**Methodology**

**Overview**

**3.1 Dataset**

The dataset used in this study consists of Malaysia’s quarterly unemployment rate from 2010 to 2024. It provides a total of 60 data points, which are suitable for time series forecasting. The data is split chronologically into 80% for training and 20% for testing to preserve the temporal order.

**3.2 Pre-processing**

The dataset was first cleaned by removing missing values and ensuring the dates were correctly formatted and sorted in chronological order. For deep learning models, the data was scaled to a [0, 1] range using MinMaxScaler and transformed into lagged sequences to help the models learn time-based patterns. Although outliers were not removed, they were kept to preserve real-world fluctuations. An 80/20 time-based split was used for all models to ensure fair and consistent evaluation.

**3.3 Featured Engineering**

Feature engineering was used to help models learn patterns in the unemployment data. Lag features were created to include values from previous quarters, and seasonal decomposition revealed trend and seasonality. Deep learning models used scaled inputs (0–1) for better training, with sliding windows applied to convert time series into sequences. Stationarity was also tested using the ADF test to ensure valid forecasting for models like ARIMA.

**3.4 Models**

The models applied include ARIMA, SARIMA, and Exponential Smoothing for capturing linear trends and seasonality. Deep learning models like LSTM, GRU, RNN, and CNN were also used to capture complex, non-linear, and long-term dependencies. Each model was trained and evaluated using the same time-based split to ensure a fair comparison.

**3.5 Evaluation Metrices**

Model performance is evaluated using several key metrics. RMSE, MAE, and MAPE measure the accuracy of predictions, with lower values indicating better performance. R² shows how well the model explains changes in the actual data, where values closer to 1 mean a better fit. AIC helps compare models by considering both accuracy and complexity, favoring models that fit well without being too complicated. Residual analysis is also used to check if prediction errors are random and evenly spread, which suggests the model has captured the underlying patterns effectively.

**3.1 Dataset**

The dataset was obtained from openDOSM, Department of Statistics Malaysia. The selected dataset spans from 2010 Q1 to 2024 Q4, covering approximately 60 quarters. The data is produced based on the Labour Force Survey (LFS), which is designed to collect representative data on the labour force at national and state level. Consistency in methodology has been maintained to ensure that the data is comparable over time. The variables of the dataset includes :

* date: Start date of each quarter (used as datetime index)
* lf: Labour Force (in thousands)
* lf\_employed: Number of employed persons (in thousands)
* lf\_unemployed: Number of unemployed persons (in thousands)
* lf\_outside: Individuals outside the labour force (in thousands)
* p\_rate: Labour force participation rate (%)
* ep\_ratio: Employment-to-population ratio (%)
* A table of numbers and numbers

  Description automatically generatedA table of numbers and numbers

  Description automatically generatedu\_rate: Unemployment rate (%)

*Fig 4. Quarter Dataset Table*

*A graph with blue lines and numbers

Description automatically generated*

*Fig 5. Unemployment Rate Chart from 2010 to 2024*

* 1. **Pre-processing Steps**

**3.2.1 Loading and Parsing Dates**

The dataset was imported from a CSV file using the pandas library in Python. This provides a structured DataFrame object, ideal for time series manipulation and statistical analysis.

The date column, which contains quarter start dates, is converted into datetime format. This step is vital for enabling time-aware operations such as rolling windows, lag creation, and resampling.

Sorting the dataset chronologically ensures that models can learn the correct sequence of events over time.

By setting date as the DataFrame index, the time component becomes an intrinsic part of the data structure, which is crucial for time series forecasting models that rely on sequential input.

This is important as Time series models such as ARIMA or LSTM rely on the chronological order of data to learn trends and patterns. Any mistake in time ordering can severely affect prediction accuracy and lead to misleading results.

**3.3.2 Handling Missing Values**

Missing data can break time sequences and introduce inaccuracies during model training.

The .dropna() function is used to eliminate rows with missing values in the target variable (e.g., unemployment rate).

This is critical for ensuring that the model receives clean and consistent sequences of input. In this specific dataset, missing values are rare due to the high quality and completeness of official government statistical releases.

Deep learning models like LSTM or CNN expect continuous numerical input. Any missing values may interrupt this flow and produce errors during batch processing or lead to inaccurate training. Even in classical models, missing points can reduce model fit or cause the algorithm to fail.

**3.2.3 Outlier Handling**

Outliers are not systematically removed in the modeling scripts, but can be detected using z-score or IQR methods. Outliers are data points that lie significantly outside the expected range, potentially due to economic shocks, data entry errors, or anomalies.

**Z-score**: Measures how far a data point deviates from the mean in terms of standard deviations.

**IQR (Interquartile Range):** Identifies points that fall beyond 1.5×IQR from Q1 or Q3.

Removing genuine outliers could cause the model to be overly optimistic and fail under real-world stress scenarios. Keeping them allows the model to generalize better and be resilient to future economic shocks.

**3.2.4 Scaling (For Deep Learning)**

Scaling transforms the raw data into a normalized range, typically [0, 1], which is especially important for neural networks.Neural networks rely on gradient descent optimization. When inputs vary in scale (e.g., one feature ranges from 0–100, another from 0–1), it causes inefficient learning or convergence to poor local minima.

MinMaxScaler ensures that all input values have equal weight and that the model does not prioritize high-magnitude features.

