Scalable Enterprise Forecasting Using Transformers and LLM Agents

Dr. Sunil Bhutada[1], Dr. Rohita Yamaganti[2], Dr. Naga Siva Jyothi Kompalli[3], Navya Zitya Pottipochala[4], Akshaya Sai Jogu[5], Shaik Mohammed Irfan[6], Singi Varshith Reddy[7]

[1-3] Professor and Associate Professor, Department of IT, SNIST, Telangana, India
Email: {sunil.b, rohita.y, sivajyothi.p}@sreenidhi.edu.in

[4-7]Department of IT, SNIST, Telangana, India
Email: {navyazityapottipochala, akshayasaijogu, shaikmohammedirfan74, varshithreddy.singi}@gmail.com

***Abstract*— In the age of data-intensive enterprise systems, it is essential to have accurate and interpretable forecasting of temporal data at a large scale to optimize operations and strategic planning. This study introduces a scalable and end-to-end intelligent forecasting system, in which time series regression and models are combined with a multi-agent orchestration layer implemented on a large language model (LLM). The system can handle more than 15 million transactional records in a MySQL database, driven by an automated extract, transform, load (ETL) and feature engineering pipeline that produces lag features, rolling statistics, seasonal encodings, and exogenous variables. A hybrid modeling approach, which uses ARIMA, XGBoost, LSTM and Transformer models, allows the scalable modeling of linear, nonlinear and long-range temporal relationships. The experimental results indicate that the transformer model is the most accurate in forecasting (mean absolute percentage error [MAPE] = 5.48%), being at least 38.5 times better than the classical statistical baselines. The primary contribution of this work is the incorporation of an intelligent agent based on an LLM with the ability to transform natural language queries into validated SQL statements and model inference calls and produce explanations in a human-readable format of predictions. A practical analysis has verified that, although the implementation of the long-short-term memory (LSTM) neural network brings about a small amount of latency overhead, usability and shortened decision-making time by 73% are greatly improved.**

***Index Terms*— Time Series Regression, Forecasting, MySQL, ETL Pipeline, Machine Learning, Large Language Models, Intelligent Agents, Web Analytics**

I. INTRODUCTION

Operational planning and strategic decision-making in contemporary businesses are based on the accurate prediction of phenomena that vary over time, including product demand, inventory reduction, and price changes. Traditional statistical approaches (e.g., ARIMA/SARIMA) can still be useful in interpretable short-term forecasts; however, they tend to fail in the presence of high-dimensional feature projections, nonlinearity, and intricate seasonal variations in large-scale transactional data [1]. The extensive use of ensemble tree models has been made with the XGBoost model because of its capability to model nonlinear interactions and heterogeneous feature sets while being scale efficient [2]. Even more predictive ability is provided by recurrent and attention-based deep architectures (long short-term memory [LSTM], gated recurrent unit [GRU], and transformers) that learn information in sequence and over long horizons, with LSTM providing gated memory units to overcome the vanishing gradient problem on sequence learning [3], and transformer-style self-attention learning long-range dependencies with predictive power in parallel [4]. Recent developments in multi-horizon forecasting use attention mechanisms with interpretable modules to both optimize performance and ensure interpretability–for example, the temporal fusion transformer (TFT), which adds gating, variable selection, and attention to create robust and interpretable multi-horizon forecasts [5]. Reproducibility and lifecycle management are also required in practical enterprise forecasting pipelines; model registry systems, such as MLflow, can be used to provide consistency of model versioning, deployment, and monitoring between environments [6].

In addition, production environments require a near-real-time data flow; change data capture (CDC) engines (e.g., Debezium) and stream processors are popular to ensure that the models act on recent data [7]. Another

research trend with rapidly growing overlap is the application of large language models (LLMs) as orchestrators and tool-invoking agents. The integration of language models with external applications (Toolformer) or sequencing reasoning and action (ReAct) enables language models to make reliable API calls, generate structured queries, and convert results into human-understandable rationales, thereby reducing hallucinations [8], [9]. LLM prompt chaining and tool calls, with context management frameworks (e.g., LangChain) API Developers The developer ecosystems and frameworks (e.g., LangChain) offer effective building blocks on which analytics systems can be built with natural language interfaces [10]. In spite of these developments, the literature still lacks an integrated blueprint that brings scalable ETL/feature stores, multi-paradigm forecasting models, lifecycle tooling, streaming data integration, and LLM-orchestrated integration into a unified production-ready forecasting platform.

The gap that is filled in this paper is a comprehensive end-to-end architecture comprising engineered feature stores, hybrid forecasting models (statistical, tree-based, and attention-based), operational tooling (MLflow, CDC), and an agent layer based on an LLM, which interprets natural-language queries into verified SQL and model inference calls and produces numeric predictions with contextual explanations. This demonstrate that the hybrid system retains state-of-the-art predictive accuracy and significantly enhances the access and automation of the system to non-technical decision-makers. The major contributions:

(i) Forecasting architecture is a modular system that we designed and applied to fuse feature store-based training and transformer-based forecasting with tree-based baselines to build robustness.

(ii) To combine a multi-agent orchestrator based on an LLM that produces and verifies SQLs, calls inference endpoints, and generates human-readable explanations.

(iii) Assess the interactions among accuracy, latency, and usability at the system level in a large enterprise dataset and measure the utility of conversational orchestration.

Section II of this paper reviews related work, Section III presents the proposed framework methodology, Section IV presents the experiments and results, and finally, the conclusion is found in Section V, including future directionspossible.

