combining the strengths of machine learning with the rich information contained in profiling datasets. By identifying correlations between key performance metrics and STM configurations, this approach facilitates informed decision-making for optimizing concurrency and scalability in real-world applications. Additionally, the study demonstrates the potential of integrating machine learning into STM performance analysis, paving the way for advancements in concurrent programming practices.

This paper presents a framework using machine learning to evaluate STM performance, focusing on the differences between nested and non-nested configurations to enhance scalability. This study is guided by the following research questions: Can machine learning models accurately classify and distinguish between nested and non-nested STM configurations based on profiling data? Among the evaluated models—Naive Bayes, Decision Tree, K-Nearest Neighbours (KNN), and Random Forest—which demonstrate the best performance for STM profiling analysis? What insights into memory allocation patterns and STM behaviour can be derived from heap profiling data, and how do these insights inform model performance? Finally, how does the proposed ML-based framework compare to traditional STM analysis methods in terms of automation, scalability, and interpretability? Addressing these questions helps establish the effectiveness and practical value of the proposed approach.

Section 2 traces the evolution of STM, compares nested and non-nested STM, and discusses profiling tools and machine learning for STM evaluation. Table 1 summarizes key STM design choices, comparing SOTA approaches and the proposed ML-based framework. Section 3 outlines the machine learning-based approach for analysing STM datasets using MATLAB [2], covering preprocessing, training, and evaluation. Section 4 presents the experimental setup for assessing the machine learning framework, ensuring reliability and reproducibility in STM analysis. Section 5 evaluates the performance of Naive Bayes, Decision Tree, KNN, and Random Forest models applied to STM profiling datasets, comparing them using key metrics like accuracy, precision, recall, and F1-score. Section 6 evaluates the framework's effectiveness, acknowledges limitations, and outlines future directions. Section 7 summarizes the findings, emphasizing model selection for STM performance improvement. Section 8 outlines future improvements for the ML-based STM framework, focusing on better data, model tuning, real-world testing, and advanced learning techniques to enhance accuracy, scalability, and adaptability.

2 Literature review

This section outlines the evolution of Software Transactional Memory (STM) from HTM to HyTM, highlighting key design parameters like granularity, conflict detection, and contention management. It contrasts nested STM, which offers modularity but higher overhead, with non-nested STM, which prioritizes simplicity and efficiency. Profiling tools and emerging machine learning techniques are discussed as methods to evaluate STM performance. The section concludes by emphasizing STM's adaptability and the potential of machine learning to drive future innovations in concurrency management. To summarize STM design choices and innovations, Table 1 compares SOTA approaches and includes the proposed ML-based framework as a scalable evaluation method.

STM has emerged as a key alternative to traditional synchronization mechanisms, providing flexibility and scalability in managing concurrency. Its evolution, ranging from basic hardware-based implementations to advanced software models, has paved the way for nested and non-nested configurations, making it a versatile solution for diverse applications. Transactional Memory (TM) originated with Herlihy and Moss, who introduced Hardware Transactional Memory (HTM) [3]. HTM achieved atomic transaction execution at the hardware level but faced challenges like hardware dependency and limited scalability. To address these issues, Shavit and Touitou proposed Software Transactional Memory (STM) [4], which removed hardware constraints and offered greater adaptability. Later, Hybrid Transactional Memory (HyTM) [5] emerged, combining HTM's performance benefits with STM's flexibility, making it a hybrid solution

for varied workloads. Nested transactions, introduced by Moss [6], and extended by Moss and Hosking, originated in databases to manage complex operations. Nested Transactional Memory [7].enhances modularity by allowing modules to call others without transactional dependency concerns, improving software composition and integration. Understanding the design parameters is essential for developers to evaluate STM systems for specific application needs, ensuring improved performance and reliability. Differences in STM designs significantly affect the programming model and system performance [3] [8] [9] ,encompassing aspects such as transaction granularity, update policies, read and write policies, acquire strategies, conflict detection mechanisms, memory management, contention management techniques, isolation levels, and nesting models. Transaction Granularity, whether word-based or object-based, defines the unit of conflict detection and affects accuracy and communication costs. Update Policies, such as direct or deferred updates, determine how transactions modify shared objects and influence memory usage and immediacy of changes. Read Policies, distinguishing between invisible and visible reads, guide access to shared resources and balance consistency with accessibility [10] .Acquire Strategies, categorized as eager or lazy, determine how transactions obtain exclusive access to memory, impacting concurrency and responsiveness [10] . Similarly, Write Policies, including write-through and buffered strategies, define how transactions handle commits and aborts, balancing simplicity with efficiency. Conflict Detection Mechanisms are categorized as early, late, or lazy, and they balance computational effort and wasted work to ensure overall consistency. Concurrency Control Methods include pessimistic approaches using blocking synchronization (e.g., locks to secure exclusive access to resources) and optimistic techniques relying on non-blocking synchronization (e.g., wait-free, lock-free, or obstruction-free methods) [11]. Effective Memory Management, encompassing allocation and deallocation strategies, safeguards against memory leaks and enhances system stability [10].

Contention Management Techniques, such as Timid [12], Polka [13], Greedy [14] and Serializer (as implemented in the RSTM project[i]),decide when to abort transactions, adapting to varying contention levels and ensuring efficient conflict resolution [10] . Isolation Levels are critical for maintaining data consistency, but weak isolation, as seen in STM Haskell, may lead to anomalies when threads concurrently access shared .The Nesting Model [8] [15] introduces various methods for managing transactions, such as Flattened Nesting, implemented in DSTM [16] and RSTM [17];Linear Nesting with Closed Nested Transactions (CNTs) [18], used in McRT-STM [19] , NORec [20], Nested LogTM [21] [22] and Haskell STM [23] [24] [25] [26] [27] [28] [29]; Open Nested Transactions (ONTs) [30] [31] ,exemplified by ATOMOS [32] and Parallel Nesting, allowing multiple active transactions within a tree structure, as implemented in NeSTM [33], HParSTM [34], NePalTM [35], CWSTM [36], PNSTM [37] and SSTM [38].These approaches cater to diverse application requirements by balancing transaction granularity, isolation, and concurrency. By addressing these interconnected design parameters, STM systems can be tailored to improve performance, scalability, and reliability, enabling developers to select and implement strategies best suited for their application-specific needs. Each design choice contributes to a comprehensive framework that shapes the behaviour of STM systems, ensuring efficient transaction processing in modern concurrent environments. To consolidate the key design choices in STM systems, Table 1 presents a comparative summary of state-of-the-art (SOTA) approaches for Software Transactional Memory (STM) configurations, along with the proposed ML-based evaluation framework.

Table 1: Comparison of SOTA approaches for software transactional memory (STM) configurations

Approach	Strengths	Weaknesses	Key Results	Examples / Systems	Why Insufficient?
Transaction Granularity	High accuracy (word-based); simplicity and lower cost (object-based).	Word-based is costly in performance; object-based sacrifices precision.	Object-based STM (e.g., STM Haskell) achieves a good balance of cost and accuracy.	Word-based STM, Object-based STM (STM Haskell).	Trade-off between accuracy and overhead, not suitable for all STM workloads.
Update Policy	Deferred update allows easy rollback and reduces overhead on shared memory.	Direct update can cause cascading aborts; deferred update can increase commit latency.	STM Haskell uses deferred update successfully for isolation and rollback.	Direct Update, Deferred Update (STM Haskell).	Some real-time systems cannot tolerate deferred commit delays.
Read Policy [10]	Invisible reads have low runtime overhead; visible reads provide	Invisible reads risk late conflicts; visible reads require more synchronization (locks/reader lists).	STM Haskell uses visible reads for better conflict resolution.	Invisible Reads, Visible Reads (STM Haskell).	No single optimal policy; trade-off depends on transaction patterns.

