Lock Prediction for Zero-Downtime Database Encryption

Mohamed Sami Rakha
Toronto Metropolitan University
Toronto, ON, Canada
rakha@torontomu.ca

Adam Sorrenti
Toronto Metropolitan University
Toronto, ON, Canada
adam.sorrenti@torontomu.ca

Greg Stager
IBM Canada Lab
Toronto, ON, Canada
gstager@ca.ibm.com

Walid Rjaibi
IBM Canada Lab
Toronto, ON, Canada
wrjaibi@ca.ibm.com

Andriy Miranskyy
Toronto Metropolitan University
Toronto, ON, Canada
avm@torontomu.ca

ABSTRACT

Modern enterprise database systems face significant challenges in balancing data security and performance. Ensuring robust encryption for sensitive information is critical for systems' compliance with security standards. Although holistic database encryption provides strong protection, existing database systems often require a complete backup and restore cycle, resulting in prolonged downtime and increased storage usage. This makes it difficult to implement online encryption techniques in high-throughput environments without disrupting critical operations.

To address this challenge, we envision a solution that enables online database encryption aligned with system activity, eliminating the need for downtime, storage overhead, or full-database reprocessing. Central to this vision is the ability to predict which parts of the database will be accessed next, allowing encryption to be applied online. As a step towards this solution, this study proposes a predictive approach that leverages deep learning models to forecast database lock sequences, using IBM Db2 as the database system under study. In this study, we collected a specialized dataset from TPC-C benchmark workloads, leveraging lock event logs for model training and evaluation. We applied deep learning architectures, such as Transformer and LSTM, to evaluate models for various table-level and page-level lock predictions. We benchmark the accuracy of the trained models versus a Naive Baseline across different prediction horizons and timelines.

The study experiments demonstrate that the proposed deep learning-based models achieve notable improvements, with average accuracy reaching up to 49% for table-level lock prediction and 66% for page-level prediction, outperforming the Naive Baseline. By anticipating which tables and pages will be locked next, the proposed approach is a step toward online encryption, offering a practical path toward secure, low-overhead database systems.

Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/mbrotos/lock-pred.

1 INTRODUCTION

Efficient database management is essential for ensuring high performance transaction processing, particularly in enterprise-scale database management systems (DBMS) such as IBM Db2 [5]. One of the most pressing challenges for modern organizations is ensuring robust data security through encryption. As cyber threats become more sophisticated and data privacy regulations become stricter, encrypting databases has become an essential requirement for organizations handling sensitive information [2]. The urgency is further amplified by the impending threat of quantum computing, which has the potential to break widely used encryption schemes [43, 44].

Database encryption can be applied at different levels, including file system encryption, tablespace encryption, and column-level encryption [32]. However, these approaches often require trade-offs between security, storage efficiency, and performance. File-system encryption, for instance, does not fully secure data stored at rest within a database, while column-level encryption, though highly granular, can degrade performance due to computational overhead. The most effective approach to balancing security and efficiency is holistic database encryption, in which the entire database is encrypted at once. This method ensures comprehensive protection without requiring significant application changes [22].

However, holistic database encryption presents a major challenge: online encryption or continuous re-encryption of existing databases while maintaining zero downtime and avoiding substantial storage overhead [27, 28]. Currently, enabling online encryption on an operational database often requires a costly and time-consuming backup and sometimes a full restore process, making the database temporarily unavailable for queries [14]. The primary technical challenge is to online encrypt or re-encrypt the entire database without disrupting active transactions or requiring additional storage resources.

To address this challenge, we envision a solution for online database encryption that applies encryption in sync with actual system activity, avoiding the need for full database downtime or added storage overhead. A key component of this solution is the ability to anticipate which parts of the database will be accessed and locked next, allowing encryption tasks to be scheduled just in time. As a step towards this solution, this study explores a deep learning-based approach for predicting database lock sequences in IBM Db2. The ability to forecast which tables or data pages will be locked next may enable proactive encryption strategies, ensuring data security while minimizing performance degradation [12, 16, 46]. Specifically, we propose a Transformer-based model [26] and an LSTM-based model [30] to predict (1) the next table to be locked and (2) the next data page to be locked. To evaluate the predictive capabilities of these models, we conduct experiments across multiple forecasting horizons, specifically assessing their performance in predicting sequences of the next 2, 3, and 4 locks. This multi-horizon analysis provides insights into the models' scalability and effectiveness in capturing longer-term dependencies in lock sequences. We generalize predictions across multiple lock types (e.g., shared, exclusive, intent) [6].

For benchmarking purposes, we compare the performance of our deep learning models against a Naive Baseline that assumes the next lock will be identical to the previous one. The lock prediction is a step toward achieving high-performance database management systems that proactively mitigate encryption-induced overheads such as downtime and data storage. In this study, we address the following research questions:

RQ1: **How can we predict the next table locked?** For table-level lock prediction, the deep learning models outperform the Naive Baseline across all horizons, capturing complex locking patterns more effectively.

RQ2: **How can we predict the next data page locked?** For page-level lock prediction, the global deep learning models demonstrate higher accuracy and greater stability. Surprisingly, the Naive Baseline remains highly competitive when compared to local deep models, often matching or surpassing their performance in certain scenarios.

RQ3: **How does the lock prediction changes?** The results show that lock prediction accuracy remains stable over time, with no significant degradation across different forecasting horizons.

This study contributes to the growing body of work on intelligent database management, with a step towards online encryption with zero downtime and storage overhead. In addition to online encryption, the findings of this research have significant implications for database performance optimization, proactive concurrency control, and adaptive encryption scheduling. By integrating predictive lock management into Db2, enterprises can achieve workload-aware caching, dynamic deadlock avoidance, and encryption-aware access control — advancing the vision of self-tuning, security-enhanced databases [34]. The contributions of this study can be summarized as follows.

- **Deep Learning for Lock Prediction:** We explore the performance of deep learning models to predict database lock sequences. We benchmark the reported accuracy against a simple Naive Baseline for lock predictions.
- **Support Different Lock Levels:** We build prediction models for different lock levels (table-level and page-level), ensuring a step toward online encryption while maintaining performance efficiency.
- **Empirical Insights:** We validate our approach on real-world Db2 traces and TPC-C [36] benchmark workload, demonstrating the models' accuracies in predicting table-level and page-level locks.
- **Temporal Evolution of Lock Patterns:** We analyze how lock prediction accuracy and patterns change over time, providing insights into how lock predictions react to workload dynamics.

The rest of the paper is organized as follows. Section 2 reviews related work on database lock prediction, workload modelling, and buffer pool optimization. Section 3 outlines the methodology, including data collection, feature extraction, and model selection. Section 4 presents the experimental results and evaluates the performance of the proposed predictive models. Section 5 discusses the implications of the findings, while Section 6 addresses key threats to validity, limitations, and directions for future research. Finally, Section 7 concludes the paper by summarizing the main contributions.

2 RELATED WORK

The prediction of data access patterns has been a subject of interest in database management, particularly in optimizing concurrency control, buffer management, and security mechanisms [4, 9, 10, 20, 31]. Recent advancements in machine learning have enabled more sophisticated approaches to forecasting lock behavior, facilitating proactive concurrency control and workload-aware resource allocation. Several studies have investigated data access prediction and resource management in relational database systems, leveraging machine learning techniques to enhance performance, mitigate contention, and strengthen security mechanisms [3, 16, 21, 25, 27]. This section reviews relevant research in the domains of database lock prediction, workload modeling, and adaptive buffer pool optimization.

2.1 Lock Contention and Concurrency Control

Traditional concurrency control mechanisms such as two-phase locking (2PL) and timestamp-based protocols have been widely used to manage locks in relational databases [1]. However, these approaches are based on static or rule-based policies that do not adapt to dynamic workloads. Recent advances in machine learning have enabled predictive models that anticipate lock requests to mitigate contention and deadlocks. Lock contention is a critical issue in database management, affecting transaction throughput and system efficiency [25, 35, 39].

Yu and Jagadish [42] analyzed resource contention in multitenant databases, highlighting its impact on performance. McWherter et al. [23] examined the online transaction processing (OLTP) locking patterns and introduced the Preempt on Wait (POW) policy to enhance lock management. By statistically characterizing lock wait times, POW selectively preempts lower-priority transactions to reduce contention. This approach enhances lock scheduling efficiency and minimizes delays for high-priority transactions. Tian et al. [35] presented a novel approach to managing lock contention in transactional databases. The authors introduce the Largest-Dependency-Set-First (LDSF) algorithm, which prioritizes transactions that block the most others, thereby reducing overall transaction latency. The study presents a formal analysis of lock scheduling challenges and demonstrates that implementing LDSF in a real-world database management system can lead to significant performance improvements. Zhou et al. [38] introduced a predictive transaction scheduling approach designed to mitigate lock contention in databases. Their method uses historical workload patterns to forecast transaction conflicts and optimize execution order, thereby reducing the likelihood of contention-related performance degradation.

