

Predicting Beer Production for the next month

Jussi Juvonen, Niko Laurén, Isuru Mulle Gamage

1. Introduction

The objective of this project work is to predict Australian beer production one month ahead using historical time series data between 1956 to 1995. The capabilities of a linear autoregressive model, a Long Short-Term Memory (LSTM) model, and a transformer model are compared in capturing the data's seasonality and long-term trends. Performance of each model is assessed with Mean Absolute Error (MAE) and Root Mean Square Error (RMSE).

First, the data is visualized and preprocessed, partitioning it chronologically into training, testing, and validation subsets. The seasonal patterns are then analyzed, and the data is formatted using sliding windows. The models are optimized to find the best settings and then compared to find the most accurate approach.

2. Initial data visualization

Figure 1 displays the raw monthly beer production series. The data exhibits strong, regular annual seasonality together with an upward long-term trend through the 1970s and a subsequent levelling-off. Variability increases in the later decades. Seasonal peaks become larger and the scatter around the trend widens. There are no obvious gaps in the record, therefore continuity is preserved.

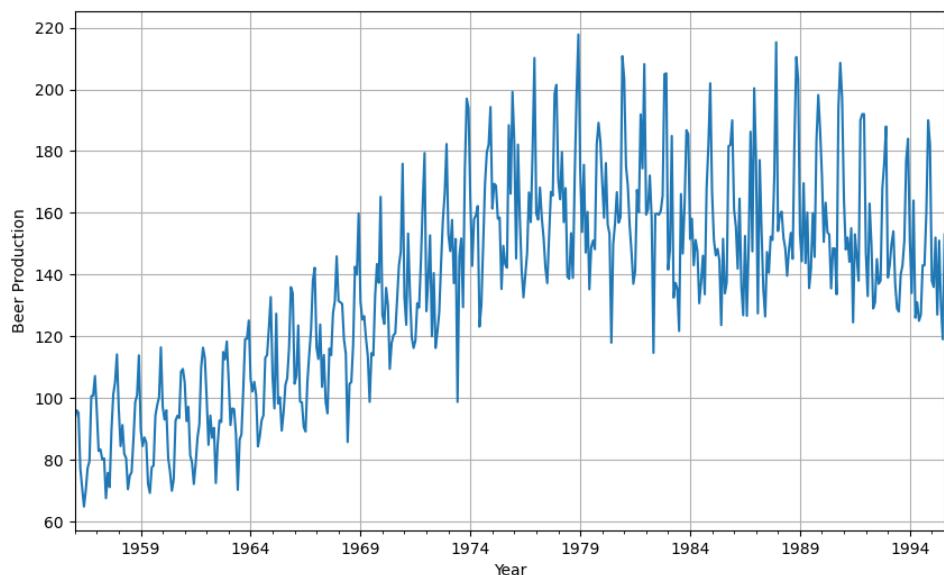


Figure 1: Raw data visualized

2.1. Time series exploration with visualization and decomposition of trend, seasonality, and residuals

The time series data represents monthly beer production in Australia, with no missing months or duplicate timestamps. The dataset spans from January 1956 to August 1995. Figure 2 shows a Seasonal-Trend decomposition using LOESS (STL) analysis of the series.

The observed data shows annual seasonality and a gradual long-term increase in production until the late 1970s, followed by a slight decline. The trend component captures the long-term pattern, and both robust and non-robust decompositions produce nearly identical trends, indicating stable estimates with minimal outlier influence. The seasonal component shows a consistent, repeating yearly pattern with roughly constant amplitude. The residuals fluctuate around zero, which means that most systemic variation is explained by the trend and seasonal terms.