	stronger consistency guarantees.				
Acquire Policy [10]	Lazy acquire improves concurrency and works well with deferred updates.	Eager acquire reduces aborts but increases locking contention.	Lazy acquire preferred in many STMs for better buffering and flexibility.	Eager Acquire, Lazy Acquire.	Eager acquire harms scalability; lazy acquire not suitable when real-time guarantees are required.
Write Policy	Buffered write avoids unnecessary aborts and allows speculative execution.	Write-through is simpler but increases abort cost and reduces concurrency.	Buffered write used in most modern STM systems to improve commit efficiency.	Write-through / Undo, Buffered Write.	Buffered write increases commit complexity and memory overhead.
Conflict Detection	Early detection reduces wasted work; late detection improves parallelism and throughput.	Early detection requires complex tracking; late detection risks wasted work and starvation.	STM Haskell employs late/lazy detection to maximize parallelism.	Early Conflict Detection, Late/Lazy Conflict Detection (STM Haskell).	Late detection can lead to high abort rates in high-contention environments.
Concurrency Control [11]	Optimistic control allows maximum parallelism; non-blocking methods ensure progress under contention.	Pessimistic control ensures consistency but blocks threads; optimistic requires effective contention manager.	Many modern STM systems adopt optimistic control with non-blocking synchronization where possible.	Pessimistic (2PL, Lock-based STM), Optimistic STM, Non-blocking STM (Wait-free, Lock-free, Obstruction-free).	No single method works well for both low-contention and high-contention workloads; tuning is difficult.
Memory Management [10]	Efficient memory management prevents leaks and supports failure recovery.	Complex to implement when supporting nested transactions or variable-sized objects.	Memory-safe STM libraries (e.g., LibSTM) can effectively manage transactional memory.	LibSTM, Haskell STM.	High overhead in managing dynamic or complex object lifecycles in STM.
Contention Management [10]	Advanced policies like Greedy and Serializer can guarantee progress and improve fairness.	Requires tuning and may not adapt well to workload changes.	Greedy and Serializer policies outperform Timid and Polka in bounded commit scenarios.	Timid [12], Polka [13], Greedy [14] Serializer (as implemented in the RSTM project [1]),	Static policies may not adapt well to dynamic transaction workloads.
Isolation	Weak isolation increases concurrency and performance.	Risk of anomalies and subtle bugs if developers are unaware.	STM Haskell's weak isolation allows better scaling.	Weak Isolation (STM Haskell).	Harder to reason about correctness; non-transactional access can break atomicity.
Proposed ML-Based Approach	Automated, scalable, data-driven evaluation of STM configurations.	Depends on dataset quality and model tuning.	Computational experiments reveal that Naive Bayes, KNN, and Random Forest achieved 100% across all metrics; Decision Tree underperformed.	Proposed ML-based framework applied to STM profiling datasets using Naive Bayes, Decision Tree, KNN, and Random Forest.	Currently focuses on classification; predictive optimization is future work.

The comparative insights from Table 1 highlight that existing STM approaches involve significant trade-offs across performance, consistency, and scalability. This motivates the need for an automated and data-driven framework—such as the one proposed in this study—to systematically evaluate STM configurations.

Nested STM introduces hierarchical structures where subtransactions can operate independently, offering modularity and fault isolation. While this enhances concurrency, it increases memory overhead and rollback complexity. In contrast, non-nested STM offers a flat structure with simpler conflict resolution, suitable for lightweight applications. The choice between nested and non-nested configurations depends on application complexity and performance requirements .STM systems are evaluated based on execution time, abort rates, memory usage, and throughput. Profiling tools, like Haskell's heap and time profilers, have provided insights into memory allocation and execution bottlenecks. While

nested STM systems excel in handling complex interdependencies, they require higher resources, unlike non-nested configurations, which prioritize efficiency. Machine learning has opened new avenues for STM analysis and optimization. Supervised learning models like decision trees and random forests predict conflicts and classify transactions, while reinforcement learning dynamically adjusts configurations in real time.

However, integrating machine learning holistically into STM systems remains an area of ongoing research.

Recent research has explored the application of machine learning to dynamically enhance STM performance. For example, Rughetti et al. [39] proposed a self-adjusting concurrency mechanism using ML to adapt thread parallelism in STM systems and further refined this approach in a later study on ML-based thread-parallelism regulation [40], showing notable throughput improvements under varying workloads . While these works demonstrate the value of ML for STM tuning, a comprehensive, profiling-metric–driven ML framework for systematic STM configuration evaluation remains limited, motivating the approach presented in this study.

In conclusion, the literature highlights the evolution of Software Transactional Memory (STM) as a flexible and efficient concurrency management tool. The trade-offs between nested and non-nested STM, alongside critical design parameters, enable developers to tailor STM systems for diverse application needs. The integration of machine learning into STM evaluation holds significant promise, paving the way for future advancements in scalable and adaptive concurrency solutions.

3 Proposed method

This section outlines a machine learning-based approach to analyse STM profiling datasets for nested and non-nested configurations, with a focus on model design, using MATLAB R2020b for all simulations and visualizations. The overall architecture of the proposed method, as depicted in Fig. 1, integrates data preprocessing, model training, evaluation, and visualization to provide a comprehensive solution for STM performance analysis.

The methodology includes data preprocessing, model training, evaluation, and visualization. The system operates on 20 key profiling metrics and classifies data into two categories: nested STM (1) and non-nested STM (0). To improve robustness, data augmentation techniques such as Gaussian noise addition and dataset expansion are employed. Four supervised machine learning models (Naive Bayes, Decision Tree, KNN, and Random Forest) are used for classification. In this study, we used k = 3 for KNN and 50 trees for Random Forest based on common practice for small to medium-sized datasets, where such default values provide a reasonable trade-off between performance and complexity.

Preliminary tuning experiments showed minimal variation in performance across different k and tree values, so fixed values were chosen to maintain reproducibility and reduce computational overhead. The model's performance is evaluated using accuracy, precision, recall, and F1 score. Visualization tools like bar plots and confusion matrix heatmaps are used to present results, and performance metrics are saved for further analysis.

3.1 Model design

The core of the proposed system is its model design, which integrates data processing, machine learning, and visualization in a seamless workflow. The design comprises the following key components:

3.1.1 Dataset preparation

- **Metrics definition**

The system operates on datasets comprising 20 metrics (features) that encapsulate the behaviour of STM operations. These metrics, such as Total_IO, String, and TextEncoding, serve as indicators for profiling STM performance.

- **Class labels**

The dataset is classified into two categories:

  - **Nested STM**

Represented as 1.

  - **Non-nested STM**

Represented as 0.

3.1.2 Data preprocessing

Data preprocessing involved parsing the heap profiling outputs to extract relevant numerical metrics that capture STM performance characteristics. The 20 selected features were chosen based on domain knowledge and their relevance to memory allocation patterns, transaction behaviour, and concurrency effects. No explicit dimensionality reduction techniques (e.g., PCA) were applied; rather, feature selection was guided by interpretability and practical significance. In addition, Gaussian noise augmentation was employed to enhance dataset variability, mitigate overfitting, and improve model generalization.

3.1.3 Data augmentation

- To enhance the robustness of the system, the model employs **data augmentation techniques**:

  - **Gaussian noise addition**

Introduces slight variations to the data, ensuring better generalization.

  - **Extended dataset**

The original and augmented datasets are merged to increase the size and variability, reducing overfitting risks.

3.1.4 Train-Test Split

- **Partitioning**

A stratified 70/30 split was used to divide the dataset into training and test sets. Stratification ensured that both nested and non-nested STM classes were proportionally represented in both subsets, enhancing evaluation reliability.

- **Test set validation**

To prevent data leakage and preserve evaluation integrity, Gaussian noise augmentation was applied exclusively to the training set. The test set remained entirely unseen throughout model training.

3.1.5 Supervised machine learning models

The system evaluates and compares four machine learning classifiers:

- **Naive Bayes (Kernel Distribution)**

Probabilistic model leveraging the Bayes theorem.

- **Decision Tree**

Rule-based algorithm for classification.

- **K-Nearest Neighbors (KNN)**

Distance-based classification model.

- **Random Forest (TreeBagger)**

Ensemble model combining multiple decision trees.

Each model undergoes the following process:

- **Training**

Using training data (XTrain and yTrain).

- **Prediction**

Generating predictions on test data (XTest).

Theoretical time complexities of the models are as follows: Naive Bayes operates in O(n), KNN in O(n·d) for prediction, Decision Tree in O(n·log n), and Random Forest in O(m·n·log n), where n is the number of samples, d the number of features, and m the number of trees. These complexities are manageable for small datasets; however, future work will explore scalability on larger STM datasets.

3.1.6 Evaluation metrics

The model's performance is assessed using several metrics:

- **Accuracy**

Proportion of correctly classified transactions.

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{FN} + \text{TN}}$$

- **Precision**

Proportion of true positives among predicted positives.

$$\text{Precision} = \frac{\text{TP}}{\text{TP}+\text{FP}}$$

- **Recall (Sensitivity)**

Proportion of actual positives correctly identified.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

- **F1 Score**

Harmonic mean of Precision and Recall.

$$\text{F1 Score} = \frac{2\text{XPrecisionXRecall}}{\text{Precision+Recall}}$$

Where:

- **TP**: True Positives
- **FP**: False Positives
- **FN**: False Negatives
- **TN**: True Negatives

These metrics ensure a balanced assessment of the model's predictive power.

3.1.7 Visualization and reporting

- **Bar plot visualizations**

Graphical representations of evaluation metrics for all classifiers across the 20 features.

- **Confusion Matrix Heatmaps**

Visualizes model performance for each classifier.

- **Exported results**

○ Metrics and performance scores are saved in STM_Profiling_PerformanceMetrics.xlsx.