Gaffney et al. [13] introduced DIBS, a modular transaction isolation service that utilizes optimized predicate locking to replace monolithic storage managers. By locking logical predicates instead of physical data, DIBS reduces overhead and scales to 10.5 million transactions/sec while cutting SQLite writes by 90% (3x throughput) and MySQL row contention by 40%. Gaffney et al.

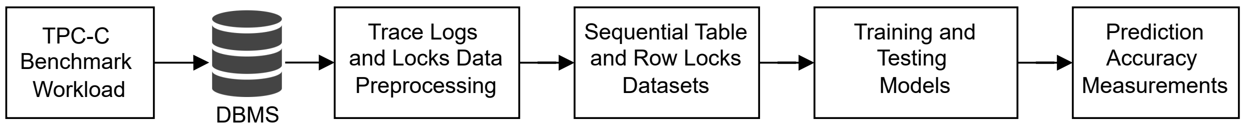
TPC-C Benchmark Workload
DBMS
Trace Logs and Locks Data Preprocessing
Sequential Table and Row Locks Datasets
Training and Testing Models
Prediction Accuracy Measurements

Figure 1: Overview of the lock prediction pipeline.

demonstrated practical, high-performance isolation via decoupled, predicate-aware design. Wang et al. [39] introduced Polyjuice a machine learning approach to optimize database concurrency control by dynamically selecting locking strategies based on transaction clusters and workload patterns. Using reinforcement learning reduces contention by predicting transaction conflicts and adaptively scheduling them, indirectly minimizing unnecessary lock waits.

2.2 Workload Modeling for Predictive Locking

Effective workload modeling is essential for predicting database access patterns and optimizing performance. Previous research has explored the characterization of workloads using statistical and machine learning techniques. For example, Martin et al. [21] developed a workload forecasting framework for autonomic database management systems, emphasizing the importance of understanding workload behavior for system optimization.

Building on this foundation, analytical models have been proposed to assess the impact of concurrency control mechanisms on system performance. Di Sanzo et al. [7] presented an analytical model of lock-based concurrency control with arbitrary transaction data access patterns, providing insights into how different workloads influence lock contention and overall throughput. Furthermore, Thomasian [34] introduced a heterogeneous data access model for concurrency control, discussing methods to handle high data contention in OLTP systems. This study highlights the need for realistic workload models to predict lock contention and suggests strategies to mitigate performance degradation under heavy load conditions. Workload management in database systems involves monitoring, managing, and controlling query execution to optimize resource utilization and achieve performance objectives. Zhang et al. [46] provide a comprehensive taxonomy of workload management techniques in modern database systems, analyzing commercial and research-based approaches while identifying open challenges in workload forecasting and optimization.

Recent advances in self-driving database systems highlight the potential of predictive workload modeling. For example, Ma et al. [20] proposed a query-driven workload forecasting approach for self-optimizing database management systems, underscoring the critical role of workload prediction in database performance tuning. Shahrivari et al. [29] presented Adaptive Approximate Query Processing (AAQP), which enhances the query processing by predicting future workloads using Recurrent Neural Networks (RNNs) trained on historical query sequences. It dynamically adjusts data synopses to minimize query execution time while maintaining accuracy. Experiments on real-world workloads show that AAQP effectively optimizes performance, approaching near-optimal execution efficiency.

2.3 Buffer Pool Optimization and Prediction

Efficient buffer pool management is crucial for improving database performance by reducing disk I/O operations. Traditional buffer replacement policies, such as Least Recently Used and Least Frequently Used, have been widely studied [24]. However, machine learning approaches have shown promise in outperforming these heuristics.

Li et al. [18] introduced DRL-Clusters, a deep reinforcement learning-based approach to buffer pool management in database systems. Their method dynamically adjusts buffer allocations based on workload changes, improving cache efficiency and overall performance. Similarly, Yang et al. [40] proposed RL-CoPref, a reinforcement learning-based coordinated prefetching controller designed to optimize buffer management by predicting future data accesses, ensuring more effective memory utilization.

Furthermore, Zirak et al. [47] developed SeLeP, a learning-based semantic prefetcher tailored for exploratory database workloads. By analyzing data access patterns, SeLeP enhances prefetching accuracy, leading to improved buffer pool utilization and reduced query latency. These studies demonstrate the effectiveness of predictive models in optimizing memory utilization, aligning with the research goals in forecasting table and page locks to enhance concurrency control in database systems.

Unlike this study, the mentioned studies focus on lock management strategies rather than explicit lock sequence prediction. In the context of IBM Db2, prior studies have examined statistical models and heuristics for lock contention analysis [45]. However, deep learning-based lock prediction remains an underexplored area, particularly for predicting table and page level locks.

3 METHODOLOGY

Our methodology is designed to develop and evaluate deep learning based models for predicting database locks in IBM Db2. This section outlines the key steps involved, including data collection, preprocessing, model architecture, training, and evaluation.

3.1 Data Collection

We collect database lock traces from an IBM Db2 environment running HammerDB [33] TPC-C benchmark workload v. 5.11 [36] while enabling the Db2 trace using the "db2trc" command. TPC-C is a widely used OLTP benchmark designed to simulate a high-volume transaction system, such as an e-commerce or inventory management system. The collected lock traces are for eight tables: customer, district, history, neworder, orderline, orders, stock, and warehouse. The orderline table is the one with the highest number of associated lock data. The trace explicitly captures Db2 functions related to lock operations, including "sqlplrq" (lock request), "sqlplrl" (lock release), and "sqlplrem" (lock removal). For the experimental workload, we apply a TPC-C benchmark configuration with 100 warehouses, 10

Table 1: Distribution of Lock Object Types Collected in Db2

Lock Type	Record Count	Percentage (%)
PAGE	2,702,900	31.00
CATALOG	2,213,913	25.40
TABLE	2,178,913	24.99
VARIATION	669,924	7.68
PLAN	642,945	7.38
SEQUENCE	300,970	3.45
INTERNAL	8,254	0.09
TABLESPACE	4	0.00
Total	8,717,823	100.00

asynchronous clients, and 100 virtual users, generating realistic transactional activity. We collect around 25GB of a trace log that is ready for further processing.

3.2 Data Preprocessing

The collected raw lock trace log undergoes several pre-processing steps to extract different information about each log lock. Preprocessing is performed by parsing the trace log and extracting each required element into a structured format for subsequent steps. Table 1 presents the distribution of the lock types extracted from the processed trace log. Below, we highlight a list of the extracted data for each lock:

- **Lock ID:** A unique identifier assigned to each lock instance in Db2. This helps track individual lock operations and distinguish between different locks in the system.
- **Start Time:** The timestamp that indicates when the lock was acquired.
- **End Time:** The timestamp that indicates when the lock was released.
- **Mode:** Specifies the type of lock acquired, such as Shared (S), Exclusive (X), Intent Exclusive (IX), or Update (U) [6]. The mode determines the level of access allowed while preventing conflicts with other transactions.
- **Lock Object:** The database resource that is being locked, which can be a data page, table, tablespace, or an index. Identifying the locked object helps to understand the contention of the transactions and access patterns.
- **Page ID:** This indicates the specific database page that is locked. Pages typically contain multiple rows. Page-level locking strikes a balance between concurrency and overhead.
- **Table Name:** The name of the table associated with the lock. Knowing which tables are affected by locking can help optimize queries, indexing strategies, and database design to minimize contention. The table-level locks include a table name, while the page-level locks include a table name and a page ID.

In this study, we focus on the table and page locks, which cover 50% of the lock types from Table 1. To prepare the data for model training, we categorize the lock data into two main groups: (1) Table Locks and (2) Page Locks. Table locks comprise all Db2 lock records where the "Lock Object" is a table, while the page locks encompass those where the "Lock Object" is a data page. To ensure our analysis focuses on the TPC-C workload, we filter out system locks associated with the "SYSIBM" schema. After applying this filter, we obtain 1.54 million table locks and 2.66 million page locks.

For each lock type (i.e., Table and Page locks), we construct sequential transaction timelines by organizing lock events into ordered sequences. We sort the locks data by the start time in ascending order. Feature encoding is applied to transform page IDs and table names into numerical representations. Continuous attributes, such as timestamps, undergo normalization, and a time-based train-test split is used to evaluate the model on future transactions.

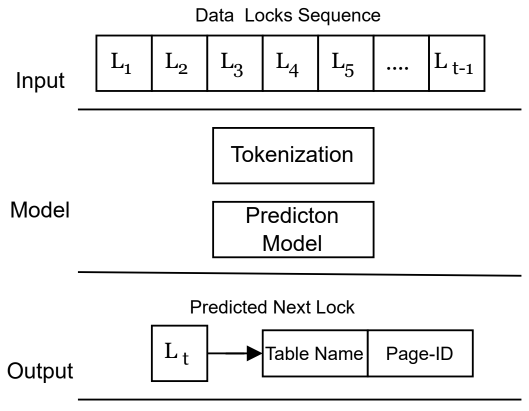

Figure 2: Predicting the next lock in a sequence.

