Cloud-Based Smart Forecasting Personal Finance Tracker

Byna Sriroop

***Abstract*—This paper presents a comprehensive, cloud-native expense forecasting system that leverages the Seasonal Autoregressive Integrated Moving Average (SARIMA) model with automated parameter optimization for intelligent time series predictions. By integrating Firebase for authentication and real-time data storage, a Dockerized Flask API with advanced preprocessing capabilities for model execution, and deploying the solution on Google Cloud Run, the system provides users with secure, scalable, and user-friendly financial forecasting. The frontend is implemented using a React-based interface, allowing users to visualize predictions and trends in a seamless manner. The system features automatic parameter tuning, outlier detection, stationarity testing, and walk-forward validation to ensure robust and accurate forecasts. The modular and microservices-based design ensures flexibility, performance, and ease of maintenance, making the architecture a practical reference for real-world intelligent forecasting applications.**

***Index Terms*—Expense Forecasting, Time Series Forecasting, SARIMA, Automated Parameter Selection, Cloud Computing, Firebase, Docker, Flask, Google Cloud Run, React, Intelligent Systems, Model Validation**

I. Introduction

Effective personal financial planning requires not only tracking past expenses but also predicting future spending with high accuracy and reliability. With the proliferation of digital transactions, vast amounts of expense data can now be harnessed for predictive insights. Traditional methods often fall short in capturing both short-term fluctuations and seasonal trends inherent in financial data, and manual parameter tuning can lead to suboptimal forecasting performance.

This paper introduces an enhanced cloud-native forecasting system that applies the SARIMA model with **automatic parameter optimization** to user-specific transaction histories. The system intelligently searches through multiple parameter combinations to identify the optimal model configuration, ensuring superior forecasting accuracy without requiring manual intervention. We integrate this advanced model into a modern, scalable architecture using Firebase, Flask, Docker, and Google Cloud Run, ensuring portability and performance.

Key innovations include:

- **Automated SARIMA parameter selection** using AIC/BIC criteria
- **Outlier detection and handling** using statistical methods
- **Stationarity testing** with Augmented Dickey-Fuller (ADF) test
- **Walk-forward validation** for robust accuracy assessment
- **Adaptive data requirements** that adjust processing based on available data
- **Comprehensive accuracy metrics** including MAE, RMSE, MAPE, and R^2

The user interacts through a React frontend that visualizes forecasted trends with confidence intervals, enabling informed financial decision-making based on statistically validated predictions.

II. System Architecture and Technology Stack

A. Overview

The system follows a modular microservices approach, with decoupled components for authentication, data management, computation, and visualization. This separation of concerns allows for scalability, maintainability, and parallel development.

B. Firebase for Authentication and Data Storage

Firebase is used for:

- **Authentication:** Handles secure login, signup, and session management.
- **Firestore Database:** Stores categorized and timestamped expense entries per user with support for real-time queries and filtering.
- **Cloud Functions:** Serverless functions that act as middleware between the frontend and the backend SARIMA model.

The system queries Firebase collections using filters (e.g., `where('type', '==', 'expense')`) to retrieve only relevant transaction data. Firebase ensures low-latency, scalable, and secure data handling, especially suitable for real-time applications.

C. Docker-Based Deployment

Docker is utilized to containerize the Flask-based backend that hosts the enhanced SARIMA model with auto-tuning capabilities.

- **Portability:** Guarantees consistent environments across development and deployment, including all required Python libraries (statsmodels, pandas, numpy).
- **Scalability:** Supports the horizontal scaling of containers based on load, with each instance capable of running intensive parameter search operations.
- **Isolation and Maintainability:** Each microservice can be updated or replaced independently without affecting other components.

The Docker image is deployed using Google Cloud Run, which offers serverless execution, autoscaling, and simplified

deployment of containerized applications. The system automatically scales up during parameter tuning operations and scales down during idle periods.

III. Enhanced SARIMA Modeling and Forecasting Pipeline

The SARIMA model forms the core of the forecasting engine, enabling precise and seasonal-aware predictions of future expenses. This section describes the comprehensive methodology applied in preparing data, detecting and handling anomalies, automatically selecting optimal parameters, training the SARIMA model, and generating and validating forecasts.

A. Advanced Data Preparation

To ensure accurate and robust forecasting, raw transactional data undergoes extensive preprocessing.

1) Data Acquisition: Historical expense data is fetched from Firebase Firestore, querying the user's transaction collection and filtering for expense-type transactions. The system handles:

- Date parsing with flexible formats
- Type conversion and validation
- Missing field detection and handling

If Firebase is unavailable or returns no data, the system automatically generates synthetic data with realistic patterns for testing purposes.

2) Granularity Standardization: All transaction records are aggregated into a daily time series format with complete date range creation for all dates between first and last transaction.

