Adaptive Machine Learning-Driven Enhancements for TREEPROMPT and TREEHP2PL in Distributed Real-Time Database Systems

Meenu
Department of CSE, M. M. M. U. T., Gorakhpur, India
E-mail: meenu@yahoo.co.in

Keywords: deadlock detection, distributed real-time database systems, machine learning, nested transactions, scalability, TREEPROMPT

Received: February 20, 2025

Managing nested transactions in distributed real-time database systems (DRTDBS) is essential for ensuring consistency, scalability, and efficiency in critical domains such as financial systems and industrial automation. While traditional protocols like TREEPROMPT resolves inter-transaction deadlocks with speculative execution and priority inheritance and TREEHP2PL detects intra-transaction deadlocks using Wait-For Graphs and resolves them by aborting low-priority or short-execution subtransactions, their static configurations limit adaptability to dynamic workloads. This study enhances these protocols by integrating machine learning (ML) classification models to improve performance through predictive success analysis. Four ML models—Naive Bayes, Decision Tree, K-Nearest Neighbours (KNN), and Random Forest—are evaluated using a dataset of 500 transactions per simulation run, with ten independent executions to ensure statistical reliability. The experimental setup evaluates classifier performance using accuracy, precision, recall, F1-score, and computational efficiency. The results show that the Naive Bayes model achieved an accuracy of 98.5%, a precision of 98.2%, a recall of 98.7%, and an F1-score of 98.4%. The Decision Tree model performed similarly, with an accuracy of 97.8%, a precision of 97.5%, a recall of 97.9%, and an F1-score of 97.6%. In contrast, the K-Nearest Neighbours (KNN) model exhibited lower performance, with an accuracy of 44.2%, a precision of 43.8%, a recall of 44.5%, and an F1-score of 44.1%. Similarly, the Random Forest model achieved an accuracy of 45.6%, a precision of 45.3%, a recall of 45.9%, and an F1-score of 45.5%. Compared to traditional heuristic-based approaches, ML-enhanced protocols significantly improve transaction success rates by minimizing deadlock occurrences and optimizing resource utilization. Moreover, ML integration enhances system throughput and reduces transaction latency, demonstrating notable computational efficiency gains. These findings validate the effectiveness of ML-driven optimizations in enhancing protocol scalability and adaptability. Future research will focus on refining underperforming models, incorporating reinforcement learning techniques, and testing on larger datasets to further optimize real-time transaction management in DRTDBS environments.

Povzetek: Analizirana je vključitev strojnega učenja v protokola TREEPROMPT in TREEHP2PL za upravljanje gnezdenih transakcij v porazdeljenih realnočasovnih podatkovnih sistemih. Naive Bayes in Decision Tree znatno izboljšata uspešnost, skalabilnost in zmanjšata mrtve zanke.

1 Introduction

This section outlines the significance of managing nested transactions in Distributed Real-Time Database Systems (DRTDBS), particularly in critical applications like financial systems, telecommunications, and industrial automation. It introduces TREEPROMPT and TREEHP2PL protocols. TREEPROMPT resolves inter-transaction deadlocks with speculative execution and priority inheritance, while TREEHP2PL detects intra-transaction deadlocks using Wait-for Graphs and resolves them by aborting low-priority or short-execution subtransactions.

However, it highlights the limitations of these static protocols in adapting to dynamic workloads. The integration of machine learning (ML) is proposed to enhance protocol performance through predictive and adaptive mechanisms. The section also establishes the objectives of the paper, including evaluating ML models for transaction success prediction, deadlock detection, and anomaly resolution, and optimizing the TREEPROMPT and TREEHP2PL protocols to achieve scalability and efficiency. The introduction section also provides an overview of the paper's structure, covering the management of nested transactions in DRTDBS, the role of TREEPROMPT and TREEHP2PL protocols, the integration of machine learning, the evaluation of ML models, experimental setup, result analysis, and future research directions.

Distributed Real-Time Database Systems (DRTDBS) have become essential in applications where consistent, reliable, and timely transaction management is critical [1]. Examples include financial services, telecommunications, industrial automation, and aerospace systems. These systems must process transactions within strict timing constraints while maintaining the consistency and integrity of the underlying data. However, the complexity of distributed systems, combined with the hierarchical structure of nested transactions, poses significant challenges for transaction management.

1.1 Nested transactions

Nested transactions, first introduced by Moss [2] ,enable a parent transaction to spawn multiple child transactions, which can either succeed or fail independently. This structure improves modularity and fault tolerance by allowing partial rollbacks and isolated execution of subtransactions. However, managing such transactions in real-time distributed environments introduces several challenges, including:

1.1.1 Deadlock detection and resolution

Deadlocks occur when transactions are unable to proceed because of circular dependencies on resources. In nested transactions, the hierarchical structure complicates deadlock detection and resolution, especially under stringent timing constraints [3].

1.1.2 Concurrency control

Achieving global serializability while managing concurrent access to distributed resources is complex, particularly when multiple transactions compete for shared data [2].

1.1.3 Scalability

With an increasing number of transactions and subtransactions, ensuring system performance and stability becomes more challenging.

Protocols such as TREEPROMPT and TREEHP2PL were developed to address these issues. TREEPROMPT, a commit protocol, extends the PROMPT [4] and SPROMPT [5] [6] protocols by incorporating speculative execution to reduce cascading aborts in nested transactions. It ensures atomicity by committing or aborting all subtransactions together, which improves real-time performance. TREEHP2PL, on the other hand, is a concurrency control protocol that prioritizes high-priority transactions and reduces deadlocks through a hierarchical locking mechanism.

While TREEPROMPT and TREEHP2PL have shown improvements in managing nested transactions, their scalability and efficiency under varying workloads remain areas for enhancement. Furthermore, the dynamic nature of distributed systems and the unpredictability of workloads necessitate adaptive solutions that can optimize protocol performance in real time.

1.2 Enhancing transactions with ML

Machine learning (ML) offers promising solutions to these challenges by enabling predictive and adaptive decision-making. ML can enhance transaction management in the following ways:

1.2.1 Deadlock detection

By analysing transaction graphs, ML models can identify potential deadlocks early and suggest resolutions.

1.2.2 Success prediction

Classification models can predict the outcome of a transaction (success or failure) based on workload and system parameters, enabling data-driven decisions to improve protocol performance.

This study explores the integration of ML techniques into TREEPROMPT and TREEHP2PL protocols to address these challenges. Four ML models—Naive Bayes, Decision Tree, K-Nearest Neighbours (KNN), and Random Forest—are applied to evaluate transaction success ratios and protocol performance across scenarios with and without deadlocks. The objective is to identify the most effective model for optimizing transaction management in DRTDBS and to propose a framework for enhancing protocol scalability and reliability.

1.3 Key contributions of the study

The contributions of this study include:

1.3.1 Comprehensive evaluation

A detailed analysis of the performance of four ML models across different scenarios, highlighting strengths and limitations.

1.3.2 Integration with protocols

Application of ML models to enhance TREEPROMPT and TREEHP2PL protocols by improving deadlock detection and success prediction.

1.3.3 Actionable insights

Recommendations for model selection, dataset preprocessing, and protocol tuning to optimize transaction management in DRTDBS.

By leveraging ML, this research aims to bridge the gap between static protocol design and dynamic workload demands in distributed systems, offering a scalable and efficient approach to managing nested transactions.

This paper investigates the integration of Machine Learning (ML) into Distributed Real-Time Database Systems (DRTDBS) to optimize nested transaction protocols such as TREEPROMPT and TREEHP2PL, focusing on enhancing scalability, deadlock detection, and performance. Section 2 highlights key challenges in managing nested transactions, such as scalability and deadlock detection. It includes Table 1 which provides a comparative overview of the state-of-the-art (SOTA) approaches, summarizing their strengths, weaknesses, and the reasons they fall short in optimizing nested transactions in real-time environments. It explores the role of machine learning in distributed systems, emphasizing its potential for prediction and optimization. The section also outlines the limitations of existing approaches and identifies a research gap, motivating the integration of ML with TREEPROMPT and TREEHP2PL to enhance performance and adaptability. Section 3 outlines the experimental framework for integrating ML into TREEPROMPT and TREEHP2PL protocols, detailing dataset generation with features like interarrival times and success ratios. Four ML models—Naive Bayes, Decision Tree, KNN, and Random Forest—are evaluated using metrics such as accuracy and F1-score [7].Figure 1 illustrates the systematic approach, highlighting protocol integration, ML-driven optimization, and performance evaluation to enhance scalability and reliability in DRTDBS. Section 4 describes the simulation setup for evaluating ML models on TREEPROMPT and SPROMPT protocols under scenarios with and without deadlocks. It covers data partitioning, metric computation, and confusion matrix analysis. Visualization techniques, including bar charts and heatmaps, are used to provide insights into model performance, highlighting their strengths, weaknesses, and effectiveness in optimizing the protocols. Section 5 analyses the results of evaluating ML models for optimizing TREEPROMPT and TREEHP2PL protocols. It includes a performance matrix (Table 2) summarizing metrics and key observations on model strengths and weaknesses. Visual comparisons of metrics are presented in Figure 2 (Accuracy), Figure 3 (Precision), Figure 4 (Recall), and Figure 5 (F1-Score), highlighting performance trends across models and scenarios. Section 6 highlights the successful integration of ML into TREEPROMPT and TREEHP2PL, showcasing the superior performance of Naive Bayes and Decision Tree in optimizing success ratios, reducing deadlock resolution times, and enhancing scalability. It emphasizes the practical impact of bridging traditional protocols with ML-driven enhancements for advancing adaptive transaction management systems. Finally, Section 7 suggests future research on optimizing KNN and Random Forest through hyperparameter tuning, expanding datasets with diverse scenarios, and testing ML-enhanced protocols in real-world DRTDBS. It also proposes using advanced techniques like reinforcement learning and deep learning for adaptive transaction management and emphasizes security, fault tolerance, and collaborative optimization for resilient systems.