○ Plots are stored as high-resolution TIFF images for documentation purposes.

In conclusion, the proposed machine learning-based approach offers a robust and scalable methodology for analysing STM profiling datasets. By leveraging various classifiers and data augmentation techniques, the system ensures accurate and reliable performance evaluation for both nested and non-nested STM configurations. The integration of MATLAB R2020b facilitates seamless simulations and visualizations, providing valuable insights into STM behaviour. This approach not only enhances the understanding of STM performance but also paves the way for optimizing concurrency and scalability in real-world applications, contributing to the advancement of software transactional memory systems.

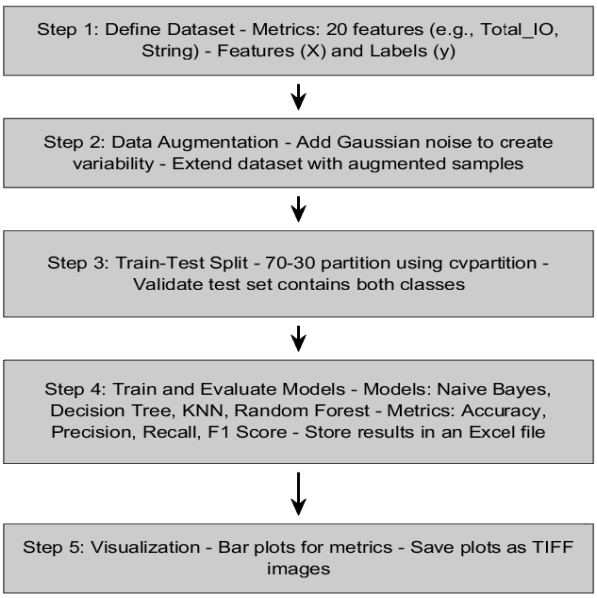

Figure 1: Architecture of the proposed method

4 Performance evaluation

This section outlines the experimental setup for assessing the proposed machine learning-based framework for STM profiling. It includes details on the hardware and software environment, dataset, machine learning models, and evaluation metrics used. The experiments aim to evaluate the performance of nested and non-nested STM configurations, ensuring reliability and reproducibility in the analysis process.

4.1 Experimental setup

The experiments were conducted in a controlled environment using MATLAB R2020b, ensuring consistency and reproducibility. The following subsections describe the key components of the experimental setup.

4.1.1 Hardware and software environment

All experiments were conducted on a system running Windows 11 Pro, equipped with an Intel Core i5-1035G1 CPU @ 1.20 GHz, 8 GB DDR4 RAM, and a 512 GB Intel 660p Series NVMe SSD (Model: SSDPEKNW512G8H). MATLAB R2020b with the Statistics and Machine Learning Toolbox was used for implementation, leveraging functions such as fitcnb, fitctree, fitcknn, and TreeBagger for model training, as well as confusionmat and cvpartition for evaluation and visualization.

4.1.2 Dataset

- **Source**

The STM profiling dataset used in this study was internally generated. Detailed heap profiling was performed on both nested and non-nested STM configurations implemented in Haskell. The profiling process captured memory allocation patterns and runtime behaviours using the runtime options +RTS -p -i0.0000000000000001 -hy, with visualization facilitated via hp2ps. The profiling outputs were systematically parsed to construct a dataset containing 20 key profiling metrics that reflect STM operation characteristics. The dataset is not publicly available due to its dependency on the specific experimental environment and codebase; however, the complete profiling methodology is provided to ensure reproducibility and transparency.

- **Features**

20 profiling metrics representing STM operations, such as Total_IO, String, and TextEncoding.

- **Classes**

Binary classification with labels:

  - 1 for Nested STM.
  - 0 for Non-Nested STM.

- **Augmentation**

Gaussian noise added to the dataset for improved variability and generalization.

4.1.3 Models and parameters

- **Machine learning models**

  - Naive Bayes (Kernel Distribution).
  - Decision Tree.
  - K-Nearest Neighbors (KNN) with k=3
  - Random Forest (TreeBagger) with 50 trees.

- **Evaluation metrics**

  - Accuracy, Precision, Recall, and F1 Score.

- **Partitioning**

70% training, 30% testing using cvpartition.

4.1.4 Implementation details

- **Confusion matrix**

Used to derive True Positives (TP), False Positives (FP), False Negatives (FN), and True Negatives (TN) for each model.

- **Visualization**

Bar plots for performance metrics saved as TIFF images for documentation.

- **Error handling**

Try-catch blocks for robust model training and evaluation.

- **Simulation tool**

MATLAB R2020b was used for all simulations, leveraging its robust computational and visualization capabilities.

In conclusion, the performance evaluation provides a robust framework for assessing the effectiveness of the proposed machine learning-based approach in analysing STM profiling datasets. By utilizing a controlled experimental setup, various machine learning models, and key evaluation metrics, the study ensures a comprehensive understanding of the performance of nested and non-nested STM configurations. The results from this evaluation serve as a solid foundation for further enhancements and improved performance in STM-based systems.

5 Results and discussion

This section evaluates the performance of four machine learning models—Naive Bayes, Decision Tree, KNN, and Random Forest—applied to STM profiling datasets, which are represented by 20 profiling metrics capturing key STM operations. The Performance Matrix presents a comparison of key evaluation metrics, including accuracy, precision, recall, and F1 score, based on these profiling metrics. This matrix provides insights into the effectiveness of each model in handling STM data. Key Observations further highlight the strengths of each model, considering their suitability for various application scenarios and their ability to evaluate STM operations. The results for each model, as summarized in the Performance Matrix (Table. 2), demonstrate both the advantages and limitations inherent in each approach. Visualizations of the evaluation metrics, including accuracy (Fig. 2), precision (Fig. 3), recall (Fig. 4), and F1 score (Fig. 5), offer a visual comparison that helps to further elucidate the models' performance.

5.1 Performance matrix

The performance of four machine learning models—Naive Bayes, Decision Tree, KNN, and Random Forest—applied to STM profiling datasets was evaluated using multiple metrics, such as Accuracy, Precision, Recall, and F1-Score. These metrics were calculated for various CV_Metrics. In this study, CV_Metrics refers to the set of profiling variables extracted from STM heap memory during runtime, serving as classification features in the machine learning models. These include Total_IO (total I/O operations), MVAR (mutable variables), TextEncoding, Handle, Buffer, and others—each representing specific memory structures and transactional behaviors within the STM system. Analysis of these metrics revealed significant efficiency advantages in nested STM configurations. For example, 48 bytes of memory usage in nested STM versus 96 bytes in non-nested STM, indicating more efficient buffer encoding/decoding. Additionally, nested STM does not use memory for BufferList, whereas non-nested STM consumes 24 bytes. For Buffer, nested STM requires 168 bytes, significantly less than the 280 bytes used by non-nested STM. The Newline metric shows equal memory usage of 24 bytes for both configurations. The Maybe datatype shows more efficient handling in nested STM, using 40 bytes compared to 64 bytes in non-nested STM. The MUT_VAR_CLEAN metric reveals 128 bytes of usage in nested STM versus 208 bytes in non-nested STM, indicating better management of mutable variables. For Handle__, nested STM uses 136 bytes compared to 272 bytes in non-nested STM, highlighting superior resource management in nested transactions. Non-nested STM consumes 48 bytes for the DEAD_WEAK metric, while nested STM does not use memory for this datatype, possibly avoiding certain weak references. Both configurations use 96 bytes for the WEAK metric, indicating identical memory usage for weak references. The ARR_WORDS metric shows a significant difference, with nested STM using 36,816 bytes compared to 61,360 bytes for non-nested STM, reflecting superior efficiency in handling arrays. In Total, nested STM consumes 38,792 bytes, while non-nested STM uses 63,080 bytes, highlighting a substantial reduction in the memory footprint for nested STM.

The following section presents the detailed evaluation results, which are summarized in Table 2. This table provides a comparative analysis of the models' performance based on the four-evaluation metrics. It includes percentage values for Accuracy, Precision, Recall, and F1-Score for each model and metric combination. Additionally, confusion matrices for each model, shown in Fig. 6 (Naive Bayes), Fig. 7 (Decision Tree), Fig. 8 (KNN), and Fig. 9 (Random Forest), further highlight their predictive performance under real-time conditions, offering deeper insights into their capabilities.

- **Naive Bayes**

Achieved consistent results with 100% accuracy, precision, recall, and F1-Score across all metrics, demonstrating its reliability for the given dataset.

- **Decision Tree**

Exhibited variability, achieving only 50% accuracy and precision while maintaining 100% recall, resulting in an F1-Score of 66.67%. The lower performance of the Decision Tree model can be attributed to its tendency to overfit small datasets with limited feature interactions. In this case, the heap profiling features exhibit complex dependencies that are better captured by ensemble methods (Random Forest) or distance-based models (KNN), while Decision Tree struggles with generalization.