3.3 Model Architecture

In this study, we use deep learning architectures, such as Transformer [37] and LSTM [15], to predict the next locks in a Db2 database system. Figure 2 presents an overview of the prediction of the next lock based on an input of the lock sequence. For both models, the input sequence consists of 25 locks each comprised of one or two tokens (Table *T* and Page ID, respectively), which are first transformed into 128-dimensional dense vector representations using an embedding layer. In the Transformer model, positional encoding is added to these embeddings to preserve the order of tokens in a sequence. The core of the architecture features a Transformer encoder layer that utilizes multi-head self-attention to capture complex, long-range dependencies among lock events. The encoder-only Transformer model we use consists of 8 self-attention heads with a hidden dimension of 512 and a dropout rate of 10%. In contrast, the LSTM model processes the embedded sequences using a recurrent layer [30] capable of learning temporal patterns and dependencies through gated memory cells, which help retain context over longer sequences. Their recurrent nature also makes LSTM models particularly effective for sequential prediction tasks involving variable-length dependencies. The LSTM model uses a hidden dimension of 256 in each gated memory cell.

In both architectures, encoded lock representations are passed through a feedforward network, including a dense layer with Rectified Linear Unit (ReLU) [26, 30] activation, and culminate in a softmax output layer that predicts the next page ID and table name to be locked. This dual-model setup allows us to compare the effectiveness of attention-based and recurrent-based architectures in the context of database lock prediction.

3.4 Model Training

The models were implemented using Python v. 3.11.5, Keras v. 3.3.3, and the TensorFlow v. 2.16.1 backend. To train the model, we preprocess the lock sequences by tokenizing and padding them to a fixed length. A single tokenizer is used for both input and output sequences, ensuring consistent vocabulary mapping. The dataset is chronologically divided into a test set, comprising 30% of the data, and a train set. A validation set, consisting of 20% of these initial training data, is then partitioned off, with the remainder used for model training. This chronological partitioning preserves the temporal order of lock requests to prevent data leakage.

All models were trained using categorical cross-entropy loss and the AdamW optimizer [19], with a constant learning rate of 0.001 and the remaining parameters set to the Tensorflow defaults. The training process involves iterating over the dataset in mini-batches of size 32. To enhance generalization, we incorporate dropout regularization to prevent overfitting. Additionally, a model checkpoint was used to monitor the validation loss for each epoch. The model weights that achieved the lowest validation loss in all 30 epochs were saved and used for the final evaluation. To ensure the robustness of our findings, each experiment was executed for a total of 10 iterations.

For the lock prediction, we adopt two modeling approaches: Global Models and Local Models. We define these model types as follows.

- **Global Model:** A single unified model is trained using data from all tables and predicts the next lock, regardless of the table. For example, in the case of page-level lock, this model learns to predict the next (Table, Page ID) combination in a sequence, enabling it to generalize across different tables and capture inter-table patterns.
- **Local Model:** For this model type, a separate model is trained for each individual database table and tested on the same table. For example, for Table *T*, we train a model using all data from Table *T* and predict the next locks for Table *T*. This is an appropriate type for the page-level locks, where a model is responsible for predicting the next Page ID within its respective table. This approach allows the model to specialize in the unique access patterns and characteristics of each table.

3.5 Model Evaluation

Performance evaluation is performed using multiple metrics, including accuracy, precision, recall, and F-measure, to assess how effectively a model anticipates locking sequences. In the following, we describe the performance metrics [8]:

- **Accuracy:** Measures the correctness of predicted locks.
- **Precision:** Measures how many of the predicted locks for a specific table or page ID are actually correct. It is defined as Precision = $\frac{\text{TP}}{\text{TP+FP}}$, where TP and FP denote the number of true positives and false positives, respectively.
- **Recall:** (also known as sensitivity) Evaluates how many of the actual lock events for a specific table or page ID were correctly predicted. It is calculated as Recall = $\frac{\text{TP}}{\text{TP+FN}}$, where FN represents the number of false negatives.
- **F1-Score:** The F1-Score is the harmonic mean of precision and recall, defined as F1 = $2 \times \frac{\text{Precision}\times\text{Recall}}{\text{Precision+Recall}}$.

These metrics are especially important in multi-class classification tasks like this study, where the number of classes (tables or page IDs) is large. High precision ensures the model does not incorrectly predict locks for the wrong table or page ID (minimizing false alarms), while high recall ensures that the model captures most of the actual lock events (minimizing missed locks). A high F1-Score indicates that the model achieves a good balance between correctly predicting lock events (recall) and minimizing incorrect predictions (precision).

To assess a model's ability to forecast locking sequences across multiple future steps, we use the term *Horizon*. In this study, we evaluate up to four horizons. For instance, a horizon of 3 indicates the model's accuracy in predicting the next three consecutive locks. By default, predicting only the immediate next lock corresponds to Horizon = 1. Overall, the predictions are compared against expected outputs, and the softmax probabilities are analyzed to interpret the model's confidence.

To ensure robustness and account for variance due to random initialization, we train each deep learning model (Transformer and LSTM) over 10 iterations, each using a different random seed. For each prediction horizon (e.g., Horizon 1 through Horizon 4), performance metrics such as accuracy, precision, recall, and F1-score are computed in each iteration. The final reported value for each metric at a given horizon is obtained by averaging the results over the 10 runs. This approach helps smooth out anomalies due to random factors and provides a more reliable estimate of model performance per horizon.

4 RESULTS

RQ1: How can we predict the next table locked?

Motivation. Achieving holistic database encryption without incurring downtime or excessive storage overhead remains a critical challenge for companies striving to comply with regulations such as GDPR, HIPAA, and PCI DSS [11, 27] as well as to ensure quantum-safety. A major bottleneck in this process is the need to encrypt or re-encrypt existing databases without disrupting active transactions. Predicting which table will be locked next is essential to integrate encryption seamlessly into live database operations. Accurate predictions allow for proactive encryption strategies that minimize performance degradation, such as aligning encryption operations with predicted lock sequences to avoid redundant re-encryption or unnecessary delays [28]. Furthermore, forecasting table-level locks is key to ensuring that online encryption processes do not interfere with critical workloads. In large-scale database systems such as IBM Db2, this predictive capability represents a meaningful step toward enabling real-time, online encryption without downtime or significant storage overhead.

Approach. To address the challenge of table-level lock prediction, we structured historical lock sequences to capture dependencies between table accesses. We trained various models on all tables' lock data to learn cross-table dependencies and generalize across the entire database. The trained models leverage the Transformer and

Table 2: Summary for the performance metrics averages for all the Table Lock predictions across the Studied Models for Horizon = 1. The reported values represent the average of the ten modelling repetitions across the metrics.

Model	Accuracy	Precision	Recall	F1
Transformer	0.475	0.488	0.359	0.455
LSTM	0.488	0.496	0.373	0.470
Naive	0.084	0.039	0.039	0.048

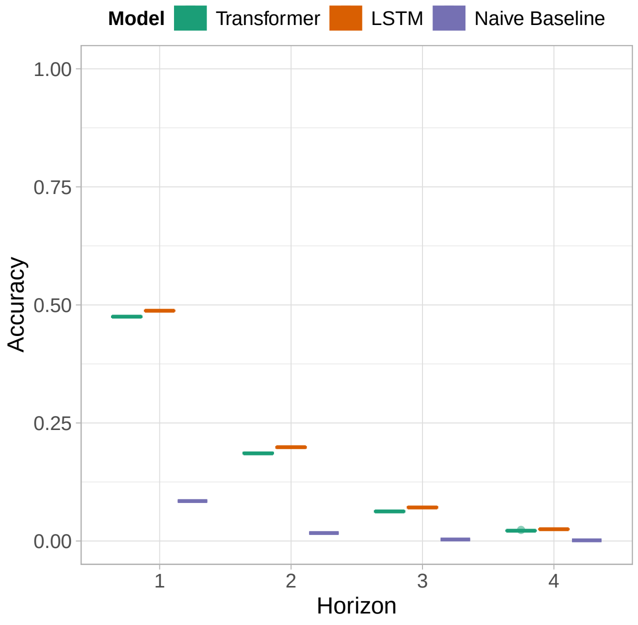

Figure 3: This figure presents the table-level prediction accuracy across different tables and horizons with a comparison of local models and naive models. The model here predicts the sequence of the next Table T to be locked. The figure shows that the deep learning approach outperforms the Naive Baseline.

LSTM architectures described in Section 3, utilizing self-attention mechanisms to model long-range dependencies in lock sequences.