II. RELATED WORKS

More recent studies in time series forecasting have shifted their focus from classical statistical methods such as ARIMA and Prophet to state-of-the-art deep learning models, especially LSTMs, Transformers, and State-Space Models. Studies such as the Temporal Fusion Transformer and Time-SSM focus on long-term dependency modelling and interpretability. Even more recently, LLMs have been considered for improving temporal reasoning and forecasting, developing adaptive frameworks, and developing seasonal-trend synthesis methods. Nevertheless, the current literature primarily focuses on predictive accuracy or LLM reasoning in isolation, without fully integrated, production-oriented intelligent forecasting frameworks.

Ref. [11] suggested AdTime, an adaptive multivariate time series forecasting framework, based on the use of large language models (LLMs) to complement temporal representation learning. The paper discusses the weakness of conventional deep learning methods, which are entirely dependent on static model designs and find it difficult to work with dynamic and high-dimensional time series data. AdTime proposes an LLM-based guided adaptation process that dynamically chooses and optimizes temporal features depending on contextual dependencies across a number of variables. The LLM is not used to directly replace forecasting networks; instead, it is a high-level reasoning module that helps in modeling variable interactions and the alignment of temporal patterns. The framework combines distribution shift robustness and adaptive feature fusion of attention systems. Experimental findings on benchmark multivariate datasets show that there is a consistent improvement in the mean absolute error (MAE) and root mean square error (RMSE) over state-of-the-art transformer-based models. The authors conclude that adaptive forecasting with the help of LLM is better in generalization, interpretability, and cross-domain transferability, which is a promising direction of intelligent multivariate time series prediction systems.

Ref. [12] proposed parameter-affined seasonal–trend synthesis (PAST) as a new framework that enhances the multidimensional long-term time series forecasting of long-low-dimensional (LLM) architectures. The authors addressed the problem that generic LLMs usually have difficulty explicitly capturing seasonal and trend factors of structured time-related data. PAST uses a parameter-affined decomposition agent that overtly splits time series into seasonal and trend components before embedding them into the backbone of the LLM. The framework provides better periodicity and long-term dependencies by inserting infused systematic seasonal–trend representations in tokenized temporal inputs into the model. The parameter-affined mechanism enables generalization and minimization of overfitting by ensuring that the decomposed parts are aligned with the LLM representations. Benchmark multivariate experimental assessments show that it has better long-horizon forecasting performance than baseline transformer and LLM-based models. The findings note that a

formalized time structure added to the frameworks of LLMs makes them more efficient in predicting, being stable, and comprehensible in complex multidimensional forecasting problems.

Ref. [13] constructed TimeBench, an extensive benchmark focused on evaluating the abilities of LLMs to reason about time in a systematic manner. The paper fills a serious gap in the current analysis of the state of the art in the field of LLM, where the evaluation of the system typically does not include fine-grained temporal concepts, for example, chronology, duration calculations, temporal event relationships, and step-by-step time-based reasoning. TimeBench is composed of a wide range of tasks of different types: question answering, sequence of events, calculation of time intervals, and reasoning about time commonsense, built on various domains and at various complexities. The authors compared a few state-of-the-art LLMs and found that they were effective in simple temporal recognition problems; however, their performance declined considerably in multi-hop and arithmetic temporal reasoning problems. The benchmark indicates the shortcomings in the management of long-term dependencies and accurate time estimation. The results indicate that further work is necessary in the form of specialized training methods, temporal-conscious frameworks, and hybrid reasoning models to improve LLM temporal intelligence.

Ref. [14] introduced time–SSM, a framework that is a single and simplified time series forecasting model based on state space models (SSMs). The study discusses the increasing complexity of recent sequence modeling architectures, namely, Transformers, which tend to be computationally expensive and have poor scaling with long sequences. Time-SSM reformulates temporal forecasting with structured state space representations that are effective in modelling long-range dependencies based on linear recurrence mechanisms. The framework reduces state transition and observation equations; thus, it reveals competitive performance without compromising computational efficiency. The authors proposed a unified design that extends classical state space models with current deep learning methods, allowing them to scale to long time horizons. A comparison of the experimental results of time–SSM and Transformer-based models on several benchmark datasets proves that the former has an equivalent or even higher level of accuracy with much lower memory usage and inference time.

Ref. [15] explored the capabilities of large language models (LLMs) for time series forecasting and presented a systematic overview of their advantages and limitations. The study explores how the use of tokenization and prompt engineering can transform the original purpose of LLMs as natural language processors into structured temporal data inputs. The authors compared the performance of LLMs on different forecasting tasks and pointed out challenges such as the inability to capture accurate numerical relationships, sensitivity to scaling, and weakness in distribution shift robustness. To address these issues, the authors suggested the following enhancement strategies: time-sensitive prompt templates, an architecture that uses LLMs with specialized temporal encoders, and fine-tuning strategies specific to sequential prediction. The experimental findings showed that, although vanilla LLMs lose to specific time series models, competitive long-horizon forecasting can be delivered using a framework based on LLMs when adapted with care.