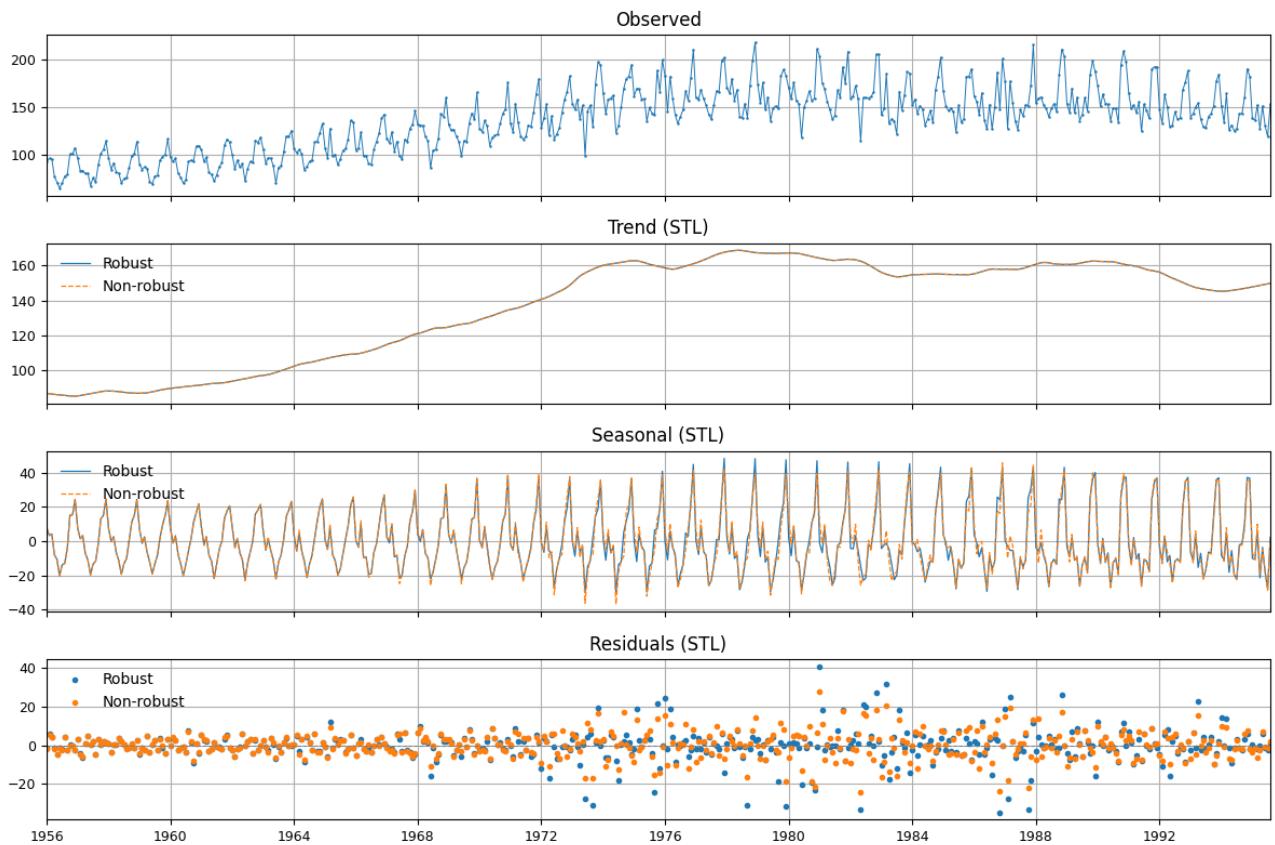


Figure 2: STL decomposition of monthly beer production time series showing the observed data, long-term trend, seasonal component, and residuals.

Figure 3 visualizes the seasonal component from the STL decomposition grouped by calendar month. The boxplots show a cyclical pattern in beer production, with the lowest seasonal

values occurring during winter months (June - July) and the highest during summer (November - December). The relatively small spread of most boxes suggests that this seasonal pattern is stable across years, with only minor year-to-year variation.

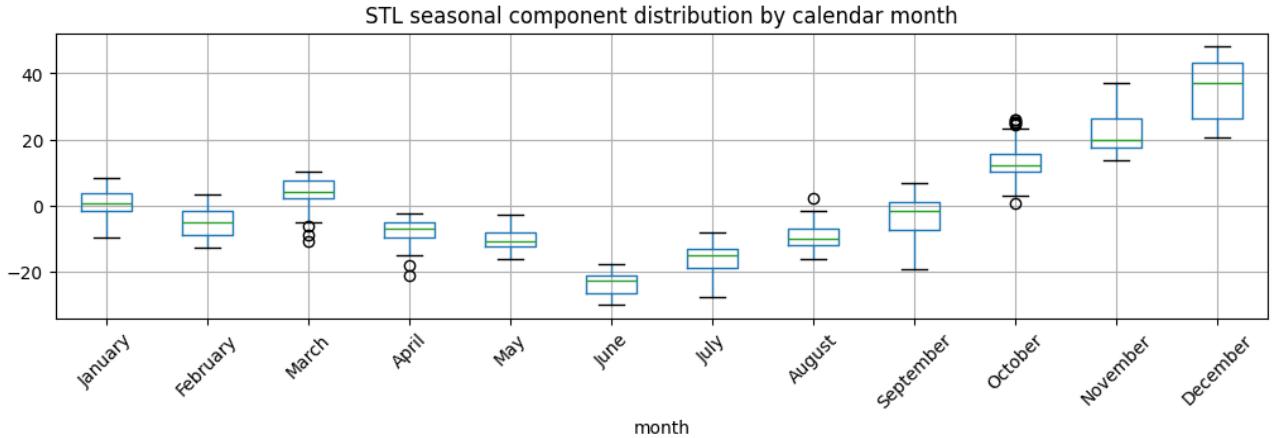


Figure 3: Distribution of the STL seasonal component by calendar month.