To evaluate performance, we compared the trained models against a native baseline that assumes the next lock is identical to the previous lock: a simple yet competitive heuristic in lock prediction tasks. Additionally, we extended our analysis beyond single-step prediction to study the model's performance across varying prediction horizons, such as predicting the next 2, 3, or 4 locks. This multi-horizon evaluation offers insights into the model's ability to forecast longer sequences, which may be crucial for optimized online database encryption.

Results. We discuss the results of this RQ as presented prediction results in Table 2 and Figure 3.

The prediction models consistently outperform the Naive Baseline. Table 2 presents the performance metrics for both deep learning models (Transformer and LSTM) compared to a Naive Baseline predictor. The results clearly show that both the Transformer and the LSTM models significantly outperform the Naive Baseline across all metrics. Deep learning models achieve higher accuracy, indicating a better overall prediction of the next lock. They also demonstrate improved precision and recall, suggesting that they are more effective at both: correctly identifying relevant locks and minimizing false positives or missed predictions. The F1-Measure, which balances precision and recall, further confirms the superior performance of the deep learning models versus the Naive Baseline.

These improvements validate the hypothesis that deep learning-based models can effectively leverage the sequential nature of database locks to make more informed predictions. In particular, the consistent superiority across multiple evaluation metrics underscores the utility of incorporating temporal patterns (captured by the Transformer's self-attention mechanism and LSTM's memory units) into predictive models for database lock behaviour. The results indicate that both the Transformer and LSTM models have the potential to serve as practical components of intelligent DBMS subsystems, particularly for online encryption of enterprise databases like DB2 without downtime or storage overhead.

The prediction performance is consistent across all horizons. Figure 3 provides a comparative analysis of predication performance across various horizons. The results clearly indicate that both the Transformer and LSTM models significantly outperform the Naive Baseline across all prediction horizons (e.g., Horizon 1, Horizon 2) and all evaluated tables. The evaluated deep learning models demonstrate a strong and consistent ability to anticipate future locks, even as the temporal distance between the observed and predicted locks increases.

For example, at Horizon 1, where the goal is to predict the immediate next lock, the Transformer model achieves an average accuracy exceeding that of the Naive Baseline, particularly on critical tables such as `customer`, `history`, and `orders`. In contrast, the Naive Baseline struggles to reach 9.1% accuracy in most cases. This performance gap becomes even more pronounced at Horizons 2-4, where the models' accuracy deteriorates. However, the Transformer and LSTM models retain a higher level of predictive accuracy, illustrating their capacity to learn and model temporal dependencies beyond the immediate future. Overall, the results reinforce the value of using sequence-aware deep learning models in lock prediction tasks. Unlike the Naive Baseline, which lacks memory or pattern recognition.

RQ2: How can we predict the next data page locked?

Motivation. While predicting table-level locks helps to optimize database-wide encryption strategies, page-level locks (represented by page IDs in Db2) offer finer control over encryption timing and transaction efficiency. In enterprise environments, where minimizing downtime and storage overhead is crucial for enabling encryption on existing databases, accurately forecasting the next page to be locked can help schedule encryption operations without interfering with active transactions. Page-level encryption must

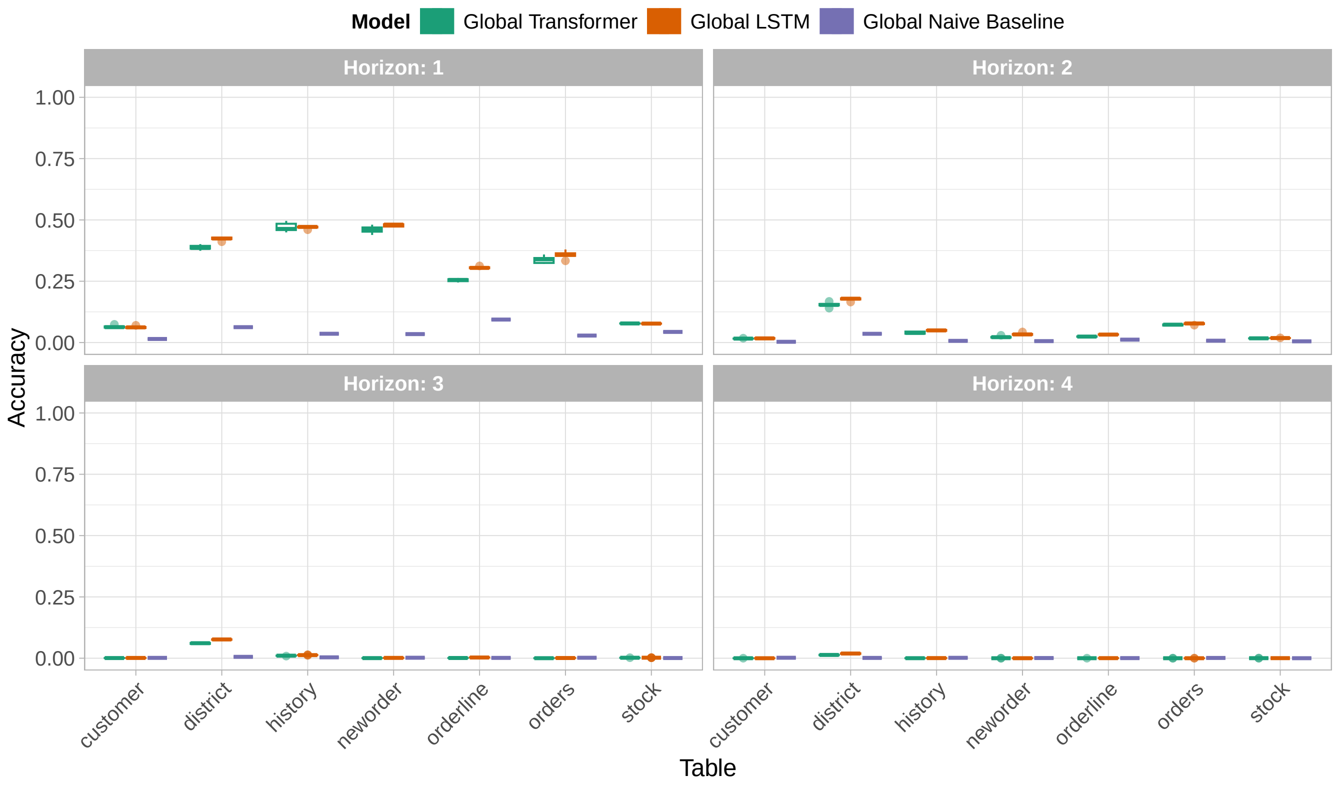

Figure 4: This figure presents the page-level prediction accuracy across different tables and horizons. The figure compares the performance of the global models across all tables and horizons. The Transformer and LSTM models outperform the Naive Baseline across all the tables and horizons. The model here predicts a sequence of next locks as a combination (Table T, Page ID Bin).

be carefully aligned with ongoing access patterns to avoid excessive performance overhead, redundant encryption operations, and unnecessary lock waiting times. By predicting the next data page to be locked, the database can preemptively encrypt data pages just before they are accessed, ensuring security policies are enforced without impacting system responsiveness. This RQ research is a crucial step toward achieving fine-grained online encryption without downtime or space overheads.

Approach. Building on the methodology of RQ1, we extend our deep learning–based framework to predict page-level locks represented by page identifiers (Page IDs) in Db2. Unlike RQ1, which focuses on table-level dependencies, this approach structures lock sequences to include both the table name and data page as tokens, enabling the model to learn fine-grained locking patterns. Specifically, the input and output sequence consists of pairs (Table T, Page ID Bin), and the model predicts the next token in this sequence (see Figure 2). We developed two distinct models: (1) a global model trained on all tables to learn cross-table dependencies and generalize throughout the database, and (2) a local model trained individually for each table to capture table-specific locking patterns (see Section 3.4). Both models leverage Transformer architectures, utilizing self-attention mechanisms to model long-range dependencies in lock sequences. Out of the eight tables mentioned in Section 3, we filter out the Warehouse table since it has only a single page ID value for all the page-level locks.

To enhance prediction performance and reduce the complexity of the output space, we applied a binning strategy to the page IDs. Instead of predicting raw page ID values, we grouped them into 10 bins based on their distribution within each table. This transformation reduces noise and enables the model to generalize better across similar page access patterns, particularly in cases where the page ID values are sparse or highly variable. This binning-based formulation enables the models to capture page-level locking behaviour more effectively, while keeping the prediction space tractable and improving model convergence in diverse database workloads. For performance evaluation, we use the same metrics as in RQ1. Additionally, we conduct a multi-horizon analysis to assess the model's ability to predict sequences of 2, 3, or 4 locks.

Results. The Results of this RQ are illustrated in Figures 4 and 5. We discuss the findings below.