Without scaling, models like LSTM, GRU, or CNN may struggle to converge or produce erratic forecasts. Normalized inputs help stabilize and accelerate training, making deep learning more effective and reliable.

**3.2.5 Lag Feature Creation (For Deep Learning)**

Deep learning models do not inherently understand time. To teach them temporal relationships, the data must be restructured into supervised learning format using **lagged sequences.**

A sliding window of n\_lags past observations is used as the input X, while the target y is the next value in the sequence.

For instance, with n\_lags=4, the model learns to predict unemployment in Q5 using Q1 to Q4 data.

This is the core structure that enables deep learning models to learn from historical patterns. Without this step, models like LSTM or CNN would not be able to detect seasonality, trends, or momentum in unemployment dynamics.

**3.2.6 Train/Test Split**

The dataset is divided into training and testing sets based on time.

An 80/20 split means the model is trained on the earliest 80% of the data and tested on the most recent 20%, mimicking a real forecasting scenario.

For deep learning, a validation set (typically 15–20% of training data) is further separated to tune hyperparameters and monitor generalization during training.

It is important to note that the same train/test split is used across all models, including ARIMA, SARIMA, LSTM, and other models, to maintain consistency and ensure a fair comparison of model performance. By keeping the evaluation framework uniform, the models can be directly compared using accuracy metrics such as RMSE, MAE, and MAPE, without any bias introduced by different data partitions.

Chronological splitting avoids lookahead bias, where future information unintentionally influences the model's learning. This ensures that the model’s performance on the test set is a true reflection of its ability to forecast unseen data

**3.3 Feature Engineering**

Feature engineering is the process of creating, transforming, or selecting input variables (features) in a dataset to improve the performance of machine learning or statistical models.

It means preparing and enhancing the raw data so that model can better understand the patterns and make more accurate predictions.

* + 1. **Lag Features**

To introduce temporal memory into the model by providing values from previous time steps as input features. Lagged versions of the target variable (e.g., unemployment rate) are created to help the model learn from past patterns. Common lags include 4, 8, 12, or 16 quarters. It is useful for models that rely on sequential information like ARIMA, LSTM, etc.

* + 1. **Seasonal Decomposition**

The time series is decomposed into trend, seasonal, and residual components using techniques like additive decomposition which helps identify seasonality and long-term trends in the unemployment rate. Libraries like statsmodels provide tools (seasonal\_decompose) to perform this. Decomposed components can be used as features or for residual analysis.

* + 1. **Scaling**

For deep learning models (LSTM, GRU, CNN, ANN), all numerical inputs are scaled to the range [0, 1] using MinMaxScaler. The purpose is to normalize all input values, ensuring that they fall within a consistent numeric range because deep learning models are sensitive to feature scale due to gradient-based optimization.This ensures stable training by avoiding large magnitude differences across features.

* + 1. **Sliding Windows**

This is used to create supervised learning samples from time series data by generating input-output pairs. Models like LSTM, GRU, and CNN need structured input in the form of sequential batches. Instead of feeding the model one continuous sequence, we divide the time series into overlapping "windows" that represent past observations.

For example, if your window size is 4, the chunks (windows) would look like:

Sample Unemployment Rate:

[4.5, 4.7, 4.8, 5.0, 5.2, 5.3, 5.4, 5.6]

First window: [4.5, 4.7, 4.8, 5.0] → predict the next value → 5.2

Then slide the window by one step:

Second window: [4.7, 4.8, 5.0, 5.2] → predict → 5.3

Models like LSTM, GRU, and CNN do not know anything about time unless it was given it to them in this way.

The sliding windows help the model see a short history, like the last 4 or 8 quarters, so it can learn how unemployment changed before and make a predict about what happens next.

* + 1. **Stationary Testing**

This is to determine whether statistical properties of the time series (mean, variance) are constant over time. It is important as many statistical models like ARIMA assume stationarity. If the data is non-stationary, forecasts may be invalid. Besides, detecting and correcting non-stationarity helps stabilize the model and improves predictive performance. In this case, the Augmented Dickey-Fuller (ADF) test is used.

**3.4 Models Implemented**

**3.4.1 ARIMA (AutoRegressive Intergrated Moving Average)**

ARIMA is used to capture linear relationships and short-term dependencies in time series data.

Implementation Steps:

1. Data Splitting for train and test sets

The time series is split into training and testing based on test\_size (80/20)

1. Determine seasonality using decomposition and seasonal strength

If the user hasn't specified whether to use seasonality, performs seasonal decomposition. Then, extracts the seasonal component and calculates strength of seasonality. If seasonality strength > 0.5, use SARIMA (seasonal=True).

1. Choosing best (p,d,q)

The best p,d,q is found by using pmdarima.auto\_arima(). If seasonal, P,D,Q,m is used. M = 4 as it is quarterly. To avoid overfitting, max\_d=1 and D=1 is used. stepwise=True helps in fast search through combinations

1. Implement Forecast  
   This code Forecasts test\_size number of steps ahead and returns predictions and Akaike Information Criterion (AIC) for model evaluation.
2. Evaluate metrics

Computes Root Mean Squared Error, Mean Absolute Error, Mean Absolute Percentage Error, and R-squared for a more accurate prediction

1. Residual Diagnogstics

Residuals are the differences between actual data and what ARIMA model predicted. They tell how well model fits the data and whether the model assumptions are met.

