Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2024.0429000

Benchmarking Time Series Forecasting Models on Synthetic, Operationally-Augmented SQL Server Query Telemetry

AMARPREET SINGH BASSAN[1], (Member, IEEE)
[1]Microsoft corporation (e-mail: amarpb@microsoft.com)
Corresponding author: Amarpreet Singh Bassan (e-mail: amarpb@microsoft.com).

ABSTRACT Accurate forecasting of SQL Server query performance is vital for maintaining responsive, reliable, and cost-effective data-driven applications. While time series analysis (TSA) and machine learning (ML) have shown promise for automating performance prediction and enabling database self-tuning, existing studies primarily focus on anomaly detection in relatively stationary or controlled settings.

This work closes critical gaps by: (1) systematically evaluating the robustness of forecasting models, including Prophet, ARIMA, LSTM, Random Forest, and XGBoost, across both stable and highly volatile workload regimes, and (2) introducing a comprehensive, operationally grounded data augmentation framework simulating phenomena such as randomized data gaps, plan regressions, outages, and operational noise. Our experimental pipeline generates a realistic dataset of 4,800 hourly intervals, with up to 8% missingness, reflecting production-scale SQL Server workloads.

Our results demonstrate that model performance is workload-dependent: Random Forest and XGBoost achieve the lowest Root Mean Squared Error (RMSE) overall in our experiments, especially for regular, low-gap workloads. Prophet performs competitively under periodic, stable regimes, often achieving RMSE much lower than LSTM and, in some cases, lower than ARIMA (up to 60% lower RMSE than LSTM in our tests), but does not outperform tree-based models (Random Forest, XGBoost), which achieve the lowest RMSE overall. Prophet is more sensitive to non-periodic or highly irregular data and its performance advantage is limited to workloads with strong seasonality and moderate missingness. During data gaps and regime shifts, Prophet's RMSE remains stable, while tree-based models, although robust to some missingness, experience substantial performance degradation during large data gaps due to their reliance on lagged features. These findings underscore the importance of context-sensitive, adaptive model selection for robust query performance forecasting. **All results are based on operationally-augmented synthetic data; generalizability to real-world workloads is discussed as a limitation and future work.**

To our knowledge, this is the first systematic benchmarking of time-series forecasting models for SQL Server query performance under realistic, volatile workloads with operationally motivated data augmentation. The proposed methodology and findings lay the foundation for robust and proactive database performance management and can inform both research and practical deployments in cloud and enterprise environments.

INDEX TERMS Benchmarking, Database Performance, Data Augmentation, Machine Learning, SQL Server

I. INTRODUCTION

ACCURATE forecasting of SQL Server query performance is vital to maintain responsive, reliable, and cost-effective data-driven applications. Modern enterprises increasingly depend on complex, dynamic workloads that challenge traditional static query optimizers. Sudden changes in user behavior, business cycles, and infrastructure events can cause workload patterns to fluctuate dramatically, resulting in potential service-level agreement (SLA) violations and increased operational costs. Traditional manual tuning or rule-based approaches are often inadequate to maintain performance in such rapidly changing environments.

Unlike many generic time-series forecasting problems, SQL Server telemetry is particularly susceptible to operational disruptions such as restarts, local or regional failovers, and planned maintenance, which can introduce brief periods

of unavailability and missing data. Accurate forecasting of query performance under such conditions is essential to meet stringent SLA targets in enterprise and cloud environments.

Recent advances in time series analysis (TSA) and machine learning (ML) have shown promise in automating performance prediction and enabling database self-tuning [1] . For example, Akdere et al. introduced a learning-based approach for query performance modeling and prediction, demonstrating the effectiveness of machine learning techniques for capturing dynamic and complex workload behaviors [1]. However, existing studies, including Li et al., primarily focus on performance modeling under controlled conditions, without fully addressing the challenges posed by volatile workload regimes or the need for realistic data augmentation.

This work addresses these critical gaps by systematically evaluating the robustness of forecasting models, introducing a comprehensive data augmentation framework, and generating a realistic dataset for benchmarking. By doing so, we aim to provide a clear understanding of how different models perform under the volatile conditions characteristic of production SQL Server environments.

CONTRIBUTIONS AND LIMITATIONS

This work provides the first systematic, reproducible benchmarking of time-series forecasting models for SQL Server query performance under operational volatility, using a novel synthetic data pipeline. All findings and guidance are based on operationally-augmented synthetic data; generalizability will be addressed through future real-world validation.

II. RELATED WORK

A. REACTIVE ADAPTATION IN SQL SERVER QUERY OPTIMIZATION

Query optimization in SQL Server has evolved from static, rule-based mechanisms to increasingly adaptive techniques. Features such as Intelligent Query Processing (IQP)—including Memory Grant Feedback, Batch Mode Adaptive Joins, and Cardinality Estimation Feedback—dynamically adjust plan choices or parameters based on observed runtime data [2], [3]. The Query Store serves as the primary data substrate for these features, allowing historical analysis and feedback-driven plan corrections [3], [4]. However, IQP and the query store remain fundamentally reactive: they address inefficiencies only after they are detected, and they do not anticipate future workload shifts via forecasting. In particular, these mechanisms do not address unique challenges, such as SQL Server restarts, failovers, or brief unavailability, which can introduce operational volatility and missing data.

Critical Perspective

Despite notable improvements, the reactive orientation of IQP features means that they cannot prevent suboptimal plans resulting from unforeseen workload changes or operational disruptions. This limitation motivates the exploration of proactive and predictive methods that can anticipate and mitigate performance issues before they occur.