The Global deep learning models outperform the Naive Baseline. Figure 4 presents the page-level prediction accuracy across different tables and prediction horizons. The results clearly show that both the Global Transformer and Global LSTM models consistently outperform the Naive Baseline across all tables and horizons, particularly at Horizon 1. At this short horizon, both deep models

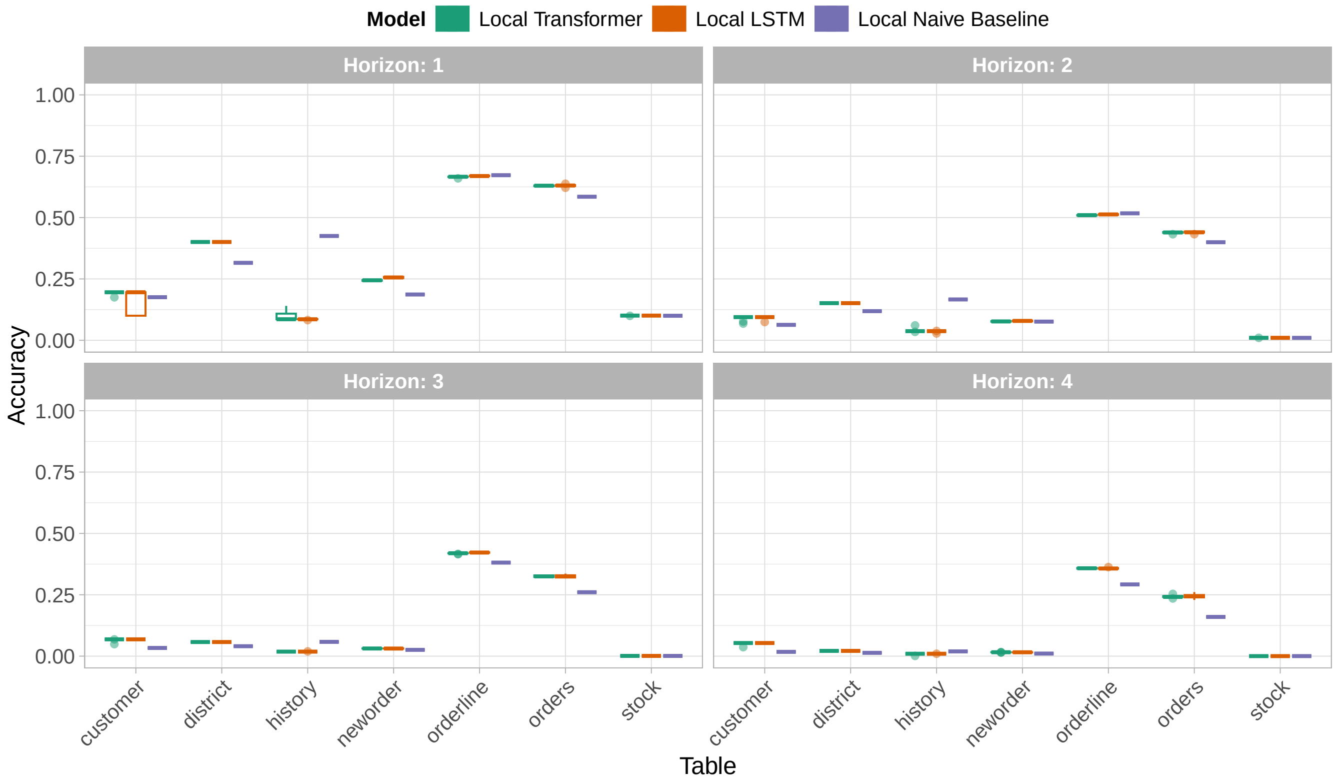

Figure 5: This figure presents the local models' prediction accuracy for page-level across different tables and horizons. The figure compares the prediction for the Transformer and LSTM models versus the Naive Baseline. The results show that the Naive Baseline outperforms the deep learning models in some cases. The model here predicts a sequence of next locks as Page ID Bin. Predicting only the next lock Page ID Bin is less challenging than the Global Model (compare with the Global models' Figure 4).

achieve noticeably higher accuracy for most tables, indicating their strong ability to model immediate next-lock patterns.

As the prediction horizon increases (Horizons 2–4), accuracy declines for all models, reflecting the growing difficulty of longer-term sequence prediction. However, even as performance drops, the deep learning models, especially the LSTM model, continue to outperform the Naive Baseline in almost all cases. The gap between the deep models and the Naive Baseline is especially prominent for tables such as `district` and `orders`, where the deep learning models demonstrate higher accuracy and robustness.

Notably, the variance of the deep models remains relatively low at Horizon 1 but increases slightly with increasing horizon depth, although it remains generally more stable than the Naive Baseline. No table shows the Global Naive Baseline outperforming the deep models at any horizon, suggesting that learning temporal and contextual locking patterns provides a strong advantage. These findings reinforce the suitability of deep learning models for predicting next-lock events at a global scope, with the Transformer and LSTM showing particular promise due to their consistent performance across both tables and horizons. One explanation for the low performance at large horizons (2-4), is due to the fact that the models predict a pair (Table T, Page ID) and not a single value as in RQ1.

Surprisingly, for a Local Model, the Naive Baseline shows a competitive performance. Figure 5 presents the page-level prediction accuracy across all tables and four using local models. In this case, every table will have its own trained model, and the prediction would focus only on the Page ID Bin (because the table name T in the "Table T, Page ID Bin" tuple stays constant). While the Local Transformer and Local LSTM models generally show improved accuracy over the Naive Baseline in structured tables (such as `orderline` and `orders`) — especially at Horizon 1 — the Naive Baseline often remains surprisingly competitive, and in some cases, even outperforms the deep models. For example, at Horizon 1, the Naive Baseline performs similarly or better than both deep models on `district`, `history`, and `neworder`. This trend persists and becomes more apparent in longer horizons (Horizon 2 to 4), particularly in low-variability or highly repetitive tables.

As the prediction horizon increases, all models exhibit a decline in accuracy due to the increased complexity of forecasting multiple future page accesses. However, the performance of deep models does not always scale favorably with horizon length. In some tables (e.g., `district`, `history`, `stock`), the Local Naive Baseline

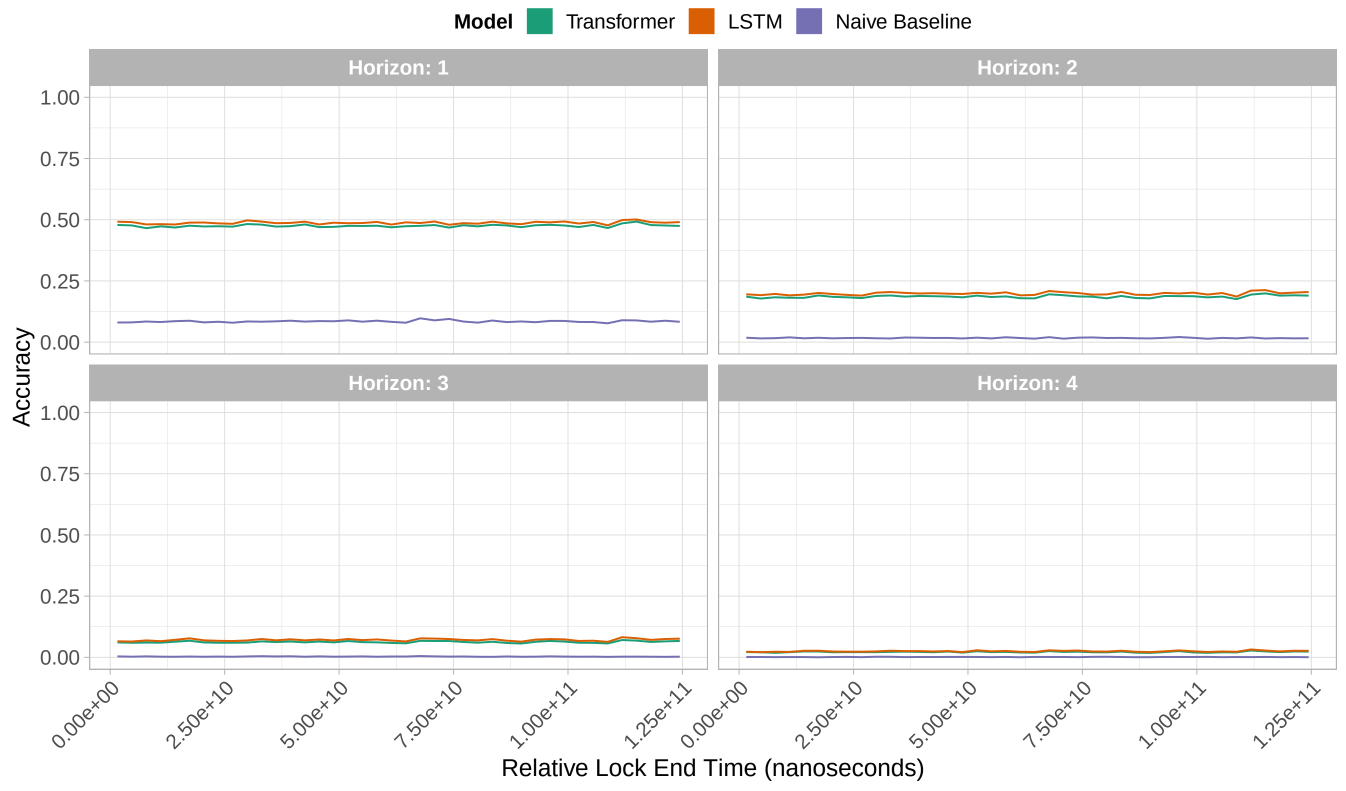

Figure 6: This figure presents the table-level prediction accuracy over the workload period and horizons. The figure highlights whether the performance accuracy changes over time. As a table-level prediction model, the model here predicts the sequence of locks as the next Table T to be locked.

maintains or exceeds the performance of both the Transformer and LSTM models across all horizons. This highlights that, in the local context, simple recency-based heuristics can be robust and effective, especially under consistent or repetitive locking behavior.