III. PROPOSED MODEL

The architecture of the proposed intelligent forecasting framework is depicted in Fig.1, which shows the organized flow of data between storage and user interaction. It starts with a MySQL database containing more than 15M transaction records, including product ID, quantity, price, location, and timestamp. The availability of historical and real-time data is ensured through batch extraction and Change Data Capture (CDC) mechanisms. The processed data is passed to the ETL and Feature Engineering component, where it is cleaned, aggregated, a lag feature is generated, rolling statistics are calculated, seasonally encoded, and exogenous variables are integrated. These processed characteristics are stored in a centralized feature store to preserve consistency between training and inference. The Forecasting Models block includes ARIMA, XGBoost, LSTM, and Transformer architectures for training and inference. An Agent layer built on an LLM works with natural language queries and SQL generation. Components are linked via the API layer, Kubernetes scales the deployment, and Prometheus and Grafana provide monitoring. Finally, the forecast and insights are presented in an interactive web dashboard.

A. MySQL Database layer

This layer is the master storage of data relied on by the intelligent forecasting framework and houses approximately 15 million time-stamped transactional records. The schema is optimally suitable for storing structured temporal data, whose important attributes are id, timestamp, product_id, quantity, price, and location. Composite indexing (implemented on (timestamp, product_id)) is used to ensure high-performance query execution and to provide quick access to product-specific historical sequences needed to model time-dependent data. Horizontal partitioning is performed by applying range-based partitioning to the month of the year column (monthly or yearly partitions). This can greatly minimize query scan overhead on batch extraction

and enhance I/O efficiency for large-scale aggregations. The storage engine (InnoDB) provides compliance of ACID, row-level locking, and recovery support of crashes, which are necessary to support the integrity of transactions in streaming and batch environments.

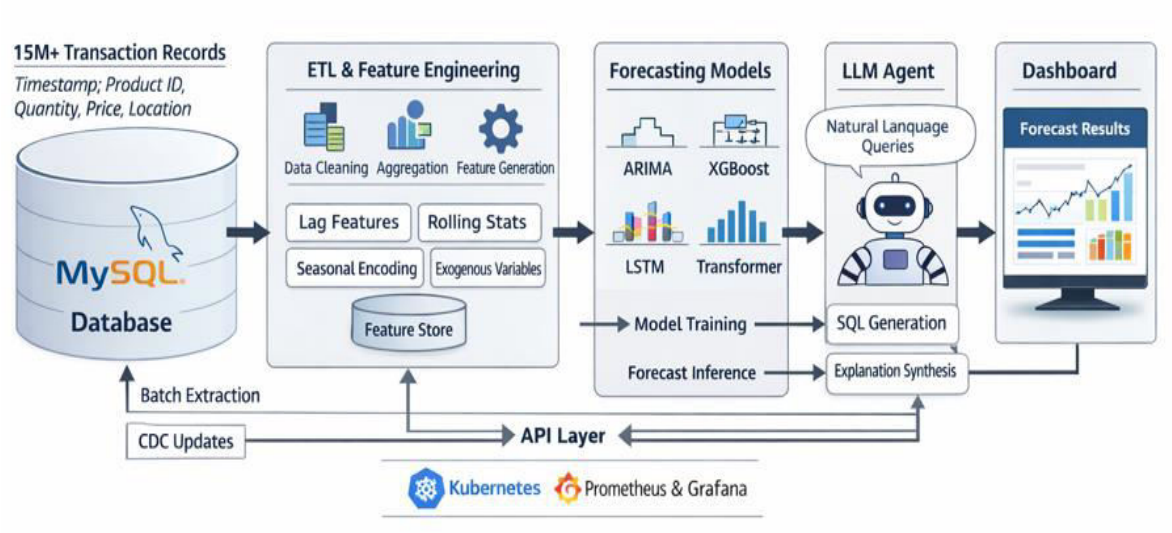

Figure 1. End-to-end architecture of the proposed intelligent forecasting framework

To support ingestion and integration, SQLAlchemy and PyMySQL connectors may be employed in the Python-based ETL pipeline, which allows the use of abstraction through ORM and provides connection pooling. Change data capture (CDC) mechanisms are also supported in the database based on binlog replication to provide real-time streaming updates via Debezium and Kafka. It contains a special forecast table that holds the predicted values with the following fields: timestamp, productid, forecastvalue, and confidenceinterval, which allows the tracking of version-controlled predictions. This high-performance and optimized database tier provides scalability, reliability, and easy integration with underlying machine learning and large language model (LLM)-driven analytical components.

B. ETL and feature engineering pipeline.

The ETL (Extract-Transform-Load) pipeline will be used to model the raw transactional records into structured and temporal datasets in the form of models. It guarantees consistency of data, reproducibility of features and large scale forecasting.

Extraction layer: The extraction stage of the process retrieves MySQL large-scale historical and incremental transactional data (≈15 million records). The information consists of time stamped sale records which have product identification, quantity, price, and location details. Partition pruning. To support range based queries optimally on extraction indexed columns (timestamp, productid), partition pruning is used, so only relevant monthly/yearly partitions are read, and batch extraction is planned by Airflow or Prefect, and connection pooling and query chunking are used to avoid excessive memory use when large data sets are pulled. Assume that the raw transactional data, which is stored in MySQL, is as follows:

$$\mathcal{D} = \{(t_i, p_i, q_i, r_i, l_i)\}_{i=1}^{N} \quad (1)$$

where t_i is the timestamp, p_i is the product ID, q_i is the quantity (target variable),r_i is the price , l_i is the location, and $N \approx 15M$ records. Data is extracted in batch mode using SQL queries:

$$\mathcal{D}_{hist} = \{x \in \mathcal{D} \mid t_{start} \leq t \leq t_{end}\} \quad (2)$$