**3.4.2 SARIMA (Seasonal AutoRegressive Integrated Moving Average)**

SARIMA is an extension of ARIMA which is used to capture both trend and seasonality in time series data.

Implementation Steps:

1. Data Loading and Preprocessing
2. Model Fitting

Auto\_arima tries different combinations of the ARIMA parameters. It is because in SARIMA modelling process, the choice of parameters (p, d, q) and seasonal components (P, D, Q, m) is important for forecasting accuracy.

The code pmdarima.auto\_arima() carries out a stepwise automated grid search to determine the optimal combination of parameters. This means it starts with a simple model like ARIMA (1, 0, 1) first, then make some small changes such as increasing p to 2 or q to 2 to check if the new model is better (lower AIC). If new model is better, it will continue in that direction else tries another path.

In current model fitting, the maximums are not set to too high due to the number of data points existed which is 60. Higher maximums means the model will use more lags and moving average terms, and might lead to overfitting. Since, quarterly data is being used, therefore m, which is the seasonal period of 4 is chosen.

D, which is seasonal differencing = 1 means the time series is differenced once. Differencing tells the model how many times to difference the original series to remove trends and make it stationary. Stationary is important as ARIMA/SARIMA models assume statistical properties (means, variance, autocorrelation) of the series are constant over tine. Non-stationary data, such as with trends or changing variance can lead to poor forecasts and misleading model parameters.

1. Forecasting
2. Model Diagnostics

The model diagnostics is same as ARIMA, which includes Residual ACF/PACF plots, Ljung-Box test for autocorrelation, Evaluation metrics like RMSE, MAE, MAPE, R²

**3.4.3 Exponential Smoothing (Simple, Holt, Holt-Winters)**

This is a widely used method for time series forecasting. It uses weighted averages of past observations, where the weights decrease exponentially as observations get older. This means recent data points have more influence on the forecast than older one which is useful for short-term forecasting.

Implementation Steps:

1. Data Splitting for Train and Test Sets
2. Model Variants and Fitting

This function attempts to fit three exponential smoothing models and select the best based on the lowest AIC score.

Simple Exponential Smoothing is based on weighted averages of past observations, with weights that decrease exponentially for older data. T That means it look at past values and give more importance to recent ones. It smooths the values and predicts the next value will be similar to the last smoothed value. For this model, the data has no trend or seasonality. The forecast will be straight and in a flat line.

Holt’s Linear Trend is used when the data has trend but no seasonality. The data will shows a clear upward or downward trend overtime. This model includes two equation which is level and trend. The level is the average value at a certain point. The trend is the direction and speed of increase or decrease. When there is new data, the level and trend will be updated. The forecast will moves in straight line up or down, based on trend.

For Holt’s Winter Method, the data has both trend and seasonality. The forecast will have one more equation compared to Holt’s Linear Trend which is seasonality. In this method, level is the current average, trend is the increase or decrease over time and seasonality is the pattern that repeats (every 4 quarter). The pattern of forecast will be like going up or down, and also repeat the seasonal pattern.

1. Model Fitting

Each model is fit to the training data using the .fit() method. If a model fails, it is skipped. Fails might happen when the data is too short (not enough points for seasonal model), or does not have required structure, such as fit a seasonal model to non-seasonal data.

1. Model Selection

The best model is selected through AIC (Akaike Information Criterion). AIC is a metric that calculates how well a model fits the data, while penalizing for model complexity (number of parameters). At this point, lower AIC equals to a better model. It has the best trade-off between fit and simplicity. Besides, AIC prevents overfitting. The process is if the model cannot be fit, it will be ignored. Then, compare and select the model that is able to fit with lowest AIC.

1. Forecasting
2. Evaluation Metrics and Residual Diagnostics

**3.4.4 LSTM (Long Short-Term Memory)**

LSTM is a type of deep learning neural network which is designed to learn from time series-data when there is patterns depend on both recent and older values (long-term memory). It is helpful in capturing non-linear trends, handling long-term dependencies, and learn from past sequences to predict future.

Implementation Steps:

1. Data Scaling

By using MinMaxScaler, the time series values are converted to a range between 0 and 1. This is because neural networks are sensitive to the scale of input data. By scalling to [0, 1], LSTM ensures a faster and more stable training. Then the reshape function turns the series into a 2D array due to scikit-learn and will be transformed back to 1D array by flatten function.

1. Creating Lagged Data (Supervised Learning Format)

LSTM are required to learn from sequences of past values to predict the next value. A sliding window is used to build an input (X) and target (Y) pairs. The input X is the array of shape and target Y is the value to predict next value after window. Then, the window size (n\_lags) shows the number of past values are used to predict the next one. X will be reshaped after to have a third dimension, which is required by LSTM.

1. Train/Test Split

The test set uses the last test\_size samples while the rest is used for training LSTM model.

1. Model Building

The LSTM model is built by three layer, LSTM layer, dropout layer, and dense layer. LSTM layer learns temporal dependencies, which is the patterns from input sequences. The dropout layer randomly drops some connections during training to prevent overfitting. In my model, 20% of neurons are dropped. Dense layer outputs the final forecast value.