Overall, these results emphasize the unexpected resilience of the Naive Baseline in local sequence prediction for Page ID Bins. Despite its simplicity and lack of learning overhead, it can rival and sometimes outperform more complex models, particularly in scenarios characterized by stability and routine. This suggests that for certain OLTP environments, lightweight heuristics may offer a practical and resource-efficient alternative to deep learning approaches.

RQ3: How does the lock prediction changes?

Motivation. Locking patterns in a database system are not static; they evolve based on workload fluctuations, query patterns, and system optimizations. Understanding how lock predictions shift over time is crucial for ensuring the long-term reliability of predictive models, particularly in the context of enabling live encryption with zero downtime and storage overhead. If lock sequences exhibit significant temporal drift, models may require adaptive learning mechanisms to maintain accuracy and effectiveness. Additionally, analyzing temporal changes in lock predictions can help identify workload trends, detect anomalies, and refine encryption scheduling strategies to minimize performance impact. By addressing this research question, we aim to evaluate the robustness of lock prediction models over time and explore a possible need for continuous optimization of the trained lock prediction models.

Approach. To analyze how lock predictions evolve over time, we explore the separate models for table-level (see RQ1) and page-level locks (see RQ2), capturing their distinct access patterns. This RQ methodology involves segmenting the DB2 lock traces into time-based windows, allowing us to assess temporal shifts in prediction accuracy and model performance. We track key metrics discussed in Section 3, across different time periods to identify potential drift in locking patterns. To further investigate variations, we compare the models' performances across different horizons.

Results. The Results of this RQ are illustrated in Figures 6 and 7. We discuss the findings below.

Table level prediction exhibits a temporal stability. Figure 6 illustrates the table-level prediction accuracy over time and across the four forecasting horizons (Horizon 1 to 4). The x-axis represents the relative lock end time in nanoseconds, while the y-axis denotes the percentage of correctly predicted locks within each time step. Each subplot corresponds to a specific prediction horizon. Across all horizons, the temporal prediction performance remains remarkably stable over time for all three models: Transformer, LSTM, and

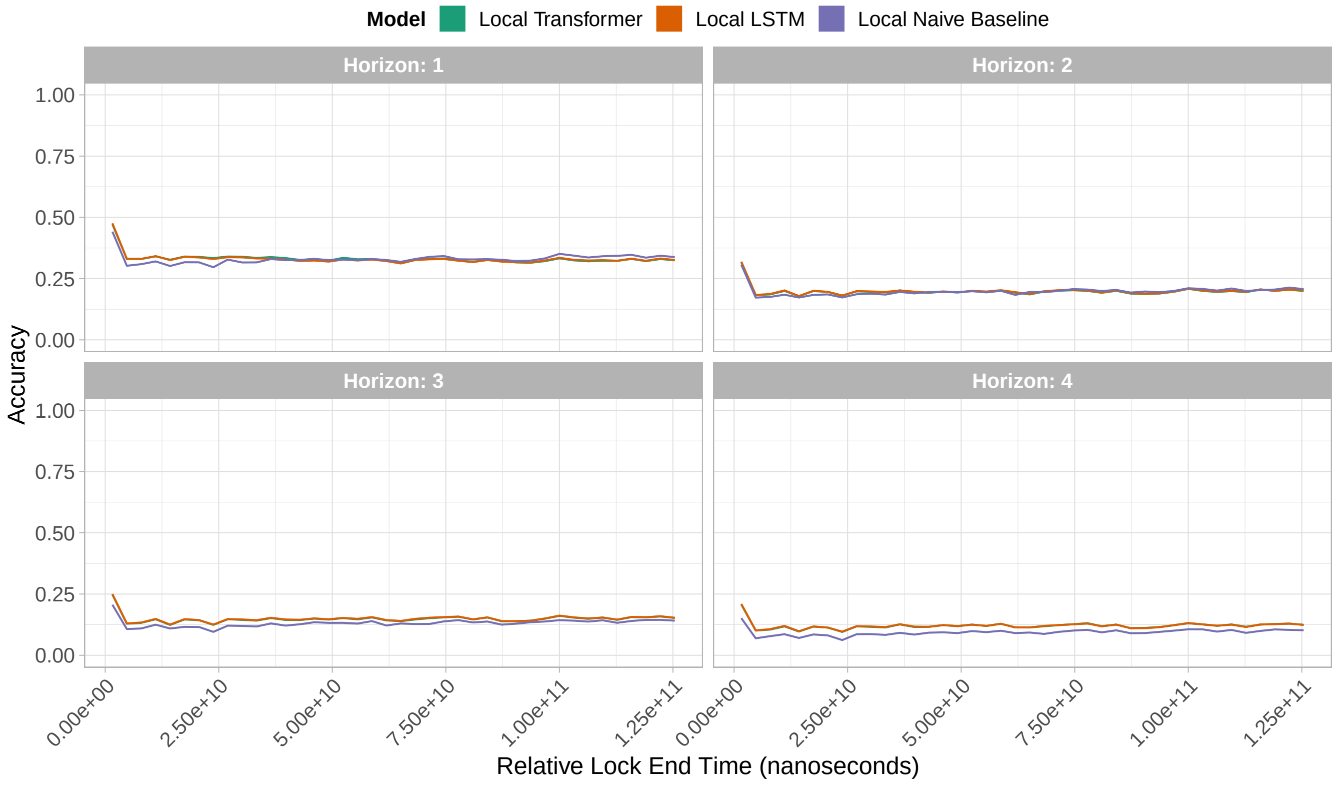

Figure 7: This figure presents the local page-level prediction accuracy over the workload period and horizons. The figure highlights whether the performance accuracy changes over time.

the Naive Baseline. No substantial drops or spikes in accuracy are observed, even as the workload progresses, suggesting that locking patterns do not experience strong concept drift over time. Some fluctuations occur only near the end of the timeline, where the number of data points diminishes. Figure 8 presents a data distribution example over time, highlighting the fluctuations at the start and end of the testing period. These results suggest that the models, once trained, generalize effectively over time and do not require frequent retraining to sustain performance.

Importantly, the relative ranking among models is also consistent throughout time and across horizons. LSTM and Transformer models significantly outperform the Naive Baseline, especially at Horizons 1 and 2, maintaining a 50% accuracy rate in Horizon 1 versus 10% for the Naive Baseline. Even at longer horizons (3 and 4), the gap remains evident, albeit at lower absolute accuracy levels, indicating increased prediction difficulty.

Overall, the prediction results demonstrate both accuracy and temporal resilience, reinforcing their potential practicality for online encryption under dynamic workloads.

Page-level prediction maintains temporal consistency. Figure 7 presents the page-level prediction accuracy across the workload period for Horizons 1 and 4. Similar to Figure 6, the x-axis represents relative lock end time in nanoseconds, while the y-axis shows prediction accuracy. Three models are compared: Local LSTM, Local Naive Baseline, and Local Transformer. The results demonstrate stable temporal performance for all models, with no significant fluctuations in accuracy as the workload progresses. We notice that the deep learning models maintain a performance advantage throughout the entire observation period, consistently outperforming the Naive Baseline model. The big drop in performance that happens at the beginning of the timeline is due to a shortage of data points at the start of the workload (See Figure 8 as an example of workload distribution). Due to space constraints, we omit the figure for global page-level models; however, they exhibit similar trends, further supporting the observation of temporal stability in lock prediction. This consistency suggests that page-level locking patterns remain relatively stationary over time, showing no clear signs of concept drift. As for table-level prediction, these results highlight that page-level lock prediction shares the temporal stability characteristics while maintaining the consistent performance advantage of deep learning models across all horizons.

5 IMPLICATIONS

The results of our study demonstrate the feasibility of using deep learning models for database lock prediction, offering several practical implications for database performance optimization, security, and resource management.