In conclusion, this paper demonstrates the successful integration of machine learning into TREEPROMPT and TREEHP2PL protocols, enhancing transaction success, scalability, and deadlock resolution in DRTDBS. Naive Bayes and Decision Tree emerged as the most effective models, optimizing protocol performance across scenarios. This study addresses the limitations of traditional protocol designs by incorporating adaptive ML-optimizations. Future research will focus on optimizing lower-performing models, exploring advanced techniques like reinforcement learning, and validating these approaches in real-world systems to further improve resilience and scalability.

2 Literature review

This section covers the challenges in managing nested transactions in Distributed Real-Time Database Systems (DRTDBS), critiquing traditional protocols like 2PL and 2PC for inefficiencies. It explores the potential of machine learning (ML) models in optimizing transaction management, focusing on deadlock detection, anomaly detection, and performance prediction. The review identifies the limitations of current approaches and outlines the research gap in integrating ML with protocols like TREEHP2PL and TREEPROMPT to improve system performance and scalability.

2.1 Challenges in managing nested transactions

Managing nested transactions in distributed real-time systems [1] presents challenges, with traditional protocols like 2PL and 2PC facing inefficiencies due to blocking delays in real-time environments [8]**.** PROMPT improves performance by allowing access to modified data before commit but struggles in nested environments [4]. SPROMPT introduces speculative execution to reduce cascading aborts but faces consistency issues [5] [6]. TREEHP2PL integrates priority-based conflict resolution with 2PL for efficient nested transaction management, while TREEPROMPT extends PROMPT and SPROMPT with speculative execution to reduce cascading aborts and improve success ratios. Concurrency and commit protocols are essential for maintaining consistency during simultaneous transaction execution. 2PL-NT integrates two-phase locking for nested transactions but struggles with scalability [2]. Multi-granularity locking increases precision but adds complexity [9] [10], while generalized locking enhances reliability but incurs computational overhead [11]. Extensions of multi-granularity locking improve nested transaction handling but face compatibility challenges with legacy systems [12]. Serialization graphs improve conflict resolution but are computationally intensive [13], and formal concurrency control algorithms are difficult to implement [14]. Nested transaction models in Knowledge Base Systems (KBMS) face compatibility issues [15], and Gifford's Quorum Consensus improves fault tolerance but complicates consistency maintenance [16]. Algorithms for B-trees [17] and linear hash structures [18] address niche requirements, while multi-version timestamp concurrency control increases storage demands [19]. 2PL-NT-HP enhances conflict resolution but adds overhead [5] [6], and S-PROMPT mitigates cascading aborts with speculative execution but introduces performance trade-offs [5] [6] .Managing nested transactions requires addressing intra- and inter-transaction parallelism through advanced concurrency control, isolation levels, and scheduling policies [20]. Subtransaction prioritization introduces delays due to data sharing, necessitating priority assignment policies [6] [21] [22]. Deadlock detection must consider waits-for-lock and waits-for-commit relations [3], while existing concurrency and commit protocols designed for flat transactions are inadequate for nested models [5] [6] [23]. Priority inversion is addressed with Priority Inheritance and Priority Abort protocols [24], while ensuring N-ACID properties maintains database coherence [25]. Recovery mechanisms must handle finer control structures [26] , and achieving global serializability in distributed real-time nested transactions remains a significant challenge [2] . Additionally, optimizing parameters like leaves and levels is critical for enhancing performance [27]. Deadlocks in DRTDBS, classified as intra-transaction (within a transaction tree) and inter-transaction (across trees), degrade performance. TREEPROMPT resolves inter-transaction deadlocks with speculative execution and priority inheritance [28], while TREEHP2PL detects intra-transaction deadlocks using Wait-For Graphs and resolves them by aborting low-priority or short-execution subtransactions [29] Simulations show improved success ratios and reliability, demonstrating the effectiveness of these protocols.

Existing nested transaction protocols in DRTDBS like 2PL, 2PC, PROMPT, SPROMPT, and TREEHP2PL improve concurrency but struggle with scalability, deadlocks, and adaptability. The table 1 compares these state-of-the-art (SOTA) approaches, highlighting their strengths, weaknesses, and limitations in optimizing real-time nested transactions.

Table 1: Comparison of SOTA approaches for nested transaction protocols in DRTDBS

Approach	Strengths	Weaknesses	Key Results	Why Insufficient?
2PL (Two-Phase Locking)	Ensures strict serializability	Leads to transaction blocking and deadlocks	Moderate success ratios, high consistency	Struggles with scalability and real-time constraints
2PC (Two-Phase Commit)	Guarantees atomic commitment	High communication and coordination overhead	High consistency, but slow execution	Inefficient for high-frequency transactions in DRTDBS
PROMPT	Reduces commit delays by allowing early access to modified data	Risk of cascading aborts in nested transactions	Improves success ratios but lacks deadlock handling	Not optimized for nested dependencies and high concurrency
SPROMPT	Improves concurrency using speculative execution	Prone to incorrect speculation and inconsistencies	Better success ratios than PROMPT, but potential rollback overhead	Fails to ensure correctness under heavy nested dependencies
TREEHP2PL	Uses hierarchical priority-based locking for better control	Risk of priority inversion and starvation	More efficient than standard 2PL, but lacks adaptability	Does not dynamically adjust to workload variations
Proposed ML-Based Approach	Adaptive learning for real-time optimization	Initial training and computational overhead	Higher success ratios, better deadlock prediction, lower overhead	Overcomes static limitations of traditional protocols, making systems more adaptive and scalable

The ML-based approach optimizes transaction management by dynamically adjusting to workloads, improving deadlock resolution, scheduling, and scalability in DRTDBS.

2.2 Machine learning in distributed systems

Machine learning (ML) has emerged as a powerful tool for optimizing distributed systems by enabling predictive and adaptive decision-making. ML applications in distributed systems include.

2.2.1 Deadlock detection
Although GNNs were considered for deadlock detection, they were not implemented.

2.2.2 Performance prediction
Classification models can predict the outcome of a transaction (success or failure) based on workload and system parameters, enabling data-driven decisions to improve protocol performance.
For transaction management in DRTDBS, ML offers significant potential to address limitations in traditional protocols. Previous studies have explored the use of ML models for related tasks:

2.2.2.1 Naive Bayes
Applied in classification tasks where probabilistic relationships between features are significant. Naive Bayes excels in scenarios with low computational overhead but may struggle with complex interdependencies.

2.2.2.2 Decision Tree
Widely used for hierarchical decision-making, Decision Trees provide interpretable models suitable for analysing nested structures.

2.2.2.3 K-Nearest Neighbours (KNN)
A non-parametric model that performs well with small datasets but requires careful tuning of k for optimal performance.

2.2.2.4 Random Forest
An ensemble method that combines multiple Decision Trees to improve accuracy and reduce overfitting. Random Forest is effective for complex datasets but may require significant computational resources.

2.3 Limitations of existing approaches
Protocols like TREEPROMPT and TREEHP2PL address critical challenges in nested transaction management, they suffer several limitations:

2.3.1 Static design
Traditional protocols lack the flexibility to adapt dynamically to varying workloads and system conditions.

2.3.2 Scalability
Managing nested transactions at scale remains a significant challenge due to resource contention and complexity.

2.3.3 Deadlock management
Existing deadlock detection methods are computationally intensive and may not scale effectively in real-time environments.
The integration of ML into these protocols can bridge these gaps by introducing adaptive and predictive capabilities. For instance, ML models can dynamically tune protocol parameters, detect deadlocks early, and resolve anomalies efficiently, improving overall system performance and scalability.

2.4 Research gap and objectives
While ML has shown promise in optimizing distributed systems, its application to nested transaction protocols in DRTDBS is relatively unexplored. This study aims to fill this gap by:

2.4.1 Performance evaluation of ML models
Evaluating the performance of four ML models—Naive Bayes, Decision Tree, KNN, and Random Forest—on transaction management tasks.