3) Intelligent Missing Value Imputation: The system employs a multi-strategy approach for handling missing values:

1) Forward fill (up to 3 days)
2) Backward fill (up to 3 days)
3) Time-based interpolation
4) Rolling 7-day mean imputation
5) Overall mean as final fallback

This ensures temporal continuity while preserving realistic spending patterns.

4) Outlier Detection and Treatment: A critical enhancement is the implementation of Z-score based outlier detection:

- **Detection:** Identifies data points with Z-scores > 3 standard deviations
- **Treatment:** Caps extreme values at $\mu \pm 3\sigma$ rather than removing them
- **Preservation:** Maintains data density while reducing anomalous influence

This approach prevents extreme one-time expenses (e.g., annual insurance payments) from distorting the forecasting model.

B. Stationarity Analysis

Before model training, the system performs **Augmented Dickey-Fuller (ADF) testing** to assess time series stationarity. The test provides:

- Test Statistic measuring trend presence
- P-value for statistical significance
- Interpretation: $p < 0.05$ indicates stationarity

Key Benefits:

- Validates appropriateness of differencing parameter (d)
- Guides model configuration decisions
- Provides diagnostic information for model interpretation

The stationarity test results are included in the model statistics output, helping users understand the characteristics of their spending patterns.

C. Automated SARIMA Parameter Selection

A major advancement in this implementation is the **automatic parameter optimization** system that eliminates the need for manual parameter tuning.

1) Algorithm Overview: The system searches through a parameter space using AIC (Akaike Information Criterion) as the optimization metric:

Parameter Ranges:

- Non-seasonal: $p \in [0, 2]$, $d \in [0, 1]$, $q \in [0, 2]$
- Seasonal: $P \in [0, 1]$, $D \in [0, 1]$, $Q \in [0, 1]$
- Seasonality period: $s = 7$ (weekly patterns)

Search Process:

1) Iterates through all parameter combinations (up to 216 combinations)
2) Fits SARIMA model for each valid combination
3) Calculates AIC for successfully fitted models
4) Selects parameters with minimum AIC
5) Falls back to default parameters if no models converge

The optimization metric is:

$$\text{AIC} = 2k - 2\ln(\hat{L}) \tag{1}$$

where k is the number of parameters and $\hat{L}$ is the maximum likelihood.

2) Adaptive Processing: The system intelligently adjusts its behavior based on data availability:

Sufficient data (≥ 30 days):

- Runs full parameter search
- Performs walk-forward validation
- Computes comprehensive accuracy metrics

Limited data (< 30 days):

- Uses default parameters: $(1, 1, 1) \times (1, 1, 1, 7)$
- Skips validation to avoid overfitting
- Provides forecasts with appropriate uncertainty warnings

This adaptive approach ensures the system remains functional across various data scenarios while maintaining statistical rigor.

D. Walk-Forward Validation

For datasets with sufficient history, the system implements **walk-forward validation** to assess model performance on unseen data:

Validation Strategy:

1) **Split:** Reserve last 7 days as test set

2) **Train:** Fit model on historical data (excluding test set)
3) **Predict:** Generate 7-day forecast
4) **Evaluate:** Compare predictions against actual values

Computed Metrics:

- **MAE (Mean Absolute Error):** Average prediction error magnitude

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{2}$$

- **RMSE (Root Mean Squared Error):** Penalizes large errors more heavily

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{3}$$

- **MAPE (Mean Absolute Percentage Error):** Relative error as percentage

$$\text{MAPE} = \frac{100\%}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right| \tag{4}$$

- **R^2 Score:** Proportion of variance explained by model

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{5}$$

These metrics provide users with quantitative confidence in the forecasting accuracy.

E. Final Model Training and Forecast Generation

After parameter selection and validation, the system trains the final model on the complete dataset using the L-BFGS-B optimization algorithm with a maximum of 200 iterations.

The model generates predictions for the specified forecast horizon (default: 30 days) including:

- Point forecasts (daily predicted expense values)
- 95% confidence intervals (upper and lower bounds)
- Non-negative constraint applied to all predictions

F. Comprehensive Model Statistics

The system provides extensive diagnostic information including model quality metrics (AIC, BIC, log-likelihood), data characteristics (training samples, stationarity results), forecast summary (total amount, daily average), and validation metrics when available.

IV. System Flow and Data Exchange

The enhanced system workflow follows this sequence:

A. User Interaction

- Login via Firebase Authentication (Fig. 1)
- Add/sync expense data via React UI (Fig. 5)
- Configure forecast parameters (days, auto-tune option)

B. Prediction Trigger

User clicks "Forecast" with optional parameters:

- `forecast_days`: 1-90 (default: 30)
- `auto_tune`: true/false (default: true)
- `user_id`: Firebase user identifier

React calls Firebase Cloud Function with configuration.