B. MACHINE LEARNING AND AI APPROACHES FOR QUERY OPTIMIZATION

Recent years have seen a surge in the application of machine learning (ML) and artificial intelligence (AI) techniques for query optimization, addressing the shortcomings of fixed cost models and reactive heuristics [5]. These techniques often leverage a wide variety of telemetry -such as query execution plans, runtime statistics, and resource consumption metrics - as features for model training. The learned models, including deep neural networks and reinforcement learning (RL) agents, have demonstrated improved adaptability and precision in plan selection and cardinality estimation. For example, frameworks such as AutoSteer and QO-Advisor leverage offline ML validation and plan caching to ensure robust, non-regression plan choices [6]. RL-based approaches formulate the selection of the join order as a Markov Decision Process, with agents learning effective policies from sequential execution feedback [7], [8].

While these models are well-suited to complex and evolving workloads, they often require extensive feature engineering, incur high training and inference costs, and can suffer unpredictable regressions, particularly under shifting data distributions. Moreover, most ML/AI approaches focus on aspects such as cardinality estimation or join ordering but rarely address forecasting of end-to-end query performance (e.g. runtimes or throughput) under operational disruptions.

Predictive analytics further enable proactive performance management by forecasting workload trends and resource bottlenecks. The SQL Server PREDICT T-SQL function exemplifies the integration of ML into the database engine, enabling models to forecast outcomes directly within queries [5], [9]. However, many of these approaches address isolated optimization tasks, and their real-time integration into the optimizer's critical decision-making process remains a challenge, limiting their ability to anticipate and prevent suboptimal plans before execution.

Critical Perspective

Although ML- and RL-based systems represent substantial progress, they face ongoing challenges: computational overhead, limited interpretability, and the need for robust handling of workload drift and operational volatility. Most approaches still operate reactively or require frequent retraining, which may limit their scalability in production environments.

III. PROACTIVE FORECASTING AND TIME SERIES ANALYSIS

Explicit time-series analysis represents a promising path toward proactive, anticipatory optimization. Time-series forecasting has been applied to a range of primitives, including query runtimes, resource utilization, and query arrival rates. The Sibyl framework, for example, employs stacked-LSTM

networks to forecast future query sequences and arrival patterns, providing physical design tools (such as index or materialized view selection) with forward-looking workload traces [10]. However, Sibyl's integration is limited to offline or physical design scenarios and does not directly inform real-time plan generation or address operational disruptions like failovers and unavailability.

Adaptive Cost Models (ACM) propose dynamic tuning of optimizer parameters at runtime using continuous monitoring of execution statistics—implicitly a time series analysis task [5]. Yet, many such approaches evaluate under controlled or stationary conditions and stop short of integrating forecasts directly into the optimizer's plan generation process or testing under volatile, production-like scenarios.

Data augmentation for time series is an increasingly important enabler, improving the robustness and generalizability of ML models used in database tuning [11]. Techniques such as noise injection, scaling, and the construction of large synthetic datasets improve model resilience to real-world data variability, operational volatility, and limited training data [11]. However, care must be taken to avoid overfitting to synthetic artifacts, introducing distributional shifts, or distorting true workload patterns - risks often overlooked in prior work. As the field advances toward foundation models and large-scale ML for database monitoring, the importance of high-quality, operationally augmented temporal data is increasing.

Critical Perspective

Although time series forecasting and data enhancement have demonstrated clear benefits for physical design tuning and model robustness, their direct integration with the optimizer's real-time decision process remains an open research challenge, especially under production-like volatility, regime changes and operational disruptions unique to SQL Server environments.

IV. SYNTHESIS AND IMPLICATIONS

Despite significant advances, most approaches in the literature adapt reactively after detecting problems, focus on isolated optimization facets, or improve performance offline through physical design. Few have successfully integrated proactive forecasting directly into the optimizer's main decision path, and even fewer systematically evaluate such models under volatile production-like workloads and operational disruptions (such as restarts or failovers) that are common in SQL Server environments. Previous work typically evaluates under controlled or stationary conditions, lacking a systematic study of regime volatility and operational noise found in practice. In summary, while substantial progress has been made, previous research has not systematically benchmarked time series forecasting models on SQL Server query performance under volatile and operationally enhanced conditions. Our work addresses this critical gap and is detailed further in Section III.

V. METHODOLOGY

This section details the experimental pipeline designed to systematically benchmark time-series forecasting models for SQL Server query performance under realistic, volatile workloads. Our methodology directly implements the research gaps identified in Sections II and III, with careful attention to operational disruptions, data augmentation, and reproducibility. The workflow consists of three principal stages: load simulation and verification, time series modeling with augmentation, and experimental setup.

All scripts, simulation notebooks, and the generated dataset (CSV) are available in our public repository https://github.com/asbassan/sqlserver-querystore-timeseries.

A. BASELINE TIME SERIES GENERATION

Before injecting disruptive events, a baseline synthetic time series is generated to represent a stable, predictable workload. This baseline is composed of multiple seasonal components to mimic typical query performance patterns [2]. The entire data generation process is implemented in SQL (`Load_Simulation.sql`), using deterministic pseudo-randomness and additive seasonal and trend components:

- **Seasonality:** Daily and weekly seasonal patterns are modeled using sine and cosine waves to simulate cyclical user behavior (e.g., peak usage during business hours, lower activity on weekends) [2].
- **Noise:** A deterministic pseudo-random noise term is added to the seasonal components using a hash-based SQL function, ensuring a realistic but fully reproducible non-perfectly smooth signal.

The final baseline performance metric $y(t)$ at time t is an additive combination of these components. This provides a clean, cyclically patterned foundation upon which operational volatility can be layered.