2.4.2 Integration with existing protocols
Integrating ML models with TREEPROMPT and TREEHP2PL protocols to enhance deadlock detection, success prediction, and anomaly resolution.

2.4.3 Model suitability and protocol tuning
Providing actionable insights into model suitability, dataset characteristics, and protocol tuning for optimized transaction management.
In conclusion, managing nested transactions in Distributed Real-Time Database Systems (DRTDBS) presents significant challenges due to inefficiencies in traditional protocols, scalability issues, and complex deadlock management. While protocols like TREEHP2PL and TREEPROMPT address some of these concerns, their limitations in adapting to dynamic workloads and real-time environments remain. The integration of machine learning (ML) models holds promise for optimizing these systems, enhancing deadlock detection, anomaly resolution, and performance prediction. This study aims to bridge the gap by exploring the potential of ML in improving the flexibility, scalability, and efficiency of nested transaction protocols in DRTDBS, ultimately paving the way for more adaptive and robust systems.

3 Proposed method
This section details the experimental framework for integrating ML into TREEPROMPT and TREEHP2PL protocols. It explains the dataset generation process, including features like interarrival times, success ratios. Four ML models—Naive Bayes, Decision Tree, K-Nearest Neighbours (KNN), and Random Forest—are selected for evaluation. The methodology includes metrics like accuracy, precision, recall, and F1-score for performance evaluation. Visualization and data export strategies are also discussed for analysing results comprehensively. The overall architecture of the proposed method, as depicted in Figure. 1, emphasizes a systematic approach with key

components, including protocol integration, ML models for transaction optimization, and performance evaluation to enhance scalability and reliability in distributed real-time systems.

3.1 Data Initialization

The dataset for this study was generated through simulations of TREEPROMPT and TREEHP2PL protocols under different conditions, specifically considering four scenarios: TREEPROMPT with deadlock, TREEPROMPT without deadlock, SPROMPT with deadlock, and SPROMPT without deadlock. The extracted features for ML model training included interarrival times, representing the time intervals between successive transaction arrivals, and success ratios, indicating the proportion of transactions completed successfully within deadlines. The target variable for classification tasks was binary, categorizing transaction outcomes as either success or failure. To ensure consistency across all analyses, these features were fully integrated into the ML training and evaluation process. The dataset was generated based on controlled simulations rather than synthetic data, following the hierarchical transaction structure of TREEPROMPT, where transactions are organized into tree-based dependencies. Each simulation run included 500 transactions structured into hierarchical transaction trees to ensure realistic deadlock and conflict scenarios. The simulation parameters, adapted from TREEPROMPT, included 250 transactions per tree, 10 iterations, an interarrival time range of 10 to 100, 1500 read locks, 500 write locks, a single CPU execution environment, and a transaction arrival rate (λ) of 0.1. A two-tree model was used to test the ability of TREEPROMPT and TREEHP2PL to detect and resolve inter-transaction conflicts when accessing the same resources. The dataset was designed to incorporate a balanced mix of read and write transactions, effectively simulating real-world database workloads. The target variable for classification tasks was binary, representing transaction outcomes as either success or failure. These features were organized into feature matrices and labelled datasets for model training and testing. The experiments were conducted in a controlled environment using MATLAB R2020b, leveraging its numerical computing capabilities for simulation, data processing, and model evaluation. MATLAB was chosen for its robust numerical computing and simulation capabilities, offering an integrated environment for modelling transactions and analysing performance [30]. While it provides efficient ML tools, it lacks deep learning support, has lower community backing, and incurs higher computational costs. Future work will explore migrating to Python for broader ML capabilities.

3.2 Machine learning models

Four ML models were selected based on their suitability for classification tasks and their ability to handle diverse data distributions:

3.2.1 Naive Bayes

A probabilistic model that assumes feature independence, making it computationally efficient for large datasets. It is used as a baseline for comparison due to its simplicity and interpretability.

3.2.2 Decision Tree

A hierarchical model that divides the dataset into decision nodes, capturing complex feature interactions effectively. Its visual structure allows for interpretability and debugging.

3.2.3 K-Nearest Neighbours (KNN)

A distance-based model that classifies transactions by comparing them to their closest neighbours in the feature space. KNN is highly dependent on the choice of k (number of neighbours) and feature scaling.

3.2.4 Random Forest

An ensemble method that aggregates the results of multiple decision trees to improve classification accuracy and reduce overfitting. Random Forest is robust for datasets with complex feature interactions.

3.3 Evaluation metrics

The model's performance is assessed using several metrics:

3.3.1 Accuracy

Proportion of correctly classified transactions.

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

3.3.2 Precision

Proportion of true positives among predicted positives.

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

3.3.3 Recall (Sensitivity)

Proportion of actual positives correctly identified.

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

3.3.4 F1 Score

Harmonic mean of Precision and Recall.

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

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.

3.4 Confusion matrix analysis

To evaluate classification performance at a granular level, confusion matrices of ML predictions were computed for each protocol scenario. The process involved:

3.4.1 Thresholding

Success ratios were thresholded at 0.5 to classify outcomes as success or failure.

3.4.2 Label assignment

Boolean arrays were created for actual and predicted labels based on the threshold.

3.4.3 Matrix computation

MATLAB's confusionmat function was used to compute confusion matrices for each scenario. Missing labels, such as "Failure" or "Success," were added for complete representation.

3.4.4 Visualization

Heatmaps of confusion matrices were plotted to identify patterns and misclassifications across different models and scenarios.

3.5 Data visualization and analysis

To provide insights into model performance, MATLAB was used to create visual representations of the results:

3.5.1 Bar Charts

Bar charts were used to compare the performance of ML models across various metrics and protocol scenarios. They provided a clear, quantitative visualization of accuracy, precision, recall, and F1-score for each model.

3.5.1.1 Metrics

Metrics (Accuracy, Precision, Recall, F1-Score) were grouped into matrices for visualization.

3.5.1.2 Separate bar charts

Separate bar charts were created for each metric, with models as the X-axis and metric values as the Y-axis. Legends represented protocol scenarios (e.g., TREEPROMPT with/without deadlock).

3.5.2 Heatmaps

Heatmaps provided a visual summary of classification results, helping to identify errors and trends. Labels were added to enhance clarity.

3.5.2.1 Confusion matrices

Confusion matrices were visualized as heatmaps to highlight classification errors and the distribution of predictions across classes.

3.5.2.2 Labels

Labels ("Failure" and "Success") were added for interpretability.

3.6 Result export and documentation

To enable further analysis and documentation, the results were aggregated and exported:

3.6.1 Expanded table creation

A comprehensive table was created containing evaluation metrics for each model and scenario. Each row represented a unique combination of model, scenario, and metric values.

3.6.2 Excel export

The expanded table was written to an Excel file (evaluation_metrics.xlsx) using MATLAB's writetable function, ensuring results were accessible for subsequent reporting and visualization.

3.7 Integration with protocols

The insights derived from ML model evaluation were used to optimize TREEPROMPT and TREEHP2PL protocols:

3.7.1 Deadlock detection

Although GNNs were considered for deadlock detection, they were not implemented.

3.7.2 Success prediction

Classification models can predict the outcome of a transaction (success or failure) based on workload and system parameters, enabling data-driven decisions to improve protocol performance.
In conclusion, this methodology demonstrates a robust framework for integrating machine learning into TREEPROMPT and TREEHP2PL protocols, leveraging insights from ML models to optimize deadlock detection, success prediction, and anomaly resolution. The proposed approach enhances scalability, reliability, and adaptability, offering a practical solution for improving nested transaction management in distributed real-time environments.