- **KNN and Random Forest**

Both models achieved perfect scores (100%) for all evaluation metrics, indicating their strong performance and suitability for this dataset.

Table 2: Performance matrix for machine learning models

CV_Metric	Model	Accuracy	Precision	Recall	F1_Score
Total_IO	Naive Bayes	100	100	100	100
Total_IO	Decision Tree	50	50	100	66.66666667
Total_IO	KNN	100	100	100	100
Total_IO	Random Forest	100	100	100	100
String	Naive Bayes	100	100	100	100

String	Decision Tree	50	50	100	66.66666667
String	KNN	100	100	100	100
String	Random Forest	100	100	100	100
TextEncoding	Naive Bayes	100	100	100	100
TextEncoding	Decision Tree	50	50	100	66.66666667
TextEncoding	KNN	100	100	100	100
TextEncoding	Random Forest	100	100	100	100
MVAR	Naive Bayes	100	100	100	100
MVAR	Decision Tree	50	50	100	66.66666667
MVAR	KNN	100	100	100	100
MVAR	Random Forest	100	100	100	100
Handle	Naive Bayes	100	100	100	100
Handle	Decision Tree	50	50	100	66.66666667
Handle	KNN	100	100	100	100
Handle	Random Forest	100	100	100	100
Word32	Naive Bayes	100	100	100	100
Word32	Decision Tree	50	50	100	66.66666667
Word32	KNN	100	100	100	100
Word32	Random Forest	100	100	100	100
PairSharp	Naive Bayes	100	100	100	100
PairSharp	Decision Tree	50	50	100	66.66666667
PairSharp	KNN	100	100	100	100
PairSharp	Random Forest	100	100	100	100
Pair	Naive Bayes	100	100	100	100

Pair	Decision Tree	50	50	100	66.66666667
Pair	KNN	100	100	100	100
Pair	Random Forest	100	100	100	100
ForeignPtrContents	Naive Bayes	100	100	100	100
ForeignPtrContents	Decision Tree	50	50	100	66.66666667
ForeignPtrContents	KNN	100	100	100	100
ForeignPtrContents	Random Forest	100	100	100	100
BufferCodec	Naive Bayes	100	100	100	100
BufferCodec	Decision Tree	50	50	100	66.66666667
BufferCodec	KNN	100	100	100	100
BufferCodec	Random Forest	100	100	100	100
BufferList	Naive Bayes	100	100	100	100
BufferList	Decision Tree	50	50	100	66.66666667
BufferList	KNN	100	100	100	100
BufferList	Random Forest	100	100	100	100
Buffer	Naive Bayes	100	100	100	100
Buffer	Decision Tree	50	50	100	66.66666667
Buffer	KNN	100	100	100	100
Buffer	Random Forest	100	100	100	100
Newline	Naive Bayes	100	100	100	100
Newline	Decision Tree	50	50	100	66.66666667
Newline	KNN	100	100	100	100
Newline	Random Forest	100	100	100	100
Maybe	Naive Bayes	100	100	100	100

Maybe	Decision Tree	50	50	100	66.66666667
Maybe	KNN	100	100	100	100
Maybe	Random Forest	100	100	100	100
MUT_VAR_CLEAN	Naive Bayes	100	100	100	100
MUT_VAR_CLEAN	Decision Tree	50	50	100	66.66666667
MUT_VAR_CLEAN	KNN	100	100	100	100
MUT_VAR_CLEAN	Random Forest	100	100	100	100
Handle__	Naive Bayes	100	100	100	100
Handle__	Decision Tree	50	50	100	66.66666667
Handle__	KNN	100	100	100	100
Handle__	Random Forest	100	100	100	100
DEAD_WEAK	Naive Bayes	100	100	100	100
DEAD_WEAK	Decision Tree	50	50	100	66.66666667
DEAD_WEAK	KNN	100	100	100	100
DEAD_WEAK	Random Forest	100	100	100	100
WEAK	Naive Bayes	100	100	100	100
WEAK	Decision Tree	50	50	100	66.66666667
WEAK	KNN	100	100	100	100
WEAK	Random Forest	100	100	100	100
ARR_WORDS	Naive Bayes	100	100	100	100
ARR_WORDS	Decision Tree	50	50	100	66.66666667
ARR_WORDS	KNN	100	100	100	100
ARR_WORDS	Random Forest	100	100	100	100
Total	Naive Bayes	100	100	100	100

Total	Decision Tree	50	50	100	66.66666667
Total	KNN	100	100	100	100
Total	Random Forest	100	100	100	100

5.2 Key observations

This section provides an overview of the key observations from the evaluation of machine learning models applied to STM profiling datasets. It highlights trends in model performance, generalization across various metrics, and comparative strengths. Additionally, it discusses the implications for model selection, emphasizing the importance of choosing the right model based on the specific requirements of the task and the characteristics of the dataset.

- **Model performance trends**

o Naive Bayes, KNN, and Random Forest consistently outperformed the Decision Tree model across all metrics and CV_Metrics categories.

o The Decision Tree model struggled with accuracy and precision, possibly due to overfitting or sensitivity to the dataset's characteristics.

- **Generalization Across CV_Metrics**

o Models such as KNN and Random Forest demonstrated robust generalization, achieving 100% performance for diverse CV_Metrics, including Buffer, Handle, ARR_WORDS, and MUT_VAR_CLEAN.

o Decision Tree's lower performance was uniform across all CV_Metrics, suggesting a limitation in its ability to handle the dataset's complexity.

- **Comparative strengths**

o While Naive Bayes achieved perfect scores, it may have benefitted from the dataset's simplicity or lack of noise, which favoured its probabilistic approach.

o Random Forest and KNN showed superior adaptability, making them ideal for applications requiring high accuracy and consistency.

- **Implications for model selection**

o For scenarios demanding high precision and robustness, Random Forest and KNN are clear choices.

o Naive Bayes may be preferred for lightweight or resource-constrained environments due to its simplicity and computational efficiency.

o Decision Tree may require optimization techniques, such as pruning or hyperparameter tuning, to improve its performance.

In conclusion, the evaluation of machine learning models applied to STM profiling datasets reveals that Random Forest and KNN are the most reliable and robust models, consistently outperforming other models across all metrics. Naive Bayes, while efficient, may be more appropriate for simpler, resource-constrained environments. The Decision Tree model, although useful, exhibited limitations and highlighted the need for further refinement to handle complex datasets more effectively. This analysis demonstrates the importance of selecting the right machine learning model based on the dataset's characteristics and the specific application requirements. Future research could explore the effects of parameter tuning and dataset variations to further enhance the performance of these models for STM profiling tasks.