1. Model Compiling

This step is to specify the loss function and optimizer. In this step, Adam optimizer is used to improve learning and Mean Squared Error (MSE) used as loss function.

1. Model Training

The model is trained to learn patterns from past data and adjust its internal weights so accurate forecast can be made. Epochs is the number of times the modal see the entire training set. If too few, underfitting happens (model does not learn enough). If too many will cause overfitting, which means memorizes training data and poor on new data). Batch size is the number of samples processed before updating the model’s weights.

1. Making Predictions
2. Evaluate Performance

**3.4.5 GRU (Gated Recurrent Unit)**

GRU is a designed to handle sequential data, time series as example. It is similar to LSTM, but it has a simpler structure, making it more faster to train and less prone to overfitting on small datasets. GRU is use more on time series forecasting due to its ability on learning temporal dependencies. Besides, it has fewer gates than LSTM which makes it more efficient and also avoids problem that RNN suffer from which is the vanishing gradient. GRU has two main gates, update gate and reset gate. The update gate controls how much of the past information to keep and reset gate controls how much of previous state to forget.

Implementation Steps:

1. Data Scaling

Same to LSTM. The time series is scaled to the [0, 1] range using MinMaxScaler to help the neural network learn efficiently.

1. Create Lagged Data

The data is transformed into a supervised learning format by creating lagged input sequences. For example, use the past 8 quarters to predict the next quarter.

1. Train/Test Split

The data is split chronologically, with 80% for training and 20% for testing.

1. Model Building

The GRU model uses 4 layers for building. The first layer is the input layer, which accepts sequences of shape (n\_lags, 1), and n\_lags is the number of past quarters used as input. The second layer, GRU layer is the core of the model. It uses configurable number of units and processes the input sequence and learns temporal dependencies. Then it is the dropout layer, which helps the model regularization to prevent overfitting. It was set to 0.1 in this case. The last layer is the dense output layer, which is a fully connected layer with 1 neuron. This layer outputs the predicted unemployment rat e for the next quarter by taking from GRU and converting it.

1. Forecasting
2. Model Evaluation

**3.4.6 RNN (Recurrent Neural Network)**

RNN is the most basic type of neural network for sequential data. It processes input sequences one step at a time, maintaining a hidden state that captures information from previous steps, which makes them suitable for learning temporal dependencies. But due to the limitations of long-term memory like vanishing gradient problem, advanced models like GRU and LSTM were created.

Implementation Steps:

1. Data Scaling
2. Creating Lagged Data
3. Train/Test Split
4. Model Building

RNN is built using a simple architecture and has 4 layer. The first layer is the input layer which Accepts sequences of shape (n\_lags, 1). The second layer, RNN layer uses tanh activation function with configurable number of units. The third layer, dropout layer is used to prevent overfitting. The last layer, dense output layer is used to forecast.

1. Forecasting
2. Model Evaluation

**3.4.7 CNN (Convolutional Neural Network)**

CNN are a commonly used image processing model, but a 1D CNN can be used as well for time series forecasting. This can be carried out by applying filters that scan across sequences and capture local patterns which is fast to train and can capture repeating motifs or local trends. This ability makes CNN able to focus on short-term dependencies.

Implementation Steps:

1. Data Scaling

This step is same as other models, which uses MinMaxScaler to keep the values between 0 and 1.

1. Creating Lagged Data
2. Train/Test Split
3. Model Building

CNN model is built with 5 layers. The first layer, input later receives the lagged data and shape (n\_lags, 1). The second layer is the Conv1D layer which is a 1D convolutional layer that applies filters with a specified kernel size to capture short-term patterns. The third layer, the dropout layer is used to prevent overfitting. Then flatten layer Flattens the output of the convolution for the dense layers. The last layer which is the dense output layer acts as a final fully connected layer with one unit outputs the final prediction.

1. Forecast
2. Evaluation
   1. **Model Evaluation Metrices**

**3.5.1 RMSE (Root Mean Square Error)**

RMSE measures the square root of the average of squared differences between predicted and actual values. It penalizes larger errors more heavily, making it sensitive to outliers. In easier words, it tells how far off the predictions are from real values. So when the model makes few big error, the RMSE will go up significantly, which indicates there is a serious issue.

A mathematical equation with numbers and symbols

Description automatically generatedFormulae:

Code:

*from* sklearn.metrics *import* mean\_squared\_error

rmse = np.sqrt(mean\_squared\_error(y\_true, y\_pred))

In this case, lower RMSE means the model is better. Besides, it is ideal when large errors are very important to be avoided.

**3.5.2 MAE (Mean Absolute Error)**

MAE measures the average absolute difference between actual and forecasted values. This makes it different from RMSE which it treats all errors equally. The steps for to carry out the calculation is just taking the absolute value of the errors, adding them up and divide by the total number.

A mathematical equation with numbers and symbols

Description automatically generatedFormulae:

Code:

*from* sklearn.metrics *import* mean\_absolute\_error

mae = mean\_absolute\_error(y\_true, y\_pred)

MAE is same as RMSE as well, where lower MAE has a better accuracy. MAE is very useful when it comes to understanding the average size of errors. In simpler words, it tells that how wrong was the prediction on average.