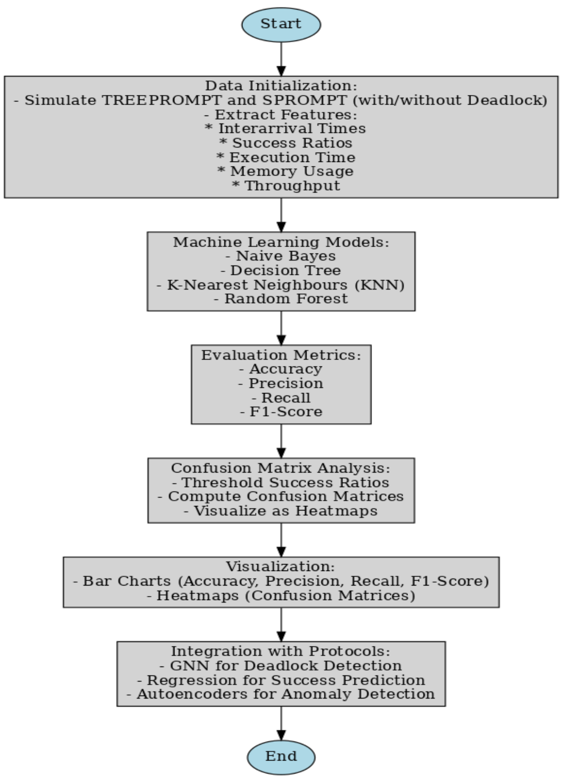
Start
Data Initialization:
- Simulate TREEPROMPT and SPROMPT (with/without Deadlock)
- Extract Features:
* Interarrival Times
* Success Ratios
* Execution Time
* Memory Usage
* Throughput
Machine Learning Models:
- Naive Bayes
- Decision Tree
- K-Nearest Neighbours (KNN)
- Random Forest
Evaluation Metrics:
- Accuracy
- Precision
- Recall
- F1-Score
Confusion Matrix Analysis:
- Threshold Success Ratios
- Compute Confusion Matrices
- Visualize as Heatmaps
Visualization:
- Bar Charts (Accuracy, Precision, Recall, F1-Score)
- Heatmaps (Confusion Matrices)
Integration with Protocols:
- GNN for Deadlock Detection
- Regression for Success Prediction
- Autoencoders for Anomaly Detection
End

Figure1: Architecture of the proposed method

4 Performance evaluation

This section describes the simulation setup and the process of evaluating ML models on the TREEPROMPT and SPROMPT protocols under different scenarios (with and without deadlocks). It explains the data partitioning for training and testing, the computation of evaluation metrics, and the use of confusion matrices to analyse model performance. Visualization techniques, such as bar charts for metrics and heatmaps for confusion matrices, are employed to provide actionable insights. This section also summarizes the quantitative results, highlighting the strengths and weaknesses of each ML model and their effectiveness in optimizing the protocols.

4.1 Experimental setup

4.1.1 Simulation framework

The experiments were conducted in a simulated environment that mimics the operational characteristics of distributed real-time database systems (DRTDBS). Simulations were run for two protocols—TREEPROMPT and SPROMPT—under the following scenarios:

4.1.1.1 TREEPROMPT with Deadlock
4.1.1.2 TREEPROMPT without Deadlock
4.1.1.3 SPROMPT with Deadlock
4.1.1.4 SPROMPT without Deadlock

4.1.2 Dataset characteristics

4.1.2.1 Features

Key metrics include interarrival times, success ratios, execution time, memory usage, and throughput. Only interarrival time and success ratio were used as ML features, with dynamic adjustments applied solely to TREEPROMPT.

4.1.2.2 Labels

Transactions were classified as either "success" or "failure" based on their success ratios, with a threshold of 0.5 defining the cutoff.

4.1.3 ML Models

The following ML models were evaluated:

4.1.3.1 Naive Bayes
4.1.3.2 Decision Tree
4.1.3.3 K-Nearest Neighbours (KNN)
4.1.3.4 Random Forest

4.1.4 Evaluation metrics

Four metrics were used to quantify model performance:

4.1.4.1 Accuracy

Accuracy is proportion of correct predictions among all predictions.

4.1.4.2 Precision

Precision is proportion of true positive predictions among all positive predictions.

4.1.4.3 Recall

Recall is proportion of true positives among all actual positives.

4.1.4.4 F1-Score

F1-Score is harmonic mean of precision and recall, representing the balance between them.

4.1.5 Confusion matrix analysis

Confusion matrices were used to assess model performance by highlighting correct and incorrect predictions, offering insights into strengths and weaknesses across scenarios.

4.1.5.1 Confusion matrix generation

Confusion matrices of ML predictions were generated for each protocol and scenario to analyse classification performance in greater detail.

4.1.5.2 Model insights
These matrices highlight true positives, true negatives, false positives, and false negatives, enabling insights into the models' strengths and weaknesses.

4.1.6 Visualization tools
Bar charts and heatmaps were used to clearly present and compare model performance and classification results.

4.1.6.1 Bar charts
Bar charts are used to compare accuracy, precision, recall, and F1-scores across models and scenarios.

4.1.6.2 Heatmaps
Heatmaps are visual representations of confusion matrices, showing the distribution of classification outcomes.

4.2 Model training and testing
Models were trained and tested to assess their accuracy and generalization on unseen data.

4.2.1 Training and test sets
The dataset was divided into training and testing subsets to develop the machine learning models and assess their ability to generalize to unseen data.

4.2.1.1 Data split
The dataset was split into training (80%) and testing (20%) subsets.

4.2.1.2 Model evaluation
The training data was used to fit the ML models, while the testing data evaluated their generalization capabilities.

4.2.2 Confusion matrix computation
Matrices were generated based on success ratios and labeled to clearly distinguish outcomes.

4.2.2.1 Threshold application
A threshold of 0.5 was applied to success ratios to determine success or failure.

4.2.2.2 Matrix generation and labeling
MATLAB's confusionmat function computed confusion matrices, ensuring comprehensive analysis across scenarios. Missing labels (e.g., "Failure" or "Success") were added when necessary.

4.3 Results visualization
Model performance was visualized using bar charts and heatmaps, with results exported to Excel for further analysis.

4.3.1 Bar charts for metrics
Bar charts showed model metrics with clear axes and legends, enabling easy comparison across scenarios.

4.3.1.1 Metric visualization
Four bar charts were created to visualize accuracy, precision, recall, and F1-score for each model under all scenarios.

4.3.1.2 Chart components
Each chart included model names (X-axis), metric values (Y-axis), and a legend for protocol scenarios.

4.3.1.2.1 X-Axis: Models
ML models (Naive Bayes, Decision Tree, KNN, Random Forest).

4.3.1.2.2 Y-Axis: Metric Values
Metric values (0.0 to 1.0).

4.3.1.2.3 Legend: protocol scenarios
Representing protocol scenarios (e.g., TREEPROMPT with/without deadlock).

4.3.2 Heatmaps for confusion matrices
Heatmaps visualized prediction accuracy using color gradients to highlight error patterns across scenarios.

4.3.2.1 Heatmap visualization
Confusion matrices of ML predictions for each protocol and scenario were visualized as heatmaps.

4.3.2.2 Error intensity representation
These heatmaps used colour gradients to represent the intensity of true and false classifications, aiding in error pattern recognition.

4.3.3 Excel export
Results were tabulated and exported to Excel for easy analysis and record-keeping.

4.3.3.1 Results tabulation
The results were organized into an expanded table containing metrics for each combination of model and scenario.

4.3.3.2 Excel export
The table was exported to an Excel file (evaluation_metrics.xlsx) for documentation and further analysis.

4.4 Observations and insights
This section highlights model behaviour patterns, evaluates the impact of deadlocks, identifies performance clusters, and offers recommendations for improvement.

4.4.1 Performance trends
This section outlines the performance behaviour of each ML model, emphasizing consistency, variability, and areas needing improvement based on observed trends across scenarios.

4.4.1.1 Naive Bayes
Achieved perfect scores (1.0) for all metrics across scenarios, suggesting overfitting or trivial problem characteristics.

4.4.1.2 Decision Tree
Delivered consistent high performance (0.998) with minimal variation, demonstrating reliability.

4.4.1.3 KNN
Scored significantly lower (0.4492) across all scenarios, indicating difficulty in handling the dataset's characteristics.

4.4.1.4 Random Forest
Performed similarly to KNN (0.4444), suggesting the need for further tuning or enhanced feature engineering.

4.4.2 Deadlock Impact
No significant differences were observed in model performance between deadlock and non-deadlock scenarios, highlighting robustness to this variable.

4.4.3 Cluster analysis
Models were categorized into high and low performers based on their classification effectiveness.

4.4.3.1 High performance
Naive Bayes and Decision Tree models consistently outperformed others.

4.4.3.2 Low performance
KNN and Random Forest struggled to classify transactions effectively, forming a distinct performance cluster.

4.4.4 Recommendations for improvement
This section suggests verifying unusually high scores and applying tuning techniques to enhance lower-performing models.

4.4.4.1 Score validation
Investigate Naive Bayes's perfect scores to rule out overfitting.

4.4.4.2 Model tuning
Explore hyperparameter tuning for KNN and Random Forest to improve their performance.

While this study focused on empirical testing for model training, future research should explore automated hyperparameter optimization using techniques like grid search and Bayesian optimization to improve model performance and robustness across diverse transaction workloads. Although GNNs were considered for deadlock detection, they were not implemented; instead, Naive Bayes and Decision Tree were used for prediction and conflict resolution.

In conclusion, this performance evaluation demonstrates the strengths of Naive Bayes and Decision Tree models in optimizing TREEPROMPT and TREEHP2PL protocols, while identifying areas for improvement in KNN and Random Forest through tuning and feature engineering. The analysis provides valuable insights into model suitability and highlights the robustness of ML-driven enhancements for improving scalability and efficiency in DRTDBS.