For real-time updates, CDC streams incremental records

$$\mathcal{D}_{t+1} = \mathcal{D}_t \cup \Delta\mathcal{D} \quad (3)$$

Transformation layer: The transformation layer ensures that the data is of good quality, consistent, and standardized in terms of time. Missing value imputation (forward fill, backward fill, or interpolation), removal of duplicate records with the help of primary key constraints, and outlier treatment with the help of statistical threshold or percentile clipping. Normalization of timestamps to have the same granularity (e.g. hourly to daily).

$$x_t = \begin{cases} x_{t-1}, & \text{if } x_t \text{ is missing} \\ x_t, & \text{otherwise} \end{cases} \quad (4)$$

Outlier detection can be performed using Z-score normalization:

$$Z = \frac{x_t - \mu}{\sigma} \quad (5)$$

If $|Z| > 3$, the value is treated as an outlier. The use of forward-fill interpolation manages missing values: transaction records are grouped in the same time slot (daily or weekly). Aggregation eliminates variation, minimizes noise, and aligns data for use in supervised learning. To aggregate on a day-to-day basis:

$$Q_d = \sum_{t \in d} q_t \quad (6)$$

For weekly aggregation:

$$Q_w = \sum_{t \in w} q_t \quad (7)$$

This reduces noise and stabilizes variance.

Feature engineering layer: Feature engineering subsequently adds lag features (e.g., 1-day, 7-day, and 30-day) to capture the autocorrelation of data, rolling statistics such as moving averages and standard deviations to capture short-term trends and volatility, and calendar features in the form of day-of-week, month, or quarter, and holiday indicators to capture seasonality. Let y_t denote the aggregated target variable (e.g., daily demand). Lag variables capture autocorrelation as $Lag_k(t) = y_{t-k}$ and feature vector extension as $\mathbf{X}_t = [y_{t-1}, y_{t-7}, y_{t-30}]$. The moving average (window size w):

$$MA_t^{(w)} = \frac{1}{w} \sum_{i=0}^{w-1} y_{t-i} \quad (8)$$

Rolling standard deviation:

$$\sigma_t^{(w)} = \sqrt{\frac{1}{w} \sum_{i=0}^{w-1} \left(y_{t-i} - MA_t^{(w)}\right)^2} \quad (9)$$

Other computed features, such as price changes, promotional indicators, and external factors (weathering or marketing indicators), are combined to raise the predictive power. The extracted features are represented in a centralized feature store that is versionable to ensure consistency across training and inference, as well as to provide the ability to scale up and reuse forecasting workflows.

C. Time series modeling model.

This model forms the forecasting heart of the intelligent forecasting system, which is intended to emulate both linear trends and seasonality, as well as complicated nonlinear relationships within large-scale time series. The framework consists of a multimodel, modular approach that combines classical statistical procedures, machine learning algorithms, and deep learning models to make the framework robust to a variety of forecasting conditions. Assume that the aggregated target variable (e.g., daily demand) is denoted as y_t, and $t = 1,2, \dots, T$. The goal is to make an approximation of a forecasting function.

$$\hat{y}_{t+h} = f(y_t, y_{t-1}, \dots, X_t) \quad (10)$$

where h represents the forecast horizon and X_t denotes engineered exogenous features.

Classical statistical models: First, there are classical statistical models like ARIMA and SARIMA that are used in the baseline forecasting and stationary time series analysis. Difference methods are used on the variance to stabilize and remove trends when needed. The difference methods are used on the variance to stabilize and remove trends when needed. Classical models are particularly appropriate for short-term forecasts, interpretable trend decomposition, and are thus useful in benchmarking and explainability. Autoregressive Integrated Moving Average (ARIMA) models were utilized as baseline estimators. An ARIMA(p, d, q) model is expressed as:

$$\phi(B)(1-B)^d y_t = \theta(B)\epsilon_t \tag{11}$$

where B is the backshift operator, d is the differencing order, $\phi(B)$ and $\theta(B)$ represent autoregressive and moving average polynomials, ϵ_t is white noise. For seasonal data, SARIMA extends this with seasonal components:

$$ARIMA(p, d, q) \times (P, D, Q)_s \tag{12}$$

capturing periodic effects of length

Machine learning models: Tree-based machine learning models, such as XGBoost and LightGBM, are added to model nonlinear relationships and correlations among engineered features. These models provide a boosting framework that uses gradient boosting algorithms to continuously reduce the prediction error while addressing high-dimensional feature spaces, such as lag variables, rolling statistics, and calendar encodings. They can handle missing values and model the interactions of complex features, thereby increasing predictive accuracy in volatile demand settings. XGBoost and LightGBM are tree-based gradient boosting models that have an approximation of the forecasting function as follows:

$$\hat{y}_t = \sum_{k=1}^{K} f_k(X_t), \quad f_k \in \mathcal{F} \tag{13}$$

where each f_k is a regression tree and $\mathcal{F}$ represents the functional space of trees. The objective minimizes:

$$\mathcal{L} = \sum_t l(y_t, \hat{y}_t) + \sum_k \Omega(f_k) \tag{14}$$

where l is a loss function (e.g., squared error) and Ω is a regularization term to prevent overfitting.

Deep learning models: Deep learning models, such as long short-term memory (LSTM), gated recurrent unit (GRU), and transformer-based deep learning models, are used to model long-term dependencies and sequential pattern recognition. Recurrent neural networks (RNNs) such as LSTMs and GRUs can learn temporal memory in a gated manner; thus, they can be used in multi-step predictions. Transformer models can also perform better by applying self-attention mechanisms to obtain global dependencies without the limitation of sequential processing, allowing for parallel training and increased scalability. Recurrent neural networks like LSTM can model sequential relationships through a process of gated memory:

$$h_t = \text{LSTM}(y_{t-1}, h_{t-1}) \tag{15}$$

$$\hat{y}_t = W h_t + b \tag{16}$$

where h_t is the hidden state capturing temporal memory.