2.2. Autocorrelation analysis

Figure 4 shows the autocorrelation structure of the time series. The Autocorrelation (ACF) plot displays a strong positive correlation that gradually decays over many lags, with a noticeable seasonal pattern repeating approximately every 12 months, confirming yearly seasonality in the data. The Partial Autocorrelation (PACF) plot shows significant spikes at lag 1 and around lag 12. This suggests short-term dependence and annual cyclical effects. Together the plots confirm that beer production is highly autocorrelated and strongly seasonal over time.

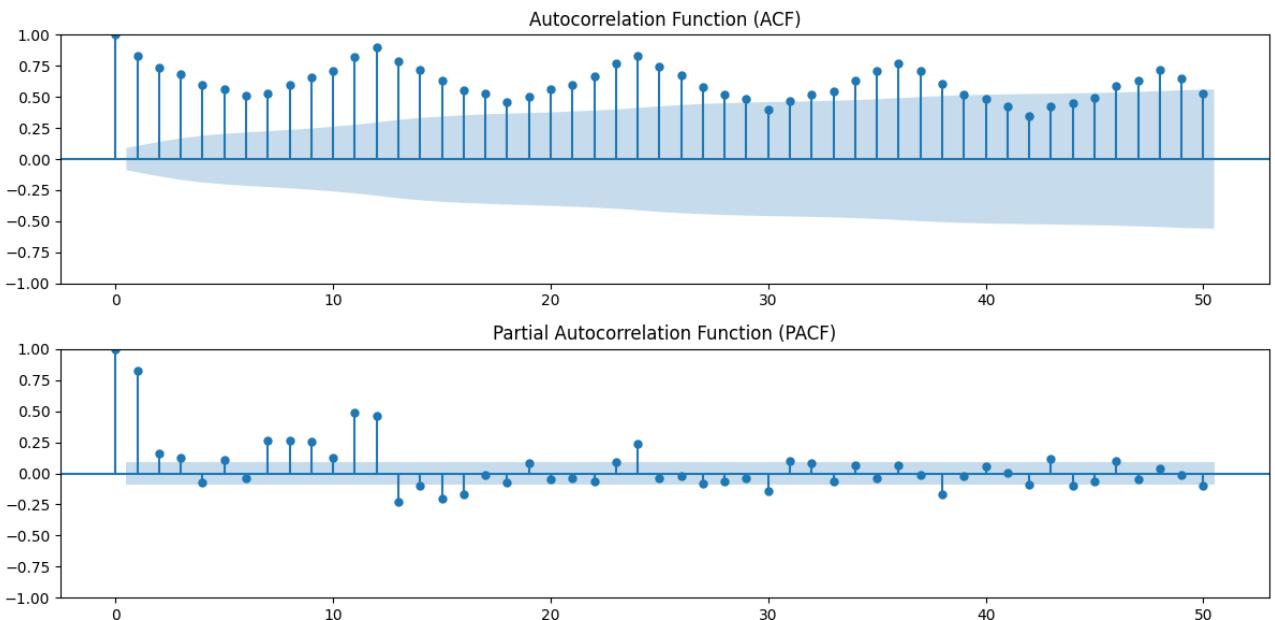


Figure 4: ACF and PACF plots of monthly beer production

2.3. Partitioning time series data

The dataset is partitioned chronologically into three subsets. The training subset is ~70% of the data covering period from January 1956 to September 1983 consisting of 333 samples. This subset is used to train the model and identify seasonal patterns in beer production. The validation subset is the following 71 samples or ~15% of the data from October 1983 to August 1989. The validation subset is used to adjust and fine-tune the parameters of the model. The remaining 72 samples (~15%) from September 1989 to August 1995 is for the test subset. The test subset is used to evaluate the model's predictive performance on unseen future data.

2.4. Data preprocessing and quality assessment

The dataset consists of uninterrupted monthly observations from January 1956 to August 1995, showing no missing or duplicate timestamps. The measurements are uniformly sampled at a constant monthly frequency. For irregular series, a uniform rate can be restored by resampling or interpolation. For missing data, short gaps can be interpolated, while longer gaps can be addressed through model-based smoothing or segment-wise estimation that preserves the temporal structure, such as Kalman filtering or spline-based local regression.