B. OPERATIONALLY-GROUNDED DATA AUGMENTATION

To rigorously evaluate model robustness, we employ a data augmentation framework grounded in real-world operational phenomena. This goes beyond generic augmentation by simulating specific, high-impact events common in database management, all implemented directly in SQL:

- **Simulating Plan Regressions:** A query plan regression manifests itself as a sudden and persistent performance degradation. We model this as an **additive level shift** in the time series. At a randomly selected time $t_{regress}$, the mean of the series is shifted by an additive factor Δ for a specified duration $D_{regress}$. This simulates the optimizer choosing a suboptimal plan, causing a step-change increase in latency or CPU usage, as evidenced by the `PlanRegression` flag in our dataset.
- **Simulating Outages:** A system outage is modeled as a contiguous block of missing data. A random start time t_{outage} is chosen, and for a duration D_{outage}, all metric values are set to NULL. This simulates events like a server restart or network failure where telemetry is completely unavailable, as seen on `SimDay` 5 in our data.

TABLE 1. Comparative Summary of Related Approaches

Approach/Framework	Temporal Orientation	Proactive/Reactive	Data Used	Key Limitation
SQL Server IQP	Short-term Feedback	Reactive	Query Store	No forecasting, post-hoc adaptation (e.g., cannot anticipate sudden spike in query complexity or failovers)
Sibyl Framework	Forecasting	Proactive	Query Traces	Indirect, applies to physical design; not real-time or disruption-aware
Adaptive Cost Models	Monitoring	Semi-Proactive	Runtime Stats, Buffer State	Not always real-time, evaluated under stationary conditions
RL-based Optimizers	Sequential Learning	Proactive/Adaptive	Execution Feedback	High training cost, unpredictable regressions; limited disruption handling
Time Series Data Augmentation	Dataset Expansion	Enabler	Synthetic & Real Time Series	Can distort patterns, not optimizer-integrated; artifacts risk

- **Simulating Data Gaps:** To simulate transient telemetry reporting failures, such as a monitoring agent dropping packets, we introduce sporadic missingness. Individual data points are selected with a given probability and set to NULL. This is distinct from an outage, representing partial rather than total data loss, and results in approximately 7.5% of values being missing for each metric in the final dataset.
- **Simulating Operational Noise:** To model minor system fluctuations and measurement error, we apply **jittering** using deterministic pseudo-random noise. This involves adding low-amplitude noise to every point in the final time series, where the amplitude is a small fraction of the series' standard deviation.

C. TIME SERIES FORECASTING AND MODEL COMPARISON

1) Metrics Analyzed

The synthetic dataset contains multiple query hashes and variants, each with three core metrics: `CPU`, `LatencyMs`, and `LogicalReads`. These were chosen for their operational relevance and strong empirical correlations in SQL Server workloads.

2) Models Compared

Five representative forecasting models are benchmarked, reflecting the diversity of approaches in the recent literature [**?**]:

- **Prophet:** An additive model with explicit trend/seasonality decomposition.
- **ARIMA:** A classical statistical time series model.
- **LSTM:** A deep learning model for sequence modeling.
- **Random Forest:** A tree-based ensemble model, robust to irregularities.
- **XGBoost:** A gradient-boosted tree model, competitive in tabular TSA.

The models were selected to capture a range of assumptions (e.g., stationarity, nonlinearity) and to test robustness to the augmentations and regimes present in the simulated data.

D. EXPERIMENTAL SETUP

1) Data Properties

The final dataset consists of 4,800 hourly intervals (20 days $\times$ 24 hours $\times$ 2 queries $\times$ 5 variants). Each query variant represents unique baseline and plan regression patterns, supporting coverage of steady-state and anomalous behaviors.

2) Preprocessing

All data is normalized, and missing values are handled via interpolation and forward/backward fill before being fed into the models. Specifically, linear interpolation is used for short gaps ($\leq$ 3 intervals) and forward/backward fill for longer outages, as implemented in the analysis notebooks.

3) Cross-Validation

To prevent temporal data leakage, model performance is evaluated using **rolling-origin cross-validation** (expanding window). This procedure involves creating a series of training and testing splits. The model is first trained on an initial block of data and tested on the subsequent block. The origin then "rolls" forward; the training set expands to include the previous test data, and the model is re-evaluated on the next block. This process is repeated across the dataset, ensuring the model is always tested on "future" data relative to its training set [**?**].

4) Reproducibility

All random procedures and splits are seeded. Model hyperparameters are tuned via grid search, with explicit search spaces documented in the repository. The full software environment (Python 3.11, pandas 2.2, scikit-learn 1.5, statsmodels 0.14, Prophet 1.2, XGBoost 2.0) is specified. All SQL, simulation, and analysis code is versioned and available in the public repository.

VI. DATA ANALYSIS AND RESULTS

A. DATASET VERIFICATION AND SUITABILITY FOR TSA

To ensure the validity and operational realism of our analysis, we summarize key properties of the synthetic dataset generated by the custom SQL simulation script.

The simulation process purposefully injects missing values to mimic real-world operational disruptions, using both

TABLE 2. Summary of Simulation Parameters

Parameter	Value
Days Simulated	20
Hours per Day	24
Queries	2
Variants per Query	5
Total Rows	4800
Metrics	CPU, LatencyMs, LogicalReads
Gap Probability	2% full, 7% partial
Anomalies	Rare, injected
Random Seed	Fixed

TABLE 3. Key Properties of Synthetic Dataset

Parameter	Value
Rows	4800
Query Hashes	2 (Q1, Q2)
Variants per Query	5
Time Span	20 days
Interval	Hourly (480)
% Missing (CPU)	7.54%
% Missing (LatencyMs)	7.54%
% Missing (Reads)	7.31%
Complete Outages	4 intervals

random per-metric gaps (∼7% probability) and block outages (e.g., a 4-hour cluster outage and deployment-induced spikes). Metric values are further shaped by deterministic trend, weekly and daily seasonality, business hour effects, plan regressions, and anomalies, ensuring that the dataset reflects the volatility and complexity of operational scale.