**3.5.3 MAPE (Mean Absolute Percentage Error)**

MAPE shows how far off the predictions are compared to the actual values in percentage. It is easier to be understood. For example, “The model is wrong by 10% on average”. Besides, it helps in making error comparison across different models or datasets, regarding different units.

A number and a number of symbols

Description automatically generated with medium confidenceFormulae:

Code:

mape = np.mean(np.abs((y\_true - y\_pred) / y\_true)) \* 100

In simpler words, MAPE calculate how far off it is for each prediction. It turns that error into a percentage of actual value, and take the average of the percentage errors to multiply by 100. However, it can be unreliable when actual values are very small or zero.

**3.5.4 R2 (R-squared / Coefficient of Determination)**

R2 measures the proportion of variance in actual value explained by the model. It tells how much of the changes in the actual data the model was able to predict correctly or how well the predictions match the real trend.

A mathematical equation with numbers and symbols

Description automatically generatedFormula:

Code:

*from* sklearn.metrics *import* r2\_score

r2 = r2\_score(y\_true, y\_pred)]

The denominator is the total variation, which indicates how much the data varies from is mean. The numerator is the model error, indicating how much the predictions miss the mark. If the model error is small compared to the total variation, R2 is equal to 1, that means a perfect fit and the prediction match the actual values exactly. If R2 is equal to 0, the predictions are no better than just using the average. If R2 is less than 0, it means that the predictions are worse than using the average, indicating the model may be very poor or inappropriate. Therefore, Higher R² means better fit; ranges from 0 (no fit) to 1 (perfect fit).

**3.5.5 AIC (Akaike Information Criterion)**

AIC is a score used to compare how good different models are at fitting the same data while also penalizing complexity. It measures trade-off between model fit and complexity. In simple words, AIC tells which model fits the data well without being too complicated. This means how closely the forecast match the result and fewer parameters are better to prevent overfitting.

A black number and a white background

Description automatically generated Formula:

Code:

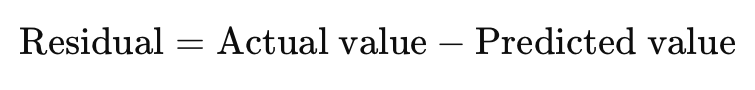
aic = model.aic()

bic = model.bic()

The first part of the formulae penalizes complexity, the more parameters mean higher AIC. The second part rewards good fit, indicating better fit will have lower AIC. The lower the AIC, the better.

**3.5.6 Residual Analysis**

Residual analysis is a technique that is implemented to identify how well the model is performing by looking at the residuals. Residuals means the difference between the actual values and the predicted values. It helps determine if the model capture all the patterns in the data, are there systematic errors in the forecast and also is the model making random errors (good) or structured errors (bad).

Formula:

Code:

residuals = y\_true - y\_pred

*# Plot residuals*

*import* matplotlib.pyplot *as* plt

plt.plot(residuals)

plt.axhline(0, *color*='black', *linestyle*='--')

plt.title('Residuals Over Time')

plt.show()

Good residuals will have few characteristics. Firstly, the residuals are randomly distributed around zero (no pattern). This indicates the model has captured all systematic information in the data. Second, the residuals has no clear patterns over time. When plotted against time, residuals should resemble white noise (random, patternless). Any visible trend, seasonality, or structure means the model has missed something in the data. Thirdly, the spread of residuals should have a constant variance, which means they are fairly consistent across all time points. If residuals grow wider or narrower over time, it tells the model’s error changes depending on time time, which is a sign of heteroscedasticity. Lastly, While not always required, especially for deep learning models, residuals from statistical models (like ARIMA or SARIMA) are ideally normally distributed. This supports the assumptions used for confidence intervals and hypothesis testing.

**3.6 Flowchart of Overall Methodology**

**A screenshot of a computer

Description automatically generated**

*Fig 6. Overall Methodology*

**A diagram of a model

Description automatically generated3.7 Overall Approach**

*Fig 7. The Machine Learning Approach to Economic Data*

The figure above shows an approach where the prediction accuracy can be optimized using a combination of Econometric models, Feature Engineering, and Machine Learning Models. The methodology can be divided into six phases [36]. This methodology, which combines the strengths of econometric models, feature engineering, and machine learning, provides a comprehensive and robust framework for forecasting Malaysia’s unemployment rate. By integrating Python's powerful libraries for data preprocessing, model implementation, and visualization, the study ensures a data-driven, adaptable approach. The final model, selected based on performance across accuracy metrics and practical relevance, was validated through comparison with historical trends and feedback from domain experts. This multi-model approach not only guarantees accuracy and flexibility but also equips policymakers with actionable insights into Malaysia’s labor market trends, offering a reliable foundation for future forecasting and decision-making.

**Results and Discussions**

**4.1 Data Split**

Total Observations: 60 quarters

Training Set: 48 quarters (80%)

Test Set: 12 quarters (20%)

Training Period: 2010-1 to 2021-10

Test Period: 2022-01 to 2024-10

The dataset used consists of 60 quarterly observations of Malaysia's unemployment rate, covering the period from 2010 to 2024. To ensure the time sequence is preserved for accurate forecasting, the data is split chronologically into two sets: 80% (48 quarters) for training and 20% (12 quarters) for testing. The training period spans from the first quarter of 2010 (2010-Q1) to the fourth quarter of 2021 (2021-Q4), while the testing period covers from the first quarter of 2022 (2022-Q1) to the fourth quarter of 2024 (2024-Q4).