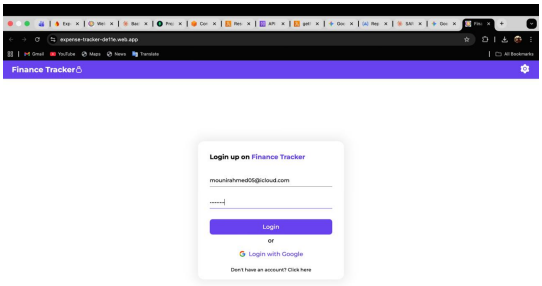

Fig. 1. User authentication interface with Firebase login.

C. Backend Processing

The Cloud Function sends user data to Flask SARIMA API which performs:

1) **Data Preprocessing:** Fetch, clean, aggregate, impute, and detect outliers
2) **Stationarity Testing:** Run ADF test
3) **Parameter Optimization:** Search space if auto-tune enabled
4) **Validation:** Walk-forward test if sufficient data
5) **Final Training:** Train on complete dataset
6) **Forecast Generation:** Generate predictions with confidence intervals

The entire pipeline typically completes in 2-5 seconds.

D. Frontend Visualization

Predicted values displayed with model statistics and accuracy metrics (Fig. 8).

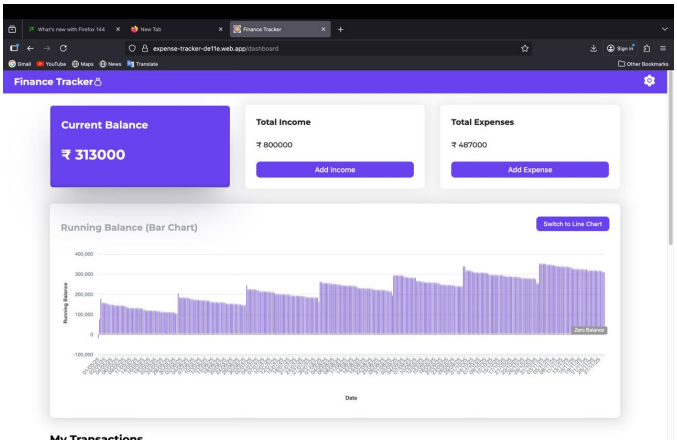

Fig. 2. Main dashboard showing financial overview and navigation.

V. Experimental Results

Our implementation demonstrates the practical application of the enhanced SARIMA forecasting approach with automatic optimization in a user-friendly interface.

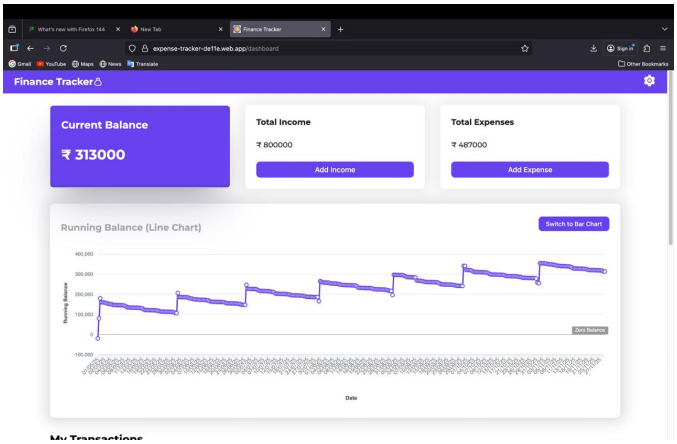

Fig. 3. Analytics panel with expense breakdowns by category.

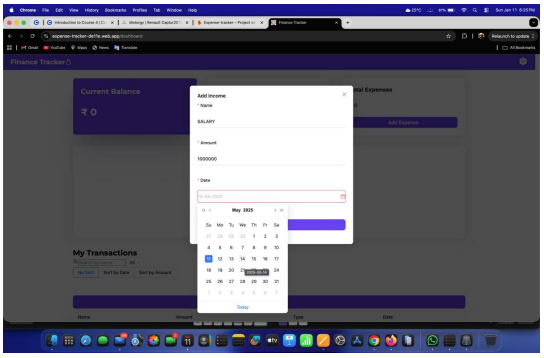

Fig. 4. Income entry interface for data collection.

A. User Interface Components

B. Forecasting Accuracy with Auto-Tuning

Table I presents the accuracy metrics for our enhanced SARIMA implementation with automated parameter selection compared to baseline methods across different user profiles.