Only one variable, monthly beer production, is recorded, making temporal alignment across variables unnecessary. In multivariate settings, synchrony can be achieved by aligning all variables to a shared time index and interpolating values where data is missing or offset.

2.5. Outlier identification using STL decomposition

STL decomposition first separates the time series into trend, seasonal, and residual components. After removing the systematic trend and seasonality, the remaining residuals capture irregular variations in the data. Outliers can then be detected by identifying residuals that deviate strongly from zero using statistical measures such as standard deviation or median absolute deviation. These points are usually flagged rather than removed to preserve the temporal structure.

3. Theory

3.1. LSTM

Long Short-Term Memory (LSTM) is a Recurrent Neural Network (RNN) that was developed to solve standard RNN disability to learn long term dependencies. In standard RNN models the error signals decay exponentially in backpropagation. This makes it difficult to connect relevant events between long timespan. The LSTM architecture has specialized memory cells that contain a mechanism called Constant Error Carousel (CEC). In CEC the constant error flows inside the cell preventing the information from degrading. To regulate the information flow, the LSTM has multiplicative input and output gates that protect the information from

irrelevant noise and when the internal state can be used to influence the model output. (Hochreiter & Schmidhuber, 1997)

3.2. Preparing Long Time Series for LSTM Forecasting: Sub Sequences, Seasonality, and Trend Handling

Dividing a long time series into shorter sub sequences is a standard preprocessing step for LSTM models. Instead of using the full sequence, the data is transformed into many fixed length sliding windows, each containing past observations and a future prediction target. This approach increases the number of training samples and helps the model learn local temporal patterns more effectively (Hewamalage et al., 2021). [1]

Seasonality strongly influences how these windows should be constructed. If the data contains annual or monthly seasonal cycles, the window length must be long enough to cover at least one full seasonal period; otherwise, the LSTM cannot learn recurring patterns (Lemke & Gabrys, 2019) [2]. Seasonal decomposition methods, such as STL, may also be used before windowing to simplify the series (Hyndman & Athanasopoulos, 2021). [3]

LSTMs can model trend and seasonality directly from raw sequences or through explicit preprocessing, such as deseasonalization or adding seasonal indicators as extra features (Bandara et al., 2020) [4].

3.3. LSTM standardization methods

LSTM has two typical standardization methods called Min-Max normalization and Z-score standardization. Min-Max transforms data into a fixed range and it works well with sigmoid and tanh functions. Z-score centers the data around zero mean with unit variance and it's suitable for dealing with outliers and varying feature scales. Min-Max produces lower Round Mean Square Error (RMSE) in LSTM-based multivariate time series forecasting when compared to Z-score. Min-Max enhances training stability and prediction accuracy in LSTM models (Pranolo et al., 2024).

3.4. Transformer

The Transformer is a sequence to sequence architecture built around a self attention mechanism, which replaces the need for recurrence in traditional models (Vaswani et al., 2017). Instead of processing the sequence step by step as RNNs do, the Transformer looks at all time steps at once and learns how strongly each past value should influence the next prediction through attention weights (Luong, Pham & Manning, 2015). This parallel attention process helps the model capture both short term patterns and long range relationships without suffering from the gradient decay issues that affect recurrent networks (Bengio, Simard & Frasconi, 1994). Within each encoder layer, multi head attention and feed forward networks work together to build rich, context aware representations of the data. Because of this design,

Transformers can learn complex seasonal behavior and long term trends very effectively, making them suited for long time-series forecasting tasks.

3.5. Preparing Time Series for Transformer Forecasting

Transformers require time series to be organized into fixed length sliding windows that provide the local temporal context needed for the self attention mechanism to learn relationships between observations (Zerveas et al., 2021).

When seasonality is present, the window must be long enough to include at least one full seasonal cycle; otherwise, the model cannot capture repeating patterns effectively (Lim et al., 2021). Because Transformers do not inherently recognize order, positional encodings must be added to each window to supply the model with information about the sequence's temporal structure (Vaswani et al., 2017).

Preprocessing steps such as deseasonalization, detrending, or standardization can further stabilize attention weights and improve learning on long time series (Zerveas et al., 2021).

3.6. Transformer standardization methods

Transformers depend on input scaling, typically Z-score or Min–Max to keep attention scores numerically stable, as large unscaled values can disrupt self-attention (Zerveas et al., 2021). Internally, Layer Normalization ensures consistent hidden-state magnitudes throughout training, while the bounded scale of positional encodings further supports stable attention computations (Ba, Kiros & Hinton, 2016; Vaswani et al., 2017).