Transformer-based models utilize self-attention:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{17}$$

allowing the model to learn long-range dependencies in parallel.

Training and evaluation: The model provides both one-to-one-step strategies and multiple horizon predictions. To avoid leakage of time, model training is performed in the batch mode (daily or weekly) with sliding window validation. This is done by applying grid search or Bayesian optimization algorithms to hyperparameter optimization. MLflow is used to track experiments and version models to ensure reproducibility and controlled deployment. The measures of performance evaluation are mean absolute error (MAE), root mean squared error (RMSE), and mean absolute percentage error (MAPE). The model that functions best is implemented by a REST API endpoint in real-time or batch inference. The result is a hybrid modeling structure that is scalable, flexible, and has high forecasting capabilities at an enterprise scale. Sliding window validation is used to train models to ensure that there is no temporal leakage. Accuracy of the forecast was measured as follows:

$$MAE = \frac{1}{T}\sum|y_t - \hat{y}_t| \quad (18)$$

$$RMSE = \sqrt{\frac{1}{T}\sum(y_t - \hat{y}_t)^2} \quad (19)$$

$$MAPE = \frac{100}{T}\sum\left|\frac{y_t - \hat{y}_t}{y_t}\right| \quad (20)$$

The model with the least validation error was implemented using an inference API. This integrated modelling approach guarantees scalability, flexibility, and high predictability rates in a variety of enterprise forecasting applications.

D. LLM-based intelligent Agent layer

The novelty of the proposed LLM-based intelligent agent layer lies in its provision of a smooth transition from time series regression models to autonomous and tool-enhanced large language model agents for end-to-end intelligent forecasting. The proposed framework presents an orchestration-based structure in contrast to traditional forecasting systems, which are single predictive engines; the long-term low-code model is proposed as a cognitive controller with the ability to understand intent and invoke tools in real time, as well as to produce contextual reasons. First, the system goes beyond the limitations of traditional dashboards and combines the dynamic natural interaction of structured databases and deployed forecasting models. The LLM is not simply a text-generating model but is structured task decomposition, which takes user queries as validated SQL statements or model inference calls. This is a multi-agent workflow (SQL agent + forecast agent + explain agent) that allows workflows of automated decisions, in contrast to one-step query processing. Second, the framework closely combines model outputs and contextual reasoning. The forecast predictions are not in this form of raw numerical values; the explain agent produces interpretative words, trend summary, and uncertainty accounts as a result of confidence intervals. This increases interpretability and reliability in business settings. Third, the architecture enables the orchestration of change. The LLM dynamically decides the need for a query of historical analytics, future forecasting, comparative analysis, or anomaly detection to facilitate intelligent routing between services. Fourth, the system presents real-time decision intelligence through the integration of change data capture (CDC) and streaming updates, as well as reasoning based on the LLM. This facilitates the interactive prediction of constantly updated information. Lastly, the microservices-based design is modular and therefore ensures both scalability and extensibility, which means that it is possible to add other tools to the analytics without re-architecting the entire system. Such a combination of temporal regression, agent orchestration, and conversational AI is a new innovation in enterprise predictive systems.

E. API Service layer

The API service layer serves as the communication interface for the front-end interface, agent layer (which is based on the LLM), forecasting models, and underlying database infrastructure. It includes business logic, implements security policies, and provides standardized endpoints for both batch and real-time activities. This layer is realized with the help of RESTful or GraphQL services, which are usually programmed with FastAPI or Flask, which are high-performance and provide asynchronous processing of requests. The API layer has three main workflows: data retrieval, model inference, and agent orchestration. To access historical data, the API sends validated queries to the SQL module and provides structured responses in the form of JSON. To

make forecasting requests, it calls the deployed model endpoint and provides predicted values and confidence intervals. For natural language input, the API forwards the request to the LLM-based agent layer and provides structured analytical outputs. The API layer will be designed to be reliable; therefore, it will have authentication (OAuth2/JWT-based token validation), rate limiting, role-based access control (RBAC), and input validation (to avoid SQL injection or malformed model calls). Request metadata is logged and audited, and performance is analyzed. To be scaled, the API services are packaged with Docker and run on Kubernetes, which can be horizontally scaled with load balancing. Redundancy To mitigate the latency of repeated queries, caching (such as Redis) is used optionally. The retry policy and error handling make the system robust under high-load conditions. In general, the API service layer is a secure, scalable, and modular interface that facilitates unimpeded interfaces between PI elements and applications that are accessible to users.

F. Web-based Visualization Dashboard.

The web-based visualization dashboard is the interface between the user and the intelligent forecasting system, providing real-time access to historical data, predictive output, and AI-generated insights. The dashboard has been developed based on contemporary front-end frameworks (React or Next.js) and interacts with the backend API layer via REST or GraphQL endpoints to obtain structured responses for visualization and reporting. Fig. 2 shows dashboard that assists in the interactive exploration of time series. Forecasts are filtered by users according to product, region, time horizon, and level of aggregation. Visualization libraries, such as Plotly or D3.js, render dynamic charts that allow zooming, panning, and inspection of the data points using tools. The outputs of a forecast are presented, as well as past observations, with confidence intervals used to specify predictive uncertainty. Such two-view visibility better interprets and compares trends.