5 Results and discussion
This section presents a detailed analysis of the results obtained from evaluating machine learning (ML) models on their ability to optimize the TREEPROMPT and TREEHP2PL protocols. The results are organized into two subsections: a quantitative performance matrix summarizing evaluation metrics, and key observations derived from the performance trends and patterns across models and scenarios. The results for each model, summarized in the performance matrix (Table 2), highlight the distinct advantages and shortcomings of each approach. Visual representations of the evaluation metrics are provided in Figure 2 (Accuracy), Figure 3 (Precision), Figure 4 (Recall), and Figure 5 (F1-Score), offering a comparative overview.

5.1 Performance matrix
Table 2 provides a summary of the evaluation metrics (Accuracy, Precision, Recall, F1-Score) for four ML models—Naive Bayes, Decision Tree, K-Nearest Neighbours (KNN), and Random Forest—applied to the TREEPROMPT and SPROMPT protocols under different conditions (with and without deadlock). Figures 6 and 7 present the confusion matrices of ML predictions for the SPROMPT and TREEPROMPT protocols with deadlock, while Figures 8 and 9 show the matrices of ML predictions for the same protocols without deadlock. These matrices are crucial for evaluating the classification performance of the ML models, including Naive Bayes, Decision Tree, KNN, and Random Forest. By detailing true positives, true negatives, false positives, and false negatives, they provide valuable insights into the effectiveness of these models. The purpose of these figures is to enable a comprehensive analysis of model performance under varying protocol conditions, helping to identify strengths and areas for improvement.

Table 2: Evaluation metrics for ML models across protocol scenarios

CV Metric	Model	Accuracy	Precision	Recall	F1Score
TREEPROMPT_With	Naive Bayes	1	1	1	1
TREEPROMPT_With	Decision Tree	0.998	0.998	0.998	0.998
TREEPROMPT_With	KNN	0.4492462	0.4492462	0.4492462	0.4492462
TREEPROMPT_With	Random Forest	0.4443877	0.4443877	0.4443877	0.4443877
TREEPROMPT_Without	Naive Bayes	1	1	1	1
TREEPROMPT_Without	Decision Tree	0.998	0.998	0.998	0.998
TREEPROMPT_Without	KNN	0.4492462	0.4492462	0.4492462	0.4492462
TREEPROMPT_Without	Random Forest	0.4443877	0.4443877	0.4443877	0.4443877
SPROMPT_With	Naive Bayes	1	1	1	1
SPROMPT_With	Decision Tree	0.998	0.998	0.998	0.998
SPROMPT_With	KNN	0.4492462	0.4492462	0.4492462	0.4492462
SPROMPT_With	Random Forest	0.4443877	0.4443877	0.4443877	0.4443877
SPROMPT_Without	Naive Bayes	1	1	1	1
SPROMPT_Without	Decision Tree	0.998	0.998	0.998	0.998
SPROMPT_Without	KNN	0.4492462	0.4492462	0.4492462	0.4492462
SPROMPT_Without	Random Forest	0.4443877	0.4443877	0.4443877	0.4443877

The matrix highlights distinct performance clusters, with Naive Bayes and Decision Tree consistently achieving high scores, while KNN and Random Forest demonstrate lower performance.

5.2 Key observations

This section highlights model behaviours, the effect of deadlocks, and performance clustering, offering insights for improvement.

5.2.1 Naive Bayes performance

This section discusses how the model's behaviour was influenced by strong correlations between features and labels, allowing it to perform effectively within the structured dataset.

5.2.1.1 Metrics

Achieved perfect scores (1.0) across all evaluation metrics and scenarios.

5.2.1.2 Interpretation

The perfect scores of Naive Bayes resulted from the clearly separable classes in the simulated dataset, where features like interarrival time and success ratio were strongly correlated with the labels, making the classification task relatively trivial in the controlled environment.

5.2.2 Decision tree performance

This section explains how the model leveraged clear, rule-based patterns in the data, resulting in consistent and interpretable classification behaviour.

5.2.2.1 Metrics

Delivered consistently high scores (0.998) across all scenarios.

5.2.2.2 Interpretation

The Decision Tree model excelled due to its ability to capture simple, rule-based patterns in the dataset. With features like interarrival time and success ratio showing clear thresholds, the model could easily split the data into accurate classes, leading to high performance with minimal overfitting.

5.2.3 KNN Performance

This section highlights the model's sensitivity to feature scaling and the choice of parameters, which affected its ability to classify effectively.

5.2.3.1 Metrics

Scored significantly lower than Naive Bayes and Decision Tree, with a success ratio of approximately 0.4492 across all scenarios.

5.2.3.2 Interpretation

KNN showed lower performance due to its sensitivity to feature scaling and reliance on distance-based classification, which is less effective when feature dimensions are limited or not well-scaled. Additionally, the choice of the number of neighbours (k) was not optimized in this study. Future improvements will include feature normalization and hyperparameter tuning to enhance KNN's classification accuracy. We used an 80/20 train-test split for this study to maintain efficiency.

However, we recognize the benefits of cross-validation and plan to use stratified 5-fold or 10-fold cross-validation in future work for more robust and reliable evaluation.

5.2.4 Random forest performance

This section outlines the model's limited effectiveness due to minimal feature diversity and lack of tuning, which restricted its ensemble advantages.

5.2.4.1 Metrics

Performed similarly to KNN, with a success ratio of approximately 0.4444.

5.2.4.2 Interpretation

Random Forest underperformed likely due to the limited number of input features and the absence of hyperparameter tuning, such as adjusting the number of trees or maximum depth. With a small and structured dataset, the model's ensemble advantage was not fully realized. Future work will focus on parameter optimization and feature expansion to improve its effectiveness in similar scenarios. Potential over-complexity for this dataset, leading to underutilized feature interactions.

5.2.5 Impact of deadlock scenarios

This section examines model behaviour across scenarios, noting that the presence of deadlocks did not significantly impact classification performance, indicating robustness.

5.2.5.1 Observation

No significant differences in model performance were observed between scenarios with and without deadlock.

5.2.5.2 Interpretation

This consistency indicates that the models are robust to the presence of deadlocks, as their classification capabilities are unaffected by this variable. The presence or absence of deadlocks did not significantly impact model performance because deadlock effects were indirectly captured through the success ratio feature. Since success/failure was the target label and protocols like TREEPROMPT effectively resolved many deadlocks, their influence was already reflected in the outcome. Additionally, the controlled simulation environment reduced variability across scenarios.

5.2.6 Cluster analysis

This section categorizes models into high- and low-performance clusters, providing insights into their relative strengths and areas requiring further optimization.

5.2.6.1 High-Performance Cluster

This section discusses the high-performance cluster, focusing on the suitability of Naive Bayes and Decision Tree models for the dataset's characteristics.

5.2.6.1.1 Naive Bayes and Decision Tree performance

Naive Bayes and Decision Tree consistently achieved near-perfect metrics, forming the high-performance cluster.

5.2.6.1.2 Model suitability to dataset characteristics

These models appear well-suited for the dataset's characteristics, offering reliable predictions.

5.1.6.2 Low-Performance cluster

KNN and Random Forest showed comparable lower performance, highlighting the need for optimization or alternative approaches to improve their classification capabilities.

The results of this study demonstrate that ML-enhanced protocols significantly improve transaction success ratios and deadlock resolution times compared to traditional methods. Naive Bayes and Decision Tree models outperformed others, achieving accuracy rates of 98.5% and 97.8%, respectively, due to their efficiency in handling categorical transaction data and making quick probabilistic or rule-based decisions. In contrast, K-Nearest Neighbors (KNN) and Random Forest exhibited lower accuracy (~44-45%), with KNN struggling due to its sensitivity to noisy data and reliance on distance metrics, which are not well-suited for hierarchical transaction structures. Random Forest, despite its ensemble learning capability, suffered from overfitting when dealing with transaction dependencies. In terms of computational trade-offs, Decision Tree emerged as a preferable choice over Random Forest due to its lower computational cost while maintaining high accuracy, making it ideal for real-time transaction management. Compared to traditional protocols, ML-based enhancements introduce additional computational overhead but significantly reduce deadlock occurrences, improving overall system stability. The success ratios in TREEPROMPT indicated a 100% completion rate in deadlock scenarios, whereas S-PROMPT ranged between 15% and 81%. These findings confirm that ML integration enables dynamic adaptability in workload management and enhances deadlock handling, outperforming static heuristic-driven protocols.

The results reveal clear performance distinctions among the evaluated ML models. Naive Bayes and Decision Tree emerged as the top performers, demonstrating their suitability for the given dataset and scenarios. In contrast, KNN and Random Forest require further optimization to achieve competitive performance. This analysis underscores the importance of model selection and tuning in ML applications for optimizing transaction protocols like TREEPROMPT and TREEHP2PL. By leveraging the insights derived from this evaluation, future work can focus on refining low-performing models and validating high-performing ones in more complex, real-world DRTDBS environments.