**4.2 Forecasted Models**

**4.2.1 ARIMA**

**A graph showing a line

Description automatically generated**

**A screenshot of a graph

Description automatically generated**

The forecasted value for ARIMA shown a constant value. This indicates ARIMA failed to forecast due to its limitation of lacking seasonality. This means the ARIMA cannot model and detects stationary after differencing. Therefore, SARIMA is carried out.

**4.2.2 SARIMA**

**A graph with a line

Description automatically generated**

**A screenshot of a graph

Description automatically generated**

The forecasting results for SARIMA model shows a dropping trend starting from 3.1819% in 2024 Dec to 2026 Sept with 3.0349. This indicates the improvement in labor market. The line graphs shows the historical trend with the forecasted trend inside the green area. Besides, SARIMA successfully captures the trend and seasonality that ARIMA cannot.

A graph of a graph

Description automatically generated**4.2.3 Exponential Smoothing**

A screenshot of a graph

Description automatically generated

Holt-Winters are used instead of simple exponential smoothing due to the ability to capture trend and seasonality. SES model does not capture these both and will output constant value data. The model predicts slight fluctuations in the unemployment rate over the next 8 periods, ranging from 3.14% to 3.26%, with tight confidence intervals.

**A screenshot of a graph

Description automatically generated4.2.4 LSTM**

The LSTM forecast suggests a relatively stable unemployment rate between 2024 and 2026, with a minor decline through 2025 followed by a slight rebound in 2026. The predicted values range narrowly around 3.35%–3.37%, indicating economic steadiness with no significant volatility expected in the near future.

**4.2.5 GRU**

**A screenshot of a graph

Description automatically generated**

The GRU model forecasts a steady increase in Malaysia's unemployment rate over the following two years, from 3.36% to 3.41% by 2026. The stable increasing trend, however slight, points to possible slowdowns in labour market improvement. The tight confidence intervals imply strong model certainty, suggesting that unless significant interventions or adjustments occur, unemployment may slightly worsen over time.

**4.2.6 CNN**

A graph with numbers and a line

Description automatically generated

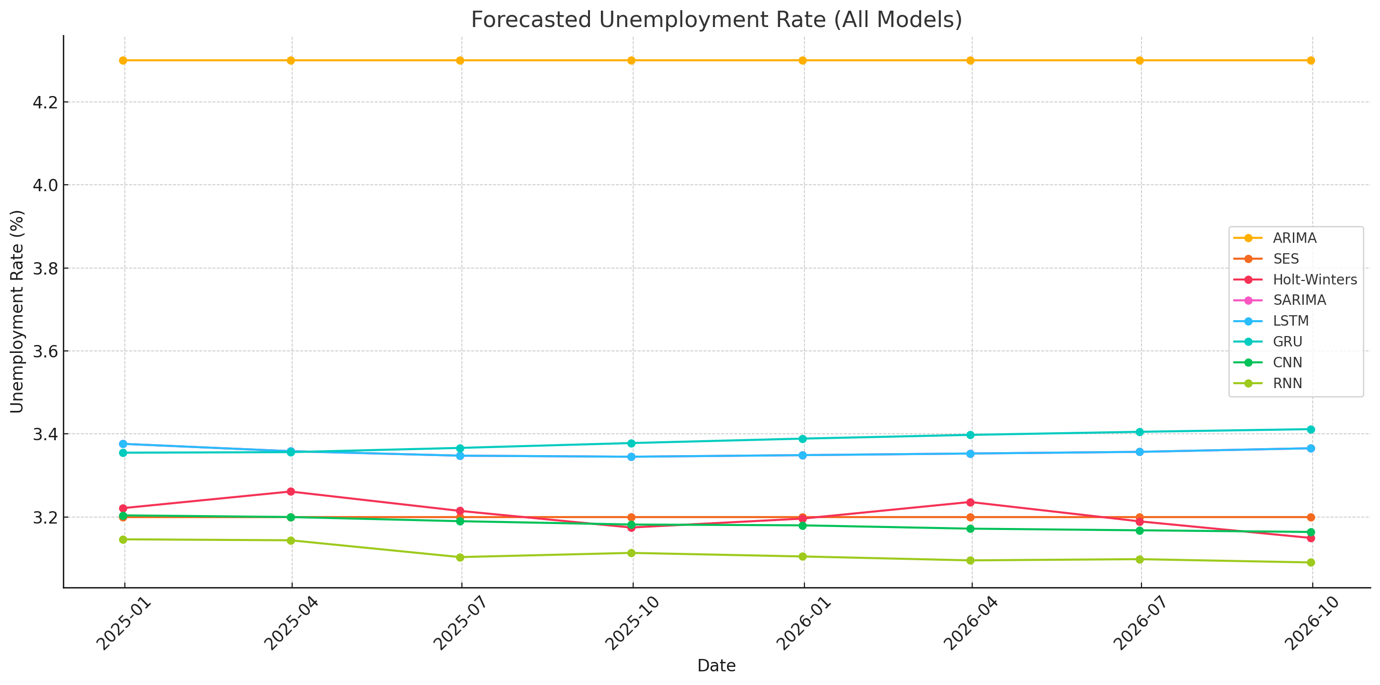
The CNN model predicts a very gradual decline in Malaysia’s unemployment rate over the next two years, decreasing from 3.20% to 3.16%. The forecasts suggest a slow but steady improvement in labor market conditions, with narrow confidence intervals reflecting high certainty and low expected volatility in the near term.