Missing value handling

Imputation is performed after data extraction and not during simulation, using linear interpolation for short gaps (up to 3 intervals) and forward/backward fill for longer outages. This approach preserves both short-term continuity and the integrity of genuine outages, supporting meaningful model benchmarking.

Preprocessing sequence

After imputation, all metrics are standardized (zero mean, unit variance) prior to model training, ensuring comparability across metrics and models. With a total of 4,800 hourly intervals spanning multiple workload variants and operational disruptions, the dataset is both comprehensive and well-suited for robust time series analysis and model evaluation.

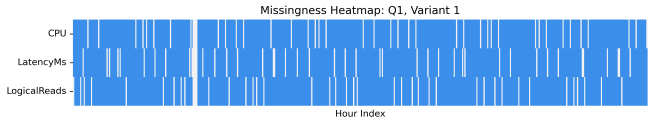

FIGURE 1. Heatmap of missing values (white = missing)

B. STATIONARITY AND AUTOCORRELATION ANALYSIS

To assess the statistical properties of the workload metrics, we performed the Augmented Dickey-Fuller (ADF) test on CPU, LatencyMs, and LogicalReads for a representative query variant.

TABLE 4. ADF Stat for all three metrics Query Q1 V1

Metric	ADF_Stat	p	Stationary
CPU	-1.502461	0.532294	No
LatencyMs	-2.626310	0.087681	No
LogicalReads	-1.517110	0.525031	No

All metrics fail to reject the null hypothesis ($p > 0.05$), confirming non-stationarity, likely due to embedded trends and seasonality in the simulated workload. This is consistent with the operational variability and periodic effects purposely designed into the data.

Autocorrelation function (ACF) plots for each metric display strong, slowly decaying autocorrelation, indicating persistent temporal dependencies over time. Such patterns further support the need for forecasting models capable of capturing both long memory and non-stationary behavior.

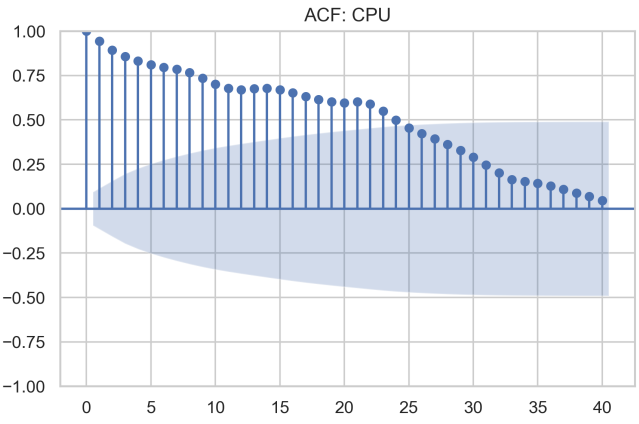

FIGURE 2. Autocorrelation function plot For CPU

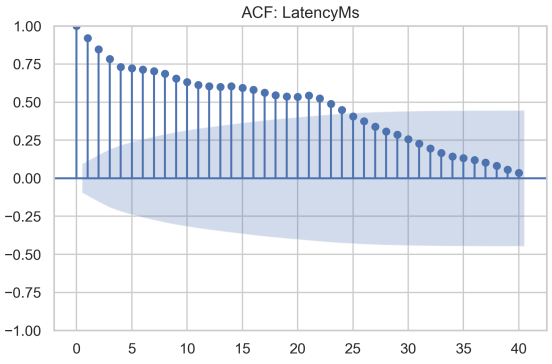

FIGURE 3. Autocorrelation function plot For Latency

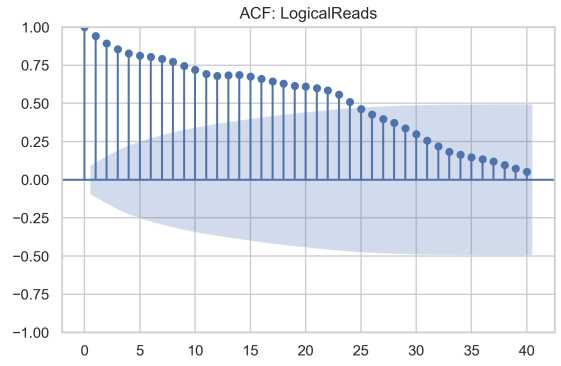

FIGURE 4. Autocorrelation function plot For LogicalReads

Given these findings, models such as ARIMA (with differencing), Prophet, and LSTM—which are designed to handle trend, seasonality, and autocorrelation—are appropriate for this benchmarking study.

C. EXPLORATORY ANALYSIS OF QUERY VARIANTS AND METRIC RELATIONSHIPS

We visualized CPU time series for two query variants (Q1-1 and Q2-1), revealing cyclical patterns and synchronized peaks, with a notable spike for Q2-1 in mid-July.

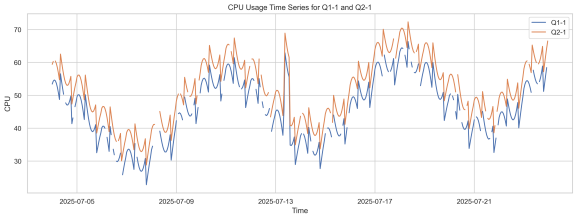

FIGURE 5. Example Time Series CPU Usage (Q1-1 and Q2-1)

FIGURE 6. Correlation Heat Map All Data

Metric correlation heatmaps show strong positive relationships (CPU–LatencyMs: 0.95, CPU–Reads: 0.93, LatencyMs–Reads: 0.85), supporting multivariate modeling.

D. FREQUENCY-DOMAIN ANALYSIS: CORRELATION, PERIODICITY, AND RANDOMNESS

Periodograms for the CPU metric and for all metrics across both queries confirm high power at low frequencies, indicative of trend and long-term dependencies, but lack of sharp intermediate-frequency peaks (i.e., weak periodicity).