4. Implementation and Results

4.1. Baseline model

The autoregressive baseline model was tuned using a random search over 30 trials. Each trial sampled a set of hyperparameters from the ranges listed in Table 1. The model consists of a single linear layer that maps the most recent T observations to a one-step-ahead forecast. A chronological split was used to create training, validation, and test segments, with all scaling fitted on the training data only. The model was optimized using mean squared error loss, and early stopping monitored validation loss to select the best state for each trial. The final configuration with the lowest validation loss across all trials was retained for forecasting.

Hyperparameter	Search range / distribution
Input window T	6, 12, 24, 36
Learning rate	Log-uniform on $[10^{-4}, 5 \times 10^{-3}]$
Batch size	16, 32, 64, 128

Table 1: Hyperparameter search space for the baseline model.

Figure 5 shows the training and validation losses for the best model. The training curve steadily decreases and remains smoother throughout, while the validation curve starts at a higher level and stays consistently above the training loss. The gap between the two narrows towards later epochs, indicating that the model gradually adapts to the validation distribution.

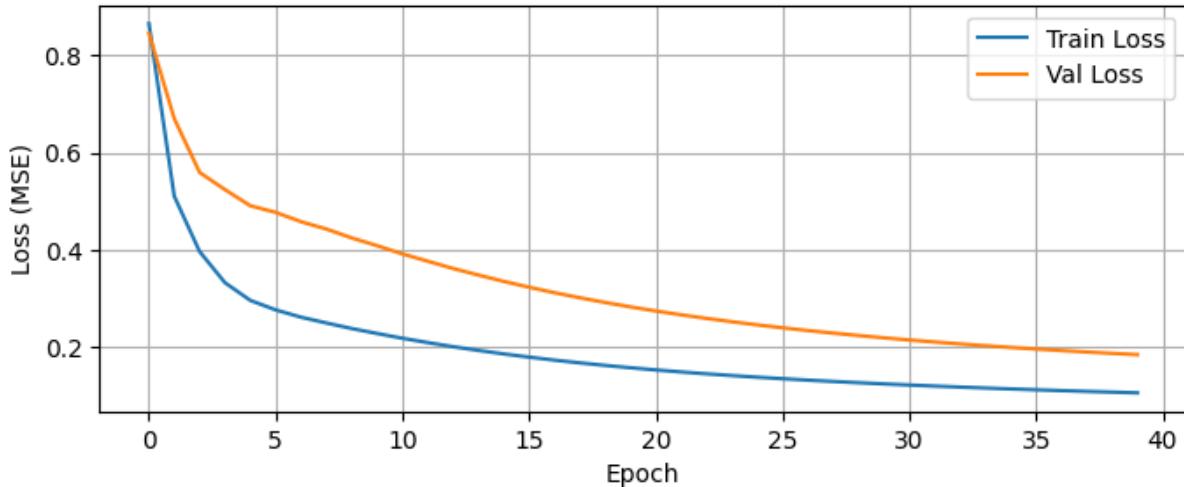


Figure 5: Training and validation MSE across epochs for the autoregressive baseline model.

The resulting multi-step forecast on the test set is shown in Figure 6. The model reproduces the general seasonal rhythm of the series, but the predictions often appear slightly out of phase with the target. Several peaks are forecast too late and local maxima are occasionally overestimated or underestimated. Overall, the baseline follows the broad pattern but doesn't have enough detail to stay accurately aligned over longer periods.