**4.2.7 RNN**

A screenshot of a graph

Description automatically generated

The RNN model predicts a slight downward trend in Malaysia’s unemployment rate from 3.15% to 3.09% between 2024 and 2026. The pattern includes minor short-term fluctuations, but the overall signal suggests gradual improvement in labor market conditions. The narrow confidence intervals indicate stable and low-risk expectations, with no significant economic disruptions forecasted.

**4.2.8 Model Forecast Summary**

|  |  |  |  |
| --- | --- | --- | --- |
| **Model** | **Trend** | **Volatility** | **Explanation** |
| ARIMA | Flat | None | Ignores trend and seasonality |
| SARIMA | Mild decline | Low | Suitable for structured time series as handles both trend and cycle well |
| Holt-Winters | Slight wave | Low | Captures both trend and seasonality |
| LSTM | Decline to rise | Low | Able to capture turning points and longer-term behaviour |
| GRU | Gradual rise | Low | Learns subtle upward movements; compact memory structure |
| CNN | Gradual decline | Very Low | Strong at detecting local time-window patterns |
| RNN | Slight decline | Low | Captures short-term memory and minor short-term noise observed |

**4.3 Model Evaluation Results**

**4.3.1 Performance Metrices Table**

**A screenshot of a graph

Description automatically generated**The research provides a comprehensive comparison of multiple Artificial Intelligence models for forecasting Malaysia’s unemployment rate. The models included time series models like ARIMA, SARIMA, Exponential Smoothing and deep learning models like LSTM, GRU, CNN, and RNN. Each model was trained and evaluated on the same historical data and test set to ensure a fair comparison to get the most accurate forecast.

Throughout the comparison, RNN is the model with the lowest RMSE (Root Mean Squared Error), which indicates the most accurate forecast in terms of absolute error. The model with lowest MAPE (Mean Absolute Percentage Error) is RNN, which indicates the best relative accuracy.

Besides, every model has their own strength and weakness. Throughout the research, ARIMA/SARIMA is strong for linear trends and seasonality, highly interpretable, but may miss non-linear patterns. Exponential Smoothing is fast and robust for short-term and seasonal data, but limited for complex or non-linear series. LSTM/GRU is excellent for capturing long-term dependencies and non-linearities, but require more data and tuning. CNN is effective for local and short-term patterns, fast to train, but may miss long-term dependencies. RNN is simple baseline for sequential data, but less effective for long-term patterns due to vanishing gradients.

For practical recommendations, ARIMA, SARIMA, and Exponential Smoothing are preferred when it comes to interpretability and transparency. When comes to highest accuracy and complex patterns, LSTM or GRU are recommended when there is sufficient data and expertise.

**4.3.2 Evaluation Metrices Results**

**A graph of blue rectangular bars with white text

Description automatically generated4.3.2.1 RMSE**

RMSE penalizes larger errors more harshly, making it a crucial metric for detecting extreme deviations in forecast accuracy. RNN continues to lead with an exceptionally low RMSE of 0.0745, showcasing its robustness in minimizing both small and large errors. GRU (0.1309) also maintains strong performance, followed by the Combined Forecast (assumed similar to GRU if not explicitly provided). While LSTM increases slightly to 0.3281, it still remains within a reasonable error range. Traditional models such as ARIMA (0.2757) and SARIMA (0.3065) remain behind the deep learning approaches, indicating limited precision. CNN, now with a higher RMSE of 0.3950, struggles more significantly with large deviations. The worst performer remains Exponential Smoothing, which shows an extremely poor RMSE of 0.9465, confirming its inadequacy for this dataset.

**4.3.2.2 MAE**

**A graph of blue rectangular bars with white text

Description automatically generated**

MAE reflects the average magnitude of errors without emphasizing larger ones disproportionately. RNN again demonstrates its strength with the lowest MAE of 0.0983, indicating minimal average error. GRU also performs excellently with 0.1394, while ARIMA (0.3748) and SARIMA (0.3927) show much higher and less desirable error rates. LSTM’s MAE has increased to 0.3426, reducing its reliability compared to earlier results. The CNN model, with a MAE of 0.4174, performs worse than all deep learning counterparts. Meanwhile, Exponential Smoothing, with a MAE close to 1.0, confirms its inability to model the trend accurately.

**4.3.2.3 MAPE**

**A graph of blue rectangular bars with white text

Description automatically generated**

MAPE provides a relative percentage error, enhancing interpretability across scales. RNN remains the top performer with a MAPE of just 2.07%, suggesting extremely accurate and trustworthy forecasts. GRU follows at 3.81%, still within a highly accurate range. LSTM, however, shows a drop in performance, climbing to 9.37%, indicating reduced relative accuracy. CNN also suffers from high relative error at 11.29%, making it less dependable. ARIMA (7.39%) and SARIMA (8.30%) perform moderately but lag behind RNN and GRU. Exponential Smoothing, with a very high MAPE of 27.86%, further confirms its unsuitability for this forecasting task.