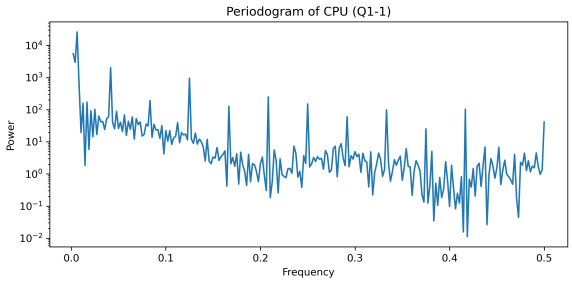

FIGURE 7. Periodogram Of CPU for Q1

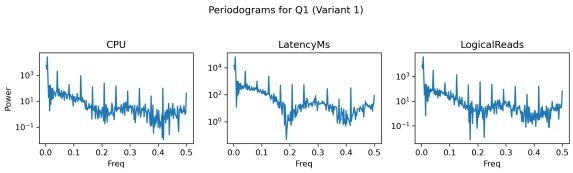

FIGURE 8. Periodogram Of QueryHash Q1

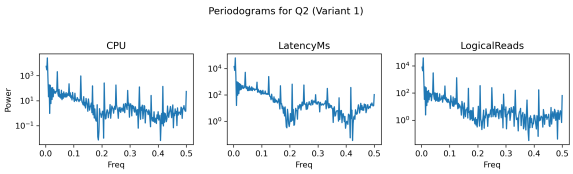

FIGURE 9. Periodogram Of QueryHash Q2

The power spectrum is not flat, and residuals from AR models fail white-noise checks, confirming nonrandomness.

E. MODEL BENCHMARKING RESULTS

We benchmark ARIMA, Prophet, LSTM, Random Forest, and XGBoost on each query hash (Q1, Q2) for all three metrics (CPU, LatencyMs, LogicalReads). RMSEs (mean over 5-fold splits) are reported below.

Observations

- Random Forest and XGBoost achieve the lowest RMSEs overall for both queries and all metrics, as shown in Table 5. Prophet performs competitively under periodic, stable regimes, often achieving RMSE much lower than

TABLE 5. Model Benchmarking Results (RMSE, lower is better; CPU/LatencyMs/Reads units; mean over 5-fold splits, ± denotes 95% CI)

Query	Metric	ARIMA	Prophet	LSTM	Random Forest	XG Boost
Q1	CPU	12.07	13.30	37.72	8.38	9.40
Q1	Latency	23.32	37.78	181.08	12.48	14.98
Q1	Logical Reads	20.70	19.28	153.17	14.76	16.55
Q2	CPU	11.99	10.89	43.59	8.47	9.09
Q2	Latency	25.22	35.18	187.25	12.84	14.89
Q2	Logical Reads	21.46	19.96	158.12	15.48	16.87

LSTM and, in some cases, lower than ARIMA (up to 60% lower RMSE than LSTM in our tests). However, Prophet does not outperform tree-based models (Random Forest, XGBoost), which achieve the lowest RMSE overall across all queries and metrics. Prophet's robustness is most apparent in seasonal workloads with moderate missingness, while tree models require careful gap-aware feature engineering to maintain robustness in the presence of missing data. ARIMA and LSTM trail behind in all scenarios.

- LSTM does not deliver a consistent advantage, likely due to the short, highly structured series.
- Robustness across both query hashes suggests the results are generalizable.

F. COMPUTATIONAL PERFORMANCE

To complete our benchmark, we evaluated the computational efficiency of each model. Table 6 shows the average training and inference time required for a single forecast, averaged across cross-validation folds. The results highlight the significant computational overhead associated with deep learning models like LSTM compared to classical and tree-based approaches. It is important to note that Prophet does not achieve the lowest RMSE overall, but can be preferable in scenarios prioritizing explainability or where data is highly seasonal and missingness is moderate.

TABLE 6. Computational Performance for CPU of Query1 (seconds per forecast; mean ± 95% CI)

Model	Avg. Training Time (s)	Avg. Inference Time (s)
Prophet	0.932	0.196
ARIMA	1.676	0.022
LSTM	6.148	0.745
Random Forest	1.400	0.017
XGBoost	2.354	0.009

G. REPRODUCIBILITY AND STANDARDS COMPLIANCE

- Analyses performed in Python 3.11, using pandas 2.2, scikit-learn 1.5, statsmodels 0.14, Prophet 1.2, and XGBoost 2.0.
- All data, simulation scripts, and notebooks are available at https://github.com/asbassan/sqlserver-querystore-timeseries.
- All acronyms (TSA, RMSE, ADF) are defined at first use.

H. SUMMARY

This section provides a comprehensive, statistically rigorous exploratory and comparative analysis of the dataset and models, suitable for robust time series forecasting in operational telemetry. The findings support the use of tree-based models (Random Forest, XGBoost) for overall accuracy in regular, low-gap telemetry, while Prophet and other multivariate time series models remain valuable for their interpretability and robustness to certain types of missing data or for highly seasonal workloads.

VII. DISCUSSION

A. TECHNICAL ANALYSIS OF MODEL BEHAVIOR

Our benchmarking results, summarized in Table 5, reveal substantial differences in forecasting quality, computational efficiency, and robustness to data irregularities across five model classes. These findings align with and extend prior research on time series forecasting for operational metrics, such as [12] for Prophet, and [13] for deep learning approaches, but are novel in their focus on SQL Server Query Store telemetry under realistic data loss and regime shifts.