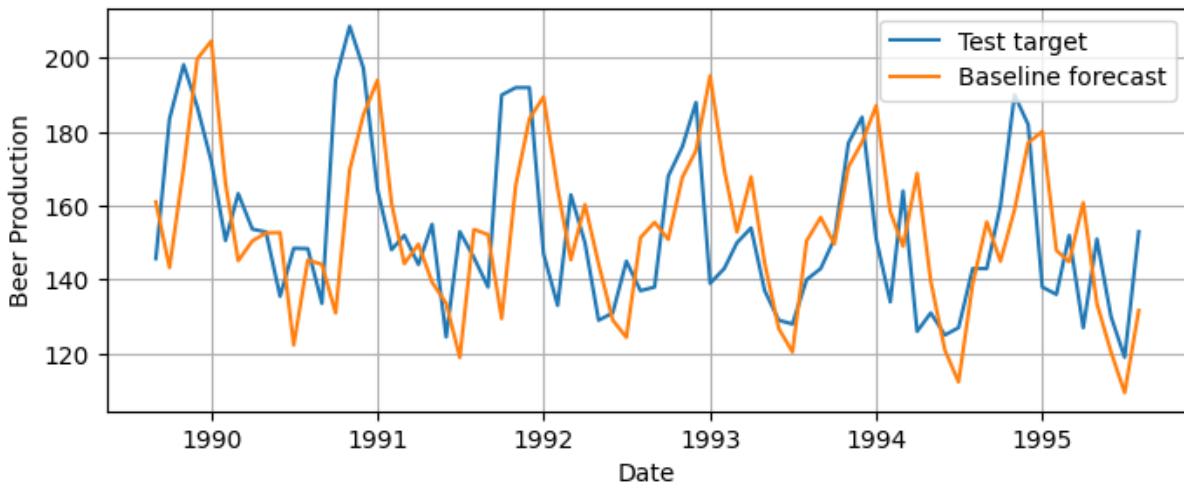


Figure 6: Iterative multi-step forecast compared with the validation target for the baseline autoregressive model.

The best baseline model uses a 36-step input window, a moderate learning rate, and a batch size of 16. It achieved a validation loss of 0.1839 and, on the held-out test set, produced

$MAE = 18.09$ and $RMSE = 22.90$. The top-performing configurations show variation in both window length and learning rate.

4.2. LSTM model

The LSTM model was tuned using a random search over 200 trials. Each trial sampled hyperparameters from the ranges shown in Table 2. The model was trained on a chronological train and validation split, with the test set reserved for final evaluation. Z-score scaling was fitted on the training data and applied to the validation and test sets to avoid leakage. The models were optimized using Adam algorithm with mean squared error loss. Early stopping was based on the validation loss. For every trial, the model state with the lowest validation loss was recorded and the best overall configuration was selected for forecasting.

Hyperparameter	Search range / distribution
LSTM units	16, 32, 64, 128, 256
Number of layers	1-5
Dropout	Uniform on [0.0, 0.3]
Learning rate	Log-uniform on $[10^{-4}, 5 \times 10^{-3}]$
Input window T	6, 12, 24, 36
Batch size	16, 32, 64, 128

Table 2: Hyperparameter search space for the LSTM model.

Training curves for the best model are shown in Figure 7. The training and validation losses decrease rapidly during the first epochs and stabilize around epoch 10 to 15. The two curves remain close throughout training, which indicates that the model fits the data without clear overfitting. Small fluctuations in the validation loss appear in the later epochs but remain limited.

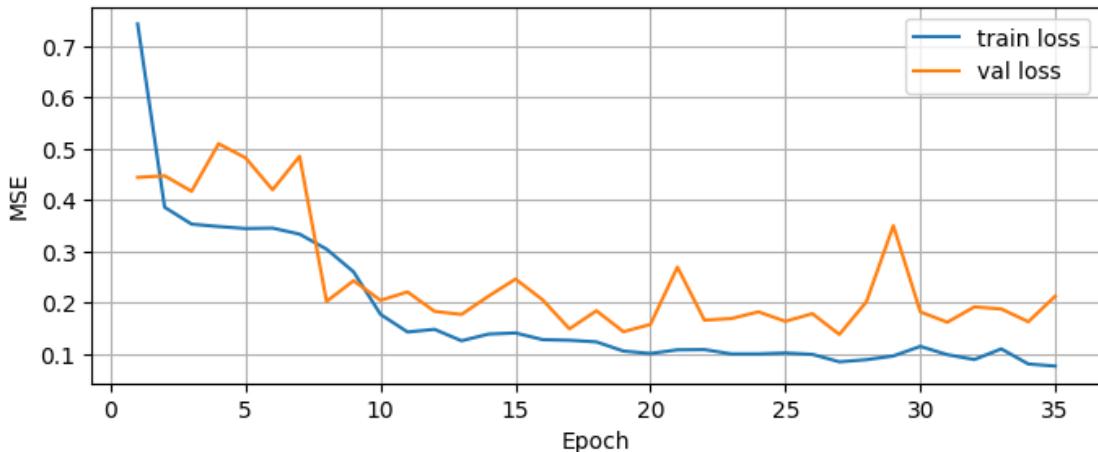


Figure 7: Training and validation MSE across epochs for the LSTM model.

The resulting test forecast is presented in Figure 8. The model reproduces the main seasonal pattern and the approximate shape of the target series. A slight timing offset remains, because some predicted peaks occur marginally later than the observed values. The model tends to overpredict several local maxima and underpredict some local minima, which produces a vertical shift relative to the target. Overall, the tuned LSTM captures the long-term structure of the series but has a moderate amplitude and phase errors.