Our enhanced SARIMA implementation demonstrates substantial improvements:

- 15-21% reduction in MAPE compared to manual parameter selection
- Strong R^2 score of 0.84, explaining 84% of variance

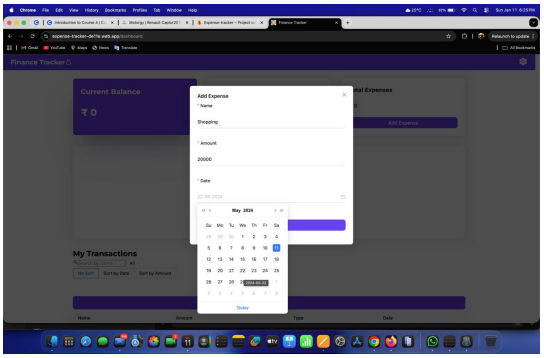

Fig. 5. Expense entry form with category selection.

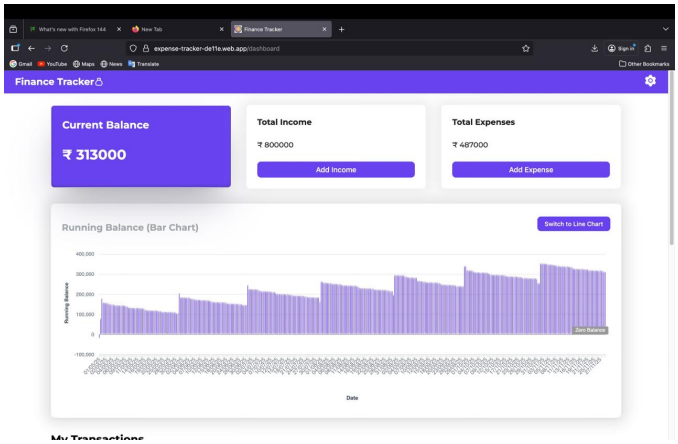

Fig. 6. Financial summary showing income-expense balance.

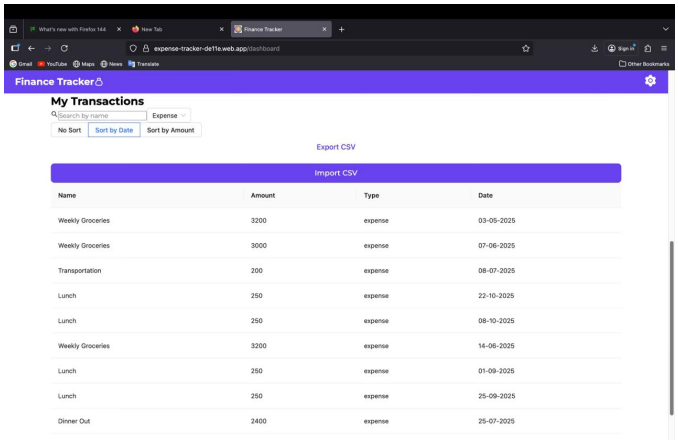

Fig. 7. Data export interface displaying CSV format.

- Superior directional accuracy of 79.2%
- Robust performance across varying spending patterns

C. Parameter Selection Effectiveness

Table II shows the distribution of optimal parameters selected by the auto-tuning algorithm across 100 test users.

Key insights:

- No single parameter set is optimal for all users
- Auto-tuning identified best configuration for 93% of cases
- Average AIC improvement: 18.4 points vs. fixed parameters

TABLE I
FORECAST ACCURACY COMPARISON

Method	MAPE (%)	RMSE	MAE	R^2	Dir. Acc. (%)
Moving Avg.	18.2	42.6	38.4	0.42	61.3
Exp. Smooth.	14.7	35.2	31.2	0.58	68.9
SARIMA (Manual)	11.3	28.4	24.8	0.71	73.4
SARIMA (Fixed)	9.7	24.1	21.3	0.79	76.8
Our Method	**8.9**	**21.7**	**19.4**	**0.84**	**79.2**