(1) Improved Concurrency Control: By accurately predicting which data pages and tables will be locked next, database management systems (DBMS) can implement proactive concurrency control

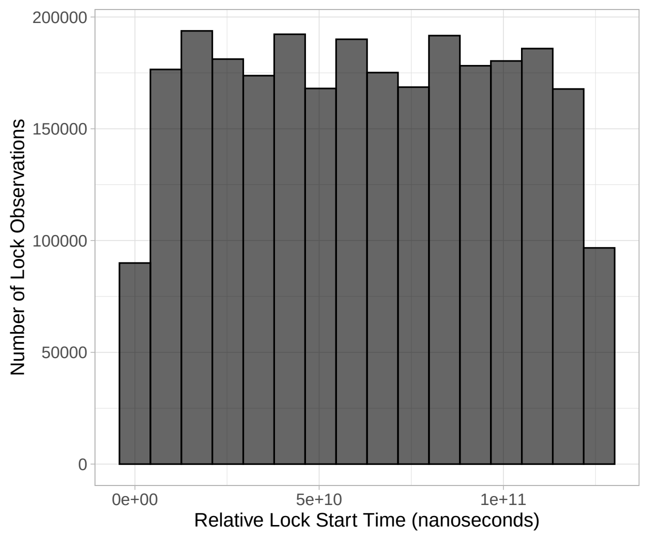

Figure 8: Distribution of table-level testing observations over time (in nanoseconds) based on relative lock start time. The histogram shows the number of recorded table lock instances in fixed-width time intervals for horizon = 1. The relatively uniform distribution indicates consistent lock activity across the monitored testing period, with slight variations near the boundaries.

strategies. This can reduce contention, minimize transaction delays, and lower the likelihood of deadlocks. Future work could explore adaptive locking mechanisms that adjust transaction scheduling based on predicted lock conflicts.

(2) Optimized Buffer Pool Management: The study results suggest that predictive caching can be used to preload frequently accessed or soon-to-be-locked data pages into memory, reducing disk I/O and improving query responsiveness. Similarly, predictions can prevent premature eviction of pages that are likely to be reused shortly. These capabilities offer strong implications for buffer pool optimization in systems like Db2, aiding with data-driven prefetching and eviction policies based on lock access patterns. However, it is important to note that our work focuses primarily on write operations, whereas buffer pool management typically considers both reads and writes. As such, more research is needed to fully integrate predictive locking into comprehensive buffer pool strategies.

(3) Enhanced Security and Access Control: In security-sensitive environments, predicting locks can help optimize online encryption and decryption of sensitive database content. By anticipating access patterns, encryption overhead can be reduced, ensuring that critical data remains protected without excessive performance costs. Future research could integrate lock prediction with dynamic security policies, adjusting encryption strategies based on predicted access patterns.

(4) Generalizability to Other DBMS and Workloads: Although our study focuses on IBM Db2, the methodology may be extended to other RDBMS, such as MySQL, MS SQL, Oracle DB, and PostgreSQL. Potentially, the approach can be adapted to different database workloads, including OLTP, OLAP (Online Analytical Processing), and hybrid workloads.

(5) Potential Integration with AI-Driven Optimization: With the rise of AI-assisted database management, lock prediction models could be integrated into self-tuning database systems. By combining lock prediction with other AI-driven optimizations, such as query plan adjustments and index tuning, databases can achieve higher autonomy and efficiency [17, 20].

(6) Trade-off between using local and global models: A global model simplifies the machine learning operations by requiring only one model to be trained and maintained, which can then be applied across all tables. This approach reduces complexity and overhead in deployment and monitoring. However, local models typically offer better performance, as they are tailored to the specific characteristics of each individual table. The downside is that each table requires its own separate model, increasing the workload for training, deployment, and ongoing maintenance.

(7) Naive vs. Deep Learning Model: As shown in the RQ3 answer, the Naive Baseline sometimes performs only slightly worse than more sophisticated machine learning models. This makes it an attractive option for integration into the database engine code, as it introduces minimal computational overhead (especially compared to running machine learning models). However, if accuracy is critical, machine learning models are more suitable.

While our model achieves promising results, future research could explore several avenues to further enhance lock prediction in database management systems. Incorporating additional workload features into model training, such as query execution plans and transaction logs, could improve predictive accuracy by providing richer contextual information. Additionally, developing real-time adaptive models that dynamically adjust to workload shifts would enhance robustness and applicability in dynamic database environments. Another key direction is the integration of lock prediction within DBMS query optimizers, enabling real-world deployment and validation of predictive locking mechanisms. By addressing these challenges, lock prediction models could evolve into a fundamental component of next-generation intelligent database management systems, optimizing performance and resource utilization.

6 THREATS TO VALIDITY

While our study demonstrates the effectiveness of deep learning models in predicting database locks, several potential threats to validity must be considered. We categorize these threats into internal, external, and construct validity concerns as per [41].

6.1 Internal Validity

Internal validity refers to factors that may introduce bias or affect the correctness of our results. One potential threat may arise from the use of a specific IBM Db2 environment, which may introduce data collection bias and cause the models to overfit to environment-specific workload patterns. Additionally, the choices made during feature selection and encoding may fail to capture important latent variables that influence lock behavior, thereby affecting the predictive accuracy of the models. Another factor influencing internal validity is the tuning of model hyperparameters, which may unintentionally bias the models toward particular characteristics of the training data, limiting generalizability even within the same environment. While we attempt to mitigate these concerns through consistent data preprocessing and multiple training iterations using different random seeds, these limitations should be considered when interpreting the results.

6.2 External Validity

External validity concerns the generalizability of our findings beyond the studied environment. Our approach is tested on IBM Db2, and its applicability to other DBMS, such as MySQL or Oracle, remains to be validated. Furthermore, the scalability of our model in high-load transactional systems and its adaptability to different workload types, such as OLTP versus OLAP, require further investigation. However, the same empirical evaluation and analysis methods can be applied to other software products, provided that well-designed and controlled experiments are conducted.

6.3 Construct Validity

Construct validity assesses whether our study accurately measures what it intends to. Predicting locks is a proxy for performance improvements; however, additional experiments are needed to measure the real-world benefits, such as reduced transaction latency. Our success metrics focus on accuracy and sequence prediction, but alternative evaluations, such as query execution time reduction, could offer deeper insights. Additionally, the model's effectiveness over time must be validated against evolving database workloads.

6.4 Mitigation Strategies

To address these threats, future research should validate our model on multiple DBMS platforms, conduct real-world deployment testing, and implement adaptive learning techniques to handle shifting transaction behaviors. These strategies will help refine our approach and improve the reliability of predictive models for database lock management.

7 CONCLUSION

In many enterprise settings, holistic database encryption provides strong protection but may require prolonged downtime and increased storage overhead. These demands make it difficult to implement online encryption in high-throughput database environments without disrupting critical operations. To address this challenge, we envision a solution that enables online database encryption aligned with system activity, eliminating the need for downtime, excess storage, or full-database reprocessing. Central to this vision is the ability to predict which parts of the database will be accessed next, enabling encryption to be applied incrementally and just in time.

This study takes a step toward this solution by proposing a deep learning-based approach to predict database locks in IBM Db2 workloads, a key step in online database encryption with minimal overhead. By addressing key research questions, we demonstrated the feasibility of forecasting both the next table and data page to be locked while maintaining prediction stability over time. Our findings highlight that historical lock sequences can be leveraged to improve the accuracy of lock predictions, providing a potential pathway toward enabling online database encryption with minimal performance overhead or service disruption.

The implications of our work extend to secure and efficient database management, including adaptive encryption scheduling, contention-aware transaction processing, and workload-driven key management. By anticipating lock patterns, database systems can dynamically optimize encryption operations, ensuring data security while minimizing disruptions to ongoing transactions. Future research can explore the integration of our approach in real-time database environments to validate its impact on live encryption processes and workload. Further optimizations in model performance and computational efficiency could help bridge the gap towards optimal online database encryption features.

ACKNOWLEDGMENTS

This work was partially supported by the Natural Sciences and Engineering Research Council of Canada (Grant # RGPIN-2022-03886) and the IBM Centre for Advanced Studies (Project #1121). The authors thank the Digital Research Alliance of Canada for providing computational resources.

REFERENCES

[1] Philip A Bernstein, Vassos Hadzilacos, Nathan Goodman, et al. 1987. *Concurrency control and recovery in database systems*. Vol. 370. Addison-wesley Reading.

[2] Elisa Bertino and Ravi Sandhu. 2005. Database security-concepts, approaches, and challenges. *IEEE Transactions on Dependable and secure computing* 2, 1 (2005), 2–19.

[3] Yu Chen, Yong Zhang, Jiacheng Wu, Jin Wang, and Chunxiao Xing. 2021. Revisiting data prefetching for database systems with machine learning techniques. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 2165–2170.

[4] Asit Dan, Philip S Yu, and Jen-Yao Chung. 1995. Characterization of database access pattern for analytic prediction of buffer hit probability. *The VLDB Journal* 4 (1995), 127–154.

[5] DB2. 1983. DB2 is a relational database with advanced features developed by IBM. https://www.ibm.com/db2.