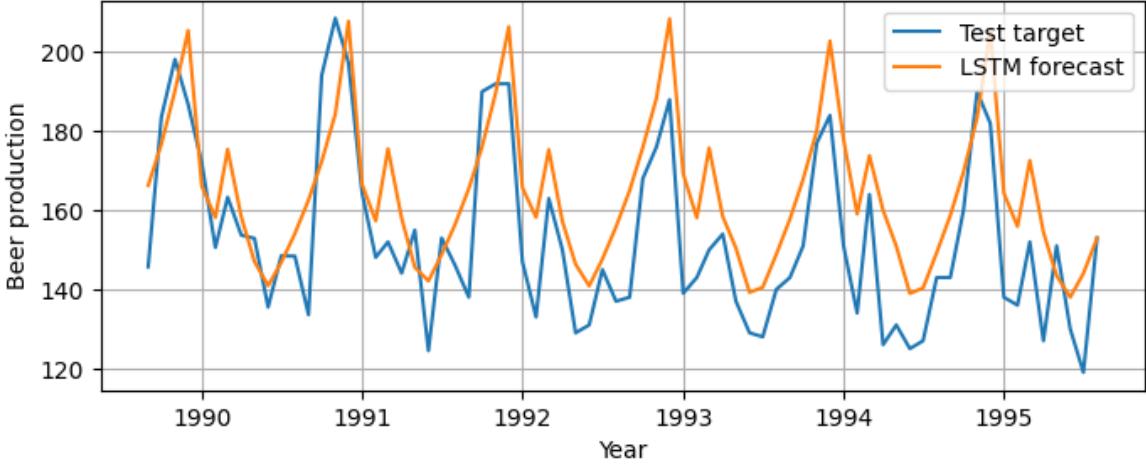


Figure 8: Iterative multi-step LSTM forecast compared with the holdout target.

Figure 9 shows boxplots of validation MAE grouped by each LSTM hyperparameter across the 200 trials. Overall performance is relatively stable, although some patterns are visible. Hidden size shows an improvement up to about 64 units, after which gains are unclear. All batch sizes show similar medians and spread. The number of layers appears to have little effect on the median error, although the 4-layer models show a noticeably narrower interquartile range. Window size of 12 seems to have the lowest average MAE, dropout seems to have minimal impact on the model, while higher learning rates seem to lower the MAE.

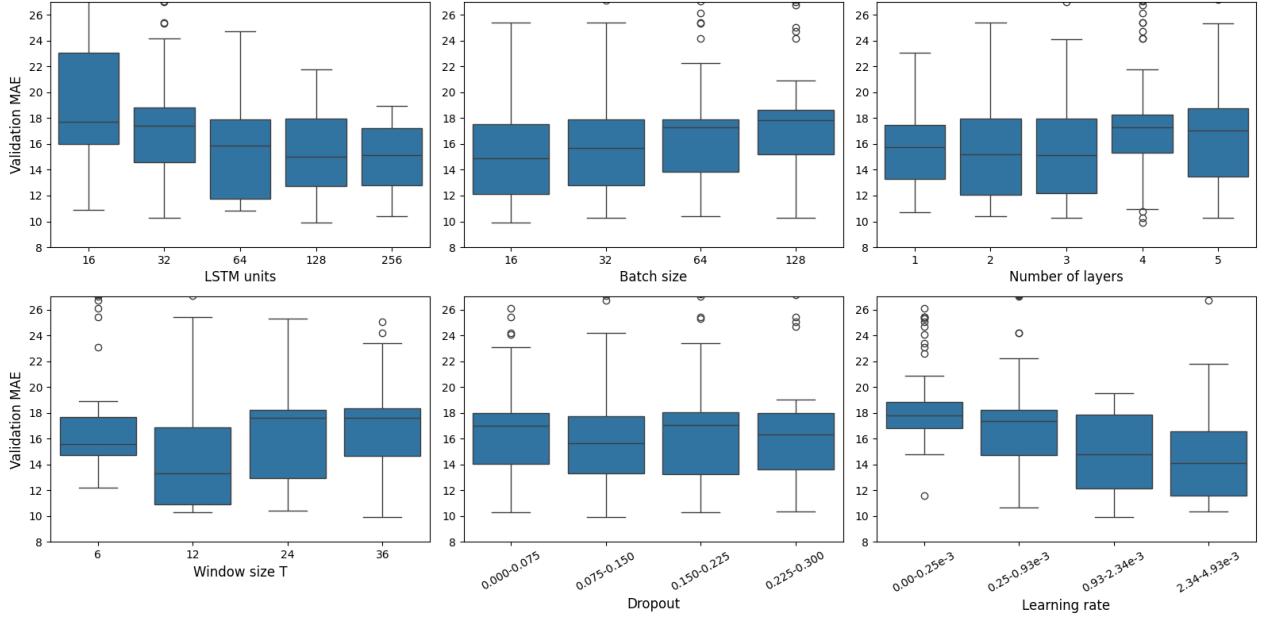


Figure 9: Validation MAE distributions across trials grouped by hyperparameter values.