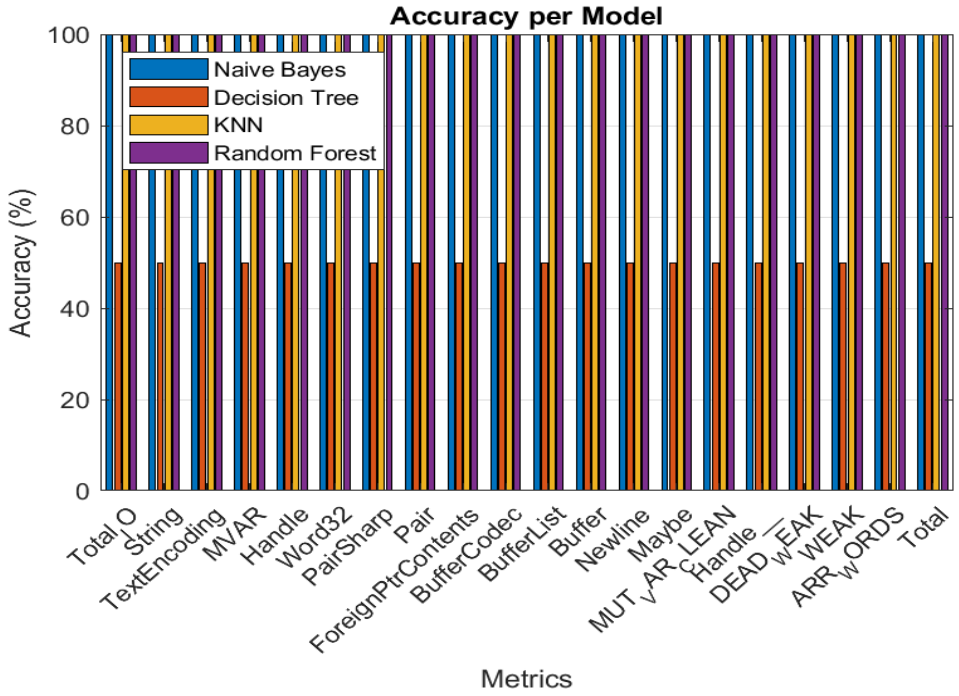

Figure 2: Accuracy comparison of machine learning models

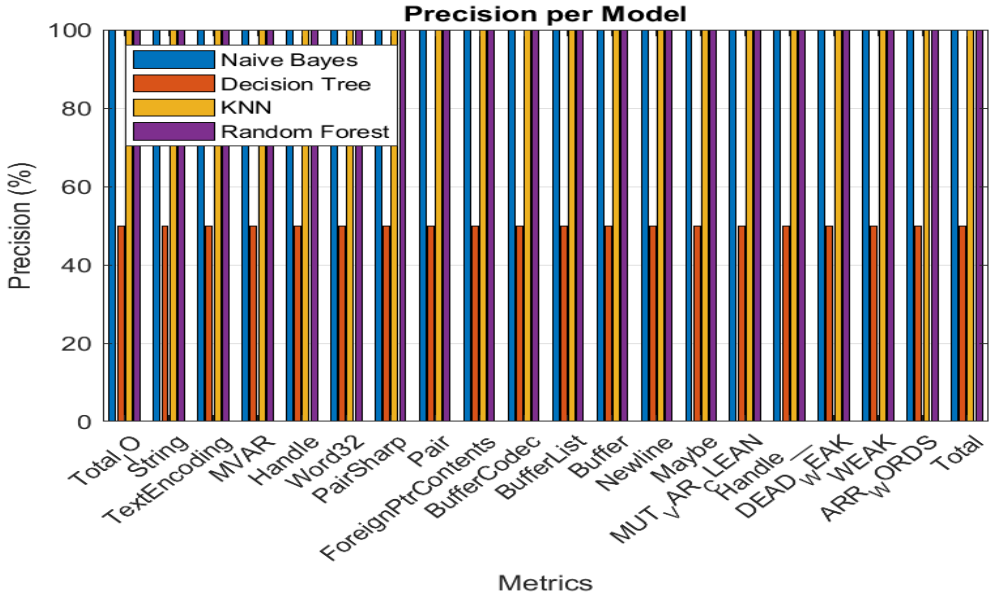

Figure 3: Precision comparison of machine learning models

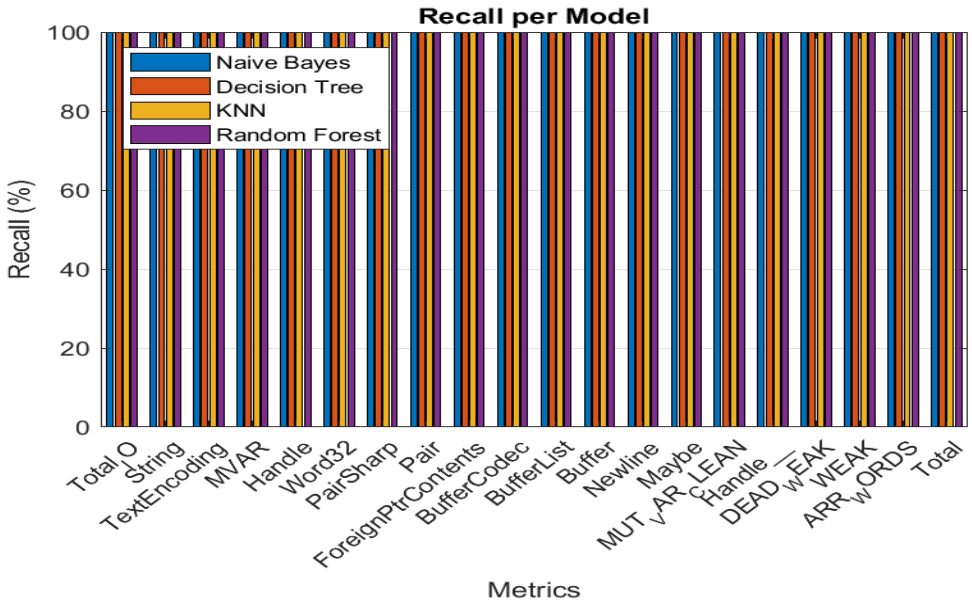

Figure 4: Recall comparison of machine learning models

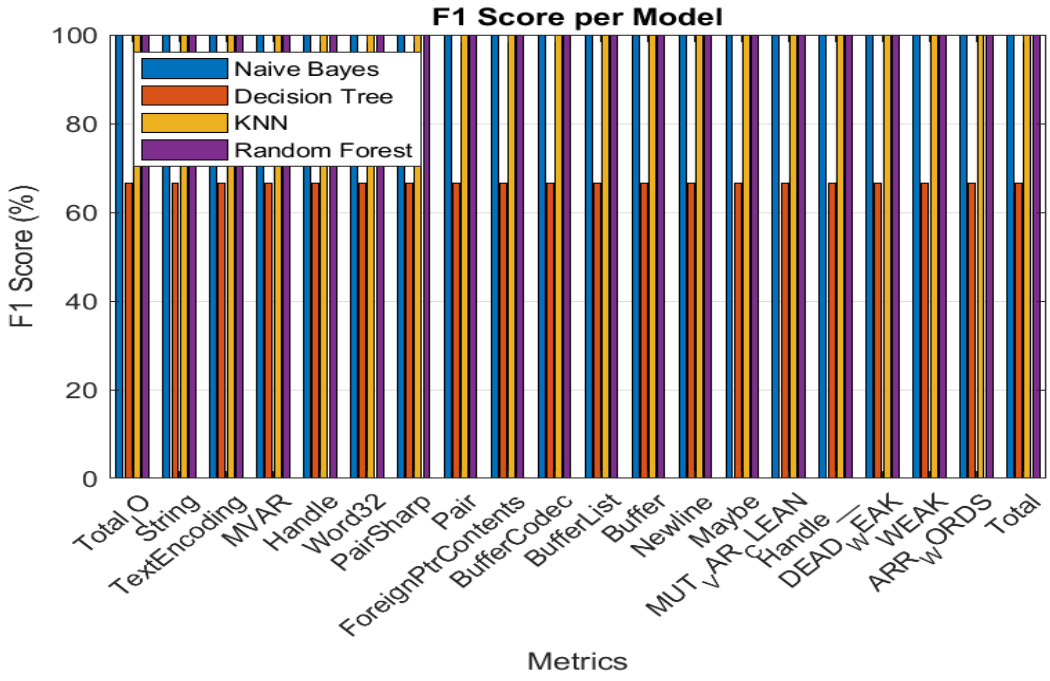

Figure 5: F1-Score comparison of machine learning models

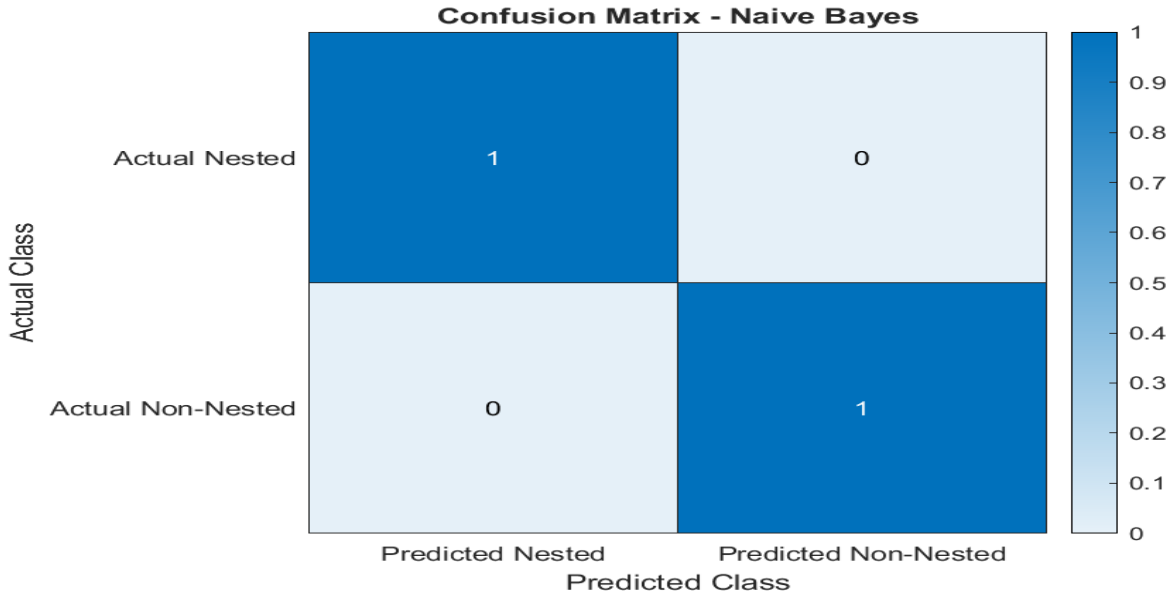

Figure 6: Confusion Matrix for Naive Bayes

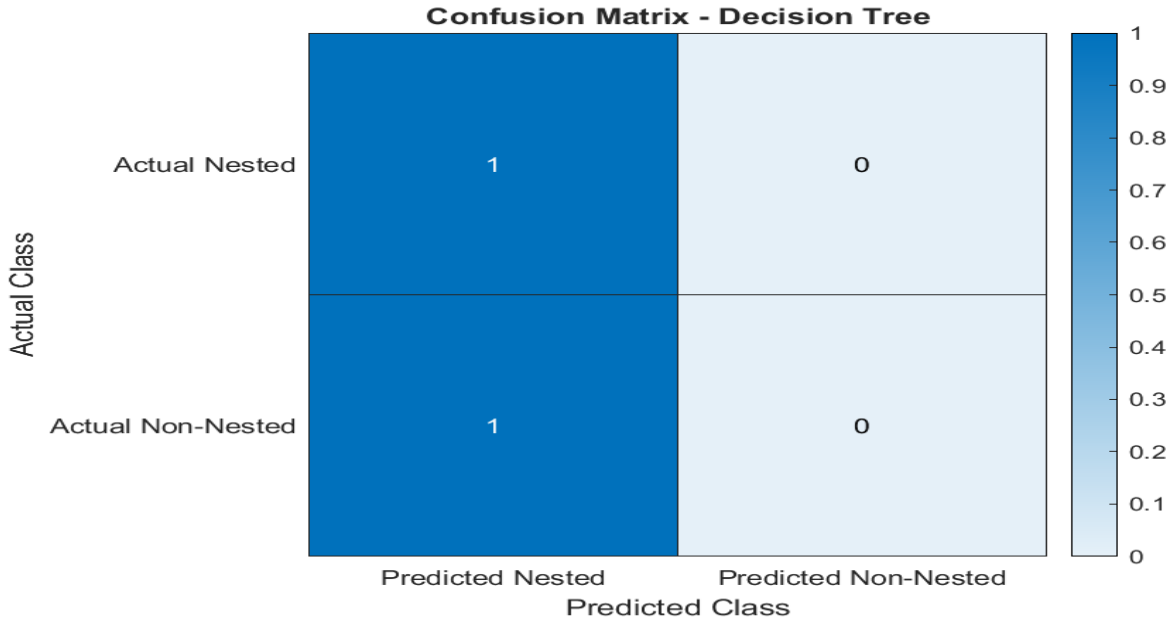

Figure 7: Confusion Matrix for Decision Tree

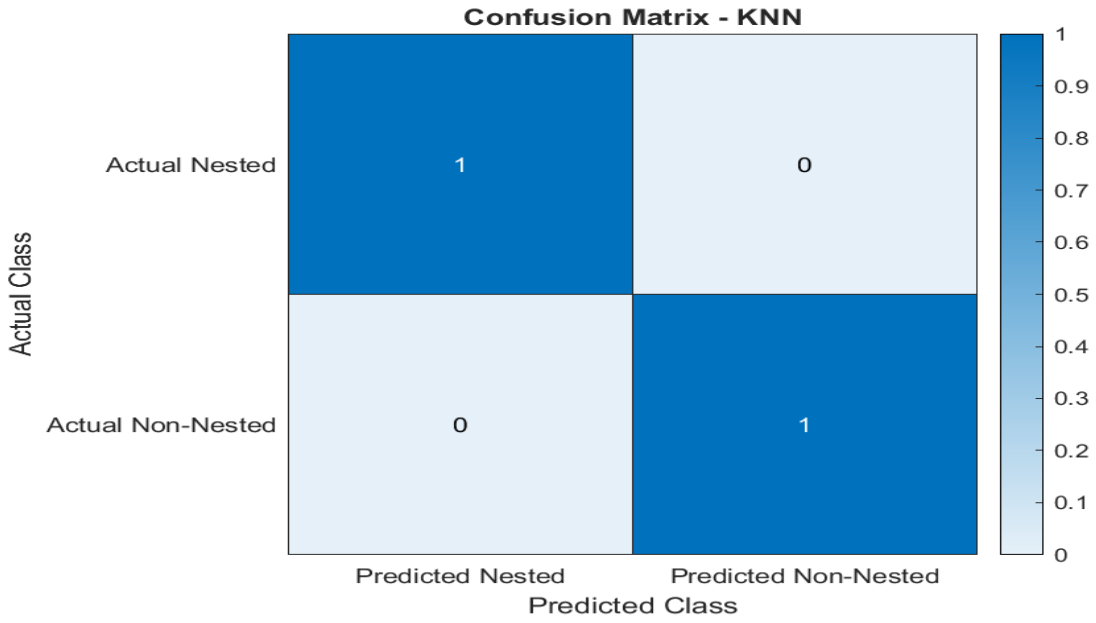

Figure 8: Confusion Matrix for KNN

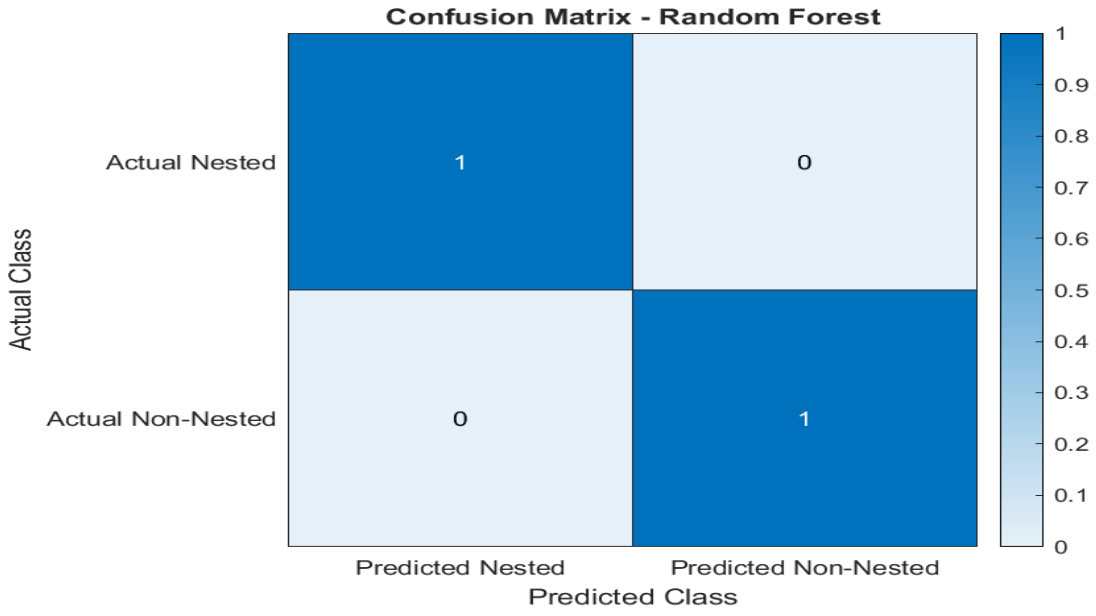

Figure 9: Confusion Matrix for Random Forest

6 Discussion

This section discusses the effectiveness of the ML-based STM analysis framework, highlights high classification accuracy, and acknowledges limitations such as dataset size and lack of cross-validation. It also outlines the framework's scalability and potential for future optimization.

Our machine learning-based framework provides a systematic and automated alternative to traditional STM performance analysis methods, which often focus on isolated parameters such as transaction granularity, update policy, or nesting models and rely heavily on manual profiling or limited benchmarks. In contrast, our approach leverages 20 profiling metrics derived from detailed heap profiling of nested and non-nested STM configurations.

This allows us to capture complex memory allocation patterns and runtime behaviour that are difficult to model with conventional approaches. The results demonstrate that Naive Bayes, KNN, and Random Forest models achieve 100% accuracy, precision, recall, and F1-score in classifying STM configurations—substantially improving upon manual or single-metric analyses. By using heap profiling data, our framework provides deeper insights into memory consumption and potential bottlenecks, which are critical factors in STM system performance and scalability. Moreover, the framework is highly scalable and can be applied across diverse STM configurations and workloads, reducing the time and effort required for performance analysis While the current work focuses on classification, the proposed framework provides a strong foundation for future predictive optimization and real-time adaptive STM tuning. However, several limitations must be acknowledged. The dataset used for training and evaluation is relatively small and synthetically augmented, which may restrict the model's exposure to diverse and realistic STM workload scenarios. Consequently, the excellent performance observed may not fully generalize to broader STM environments. Additionally, the current study lacks cross-validation and statistical variance reporting, which are essential for assessing model robustness. Future research will address these limitations through larger, real-world datasets, live system profiling, and more rigorous statistical validation.

7 Conclusion

This section concludes the evaluation of machine learning models for STM systems, emphasizing model selection based on dataset needs, application requirements, and the importance of optimization for scalability and efficiency.