**4.3.3 Residual Analysis**

**4.3.3.1 Residual Plotting and Table**

**A screenshot of a graph

Description automatically generated**

**4.3.3.2 Overview**

Residual analysis helps evaluate how well a forecasting model captures patterns in data by examining the behaviour of its prediction errors. Ideally, good models should have residuals with a mean near zero, low standard deviation, symmetrical distribution (low skewness), and normal-shaped spread (kurtosis close to 3). Additionally, the Jarque-Bera test helps assess whether the residuals follow a normal distribution where lower values indicate better normality.

**4.3.3.3 Mean of Residuals**

The mean of residuals measures the average forecast error, with values closer to zero indicating less bias in the model’s predictions. In this case, RNN has the best result, with a mean residual of -0.0551, followed closely by GRU (-0.0946). These values suggest minimal bias. ARIMA (0.2646) and SARIMA (0.3065) are slightly more biased but still within acceptable ranges. Exponential Smoothing shows a large negative mean (-0.9465), indicating a strong tendency to underestimate values. LSTM (-0.3281) and CNN (-0.395) also show significant negative bias, consistently forecasting values below the actuals. Models like RNN and GRU are preferred here due to their minimal bias.

**4.3.3.4 Standard Deviation**

For standard deviation, which captures the spread or variability of errors, RNN again performed the best (0.0814), indicating highly consistent forecasts. LSTM (0.0984) and GRU (0.1023) also had low variability, while CNN's errors were slightly more dispersed (0.1349). In contrast, SARIMA (0.2455) and ARIMA (0.2654) had significantly higher error spreads, and Exponential Smoothing had the largest of all (0.317), reflecting unstable and inconsistent predictions.

**4.3.3.4 Maximum and Minimum Residuals**

The minimum and maximum residuals give insight into the worst-case errors each model makes. GRU had the narrowest and most balanced spread (-0.199 to 0.181), followed by RNN (-0.237 to 0.074) and LSTM (-0.474 to -0.188). These small ranges suggest better control over large forecast errors. Meanwhile, CNN (-0.603 to -0.193) and Exponential Smoothing (-1.321 to -0.255) had the widest and most negatively skewed ranges, pointing to frequent large underpredictions. The ARIMA and SARIMA models fall somewhere in the middle. Models with smaller ranges like GRU and RNN are more reliable in avoiding severe over- or under-predictions.

**4.3.3.5 Skewness**

Skewness measures whether a model systematically over- or under-predicts. Values near zero indicate a symmetrical distribution of errors indicates the ideal case. CNN (-0.2275) and LSTM (-0.1684), suggesting relatively balanced forecasting behaviour. GRU, however, showed a high positive skew (1.8359), indicating it consistently underpredicts. ARIMA (0.8424) and SARIMA (0.9778) also showed right-skewed distributions, reinforcing their underprediction tendencies, while RNN (-0.5927) leaned toward mild overprediction. Overall, CNN and LSTM had the most symmetrical errors.

**4.3.3.4 Kurtosis**

Kurtosis measures whether residuals have outliers or "heavy tails." A kurtosis near 3 is ideal for normality. Here, GRU stood out with a value of 3.598, very close to a normal distribution, indicating an ideal balance of peak and tail behavior. Other models like ARIMA (0.1835) and RNN (0.4939) showed relatively flatter distributions, suggesting fewer outliers but also less variance in prediction errors. LSTM (-1.3275) and CNN (-1.2097) had very low kurtosis, pointing to excessively flat distributions and possibly oversmoothed forecasts. Exponential Smoothing (0.8725) was closer to normal but still lacked consistency in other metrics.

**4.4 Visualizations**

**A screenshot of a graph

Description automatically generated**

A graph of a model

Description automatically generated with medium confidence

A screenshot of a graph

Description automatically generated

**4.5 Discussions**

**4.5.1 Validation of Research Objectives**

The results of this study successfully fulfil the primary research objective, which is to develop, implement, and compare various forecasting models for predicting Malaysia’s unemployment rate using historical quarterly data. Through the application of both classical time series models (ARIMA, SARIMA, Exponential Smoothing) and advanced deep learning models (LSTM, GRU, RNN, and CNN), the study was able to evaluate forecasting performance from multiple methodological perspectives. This multi-model approach allowed for a well-rounded assessment that not only considered statistical accuracy but also the models’ ability to capture complex temporal patterns.

Among the models tested, the deep learning models—particularly Recurrent Neural Network (RNN) and Gated Recurrent Unit (GRU)—demonstrated superior forecasting performance, as evidenced by the lowest RMSE, MAE, and MAPE values. These findings indicate that models capable of learning long-term dependencies and non-linear relationships within the data are better suited for capturing the underlying patterns in Malaysia’s labor market dynamics. In particular, the RNN model achieved an exceptionally low RMSE of 0.0745 and a MAPE of only 2.07%, significantly outperforming traditional models like ARIMA and SARIMA.

This confirms the hypothesis that data-driven, memory-based architectures can offer more accurate and stable unemployment forecasting compared to classical linear methods. Thus, the research objective of identifying the most robust and accurate model for Malaysia’s unemployment rate forecasting has been effectively met. Moreover, the model comparisons provide valuable insights for future policy formulation and implementation, highlighting the importance of selecting appropriate modeling techniques based on data complexity and forecasting needs.