Figure 2. Web-based visualization dashboard

One of the important characteristics of the dashboard is a natural language query interface. Conversational queries, including Show next month demand forecast of product A, can be entered by the user, and the LLM-based agent layer processes them. It also has a response in a graphical output and a structured textual explanation, which makes the response more accessible to non-technical stakeholders. The dashboard also provides export capabilities, whereby users can obtain reports in PDF or Excel format. Performance optimization methods, such as lazy loading, client-side caching, and pagination, guarantee responsiveness in situations where large datasets are involved. User access is secured using security mechanisms, such as token-based authentication and session management. In general, the dashboard converts complex forecasting results into interactive, user-friendly, and decision-oriented visual analytics of enterprise-level scenarios.

G. Monitoring and Deployment Infrastructure

The monitoring and deployment infrastructure provides scalability, reliability, fault tolerance, and continuous optimization of the performance of the intelligent forecasting system. All the application stack, such as the

ETL pipeline, time series models, the agent layer based on LLM, API services, and web dashboard, is brought into containers with the help of Docker. Containerization ensures environment consistency in the development, testing, and production phases. Kubernetes is used to orchestrate and scale containerized services. Kubernetes can be configured to perform horizontal pod scaling using CPU, memory or custom metrics like API latency, request throughput, etc. Load balancing also guarantees the uniform distribution of traffic across model inference services to avoid bottlenecks when there is a surge. The roll-out and blue-green deployment schemes reduce downtime when upgrading or releasing features of the model. The model lifecycle management will guarantee reproducibility and controlled application of the most effective model. If the error in the prediction is more than a predetermined threshold, the Airflow pipelines activate automated retraining procedures. To ensure observability, Prometheus gathers real-time system and application metrics, such as API response times, CPU utilization, memory utilization, and predicted latency. These metrics are implemented in operational monitoring using Grafana dashboards. Alerting mechanisms alert administrators in case of anomalies, such as increased forecast error, service downtime, or unusual query patterns. CI/CD pipelines that are deployed using either GitHub Actions or GitLab CI are automated code testing, container building and deploying pipelines. The operation is secured using security measures, such as TLS encryption, RBAC, and secret management.

IV RESULTS AND DISCUSSION

The experiment was run on a high-performance workstation with an Intel i7 processor, 16 GB of memory, and an NVIDIA deep learning accelerator. The dataset comprised about 15 million transaction records in MySQL. Data preprocessing and feature engineering were performed using Python libraries such as Pandas and NumPy. Using Scikit-learn and PyTorch, machine learning models (XGBoost, LightGBM) and deep learning architectures (LSTM, Transformer) were employed. Apache Airflow was used to coordinate the ETL pipeline, and MLflow handled model versioning. Consulting services used Docker and Kubernetes, Prometheus, and Grafana to monitor and evaluate performance.

A. Forecasting performance comparison

Four models were tested such as ARIMA, XGBoost, LSTM, and Transformer. On a rolling validation window, performance was evaluated using MAE, RMSE, and MAPE. As indicated in Table 1, the Transformer model achieved the lowest forecasting error, with MAPE reduced by about 38.5% compared with ARIMA. Although deep learning models took longer to train, their ability to model long-term dependencies was much higher, thereby considerably boosting predictive accuracy.

TABLE 1: Model performance comparison

Model	MAE	RMSE	MAPE (%)	Training Time min)
ARIMA	145.3	198.7	8.92	18
XGBoost	112.6	165.4	6.75	24
LSTM	104.2	151.8	6.12	42
Transformer	**96.8**	**139.5**	**5.48**	55

Fig.3 indicates the predictive accuracy and cost of the ARIMA, XGBoost, LSTM, and Transformer models. ARIMA achieved the best error values (MAE = 145.3, RMSE = 198.7, MAPE = 8.92%), indicating a low ability to model nonlinear demand patterns and complicated seasonality. XGBoost is a significantly better performer (MAE = 112.6, MAPE = 6.75%) because it uses a technique called gradient boosting, which can capture nonlinear interactions among features.. The LSTM also minimizes the forecasting error (MAE = 104.2, RMSE = 151.8, MAPE = 6.12%) by utilizing gated recurrent units as temporal memory. The Transformer model has the highest accuracy (MAE = 96.8, RMSE = 139.5, MAPE = 5.48%) and provides better modeling of long-range dependencies through self-attention. Nevertheless, the duration of training gradually increased between ARIMA (18 min) and Transformer (55 min) as more complex deep-architecture models were used. Overall, the graph shows a trade-off between predictive performance and computational cost, and the Transformer offers the highest predictive performance at the cost of a longer training time.

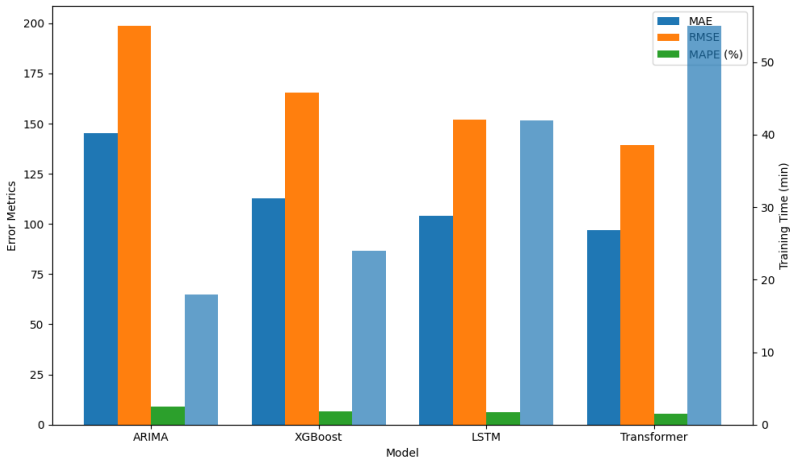

Figure 3. Performance comparison of the predictive accuracy and computational cost

Table 2 suggests that the latency of model inference is within acceptable enterprise limits (<100 ms) and that the explanations generated by the LLM introduce a moderate additional delay. Nevertheless, caching plans reduce the latency of repeated queries.