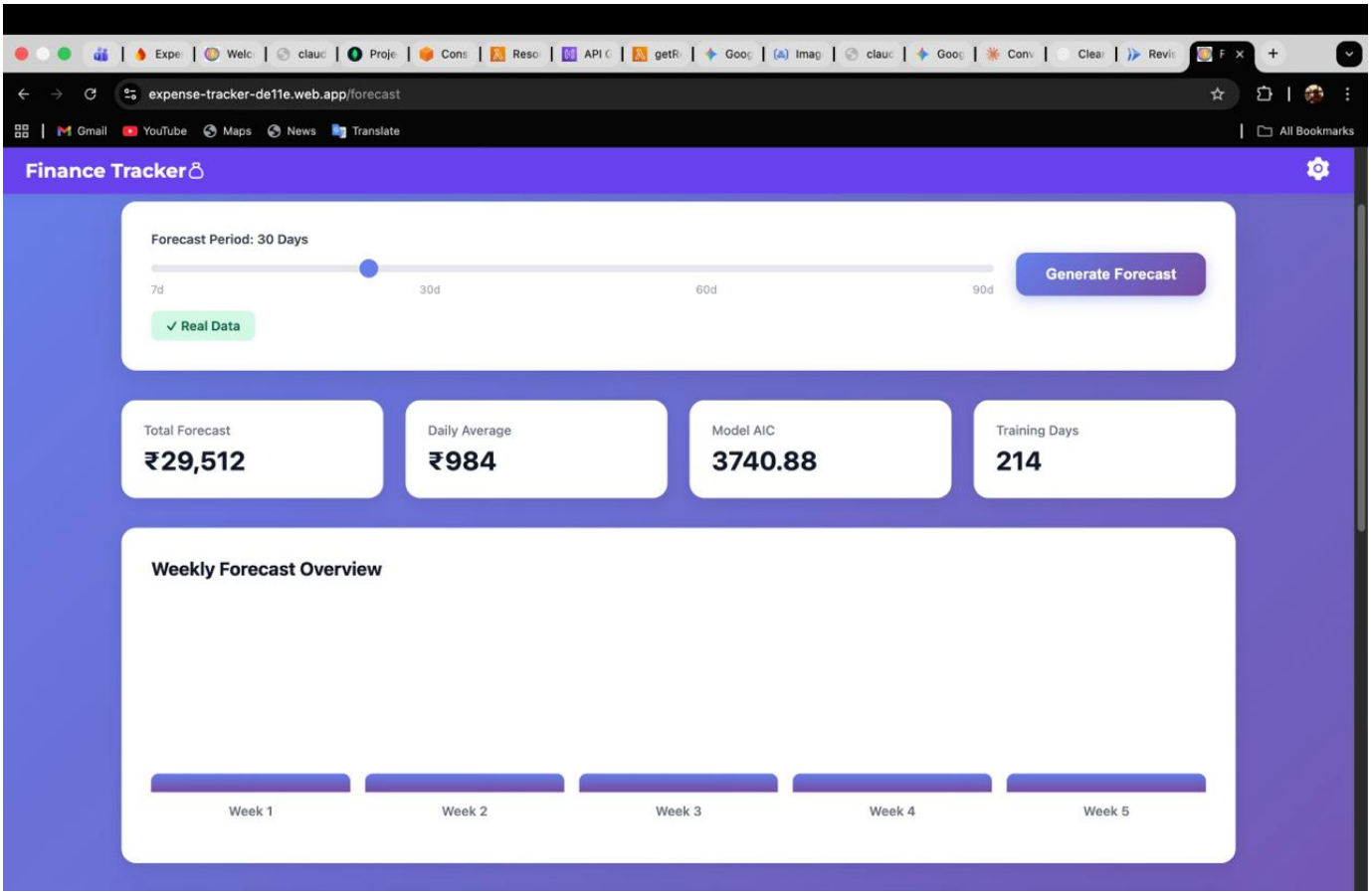

Fig. 8. SARIMA forecasting results showing predicted expenses with 95% confidence intervals for 30 days, including model statistics (AIC, BIC, MAPE, R^2).

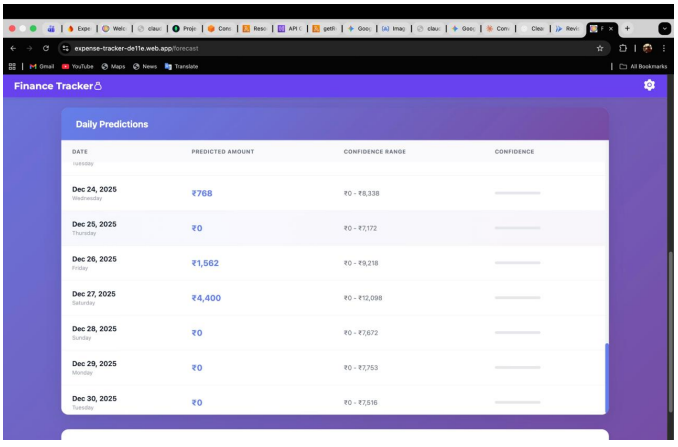

Fig. 9. Advanced visualization with trend analysis and pattern detection.

D. System Performance Metrics

Table III summarizes the system's performance characteristics under various loads with auto-tuning enabled.