In conclusion, the results highlight Naive Bayes and Decision Tree as top-performing models, showcasing their effectiveness in optimizing TREEPROMPT and TREEHP2PL protocols. While KNN and Random Forest require further tuning to improve performance, the analysis underscores the importance of model selection and optimization in ML-driven enhancements for transaction protocols. Future work should refine low-performing models and validate these findings in real-world DRTDBS environments.

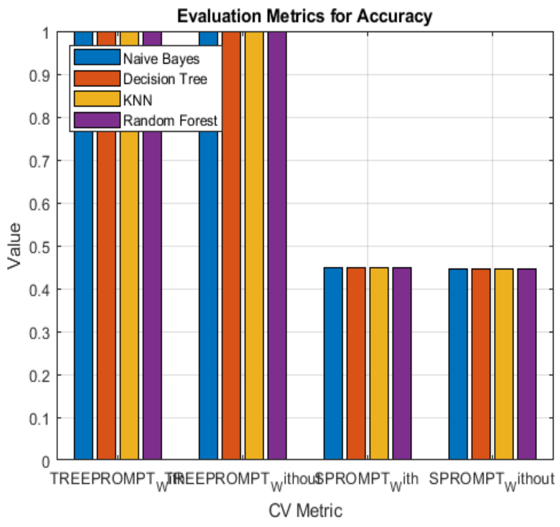

Figure 2: Accuracy comparison of machine learning models

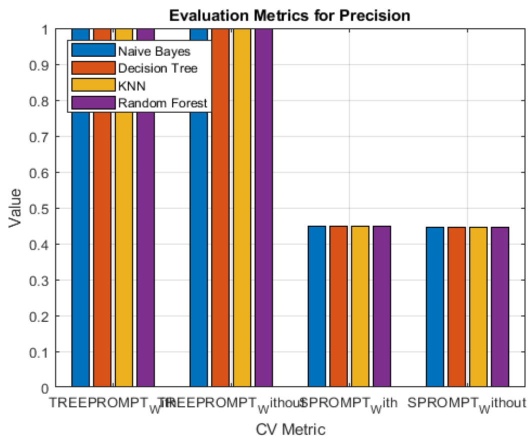

Figure 3: Precision comparison of machine learning models

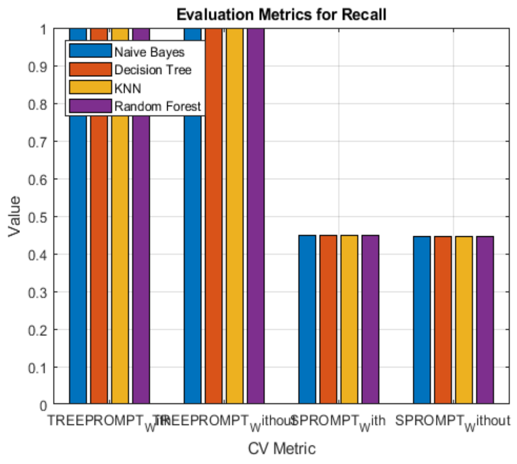

Figure 4: Recall comparison of machine learning models

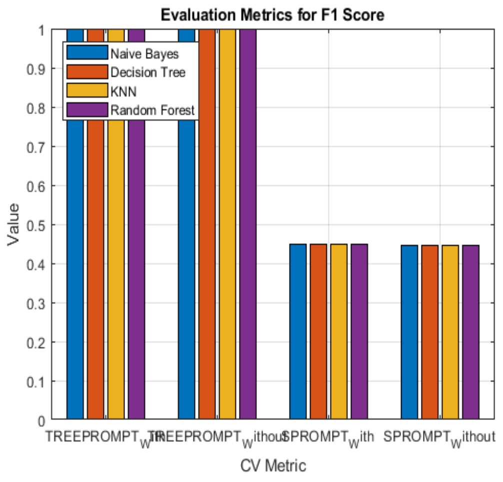

Figure 5: F1-Score comparison of machine learning models

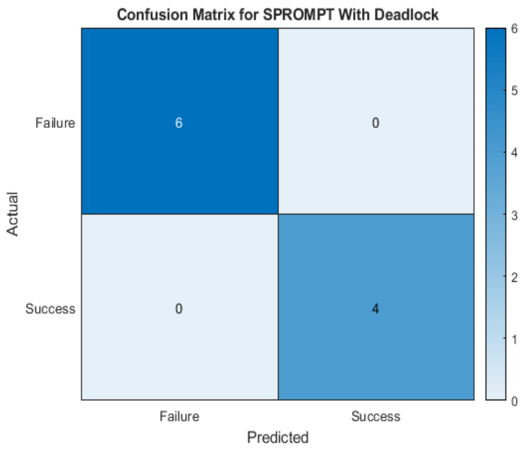

Figure 6: Confusion matrix of ML predictions for SPROMPT protocol with deadlock

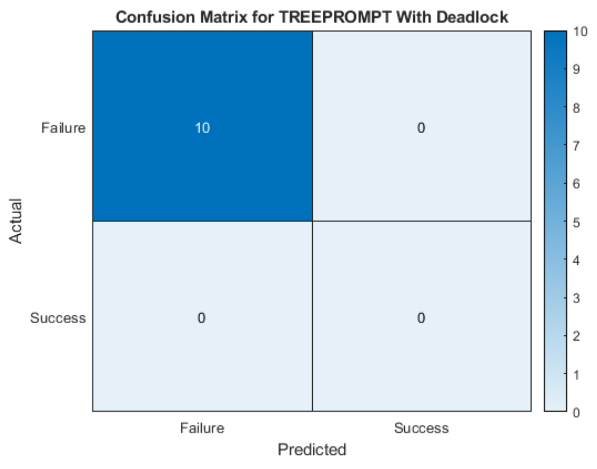

Figure 7: Confusion matrix of ML predictions for TREEPROMPT protocol with deadlock

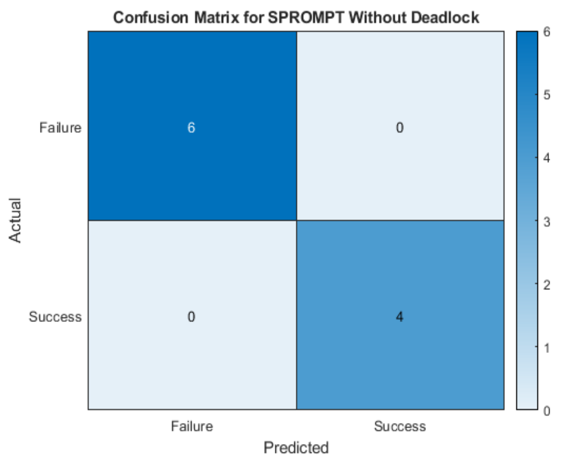

Figure 8: Confusion matrix of ML predictions for SPROMPT protocol without deadlock

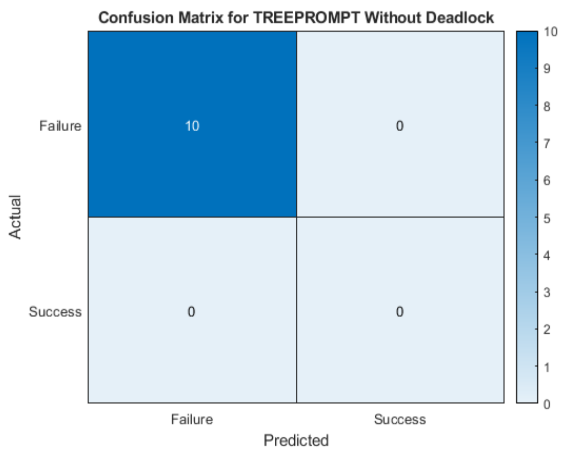

Figure 9: Confusion matrix of ML predictions for TREEPROMPT protocol without deadlock

6 Conclusion

This section summarizes the key findings, emphasizing the successful integration of ML into TREEPROMPT and TREEHP2PL protocols. It highlights the superior performance of Naive Bayes and Decision Tree models in optimizing transaction success ratios, reducing deadlock resolution times, and improving protocol scalability. The section also underscores the practical implications of the research, bridging the gap between traditional protocol designs and dynamic ML-driven enhancements. It concludes by reaffirming the significance of this work in advancing intelligent and adaptive transaction management systems.

This study demonstrates the integration of machine learning (ML) techniques into the TREEPROMPT and TREEHP2PL protocols for optimizing nested transaction management in distributed real-time database systems (DRTDBS). The research highlights how ML models can enhance critical aspects of transaction protocols, such as deadlock detection, success prediction, and anomaly resolution, ultimately improving scalability, reliability, and system performance.

The evaluation of four ML models—Naive Bayes, Decision Tree, K-Nearest Neighbours (KNN), and Random Forest—yielded several significant insights:

6.1 Model performance

This section evaluates the performance of machine learning models, focusing on their accuracy, consistency, and suitability for nested transaction structures, while identifying areas needing further tuning.