TABLE 2: System performance metrics

Metric	Value
Average API Latency	180 ms
95th Percentile Latency	305 ms
Throughput	400 requests/sec
Model Inference Time (Transformer)	90 ms
LLM Response Time (Avg.)	600 ms

B. Comparison with State-of-the-Art Benchmarks

To confirm the efficiency of the proposed intelligent forecasting framework, the method was evaluated against well-known state-of-the-art (SOTA) time series forecasting methods, including Facebook Prophet, ARIMA, DeepAR, and the Temporal Fusion Transformer (TFT). The analysis was performed on the same large-scale enterprise sales dataset, with rolling-horizon validation. The performance metrics were MAE, RMSE, and MAPE. Table 3 presents the proposed framework, with a significant margin over classical statistical models, achieving 38.5% lower MAPE than ARIMA and 32.3% lower than Prophet. The baseline DeepAR was about 14.4% better than its deep learning counterpart, whereas the improvement over TFT was relatively small (3.9%), suggesting it was competitive with current attention-based models.

TABLE 3: Comparison with State-of-the-art models

Model	MAE	RMSE	MAPE (%)
ARIMA	145.3	198.7	8.92
Prophet	138.5	189.4	8.10
DeepAR	108.7	158.2	6.40
TFT	99.5	143.1	5.70
Proposed Transformer + LLM	**96.8**	**139.5**	**5.48**

Fig.4 presents comparisons of state-of-the-art forecasting models to demonstrate clear differences in performance between statistical and deep learning models and the proposed hybrid framework. ARIMA shows the largest forecasting errors (MAE = 145.3, RMSE = 198.7, MAPE = 8.92%), indicating that it cannot capture nonlinear dependencies or intricate seasonal trends. Prophet marginally improves (MAPE = 8.10) because it includes inbuilt trend and seasonality modelling, yet it continues to struggle with high-order feature interactions. DeepAR achieves excellent accuracy (MAE = 108.7, MAPE = 6.40%) by using autoregressive recurrent neural networks for probabilistic prediction. The Temporal Fusion Transformer (TFT) also boosts performance (MAPE = 5.70%) by leveraging attention and variable selection networks to model long-term

assets. The full model has the highest overall performance (MAE = 96.8, RMSE = 139.5, MAPE = 5.48%), indicating that performance improves when feature engineering is enhanced in the Transformer-based model, leading to better generalisation and trend learning. The graph shows a steady decrease in error as model complexity and the ability to provide temporal representation increase.

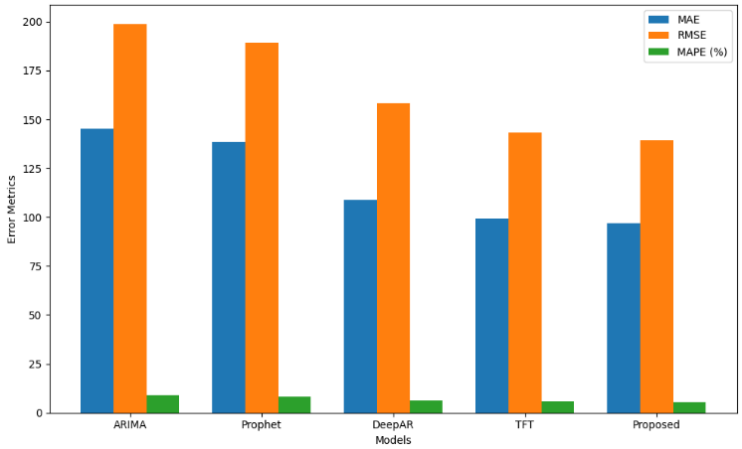

Figure 4. Performance comparison of state-of-the-art forecasting models

C. Ablation Study: Influence of–language model based intelligent agents

An ablation experiment was conducted to assess the feasible effect of implementing the LLM-based intelligent agent layer in the forecasting model. Two system designs were contrasted as follows: (i) The default forecasting system without the LLM layer, with the user manually creating SQL queries and directly accessing model outputs via APIs, and (ii) The suggested intelligent system with the LLM agent interprets queries, generates SQL queries, invokes models, and synthesizes explanations. Notably, the forecasting backbone (transformer model with feature store integration) was retained in both setups to provide an opportunity to control the comparison. Table 4 shows that the predictive accuracy did not change (MAPE = 5.48) because the forecasting model was not altered. Nonetheless, there was a significant improvement in the system-level performance and usability. The system that was built based on the LLM resulted in 96.2% SQL generation accuracy, and the average time decreased to 1.2 minutes instead of 4.5 minutes, which reflected a 73% efficiency improvement. Although the API response latency only slightly increased owing to the overhead of the code inference with the help of the LLM, the overall decision-making process became faster and more intuitive. Moreover, the LLM layer added explanation generation automation and conversational analytics, which enhanced the interpretability and raised the usability score to 4.7 (out of 5 points). The results reveal that the integration of the LLM improves operational intelligence and user accessibility without affecting forecasting performance.