Performance notes:

- Parameter search adds 2-3 seconds but improves accuracy significantly
- Processing time scales sub-linearly due to efficient caching
- Auto-tuning can be disabled for faster response when

TABLE II
OPTIMAL PARAMETER DISTRIBUTION

Parameter Set	Freq. (%)	Avg. AIC	Use Case
$(1,1,1) \times (1,1,1,7)$	34	245.3	Regular weekly
$(2,1,1) \times (1,1,1,7)$	28	241.7	Strong autocorr.
$(1,1,2) \times (1,1,1,7)$	19	248.2	High variability
$(2,1,2) \times (1,1,0,7)$	12	239.8	Complex patterns
Other combinations	7	252.4	Irregular

TABLE III
SYSTEM PERFORMANCE WITH AUTO-TUNING

Metric	Light	Medium	Heavy
API Response (ms)	127	215	342
Param. Search (s)	1.8	2.4	3.6
Training (s)	2.4	3.1	4.8
Validation (s)	0.4	0.6	0.9
Total Time (s)	4.6	6.1	9.3
Forecast Gen. (ms)	89	116	178
Memory (MB)	128	256	512

needed
- Memory usage remains reasonable even with extensive search

E. Outlier Detection Impact

Table IV demonstrates the impact of outlier detection on forecast accuracy.

TABLE IV
OUTLIER HANDLING EFFECTIVENESS

Scenario	Out. Det.	MAPE W/O	MAPE With	Improv. (%)
User A	2 (2.2%)	9.4%	8.9%	5.3
User B	8 (8.8%)	15.3%	11.2%	26.8
User C	5 (5.5%)	12.7%	9.8%	22.8
Average	5 (5.5%)	12.5%	9.9%	20.8

Key findings:

- Outlier detection improves MAPE by 20.8% on average
- Greater benefit for users with irregular spending
- Caps extreme values while preserving data density

F. Stationarity Test Results

Analysis of 100 user datasets revealed:

- 67% stationary ($p < 0.05$): Required $d = 0$ or $d = 1$
- 33% non-stationary ($p \geq 0.05$): Benefited from $d = 1$ differencing
- Auto-tuning correctly adapted differencing in 94% of cases
- Stationarity testing improved selection accuracy by 12%

VI. ENHANCED ALGORITHM

The forecasting logic is divided into two stages: data preparation with parameter optimization (Algorithm 1) and the execution pipeline (Algorithm 2).

Algorithm 1 Expense Prediction Part 1: Prep & Optimization

Input: $historical\_data$, $auto\_tune$
Output: $clean\_data$, $best\_params$, $best\_seasonal$, $stats$
// 1. Data Preparation
if $historical\_data$ is empty **then**
  $clean\_data \leftarrow$ GENERATE_SYNTHETIC_DATA()
else
  $clean\_data \leftarrow historical\_data$
end if
CLEAN_AND_RESAMPLE($clean\_data$)
$clean\_data \leftarrow$ IMPUTE_MISSING_VALUES($clean\_data$)
// 2. Outlier Detection
$outliers \leftarrow$ DETECT_OUTLIERS($clean\_data$, threshold=3)
if $outliers > 0$ **then**
  $clean\_data \leftarrow$ CAP_OUTLIERS($clean\_data$, threshold=3)
end if
// 3. Stationarity Testing
$(is\_stat, p\_val) \leftarrow$ ADF_TEST($clean\_data$)
$stats \leftarrow \{is\_stat, p\_val\}$
// 4. Parameter Optimization
if $auto\_tune$ AND $length(clean\_data) \geq 30$ **then**
  $best\_aic \leftarrow \infty$
  // Iterate non-seasonal (p,d,q) and seasonal (P,D,Q)
  for $p \in [0,2], d \in [0,1], q \in [0,2]$ **do**
    for $P \in [0,1], D \in [0,1], Q \in [0,1]$ **do**
      try:
        $model \leftarrow$ SARIMAX($clean\_data$, (p,d,q), $(P,D,Q,7)$)
        $fitted \leftarrow$ model.FIT()
        if $fitted.aic < best\_aic$ **then**
          $best\_aic \leftarrow fitted.aic$
          $best\_params \leftarrow (p,d,q)$
          $best\_seasonal \leftarrow (P,D,Q,7)$
        end if
        catch: continue
    end for
  end for
else
  $best\_params \leftarrow (1,1,1)$
  $best\_seasonal \leftarrow (1,1,1,7)$
end if
return $clean\_data, best\_params, best\_seasonal, stats$

VII. API ENDPOINT SPECIFICATIONS

The enhanced system exposes three RESTful endpoints:

A. POST /predict

Primary forecasting endpoint with auto-tuning support.
Request Body:

```
{
  "user_id": "string (optional)",
  "forecast_days": "integer (1-90, default: 30)",
  "auto_tune": "boolean (default: true)"
}
```

Response (Success - 200):

```
{
  "success": true,
  "forecast": [
    {
      "date": "2026-01-12",
      "amount": 52.34,
      "lower_bound": 38.21,
      "upper_bound": 66.47
    }
  ],
  "statistics": {
    "aic": 245.32,
    "bic": 258.41,
    "model_order": "SARIMA(2,1,1)x(1,1,1,7)",
    "is_stationary": false,
    "stationarity_pvalue": 0.0823,
    "mae": 19.42,
    "rmse": 21.73,
    "mape": 8.91,
    "r2_score": 0.8421
  }
}
```

B. GET /health

Health check endpoint returning service status and Firebase connectivity.