This study presents a machine learning-based framework for the performance analysis of nested and non-nested Software Transactional Memory (STM) configurations. By leveraging models such as Naive Bayes, Decision Tree, K-Nearest Neighbours (KNN), and Random Forest, the research provides valuable insights into their strengths, limitations, and suitability for optimizing concurrency control and transaction management in STM. The evaluation reveals that machine learning models can play a pivotal role in addressing key STM challenges, including scalability, conflict resolution, and efficient resource utilization. Naive Bayes showed excellent classification performance; however, further work is needed to assess reliability in terms of robustness and stability across diverse STM workloads. Its ability to process datasets with minimal computational overhead positions it as an excellent choice for lightweight and real-time applications. However, the model's reliance on the assumption of feature independence may restrict its effectiveness in scenarios where feature interactions are highly complex. Random Forest, an ensemble-based learning method, also emerges as a top performer, excelling in accuracy and precision. Its robustness against overfitting and capacity to handle diverse datasets make it a highly adaptable choice for STM systems. The ensemble nature of Random Forest, which aggregates predictions from multiple decision trees, enhances its ability to generalize and maintain reliability. Despite its high performance, the computational demands of Random Forest may pose challenges for resource-constrained environments, necessitating trade-offs between performance and computational efficiency. KNN, a non-parametric and instance-based model, exhibits significant adaptability in dynamic STM environments. Its simplicity and ability to capture the underlying data structure contribute to its strong performance in scenarios requiring quick adaptation to changes. However, the efficiency of KNN is contingent upon the choice of distance metrics and the number of neighbours, which require careful tuning to improve performance for larger datasets and real-time applications. In contrast, the Decision Tree model demonstrates notable limitations in handling datasets with high variance or intricate feature relationships. Its comparatively lower performance underscores the need for optimization through hyperparameter tuning or ensemble techniques like boosting to enhance its effectiveness. These findings emphasize that while Decision Tree can serve as a baseline model, it requires further refinement to compete with more advanced methods.

In conclusion, this study underscores the potential of machine learning models in enhancing STM systems, with Naive Bayes and Random Forest excelling in accuracy and precision, KNN in adaptability, and Decision Tree requiring optimization. Model selection tailored to dataset and application needs, along with continuous refinement, is crucial for addressing STM challenges like scalability, conflict resolution, and resource efficiency, paving the way for future advancements.

8 Future directions

This section outlines future work to improve the ML-based STM framework through dataset growth, model tuning, real-world testing, statistical validation, and advanced learning techniques to enhance adaptability, accuracy, and practical deployment.

The findings of this study—specifically the high classification accuracy achieved using Naive Bayes, KNN, and Random Forest on STM profiling data—demonstrate that machine learning can effectively distinguish between nested and non-nested STM configurations. Building on this foundation, the following future directions propose ways to extend the current classification framework toward predictive optimization, real-time adaptation, and integration with advanced STM concurrency mechanisms. These directions aim to evolve the work from performance evaluation to practical, deployable tools for intelligent STM decision support in real-world environments.

8.1 Dataset expansion

One of the critical aspects of improving machine learning models is training them on larger and more diverse datasets. Future studies can focus on curating datasets that mirror real-world STM workloads, including diverse transaction types, varying conflict scenarios, and workload intensities. Such datasets can enhance model generalization, ensuring their applicability across a broader range of STM applications and environments.

8.2 Hyperparameter optimization

The performance of machine learning models like K-Nearest Neighbours (KNN) and Decision Trees is highly sensitive to hyperparameters such as the number of neighbors, tree depth, and splitting criteria. Systematic tuning of these parameters through grid search, random search, or advanced techniques like Bayesian optimization can significantly improve their accuracy and adaptability. This optimization will ensure that models perform efficiently in both static and dynamic STM environments.

8.3 Algorithmic enhancements

Traditional machine learning models, while effective, may face limitations in handling large-scale and highly dynamic STM systems. Future research can focus on developing hybrid models that combine traditional approaches with advanced techniques like deep learning. For instance, integrating neural networks with ensemble methods like Random Forest or using recurrent neural networks (RNNs) for capturing temporal dependencies in transactional data can enhance scalability and prediction accuracy.

8.4 Real-world application testing

While the current study provides valuable theoretical insights, testing these models in live STM environments remains a critical next step. Future research should focus on deploying these models in operational STM systems to evaluate their real-time impact on key performance metrics such as throughput, latency, and conflict resolution efficiency. This will also help uncover practical challenges including computational overhead in embedded or constrained environments, limited data availability for model retraining, and the ongoing need for adaptive model maintenance as transaction patterns evolve.

8.5 Dynamic adaptation

STM systems often operate in dynamic environments where workloads and transaction patterns change rapidly. Future efforts can explore the development of machine learning models capable of self-adaptation. These models should automatically adjust to changing conditions without requiring manual retraining. Techniques like online learning and reinforcement learning can be particularly useful for this purpose.

8.6 Ensemble and Meta-Learning approaches

Ensemble methods, which combine multiple models to leverage their strengths, can offer improved accuracy and robustness. Future research can explore meta-learning strategies that enable models to learn from the strengths and weaknesses of different algorithms. Techniques such as boosting, bagging, and stacking can be employed to enhance predictive performance while maintaining resilience against noisy or incomplete data.

8.7 Exploration of unsupervised and semi-supervised learning

Given the challenges of obtaining labelled transactional data, future work can investigate the potential of unsupervised and semi-supervised learning approaches. Clustering methods or representation learning could identify patterns in transactions and conflicts without requiring extensive labelling, thereby reducing the dependency on annotated datasets.

8.8 Integration with advanced concurrency control protocols

Machine learning models can be integrated with advanced STM concurrency control protocols to more informed lock management and conflict resolution. By predicting potential conflicts and dynamically adjusting transaction priorities, such integration can enhance the system's overall efficiency and scalability.

8.9 Rigorous statistical validation

Future work will also focus on performing rigorous statistical validation of the proposed machine learning framework. This includes incorporating k-fold cross-validation and reporting variance and confidence intervals for all key performance metrics to ensure statistical reliability, model robustness, and generalizability across different STM configurations and datasets.

8.10 Feature importance analysis

Future work will include a detailed feature importance analysis to identify which STM profiling metrics contribute most to classification accuracy. Techniques such as Random Forest's Gini importance and permutation-based methods will be used to rank the impact of each feature. This analysis will improve interpretability, guide metric selection, and help in reducing model complexity while maintaining predictive performance.

In conclusion, the proposed machine learning framework offers a promising foundation for STM analysis, with future directions aimed at enhancing its accuracy, adaptability, and real-world applicability. By expanding datasets, refining algorithms, incorporating advanced

learning techniques, and ensuring rigorous validation, this work paves the way for intelligent, data-driven STM systems that can scale with the complexity of modern transactional workloads. These efforts will move the framework beyond classification toward fully optimized and deployable STM performance solutions.

References

[1] Q. &. Z. J. Li, "A Comparative Analysis of Extreme Gradient Boosting, Decision Tree, Support Vector Machines, and Random Forest Algorithm in Data Analysis of College Students' Psychological Health," Informatica (Slovenia), vol. 49, 2025. https://doi.org/10.31449/inf.v49i15.7004.

[2] I. a. P. O. Awoyelu, "Mathlab Implementation of Quantum Computation in Searching an Unstructured Database," Informatica (Slovenia), vol. 36, no. 3, pp. 249-254, 2012. https://www.researchgate.net/publication/292016488.

[3] M. H. a. J. E. B. Moss, "Transactional memory: architectural support for lock-free data structures," in Proceedings of the 20th annual international symposium on Computer architecture (ISCA '93)., May 1993.https://doi.org/10.1145/165123.165164.

[4] N. &. T. D. Shavit, "Software transactional memory," in Proceedings of the 14th Annual ACM Symposium on Principles of DistributedComputing, Ottawa, Can, 1995.https://doi.org/10.1145/224964.224987.

[5] A. F. Y. L. V. L. M. M. D. N. Peter Damron, "Hybrid transactional memory," in Proceedings of the 12th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2006, San Jose, CA, USA, October 21-25, 2006.https://doi.org/10.1145/1168857.1168900.

[6] J. E. B. Moss, "Nested Transactions: An Approach to Reliable Distributed Computing," Ph.D. Thesis, Technical Report MIT/LCS/TR-260, MIT Laboratory for Computer Science, Cambridge, MA, April 1981.https://publications.csail.mit.edu/lcs/pubs/pdf/MIT-LCS-TR-260.pdf.

[7] A. L. H. J. Eliot B. Moss, "Nested transactional memory: Model and architecture sketches," Science of Computer Programming, vol. 63, no. 2, pp. 186-201, 2006.https://doi.org/10.1016/j.scico.2006.05.010.