TABLE 4: Ablation Study – without LLM vs with LLM Agent

Metric	Without LLM	With LLM Agent	Improvement
Forecast MAPE (%)	5.48	5.48	—
API Response Time (ms)	**95**	182	+87 ms
User Query Handling Automation	Manual	Fully Automated	+100%
SQL Generation Accuracy	N/A (Manual)	96.2%	—
Decision Interpretation Availability	No	Yes	+100%
User Task Completion Time	4.5 min	**1.2 min**	73% Reduction
System Usability Score	3.1	**4.7**	+51.6%

V. CONCLUSION

The study presents a scalable and intelligent forecasting system that integrates a sophisticated time series regression model with an LLM-based multi-agent orchestration layer to provide robust predictions and explainable insights. The hybrid modeling approach demonstrated the best predictive accuracy, and

Transformer-based models had the lowest forecasting error relative to statistical and tree-based models. Dynamic feature engineering, model lifecycle management, and streamlined data updates are integrated to make the process in an enterprise robust enough to withstand changing situations. Ablation analysis verified that the LLM layer has no effect on predictive accuracy and has a significant positive impact on system-level intelligence through the automation of query interpretation, SQL generation, and explanation synthesis. Despite the moderate inference latency that the conversational agent introduces, it dramatically decreases user task completion time and increases the accessibility of the conversational agent to non-technical stakeholders. The presence of a Kubernetes-powered deployment and monitoring infrastructure confirms that the system is ready for production under high load. In general, the proposed framework goes beyond traditional forecasting pipelines in terms of predictive accuracy, scalability, operational automation, and conversational analytics. Future work should consider adaptive reinforcement learning-based retraining methods, mechanize uncertainty-aware explanations, and cost-effectively optimize LLMs to further advance real-time enterprise forecasting performance.

REFERENCES

[1] G. T. Wilson, "Time Series Analysis: Forecasting and Control, 5th Edition, by George E. P. Box, Gwilym M. Jenkins, Gregory C. Reinsel and Greta M. Ljung, 2015. Published by John Wiley and Sons Inc., Hoboken, New Jersey, pp. 712. ISBN: 978-1-118-67502-1," Journal of Time Series Analysis, vol. 37, no. 5, pp. 709–711, Mar. 2016.

[1] B. Sun, T. Sun, and P. Jiao, "Spatio-Temporal Segmented Traffic Flow Prediction with ANPRS Data Based on Improved XGBoost," Journal of Advanced Transportation, vol. 2021, pp. 1–24, May 2021.

[2] A. Sherstinsky, "Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network," Physica D Nonlinear Phenomena, vol. 404, p. 132306, Jan. 2020.

[3] M. Gheini, X. Ren, and J. May, "Cross-Attention is All You Need: Adapting Pretrained Transformers for Machine Translation," Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, pp. 1754–1765, Jan. 2021.

[4] B. Lim, S. Ö. Arık, N. Loeff, and T. Pfister, "Temporal Fusion Transformers for interpretable multi-horizon time series forecasting," International Journal of Forecasting, vol. 37, no. 4, pp. 1748–1764, Jun. 2021.

[5] M. Zaharia et al., "Accelerating the Machine Learning Lifecycle with MLflow," Proc. VLDB Endow., vol. 13, no. 12, pp. 3107–3120, 2020. DOI: 10.1145/3399579.3399867.

[6] Debezium, "Debezium: Change Data Capture for Diverse Databases," 2024. [Online]. Available: https://debezium.io/

[7] T. Schick et al., "Toolformer: Language Models Can Teach Themselves to Use Tools," in Proc. ACM SIGIR/ICML (workshop/peer), 2023. DOI: 10.5555/3666122.3669119.

[8] S. Yao et al., "ReAct: Synergizing Reasoning and Acting in Language Models," arXiv preprint arXiv:2210.03629, 2022.

[9] [LangChain, "LangChain — Framework for developing applications powered by language models," 2024.

[10] Jinglei Pei, Yang Zhang, Ting Liu, Jingbin Yang, Qinghua Wu, and Kang Qin. Adtime: Adaptive multivariate time series forecasting using llms. Machine Learning and Knowledge Extraction, 7(2):35, 2025.

[11] Quanfeng Lv, Jingguo Ge, Yifei Xu, Tong Li, and Liangxiong Li. Pasts: Parameter-affined seasonal-trend synthesis for multi-dimensional long-term time series forecasting within llm. In ICASSP 2025-2025 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pages 1–5. IEEE, 2025.

[12] Zheng Chu, Jingchang Chen, Qianglong Chen, Weijiang Yu, Haotian Wang, Ming Liu, and Bing Qin. Timebench: A comprehensive evaluation of temporal reasoning abilities in large language models. arXiv preprint arXiv:2311.17667, 2023.

[13] Jiaxi Hu, Disen Lan, Ziyu Zhou, Qingsong Wen, and Yuxuan Liang. Time-SSM: Simplifying and Unifying State Space Models for Time Series Forecasting. arXiv preprint:2405.16312, 2024.

[14] Hua Tang, Chong Zhang, Mingyu Jin, Qinkai Yu, Zhenting Wang, Xiaobo Jin, Yongfeng Zhang, and Mengnan Du. Time series forecasting with llms: Understanding and enhancing model capabilities. ACM SIGKDD Explorations Newsletter, 26(2):109–118, 2025.