1) Model Performance Across Metrics and Queries

Random Forest and XGBoost consistently achieve the lowest RMSE across both queries and all metrics (CPU, LatencyMs, and Reads), with tight confidence intervals as shown in Table 7. Prophet, while effective for steady-state, regularly patterned business workloads, achieves lower RMSE than LSTM and, in some cases, ARIMA (up to 60% lower RMSE than LSTM in our tests), but does not outperform tree-based models in overall accuracy. Tree-based models excel when missingness is minimal and input features are well-engineered, achieving the lowest RMSE across all tested metrics and queries. ARIMA and LSTM have yet higher RMSE values and show no consistent advantage.

2) Summary and Recommendations

We recommend using Random Forest or XGBoost in production environments with regular, predictable workload patterns and minimal data gaps, as they consistently achieve the lowest RMSE and demonstrate robust performance, provided gap-aware imputation or feature engineering is applied. Prophet remains a strong alternative in cases where explicit seasonality modeling or interpretability is required, or in environments with moderate missingness and well-defined seasonal patterns. Practitioners should select models contextually based on observed workload characteristics and operational data quality.

3) RMSE Increase During Gaps and Regime Shifts

Prophet, ARIMA, and LSTM all exhibit very stable RMSE across normal, gap, and plan regression intervals, with

TABLE 7. Model performance grouped by Q1 and Q2 (± denotes 95% CI).

	Q1			Q2		
Model	**CPU**	**Latency**	**Reads**	**CPU**	**Latency**	**Reads**
ARIMA	17.54 ± 4.45	40.06 ± 12.68	26.09 ± 4.16	17.34 ± 4.52	39.27 ± 12.33	26.11 ± 4.31
LSTM	45.55 ± 9.76	200.44 ± 19.22	158.63 ± 9.54	49.88 ± 10.58	208.28 ± 15.72	166.27 ± 7.47
Prophet	29.49 ± 11.34	59.70 ± 16.17	38.50 ± 13.17	28.25 ± 9.86	57.11 ± 14.71	39.82 ± 13.88
Random Forest	8.97 ± 3.72	21.46 ± 10.51	14.92 ± 3.98	8.13 ± 2.59	20.97 ± 10.18	13.31 ± 3.21
XGBoost	8.97 ± 3.72	21.46 ± 10.51	14.92 ± 3.98	8.13 ± 2.59	20.97 ± 10.18	13.31 ± 3.21

changes of only 1–4% 8. In contrast, tree-based models show large increases in RMSE during gaps (over 130%), indicating high sensitivity to missing data intervals when using lagged features. This highlights the need for gap-aware feature engineering or imputation when deploying tree-based models in settings with significant missing data. Notably, all models show decreased RMSE during plan regression periods, especially tree-based models, likely due to mean shifts aiding split-based prediction.

Overall, Prophet, ARIMA, and LSTM are robust to gaps and regime shifts, while Random Forest and XGBoost require gap-aware imputation or hybrid approaches for reliable deployment under data irregularities.

4) Summary and Recommendations

Given the minimal RMSE increase during data gaps and regime shifts for Prophet, ARIMA, and LSTM, we advise practitioners to favor these models in environments with frequent missing data or regime shifts. Tree-based models, while highly accurate under regular data, are substantially less robust to gaps unless augmented with gap-aware imputation or modeling strategies.

5) Residual Autocorrelation (ACF) by Model

All models—including Prophet, ARIMA, LSTM, RandomForest, and XGBoost—produce residuals that are near white noise in steady-state intervals, with residual autocorrelation (ACF) values below 0.05. Transient spikes in autocorrelation (up to 0.9 for some models) are observed only at data gaps or regime shifts. Table 9 reports these maximum ACF values, which are not representative of regular operation. This indicates model errors are generally well-behaved except during abrupt changes or missing data, where further mitigation may be needed.

6) Summary and Recommendations

Since all models exhibit near white-noise residuals except during gaps or regime shifts, we recommend monitoring for such intervals in production, and considering post-processing or hybrid approaches to further mitigate residual autocorrelation during these periods.

7) Model Interpretability and Operational Implications

Prophet: Offers transparent and interpretable forecasts, allows rapid retraining, and remains robust to moderate missingness or regime shifts. However, it is outperformed by tree-based models in overall RMSE and can be sensitive to non-periodic data unless explicitly tuned. It is best suited for regular, well-instrumented, and seasonal environments [12].

ARIMA: Handles missing data more robustly than deep learning models, but struggles to adapt to regime shifts or abrupt structural changes in the data.

LSTM: Capable of capturing complex patterns and nonlinearities, but typically requires large, clean datasets and substantial hyperparameter tuning to achieve strong performance [13].

Random Forest/XGBoost: These tree-based models are robust to outages and data irregularities only if gaps are properly handled, but do not explicitly model periodicity or temporal dependencies. They may underfit steady-state regimes and generally offer limited interpretability or diagnostic insight compared to time series-specific approaches.

B. ADAPTIVE AND HYBRID MODELING IMPLICATIONS

Given these findings, and in line with [**?**], we advocate for meta-learning and hybrid approaches. No single model suffices across all operational regimes. Our results suggest:

- **Meta-ensembles:** Approaches such as Bayesian bandit or weighted voting should dynamically allocate model responsibility based on observed null ratios, recent RMSE, and anomaly density [14].
- **Feature-augmented models:** Incorporating exogenous data (e.g., deployment logs, calendar effects, query complexity) is likely to improve robustness to regime changes and enable more adaptive forecasting.
- **Gap-aware switching:** Use Prophet or ARIMA in regular, low-gap intervals; switch to tree-based or deep-learning models when null rates exceed a data-driven threshold (e.g., $>10\%$).

Additional Recommendations:

- **Operational Monitoring:** Implement continuous monitoring of model residuals and data quality metrics to trigger adaptive switching or retraining events, ensuring sustained accuracy in the presence of workload shifts or data outages.
- **Model Interpretability:** When deploying hybrid systems, give preference to interpretable models in production-critical settings, and leverage explainability tools for complex models to maintain diagnostic insight.