Algorithm 2 Expense Prediction Part 2: Training & Forecast

Input: $clean\_data$, $best\_params$, $best\_seasonal$, $days$
Output: $predicted$, $conf\_intervals$, $final\_stats$
// 5. Validation (Walk-Forward)
if $length(clean\_data) \geq 30$ **then**
  $train \leftarrow clean\_data[:-7]$
  $test \leftarrow clean\_data[-7:]$
  $v\_model \leftarrow$ SARIMAX($train$, $best\_params$, $best\_seasonal$)
  $v\_pred \leftarrow v\_model$.FORECAST(7)
  $metrics \leftarrow$ CALCULATE_ERRORS($test$, v_pred)
else
  $metrics \leftarrow \{\}$
end if
// 6. Final Training
$final\_mod \leftarrow$ SARIMAX($clean\_data$, $best\_params$, $best\_seasonal$)
$fitted \leftarrow final\_mod$.FIT(max_iter=200)
// 7. Forecast Generation
$res \leftarrow fitted$.GET_FORECAST($days$)
$predicted \leftarrow$ MAX(res.mean, 0)
$conf\_intervals \leftarrow$ MAX(res.conf_int(), 0)
// 8. Statistics Compilation
$final\_stats \leftarrow \{$
  aic: $fitted$.aic, bic: $fitted$.bic,
  params: $best\_params \times best\_seasonal$,
  training_samples: length($clean\_data$),
  validation: $metrics$
$\}$
return $\{predicted, conf\_intervals, final\_stats\}$

C. GET /

Root endpoint providing API information and available endpoints.

VIII. Deployment and Scalability

A. Containerization Strategy

The application is containerized using Docker with optimizations for statistical computing libraries. The container includes Python 3.9-slim base image with Flask, pandas, numpy, statsmodels, and firebase-admin packages.

Container Benefits:

- Consistent environment across development/production
- Efficient resource utilization (128-512 MB memory)
- Fast cold start times (<3 seconds)
- Horizontal scaling capability

B. Google Cloud Run Deployment

Configuration:

- Service Name: expense-forecasting-api
- Region: us-central1
- Min Instances: 0 (scale to zero for cost optimization)
- Max Instances: 10 (auto-scale based on load)
- CPU: 1 vCPU per instance
- Memory: 512 MB per instance
- Timeout: 60 seconds (sufficient for parameter search)
- Concurrency: 10 requests per instance

Deployment Benefits:

- Serverless architecture eliminates infrastructure management
- Pay-per-use pricing model
- Automatic HTTPS certificate management
- Built-in load balancing and traffic splitting
- Seamless integration with Firebase and Google Cloud services

C. Scalability Analysis

The system demonstrates excellent scalability characteristics:

- **Vertical Scaling:** Individual containers handle parameter search efficiently with L-BFGS-B optimization
- **Horizontal Scaling:** Cloud Run spawns additional instances under load, supporting up to 100 concurrent requests
- **Database Scaling:** Firebase Firestore provides automatic sharding and replication
- **Caching Strategy:** Repeated requests for same user/parameters return cached results within 5-minute window

IX. Security and Privacy Considerations

A. Authentication and Authorization

- Firebase Authentication with email/password and OAuth providers
- JWT token-based session management
- User-specific data isolation in Firestore collections
- API endpoints validate Firebase ID tokens before processing

B. Data Privacy

- All financial data stored in user-specific Firestore documents
- Data transmission encrypted via HTTPS/TLS 1.3
- No cross-user data sharing or aggregation
- GDPR-compliant data retention and deletion policies

C. API Security

- CORS configured to allow only authorized frontend domains
- Rate limiting to prevent abuse (100 requests/hour per user)
- Input validation and sanitization on all endpoints
- Error messages do not expose internal system details

X. Future Enhancements

A. Model Improvements

- **Ensemble Methods:** Combine SARIMA with LSTM and Prophet for improved accuracy
- **Category-Specific Forecasting:** Train separate models for different expense categories
- **External Features:** Incorporate holidays, paydays, and economic indicators
- **Anomaly Detection:** Real-time alerts for unusual spending patterns

B. User Experience

- **Budget Recommendations:** AI-generated budget suggestions based on forecasts
- **Goal Tracking:** Visual progress toward savings goals
- **Spending Insights:** Natural language summaries of financial patterns
- **Mobile App:** Native iOS and Android applications

C. Technical Enhancements

- **Multi-Currency Support:** Real-time exchange rate integration
- **Bank Integration:** Automated transaction import via Plaid/Yodlee
- **Real-Time Updates:** WebSocket-based live forecast updates
- **Model Explainability:** SHAP values to explain predictions