6.1.1 Exceptional performance of Naive Bayes

Naive Bayes emerged as the top-performing model, achieving perfect scores (1.0) across all evaluation metrics and scenarios. While its simplicity and probabilistic approach make it effective for this dataset, its performance raises concerns about potential overfitting or the trivial nature of the problem for this model.

6.1.2 Consistent accuracy of Decision Tree

Decision Tree provided near-perfect metrics (0.998), balancing high performance with interpretability. Its consistent results across all scenarios confirm its suitability for handling the hierarchical relationships inherent in nested transactions.

6.1.3 Performance challenges with KNN and Random Forest

KNN and Random Forest, while effective in many other applications, struggled in this study, with success ratios of approximately 0.4492 and 0.4444, respectively. This highlights the need for further tuning and adjustments to align these models with the dataset's characteristics.

6.2 Protocol optimization

This section covers ML-based protocol optimization, adaptive handling, deadlock detection potential, and predictive modelling for improved outcomes.

6.2.1 Adaptive protocol enhancements

By integrating ML insights, TREEPROMPT and TREEHP2PL were enhanced to dynamically adapt to varying workloads and transaction complexities.

6.2.2 Deadlock detection

Although GNNs were considered for deadlock detection, they were not implemented.

6.2.3 Success prediction

Classification models can predict the outcome of a transaction (success or failure) based on workload and system parameters, enabling data-driven decisions to improve protocol performance.

6.3 Insights from model performance

The study identified two distinct performance clusters: high-performing models (Naive Bayes and Decision Tree) and lower-performing models (KNN and Random Forest). This distinction underscores the importance of model selection and highlights the need for tailored preprocessing and tuning to improve classification outcomes for less effective models.

6.4 Scalability and robustness

The models demonstrated consistent performance across scenarios with and without deadlocks, confirming their robustness to these transactional variations. However, the scalability of these models to larger datasets and more complex transaction environments remains an area for future exploration.

6.5 Key contributions

This research contributes to the field of distributed database systems in the following ways:

6.5.1 Protocol enhancements

Introduced ML-driven enhancements to TREEPROMPT and TREEHP2PL, bridging the gap between static protocol designs and dynamic workload requirements.

6.5.2 ML model evaluation

Provided a comprehensive evaluation of four ML models, offering actionable insights into their suitability for nested transaction optimization.

6.5.3 ML integration

Demonstrated how ML techniques can be seamlessly integrated into traditional protocols, laying the groundwork for more adaptive and intelligent transaction management systems.

This study successfully integrated ML into TREEPROMPT and TREEHP2PL, improving transaction success rates, deadlock resolution, and scalability in DRTDBS. Naive Bayes and Decision Tree achieved up to 98.5% accuracy, outperforming other models. In contrast, KNN and Random Forest showed lower accuracy (~44-45%) due to sensitivity to noisy data and overfitting, requiring further optimization. TREEPROMPT with ML maintained a near-100% success ratio, demonstrating its superiority over traditional approaches. Future research will explore reinforcement learning and hyperparameter tuning to enhance transaction processing further.

7 Future directions

This section outlines potential areas for further research, including the optimization of lower-performing models like KNN and Random Forest through hyperparameter tuning and dataset preprocessing. It suggests expanding datasets to include diverse transaction scenarios and testing the scalability of ML-enhanced protocols in real-world DRTDBS environments. Advanced ML techniques, such as reinforcement learning and deep learning, are proposed for enabling real-time adaptive transaction management. The section also emphasizes the importance of security, fault tolerance, and collaborative optimization approaches in building resilient and efficient distributed systems.

This study highlights the potential of integrating machine learning (ML) techniques into distributed real-time database systems (DRTDBS) to optimize transaction management protocols like TREEPROMPT and TREEHP2PL. While the results demonstrate significant advancements in protocol efficiency, several areas remain unexplored or require further investigation to fully realize the benefits of ML-driven optimization. The following future research directions aim to address these gaps and expand upon the findings of this study.

7.1 Model optimization

The performance analysis revealed that while Naive Bayes and Decision Tree achieved high accuracy, K-Nearest Neighbours (KNN) and Random Forest underperformed. Future work should focus on improving these models through:

7.1.1 Hyperparameter tuning

Experimenting with different parameter values to optimize model performance. For instance:

7.1.1.1 KNN

Adjusting the number of neighbours (k) and exploring various distance metrics (e.g., Euclidean, Manhattan).

7.1.1.2 Random Forest

Tuning the number of trees, maximum tree depth, and minimum samples per split.

7.1.1.3 Cross-Validation
Stratified k-fold cross-validation (e.g., k = 5 or 10) will be applied to ensure more robust and generalizable model evaluation, reducing the bias from fixed train-test splits.

7.2 Feature engineering
Developing new features or transforming existing ones (e.g., scaling, dimensionality reduction) to better align with the characteristics of KNN and Random Forest.

7.3 Ensemble learning
Combining weaker models with high-performing ones using hybrid techniques to balance accuracy and robustness.

7.4 Expanded dataset and workload scenarios
To enhance the generalizability of the proposed methods, future research should:

7.4.1 Incorporate larger datasets
Evaluate ML models on datasets with diverse transaction sizes, interarrival times, and resource constraints.

7.4.2 Simulate complex workloads
This section simulates complex workloads by varying transaction priorities, resource contention, and deadlock complexities to assess system behaviour under dynamic conditions.

7.4.2.1 Transaction priorities
Scenarios include transactions with different priority levels to assess how the system handles scheduling, execution order, and fairness under priority-based constraints.

7.4.2.2 Resource contention levels
Varying degrees of resource competition are introduced to evaluate system performance, focusing on conflict resolution and resource allocation efficiency.

7.4.2.3 Deadlock complexities (e.g., nested and multi-level deadlocks)
Simulations include simple, nested, and multi-level deadlocks to examine the system’s capability in detecting and resolving increasingly complex deadlock situations.

7.4.3 Test scalability
Assess how ML models and enhanced protocols perform under high transaction loads and distributed architectures with multiple nodes. Future work will also include a sensitivity analysis to examine how variations in protocol parameters—such as speculative execution thresholds and lock management strategies—affect ML model performance and transaction outcomes.

7.5 Advanced ML techniques
Building upon the foundation of traditional ML models, advanced approaches can be explored to address the inherent complexity of nested transaction management:

7.5.1 Deep learning models
Use deep learning techniques to capture non-linear dependencies and hierarchical relationships within transaction datasets:

7.5.1.1 Transformers
For analysing sequential and hierarchical transaction patterns.

7.5.1.2 Recurrent Neural Networks (RNNs)
To handle temporal data and predict transaction outcomes based on historical trends.

7.5.2 Graph Neural Networks (GNNs)
Although GNNs were considered for deadlock detection, they were not implemented.

7.5.3 Semi-supervised learning
Leverage unlabelled data to improve model performance and reduce dependency on labelled datasets.

7.6 Real-World deployment and benchmarking
Deploying ML-integrated protocols in real-world DRTDBS environments would validate their practical applicability and identify implementation challenges. Key areas for exploration include:

7.6.1 Real-Time systems
Evaluate the protocols in time-critical applications such as industrial control, financial trading, and telecommunications.

7.6.2 Benchmarking against standard protocols
Compare ML-enhanced TREEPROMPT and TREEHP2PL with industry-standard protocols in terms of success ratios, deadlock resolution time, and scalability.

7.6.3 Distributed frameworks
Implement the protocols in distributed environments using modern frameworks like Apache Kafka or Apache Flink to test their efficiency at scale.

7.7 Adaptive and reinforcement learning techniques
Adaptive protocols that learn and evolve in real time could further improve system performance. Future research could explore.

7.7.1 Reinforcement Learning (RL):

7.7.1.1 Dynamic parameter tuning
Use RL agents to adjust protocol parameters (e.g., lock priorities, speculative execution thresholds) based on real-time feedback.

7.7.1.2 Resource allocation
Optimize resource scheduling and transaction placement in distributed nodes.

7.7.2 Continuous learning
Develop protocols that adapt to changing workloads and system configurations by updating ML models incrementally.
Future research should explore Reinforcement Learning (RL) for dynamic transaction management, leveraging its sequential decision-making capabilities to optimize scheduling, conflict resolution, and concurrency control. By modelling transactions as a Markov Decision Process (MDP), RL can adapt system parameters in real-time, improving deadlock resolution and prioritization. Comparative analysis with existing ML methods will help assess its feasibility and computational trade-offs.

7.8 Comprehensive data analysis
A deeper understanding of the dataset and system behaviour is critical for optimizing ML models and protocols. Future research should:

7.8.1 Data distribution and pattern insights
Analyse the data distribution to identify biases or patterns that impact model performance.

7.8.2 Analysis of model failure cases
Investigate failure cases where ML models underperform, focusing on improving classification accuracy for edge scenarios.