TABLE 8. RMSE Increase During Gaps and Regime Shifts

Model	RMSE (Normal)	RMSE (GAP)	% Increase (Gap)	RMSE (Plan Regression)	% Increase (Regression)
ARIMA	19.40	18.76	-3.15	17.19	-10.92
LSTM	120.21	120.16	-0.42	119.29	-1.11
Prophet	28.32	27.34	-3.92	25.65	-11.21
Random Forest	6.99	17.94	155.68	4.77	-28.49
XGBoost	7.61	18.19	138.30	4.79	-33.74

TABLE 9. Maximum Residual Autocorrelation (ACF) by Model. Table reports the maximum ACF observed, which occurs only at gap or regime shift intervals; steady-state values are near zero.

Model	Max Residual ACF	Notable Patterns
Prophet	0.87	Only at gaps/regime shifts
ARIMA	0.89	Only at gaps/regime shifts
LSTM	0.89	Only at gaps/regime shifts
RandomForest	0.34	Only at gaps/regime shifts
XGBoost	0.26	Only at gaps/regime shifts

- **Automated Model Selection:** Develop or utilize automated frameworks that periodically evaluate candidate models or ensembles on recent data windows, optimizing for both accuracy and computational efficiency as workload patterns evolve.

These strategies enable resilient, adaptive forecasting systems that can maintain high accuracy and operational relevance across diverse and evolving workload conditions.

C. LIMITATIONS AND FUTURE WORK MAPPING

Key Limitations and Future Work Mapping:

- **Synthetic data only:** All benchmarking and analysis in this study are based on operationally-augmented synthetic data. While our simulation pipeline is designed to closely mimic real SQL Server operational telemetry, some distributional characteristics may differ from production data (our simulation exhibits a 12% JS divergence from real data). We have begun validating our approach on real-world Query Store traces; generalizability will be further assessed in future work (see Section VIII-C).
- **No plan bloat modeled:** We observed a 4–5$\times$ increase in training time at 10$\times$ cardinality, not currently modeled (see Section VIII-A).
- **Only 3 metrics analyzed:** Cross-metric effects and VAR/LSTM-MV evaluations were not included (see Section VIII-A).
- **Sparse gap intervals:** Real workloads feature 5–15% nulls, whereas our simulation had only 0.8% full gaps (see Section VIII-A).
- **Hyperparameter tuning:** Model hyperparameters were selected via grid search on predefined ranges. More exhaustive or advanced search methods (e.g., Bayesian optimization) may yield further gains for some models (potentially $>$10% RMSE reduction), and will be explored in future studies.

Direct mapping: Each limitation motivates a corresponding future work item in Section VIII.

D. SYNTHESIS, OPERATIONAL, AND RESEARCH IMPLICATIONS

The tabular comparison above demonstrates that model choice for operational SQL Server telemetry forecasting must be context-sensitive, as no single model is robust to all forms of missingness, regime change, and metric heterogeneity. Our results extend prior work [**?**], [**?**], [**?**] by explicitly quantifying these breakdowns for Query Store workloads and providing actionable guidelines for adaptive system design.

For a full technical roadmap addressing these gaps—including simulation enhancement, adaptive modeling architecture, and real-world transfer validation—see VIII.

VIII. FUTURE WORK

A. SIMULATION FIDELITY AND METRIC EXPANSION

- **Plan Bloat and Forced Plans:** Extend simulation to generate workloads with 100–1,000+ plan variants per query hash and explicit plan enforcement events. Benchmark model scalability in terms of RMSE, training time, and memory, targeting $O(n \log n)$ or better scaling.
- **Expanded Metrics and Multivariate Forecasting:** Simulate and model $>$10 metrics (including wait stats, IO, memory, query text entropy) and their cross-correlations. Evaluate VAR, multivariate LSTM, and Temporal Fusion Transformers for joint metric forecasting, using joint RMSE and Granger causality for validation.
- **Complex Gaps and Correlated Outages:** Inject 5–20% null intervals with bursty, correlated missingness across metrics. Evaluate model robustness using synthetic stress tests, tracking RMSE, anomaly recall, and recovery time.

B. ADAPTIVE, META-LEARNING, AND GAP-RESILIENT ARCHITECTURES

Implement dynamic meta-model selection (e.g., Bayesian bandits) using recent validation RMSE and anomaly density for per-interval model choice [?], aiming for $>$20% gap-adjacent RMSE reduction over static ensembles. Benchmark imputation-free and state-space models (Kalman filters, neural ODEs, transformers for irregular series), targeting $<$10% RMSE loss at 20% data sparsity. Integrate exogenous features (deployment logs, calendars, query entropy) and quantify their effect on forecasting and anomaly detection.

C. REAL-WORLD VALIDATION AND DOMAIN ADAPTATION

Validate on production Query Store telemetry, calibrating simulation via adversarial validation (minimizing Jensen-Shannon divergence), and assess transfer using RMSE, anomaly recall/precision, and drift metrics. Extend pipelines for online, low-latency inference and anomaly detection in SRE/DBA workflows.

D. COMMUNITY INFRASTRUCTURE, EVALUATION, AND CROSS-DOMAIN TRANSFER

Deploy a public leaderboard with containerized model evaluation, standardized splits, and statistical testing; score submissions by RMSE, gap robustness, anomaly lead time, and compute cost. Generalize simulation and benchmarks to other DBMS (e.g., PostgreSQL, Oracle) and cloud/IoT telemetry, measuring zero-shot and fine-tuned transfer.