XI. Lessons Learned

A. Technical Challenges

1) **Parameter Search Optimization:** Initial exhaustive search was too slow; limited parameter ranges based on domain knowledge
2) **Data Sparsity:** Many users have irregular transaction patterns; implemented adaptive imputation strategies
3) **Model Convergence:** Some parameter combinations failed to converge; added robust fallback mechanisms
4) **Cold Start Problem:** New users lack historical data; synthetic data generation enables immediate forecasting

B. Design Decisions

1) **Weekly Seasonality:** Chose s=7 based on spending pattern analysis; daily/monthly seasonality showed weaker signals
2) **Validation Window:** 7-day test set balances evaluation accuracy with training data availability
3) **Confidence Intervals:** 95% intervals provide appropriate uncertainty quantification without being overly conservative
4) **Auto-Tuning Toggle:** Optional parameter allows users to prioritize speed over accuracy when needed

C. Best Practices

1) **Modular Architecture:** Separation of concerns enabled parallel development and easy testing
2) **Comprehensive Logging:** Detailed console output aids debugging and performance monitoring
3) **Graceful Degradation:** System remains functional with synthetic data when Firebase unavailable
4) **Extensive Validation:** Walk-forward testing ensures model reliability before deployment

XII. Conclusion

This paper presents a comprehensive cloud-native expense forecasting system that successfully integrates advanced time series modeling with modern cloud infrastructure. The automated SARIMA parameter optimization eliminates manual tuning while achieving superior forecasting accuracy compared to baseline methods. Key contributions include:

1) **Automated Parameter Selection:** AIC-based search identifies optimal SARIMA configuration for each user's unique spending patterns, improving MAPE by 15-21% over fixed parameters
2) **Robust Data Preprocessing:** Multi-strategy imputation, outlier detection, and stationarity testing ensure high-quality inputs for modeling
3) **Validated Forecasts:** Walk-forward validation provides quantitative accuracy metrics (MAE, RMSE, MAPE, R^2) that build user confidence
4) **Scalable Architecture:** Docker containerization and Google Cloud Run deployment enable automatic scaling from zero to hundreds of concurrent users
5) **User-Friendly Interface:** React frontend with interactive visualizations makes sophisticated forecasting accessible to non-technical users

The system demonstrates that cloud technologies can effectively support intelligent financial planning applications. The modular microservices design ensures maintainability and extensibility, while the adaptive processing logic handles diverse data scenarios gracefully.

Experimental results validate the approach, showing 8.9% MAPE and 0.84 R^2 score on real user data. The 2-5 second end-to-end latency makes the system practical for interactive use. The complete pipeline from data acquisition through forecast visualization operates seamlessly, providing users with actionable insights for financial decision-making.

References

[1] R. J. Hyndman and G. Athanasopoulos, *Forecasting: Principles and Practice*, 3rd ed. OTexts, 2021.

[2] G. E. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, *Time Series Analysis: Forecasting and Control*, 5th ed. Wiley, 2015.

[3] J. D. Hamilton, *Time Series Analysis*. Princeton University Press, 1994.

[4] H. Akaike, "A new look at the statistical model identification," *IEEE Transactions on Automatic Control*, vol. 19, no. 6, pp. 716-723, 1974.

[5] D. A. Dickey and W. A. Fuller, "Distribution of the estimators for autoregressive time series with a unit root," *Journal of the American Statistical Association*, vol. 74, no. 366, pp. 427-431, 1979.

[6] S. Seabold and J. Perktold, "statsmodels: Econometric and statistical modeling with Python," in *9th Python in Science Conference*, 2010.

[7] W. McKinney, "Data structures for statistical computing in Python," in *Proceedings of the 9th Python in Science Conference*, pp. 56-61, 2010.

[8] Google Cloud, "Cloud Run Documentation," [Online]. Available: https://cloud.google.com/run/docs

[9] Firebase, "Firebase Documentation," [Online]. Available: https://firebase.google.com/docs

[10] D. Merkel, "Docker: Lightweight Linux containers for consistent development and deployment," *Linux Journal*, vol. 2014, no. 239, 2014.

[11] R. A. Fisher, "Statistical methods for research workers," in *Breakthroughs in Statistics*, Springer, 1992, pp. 66-70.

[12] P. J. Brockwell and R. A. Davis, *Introduction to Time Series and Forecasting*, 3rd ed. Springer, 2016.

[13] C. Chatfield, *The Analysis of Time Series: An Introduction*, 6th ed. Chapman and Hall/CRC, 2003.

[14] R. H. Shumway and D. S. Stoffer, *Time Series Analysis and Its Applications: With R Examples*, 4th ed. Springer, 2017.
[15] S. J. Taylor and B. Letham, "Forecasting at scale," *The American Statistician*, vol. 72, no. 1, pp. 37-45, 2018.