[8] N. C. J. Diegues, "Review of nesting in transactional memory," Tech. rep., Technical Report RT/1/2012, Instituto Superior Técnico/INESC-ID, 2012.https://scholar.tecnico.ulisboa.pt/records/14aedc27-91a7-48ec-8b8b-ff750b7934c6.

[9] G. A. Asi, "Performance Tradeoffs in Software Transactional Memory," Master Thesis Computer Science, School of Computing Blekinge Institute of Technology, No: MCS-2010-28, Sweden, May 2010.https://www.diva-portal.org/smash/get/diva2:833477/FULLTEXT01.pdf.

[10] S. Classen, "LibSTM: A fast and flexible STM Library," Master's Thesis, Laboratory for Software Technology, Swiss Federal Institute of Technology, ETH Zurich, Feb, 2008. https://library.ethz.ch.

[11] D. I. a. M. Raynal, "A Lock-Based STM Protocol That Satisfies Opacity and Progressiveness," in Proceedings of the 12th International Conference on Principles of Distributed Systems (OPODIS'08, 2008.https://doi.org/10.1007/978-3-540-92221-6_16.

[12] W. N. S. I. a. M. L. Scott, "Contention Management in Dynamic Software Transactional Memory," in Proceedings of the ACM PODC Workshop on Concurrency and Synchronization in Java Programs, Canada, July 2004.https://www.cs.rochester.edu/u/scott/papers/2004_CSJP_contention_mgmt.pdf.

[13] I. N. S. a. M. L. S. y, "Advanced contention management for dynamic software transactional memory," in Proceedings of the twenty-fourth annual ACM symposium on Principles of distributed computing, Las Vegas, NV, USA, 2005.https://doi.org/10.1145/1073814.1073861.

[14] e. a. R. Guerraoui, "Toward a theory of transactional contention managers," in Proceedings of the twenty-fourth annual ACM symposium on Principles of distributed computing, Las Vegas, NV, USA, 2005.https://doi.org/10.1145/1073814.1073863.

[15] T. H. a. S. Stipic, "Abstract nested transactions," in Second ACM SIGPLAN Workshop on Transactional Computing, 2007.https://www.cs.rochester.edu/meetings/TRANSACT07/papers/harris.pdf.

[16] M. &. L. V. &. M. M. &. S. W. Herlihy, "Software Transactional Memory for Dynamic-Sized Data Structures," in Proceedings of the Annual ACM Symposium on Principles of Distributed Computing, 2003.https://doi.org/10.1145/872035.872048.

[17] M. S. C. H. A. A. D. E. W. S. I. a. M. S. V. Marathe, "Lowering the overhead of Software Transactional Memory," in 1st ACM SIGPLAN Workshop on

Transactional Computing (TRANSACT '06), 2006.https://www.researchgate.net/publication/244434378.

[18] B. R. a. M. M. S. Alexandru Turcu, "On closed nesting in distributed transactional memory," in Seventh ACM SIGPLAN workshop on Transactional Computing, 2012.https://transact2012.cse.lehigh.edu/papers/turcu.pdf.

[19] A.-R. A.-T. R. H. C. C. M. a. B. H. B. Saha, "McRT-STM: a high-performance Software Transactional Memory system for a multi-core runtime," in SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'06), 2006.https://doi.org/10.1145/1122971.1123001.

[20] M. S. a. M. S. L. Dalessandro, "NOrec: Streamlining STM by abolishing ownership records," in Proceedings of the 15th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP '10), 2010.https://doi.org/10.1145/1693453.1693464.

[21] J. B. M. M. M. H. a. D. W. K. Moore, "LogTM: log-based transactional memory," in Proceedings of the 12th High-Performance Computer Architecture International Symposium (HPCA '06), 2006.https://doi.org/10.1109/HPCA.2006.1598134.

[22] J. B. K. E. M. L. Y. M. D. H. B. L. M. M. S. a. D. M. J. Moravan, "Supporting Nested Transactional Memory in LogTM," in 12th International Conference on Architectural Support for Programming Languages and Operating Systems in SIGPLAN Notices (Proceedings of the 2006 ASPLOS Conference), 2006.https://doi.org/10.1145/1168857.1168902.

[23] R. C. Ammlan Ghosh and Haskell, Implementing Software Transactional Memory using STM, vol. 396, Advanced Computing and Systems for Security, Springer AISC, 2016.https://doi.org/10.1007/978-81-322-2653-6_16, pp. 235-248.

[24] M. R. Y. a. M. F. Le, "Revisiting software transactional memory in Haskell," ACM SIGPLAN Notices, vol. 51, no. 12, pp. 105-113, 2016.https://doi.org/10.1145/3241625.2976020.

[25] A. Du Bois, "An Implementation of Composable Memory Transactions in Haskell," in Software Composition, SC 2011, Lecture Notes in Computer Science, Springer, Berlin, Heidelberg., 2011.https://doi.org/10.1007/978-3-642-22045-6_3.

[26] A. H. T. M. S. J. S. S. S. Discolo, "Lock Free Data Structures Using STM in Haskell," in Functional and Logic Programming, FLOPS, 2006.https://doi.org/10.1007/11737414_6.

[27] M. L. V. &. M. M. Herlihy, "A flexible framework for implementing software transactional memory," ACM SIGPLAN Notices, vol. 41, no. 10, pp. 253-262, 2006.https://doi.org/10.1145/1167515.1167495.

[28] S. M. S. P. J. a. M. H. T. Harris, "Composable memory transactions," in Proceedings of the Tenth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP '05, Chicago, IL, USA, 2005.https://doi.org/10.1145/1065944.1065952.

[29] A. G. a. S. F. S. Peyton Jones, "Concurrent Haskell," in 23rd ACM Symposium on Principles of Programming Languages (POPL'96), 1996.https://doi.org/10.1145/237721.237794.

[30] A. T. a. B. Ravindran, "On open nesting in distributed transactional memory," in 5th Annual International Systems and Storage Conference (SYSTOR) '12, 2012.https://doi.org/10.1145/2367589.2367601.

[31] V. S. M. A.-R. A.-T. A. L. H. R. L. H. J. E. B. M. S. a. T. S. Y. Ni, "Open nesting in software transactional memory," in PPoPP '07: Proceedings of the 12th ACM SIGPLAN symposium on Principles and Practice of Parallel Programming, ACM Press, New York, NY, USA, 2007.https://doi.org/10.1145/1229428.1229442.

[32] A. M. H. C. J. C. C. M. C. K. a. K. O. B. Carlstrom, "The ATOMOS Transactional Programming Language," in Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'06), 2006.https://doi.org/10.1145/1133981.1133983.

[33] N. B. C. K. a. K. O. W. Baek, "Implementing and evaluating nested parallel transactions in software transactional memory," in Proceedings of the 22nd ACM Symposium on Parallelism in Algorithms and Architectures, SPAA'10, Thira, Santorini, Greece, 2010.https://doi.org/10.1145/1810479.1810528.

[34] R. K. a. K. Vidyasankar, "HParSTM: A Hierarchy-based STM Protocol for Supporting Nested Parallelism," in 6th ACM SIGPLAN Workshop on Transactional Computing (TRANSACT '11), 2011.https://sss.cs.purdue.edu/projects/transact11/papers/Kumar.pdf.

[35] A. W. A.-R. A.-T. T. S. X. T. a. R. N. H. Volos, "NePaLTM: Design and Implementation of Nested

Parallelism for Transactional Memory Systems," in Proceedings of the 23rd European Conference on Object-Oriented Programming (ECOOP '09), 2009.https://doi.org/10.1007/978-3-642-03013-0_7.

[36] J. T. F. a. J. S. K. Agrawal, "Nested parallelism in transactional memory," in Proceedings of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP '08), 2008.https://doi.org/10.1145/1345206.1345232.

[37] A. D. P. F. R. G. a. M. K. J. Barreto, "Leveraging parallel nesting in transactional memory," in Proceedings of the 15th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP '10), 2010.https://doi.org/10.1145/1837853.1693466.

[38] H. R. a. E. Witchel, "The xfork in the road to coordinated sibling transactions," in 4th ACM SIGPLAN Workshop on Transactional Computing (TRANSACT '09), 2009.https://www.researchgate.net/publication/228400493.

[39] D. &. D. S. P. &. C. B. &. Q. F. Rughetti, "Machine Learning-Based Self-Adjusting Concurrency in Software Transactional Memory Systems," in IEEE 20th International Symposium on Modeling, Analysis and Simula, 2012.https://doi.org/10.1109/MASCOTS.2012.40.

[40] P. D. S. B. C. F. Q. Diego Rughetti, "Machine learning-based thread-parallelism regulation in software transactional memory," Journal of Parallel and Distributed Computing, vol. 109, pp. 208-229, 2017.https://doi.org/10.1016/j.jpdc.2017.06.001.