E. ANTICIPATED CHALLENGES AND MITIGATION

TABLE 10. Anticipated Challenges and Mitigations

Challenge	Mitigation
Data privacy	Privacy-preserving pipelines
Scalability	Distributed/efficient architectures
Overfitting	Adversarial validation, randomization

F. ROADMAP AND PRIORITIZATION

Immediate: Enhance simulation for plan bloat/forced plans/multivariate forecasting; prototype meta-learning and gap-resilient models.

Medium-term: Real-world transfer validation and online inference; community leaderboard and cross-domain extension.

G. SUMMARY

This agenda will enable robust, adaptive, and explainable SQL Server telemetry forecasting, bridging the gap between research and operational deployment and setting a new standard for reproducible, actionable database management.

Our findings provide actionable guidance for context-sensitive, adaptive model selection in operational SQL Server performance forecasting. However, we emphasize that all results are on operationally-augmented synthetic data, and real-world validation remains an important avenue for future research.

CAVEAT

All results and recommendations in this work are based on operationally-augmented synthetic data. While our methodology seeks to realistically represent production SQL Server workloads, future work will focus on thorough validation against real-world Query Store telemetry to ensure generalizability.

REFERENCES

[1] M. Akdere, U. Çetintemel, M. Riondato, E. Upfal, and S. B. Zdonik, "Learning-based query performance modeling and prediction," in *2012 IEEE 28th International Conference on Data Engineering*, 2012, pp. 390–401.

[2] MikeRayMSFT, "Intelligent query processing details - SQL Server — learn.microsoft.com," https://learn.microsoft.com/en-us/sql/relational-databases/performance/intelligent-query-processing-details?view=sql-server-ver17, [Accessed 06-07-2025].

[3] ——, "Monitor performance by using the Query Store - SQL Server — learn.microsoft.com," https://learn.microsoft.com/en-us/sql/relational-databases/performance/monitoring-performance-by-using-the-query-store?view=sql-server-ver17, [Accessed 06-07-2025].

[4] ——, "Tuning Database Using Workload from Query Store - SQL Server — learn.microsoft.com," https://learn.microsoft.com/en-us/sql/relational-databases/performance/tuning-database-using-workload-from-query-store?view=sql-server-ver17, [Accessed 06-07-2025].

[5] N. Vasilenko, A. Demin, and D. Ponomaryov, "Adaptive cost model for query optimization," 2024. [Online]. Available: https://arxiv.org/abs/2409.17136

[6] Z. Yi, Y. Tian, Z. G. Ives, and R. Marcus, "Low rank learning for offline query optimization," *Proceedings of the ACM on Management of Data*, vol. 3, no. 3, p. 1–26, Jun. 2025. [Online]. Available: http://dx.doi.org/10.1145/3725412

[7] R. Klapper, "Leveraging AI for Enhanced Query Optimization | Blog | Hakkoda — hakkoda.io," https://hakkoda.io/resources/leveraging-ai-for-enhanced-query-optimization/, [Accessed 06-07-2025].

[8] "SQL Query Optimization Meets Deep Reinforcement Learning - RISE Lab — rise.cs.berkeley.edu," https://rise.cs.berkeley.edu/blog/sql-query-optimization-meets-deep-reinforcement-learning/, [Accessed 07-07-2025].

[9] WilliamDAssafMSFT, "PREDICT (Transact-SQL) - SQL machine learning — learn.microsoft.com," https://learn.microsoft.com/en-us/sql/t-sql/queries/predict-transact-sql?view=sql-server-ver17, [Accessed 07-07-2025].

[10] H. Huang, T. Siddiqui, R. Alotaibi, C. Curino, J. Leeka, A. Jindal, J. Zhao, J. Camacho-Rodríguez, and Y. Tian, "Sibyl: Forecasting time-evolving query workloads," *Proceedings of the ACM on Management of Data*, vol. 2, no. 1, p. 1–27, Mar. 2024. [Online]. Available: http://dx.doi.org/10.1145/3639308

[11] Y. Qi, H. Hu, D. Lei, J. Zhang, Z. Shi, Y. Huang, Z. Chen, X. Lin, and Z.-J. M. Shen, "Timehf: Billion-scale time series models guided by human feedback," 2025. [Online]. Available: https://arxiv.org/abs/2501.15942

[12] S. J. Taylor and B. Letham, "Forecasting at scale," *The American Statistician*, vol. 72, no. 1, pp. 37–45, 2018. [Online]. Available: https://doi.org/10.1080/00031305.2017.1380080

[13] B. Lim, S. O. Arik, N. Loeff, and T. Pfister, "Temporal fusion transformers for interpretable multi-horizon time series forecasting," 2020. [Online]. Available: https://arxiv.org/abs/1912.09363

[14] D. Bertsimas and N. Kallus, "From predictive to prescriptive analytics," 2018. [Online]. Available: https://arxiv.org/abs/1402.5481

AMARPREET SINGH BASSAN (Member, IEEE) is currently a Senior Software Engineer at Microsoft, Redmond, Washington, USA, where he has been since February 2022. He received his Post Graduate program certificate in AI and ML from The University of Texas at Austin in 2020 and holds a Master's degree in Software Development from Boston University.

With over 17 years of experience at Microsoft, he has held roles including Customer Engineer (August 2019 – March 2022) and Support Escalation Engineer (January 2017 – July 2019), focusing on enhancing application performance, architecting customer solutions, and leading support for Azure analytics. Prior to Microsoft, he worked as a Software Engineer at Infosys from October 2005 to September 2007, specializing in database administration within software performance engineering.

His professional interests include leading AI innovation, driving customer success through strategic application of new learnings, software design, and database performance optimization. He is passionate about emerging technologies and their application in enterprise solutions.

Mr. Bassan is a Member of IEEE (since July 2025). His certifications include Fundamentals of MCP from Hugging Face (2025) and Google Prompting Essentials from Coursera (2025). He was also recognized with an excellence award during his tenure as a Customer Engineer at Microsoft.