[6] DB2. 2025. Lock modes and compatibility. https://www.ibm.com/docs/en/db2-for-zos/12.0.0?topic=locks-lock-modes-compatibility.

[7] Pierangelo Di Sanzo, Roberto Palmieri, Bruno Ciciani, Francesco Quaglia, and Paolo Romano. 2010. Analytical modeling of lock-based concurrency control with arbitrary transaction data access patterns. In *Proceedings of the first joint WOSP/SIPEW International Conference on Performance Engineering*. 69–78.

[8] Kingma Diederik. 2014. Adam: A method for stochastic optimization. *(No Title)* (2014).

[9] Jennie Duggan, Ugur Cetintemel, Olga Papaemmanouil, and Eli Upfal. 2011. Performance prediction for concurrent database workloads. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*. 337–348.

[10] Said S Elnaffar and Patrick Martin. 2004. An intelligent framework for predicting shifts in the workloads of autonomic database management systems. In *Proc of 2004 IEEE International Conference on Advances in Intelligent Systems–Theory and Applications*. 1–8.

[11] Adebola Folorunso, Ifeoluwa Wada, Bunmi Samuel, and Viqaruddin Mohammed. 2024. Security compliance and its implication for cybersecurity. *World Journal of Advanced Research and Reviews* 24, 01 (2024), 2105–2121.

[12] Marin Fotache, Adrian Munteanu, Cătălin Strîmbei, and Ionuț Hrubaru. 2023. Framework for the assessment of data masking performance penalties in SQL database servers. Case Study: Oracle. *IEEE Access* 11 (2023), 18520–18541.

[13] Kevin P Gaffney, Robert Claus, and Jignesh M Patel. 2021. Database isolation by scheduling. *Proceedings of the VLDB Endowment* 14, 9 (2021).

[14] Rick Greenwald, Robert Stackowiak, and Jonathan Stern. 2013. *Oracle essentials: Oracle database 12c*. " O'Reilly Media, Inc.".

[15] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Comput.* 9, 8 (1997), 1735–1780. https://doi.org/10.1162/NECO.1997.9.8.1735

[16] Oracle Inc. 2025. Oracle Online Encryption. https://www.oracle.com/technetwork/database/options/advanced-security/overview/advanced-security-tde-faq-2995212.pdf.

[17] Sejun Kim, Heeyoung Noh, and Bongki Lee. 2021. DeepDB: A Deep Learning Approach to Query Performance Prediction. *ACM Transactions on Database Systems* 46, 2 (2021), 1–28.

[18] Kai Li, Qi Zhang, Lei Yu, and Hong Min. 2021. DRL-Clusters: Buffer management with clustering based deep reinforcement learning. In *Workshop on Databases and AI*.

[19] Ilya Loshchilov and Frank Hutter. 2019. Decoupled Weight Decay Regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net. https://openreview.net/forum?id=Bkg6RiCqY7

[20] Lin Ma, Dana Van Aken, Ahmed Hefny, Gustavo Mezerhane, Andrew Pavlo, and Geoffrey J Gordon. 2018. Query-based workload forecasting for self-driving database management systems. In *Proceedings of the 2018 International Conference on Management of Data*. 631–645.

[21] Patrick Martin, Said Elnaffar, and Ted Wasserman. 2006. Workload models for autonomic database management systems. In *International Conference on Autonomic and Autonomous Systems (ICAS'06)*. IEEE, 10–10.

[22] Ulf T Mattsson. 2005. Database encryption-how to balance security with performance. *Available at SSRN 670561* (2005).

[23] David T McWherter, Bianca Schroeder, Anastassia Ailamaki, and Mor Harchol-Balter. 2005. Improving preemptive prioritization via statistical characterization of OLTP locking. In *21st International Conference on Data Engineering (ICDE'05)*. IEEE, 446–457.

[24] Nimrod Megiddo and Dharmendra S Modha. 2003. ARC: A Self-Tuning, low overhead replacement cache. In *2nd USENIX Conference on File and Storage Technologies (FAST 03)*.

[25] Adhish Nanda, Shalini Bhaskar Bajaj, and Swati Gupta. 2020. A Comprehensive Survey of Machine Learning in Scheduling of Transactions. In *2020 4th International Conference on Trends in Electronics and Informatics (ICOEI)(48184)*. IEEE, 740–745.

[26] Khalid Nassiri and Moulay Akhloufi. 2023. Transformer models used for text-based question answering systems. *Applied Intelligence* 53, 9 (2023), 10602–10635.

[27] Walid Rjaibi. 2018. Holistic Database Encryption. In *15th International Joint Conference on e-Business and Telecommunications, ICETE 2018 - Volume 2: SECRYPT*. SciTePress, 638–643.

[28] W. Rjaibi, A. French, K. Chin, K. Mirshahi, and J. Hwang. 2024. Encrypting Existing Databases with Zero Downtime and Zero Storage Overhead. IBM Patent Reference P202305543US01.

[29] Hamid Shahrivari, Odysseas Papapetrou, and George Fletcher. 2022. Workload prediction for adaptive approximate query processing. In *2022 IEEE international conference on big data (big data)*. IEEE, 217–222.

[30] Alex Sherstinsky. 2020. Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network. *Physica D: Nonlinear Phenomena* 404 (2020), 132306.

[31] Zhan Shi, Akanksha Jain, Kevin Swersky, Milad Hashemi, Parthasarathy Ranganathan, and Calvin Lin. 2021. A hierarchical neural model of data prefetching. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 861–873.

[32] Erez Shmueli, Ronen Vaisenberg, Ehud Gudes, and Yuval Elovici. 2014. Implementing a database encryption solution, design and implementation issues. *Computers & security* 44 (2014), 33–50.

[33] Shaw Steve. 2008. HammerDB is a trusted open source database benchmarking application to the global database industry. https://www.hammerdb.com.

[34] Alexander Thomasian. 2024. Heterogeneous Data Access Model for Concurrency Control and Methods to Deal with High Data Contention. *arXiv preprint arXiv:2404.02276* (2024).

[35] Boyu Tian, Jiamin Huang, Barzan Mozafari, and Grant Schoenebeck. 2018. Contention-aware lock scheduling for transactional databases. *Proceedings of the VLDB Endowment* 11, 5 (2018), 648–662.

[36] Transaction Processing Performance Council. 2010. TPC Benchmark™ C Standard Specification, Revision 5.11. https://www.tpc.org/TPC_Documents_Current_Versions/pdf/tpc-c_v5.11.0.pdf.

[37] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (Eds.). 5998–6008. https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html

[38] Donghui Wang, Peng Cai, Weining Qian, and Aoying Zhou. 2020. Predictive Transaction Scheduling for Alleviating Lock Thrashing. In *Database Systems for Advanced Applications: 25th International Conference, DASFAA 2020, Jeju, South Korea, September 24–27, 2020, Proceedings, Part I 25*. Springer, 139–156.

[39] Jiachen Wang, Ding Ding, Huan Wang, Conrad Christensen, Zhaoguo Wang, Haibo Chen, and Jinyang Li. 2021. Polyjuice:{High-Performance} transactions via learned concurrency control. In *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*. 198–216.

[40] Huijing Yang, Juan Fang, Xing Su, Zhi Cai, and Yuening Wang. 2024. RL-CoPref: a reinforcement learning-based coordinated prefetching controller for multiple prefetchers. *The Journal of Supercomputing* 80, 9 (2024), 13001–13026.

[41] Robert K Yin. 2017. *Case study research: Design and methods* (6th ed.). Sage.

[42] Xiaochun Yu and H. V. Jagadish. 2014. Predicting System Resource Contention for Multitenant Databases. *Proceedings of the VLDB Endowment* 7, 6 (2014), 529–540.

[43] Lei Zhang, Andriy Miranskyy, and Walid Rjaibi. 2020. Quantum advantage and the Y2K bug: A comparison. *IEEE Software* 38, 2 (2020), 80–87.

[44] Lei Zhang, Andriy Miranskyy, Walid Rjaibi, Greg Stager, Michael Gray, and John Peck. 2023. Making existing software quantum safe: A case study on IBM Db2. *Information and Software Technology* 161 (2023), 107249.

[45] Mingyi Zhang, Patrick Martin, Wendy Powley, Paul Bird, and Keith McDonald. 2012. Discovering indicators for congestion in DBMSs. In *2012 IEEE 28th International Conference on Data Engineering Workshops*. IEEE, 263–268.

[46] Mingyi Zhang, Patrick Martin, Wendy Powley, and Jianjun Chen. 2017. Workload management in database management systems: A taxonomy. *IEEE transactions on knowledge and data engineering* 30, 7 (2017), 1386–1402.

[47] Farzaneh Zirak, Farhana Choudhury, and Renata Borovica-Gajic. 2024. SeLeP: Learning Based Semantic Prefetching for Exploratory Database Workloads. *Proc. VLDB Endow.* 17, 8 (2024), 2064–2076.