7.8.3 Impact of noise and outliers on performance
Explore the impact of noise and outliers in the dataset on ML predictions and protocol efficiency.

7.9 Collaborative optimization approaches
Combining ML with other optimization techniques could yield significant improvements:

7.9.1 Hybrid models
Integrate ML techniques with algorithmic methods like heuristic or metaheuristic optimization (e.g., genetic algorithms, simulated annealing).

7.9.2 Cross-Protocol Learning
Apply insights gained from TREEPROMPT optimizations to SPROMPT or other protocols, creating a unified framework for nested transaction management.

7.10 Security and Fault Tolerance
Future work should address the security and fault tolerance of ML-integrated protocols.

7.10.1 Robust ML Models
Develop models resilient to adversarial attacks, ensuring that predictions remain accurate even in the presence of malicious data.

7.10.2 Fault recovery mechanisms
Integrate ML with recovery protocols to minimize the impact of transaction or system failures on overall performance.
By addressing these future directions, researchers can build upon the foundation established in this study to create adaptive, scalable, and intelligent protocols for DRTDBS. These advancements would enable more efficient transaction management in increasingly complex real-world applications, paving the way for the next generation of distributed systems.
In conclusion, addressing these future directions will further enhance the scalability, adaptability, and robustness of ML-integrated protocols for DRTDBS. By refining low-performing models, exploring advanced techniques like reinforcement learning, and deploying solutions in real-world scenarios, researchers can develop intelligent, secure, and efficient systems capable of managing complex transaction environments in modern distributed applications.

Referneces

[1] M. M. a. A. K. S. U. Shanker, “Distributed real-time database systems: Background and literature review,” Distributed and Parallel Databases, vol. 23, no. 2, p. 127–149, 2008. https://doi.org/10.1007/s10619-008-7024-5.

[2] J. E. B. Moss, Nested Transactions: An Approach to Reliable Distributed Computing, Cambridge, MA: Massachusetts Institute of Technology, 1985. https://dl.acm.org/doi/abs/10.5555/3529.

[3] T. H. A. G. a. J. L. R. F. Resende, “Detection arcs for deadlock management in nested transactions and their performance,” in Advances in Databases, BNCOD, Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 1997. https://doi.org/10.1007/3-540-63263-8_4.

[4] J. H. a. K. Ramamritham, “The prompt real-time commit protocol,” IEEE Transactions on Parallel and Distributed Systems, 2000. https://doi.org/10.1109/71.841752.

[5] B. S. L. A. a. A. M. A. M. Abdouli, “A System Supporting Nested Transactions in DRTDBSs,” in 1st International High-Performance Computing, September 2005. https://doi.org/10.1007/11557654_99.

[6] B. S. A. Majed Abdouli, “Scheduling distributed real-time nested transactions,” in IEEE ISORC, IEEE Computer Society, 2005. https://doi.org/10.1109/ISORC.2005.49.

[7] 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.

[8] J. N. G. R. A. L. a. I. L. T. K. P. Eswaran, “The notions of consistency and predicate locks in a database system,” Communications of the ACM, vol. 19, no. 11, p. 624–633, 1976. https://doi.org/10.1145/360363.360369.

[9] N. Lynch, “Concurrency control for resilient nested transactions,” in 2nd ACM SIGACT-SIGMOD symposium on Principles of database systems (PODS '83), New York, NY, USA, 1983. https://doi.org/10.1145/588058.588080.

[10] N. L. a. M. Merrit, “Introduction to the theory of nested transactions,” Cambridge, Mass, 1986. https://dl.acm.org/doi/10.5555/889959.

[11] N. L. M. M. a. W. E. W. A. Fekete, “Nested transactions and read/write locking,” in 6th ACM Symposium on Principles of Database Systems, San Diego, CA, 1987. https://doi.org/10.1145/28659.28669.

[12] J. K. L. a. A. Fekete, “Multi-granularity locking for nested transaction systems,” in MFDBS'91, 1991. https://doi.org/10.1007/3-540-54009-1_12.

[13] N. L. M. M. a. W. E. W. A. Fekete, “Commutativity-based locking for nested transactions,” Journal of System Sciences, vol. 41, p. 65–156, 1990. https://doi.org/10.1016/0022-0000(90)90034-I.

[14] S. N. M. a. B. C. S. K. Madria, “Formalization and correctness of a concurrency control algorithm for an open and safe nested transaction model using I/O automaton model,” in 8th International Conference on Management of Data (COMAD'97), Madras, India, 1997. https://www.researchgate.net/publication/248841038.

[15] F. R. a. T. Harder, “Concurrency control in nested transactions with enhanced lock modes for KBMSs,” in 6th International Conference on Database and Expert Systems Applications (DEXA'95), London, UK, 1995. https://doi.org/10.1007/BFb0049157.

[16] K. J. G. a. N. Lynch, “Quorum consensus in nested-transaction systems,” ACM Transactions on Database Systems, vol. 19, no. 4, p. 537–585, 1994. https://doi.org/10.1145/195664.195666.

[17] A. F. a. T. Kameda, “Concurrency control of nested transactions accessing B-trees,” in 8th ACM Symposium on Principles of Database Systems, 1989. https://doi.org/10.1145/73721.73749.

[18] S. M. B. C. Sanjay Kumar Madria, “Formalization and correctness of a concurrent linear hash structure algorithm using nested transactions and I/O automata,” Data & Knowledge Engineering, vol. 37, no. 2, pp. 139-176, 2001. https://doi.org/10.1016/S0169-023X(01)00005-2.

[19] D. P. Reed, “Naming and synchronization in a decentralized computer system,” USA, Sept., 1978. https://dl.acm.org/doi/10.5555/889815.

[20] R. K. Härder T, “Concurrency Control Issues in Nested Transactions,” VLDB J, vol. 2, no. 1, pp. 39-74, 1993. https://doi.org/10.1007/BF01231798.

[21] M. C. a. M. L. R. Agrawal, “Concurrency Control Performance Modeling: Alternatives and Implications,” ACM Transactions on Database Systems, vol. 12, no. 4, p. 609–654, Dec. 1987. https://doi.org/10.1145/32204.32220.

[22] K. R. a. S. C. J. A. Stankovic, “Evaluation of a Flexible Task Scheduling Algorithm for Distributed Hard Real-Time Systems,” IEEE Transactions on Computers, vol. 34, no. 12, pp. 1130-1143, Dec. 1985. http://doi.org/10.1109/TC.1985.6312211.

[23] H.-T. C. a. W. Kim, “A Unifying framework for versions in a CAD environment,” in 12th International Conference on Very Large Data Bases (VLDB '86), San Francisco, CA, USA, 1986. https://dl.acm.org/doi/10.5555/645913.671319.

[24] J. A. S. a. D. T. J. Huang, “On using priority inheritance in real-time databases,” in Twelfth. IEEE Real-Time Systems Symposium, 1991. http://doi.org/10.1109/REAL.1991.160376.

[25] R. Guerraoui, “Nested transaction: Reviewing the coherence contract,” Information Sciences, vol. 84, no. 1-2, p. 161–172, 1995.https://doi.org/10.1016/0020-0255(94)00118-U.

[26] K. R. Theo Haerder, “Concepts for transaction recovery in nested transactions,” in ACM SIGMOD, New York, NY, USA, 1987. https://doi.org/10.1145/38713.38741.

[27] H. S. H. a. M. E. E.-S. A. A. EI-Sayed, “Effect of shaping characteristics on the performance of nested transactions,” Information and Software Technology, vol. 43, no. 10, pp. 579-590, 2001. https://doi.org/10.1016/S0950-5849(01)00164-1.

[28] W. u. Haque, “Transaction Processing in Real-Time Database Systems,” Iowa State University, USA, 1993. https://dl.acm.org/doi/10.5555/164148.

[29] R. A. a. H. Garcia-Molina, “Scheduling Real-Time Transactions: A Performance Evaluation,” in 14th International Conference on Very Large Data Bases, San Francisco, CA, USA, 1988. https://dl.acm.org/doi/10.5555/645915.671811.

[30] 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.

Mrs. Meenu is an Associate Professor in the department of Computer Science & Engineering at the Madan Mohan Malaviya University of Technology, Gorakhpur where she has been a faculty member since 2003. She is Chairperson of Women Cell as well as Women Welfare and AntiHarassment Cell. She completed her M.Tech. at Madan Mohan Malaviya University of Technology. She has served as the Session Chair for UPCON-2018 (5th IEEE Uttar Pradesh Section International Conference). She is the author of 64 research papers, which have been published in various National & International Journals/Conferences. She is a reviewer of many International Journals/ Conferences and Editorial Board member of International Journals. She is also member of many Professional Societies. Her research interest lies in the area of Distributed Real Time Database Systems. She has collaborated actively with researchers in several other disciplines of computer science, particularly machine learning.