The best LSTM model uses 128 units, 3 layers, modest dropout and a 36-step input window. It achieved validation loss of 0.1381 and on the held-out test set produced $MAE = 14.32$ and $RMSE = 16.53$. The top three models show noticeable variation in hyperparameters while delivering similar performance. This probably happens because some settings balance each other out and random search adds chance, rather than there being one perfect combination.

4.3. Transformer model

The transformer model is implemented as a one-ahead forecasting model with a chronological train and holdout split. Z-score scalers are fitted on the training fold and then applied to validation and holdout to avoid data leakage. The model projects a single input feature into a dimensional embedding, adds sinusoidal fixed positional encodings, and passes the sequence through an encoder before using the final time-step encoder output to predict the next value with a linear head. Training minimizes MSE using the Adam optimizer. Gradient norm clipping is used to stabilize updates. Early stopping is used to stop training when validation loss did not improve for 8 epochs.

A random hyperparameter search was performed over 200 trials, Table 3 shows all 8 hyperparameters tuned and their search ranges or distributions. For each trial we trained until early stopping and kept the model state with the best validation loss. The same chronological split and scaler practice is used as in the LSTM model.

Hyperparameter	Search range / distribution
Model dimension d_{model}	16, 32, 48, 64, 128, 256
Attention heads	1, 2, 4, 8, 16
Encoder layers	1-5
Feedforward dimensions	$d_{model} \times \{2, 4\}$
Dropout	Uniform on [0.0, 0.3]
Learning rate	Log-uniform on $[10^{-4}, 5 \times 10^{-3}]$
Input window	6, 12, 24, 36
Batch size	16, 32, 64, 128

Table 3: Tunable hyperparameters for the transformer model.

Figure 10 shows the training and validation loss curves for the chosen best trial. Training and validation losses seem to both decrease early and quickly flatten. The model is performing worse on the validation dataset with small peaks in loss values.

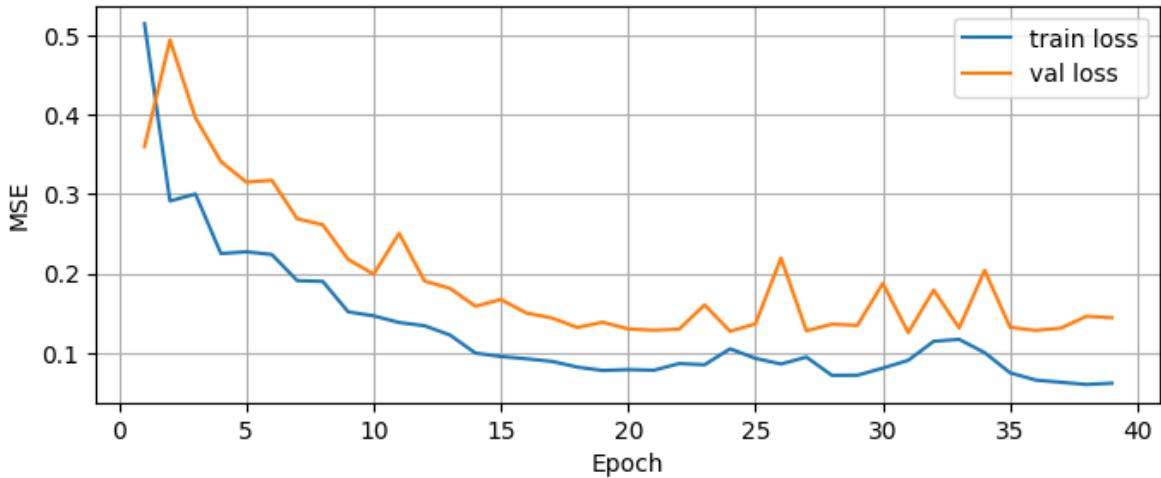


Figure 10: Training and validation MSE across epochs for the transformer model.

Figure 11 shows the best model's 1-ahead forecasts plotted against the holdout series. The transformer captures the dominant seasonal pattern and timing. The forecast has a smaller amplitude for some seasonal peaks when compared to the target. The forecasts plotted are one-ahead predictions produced by applying the trained model to each test input window and inverse-transforming the outputs back to the original scale for plotting.

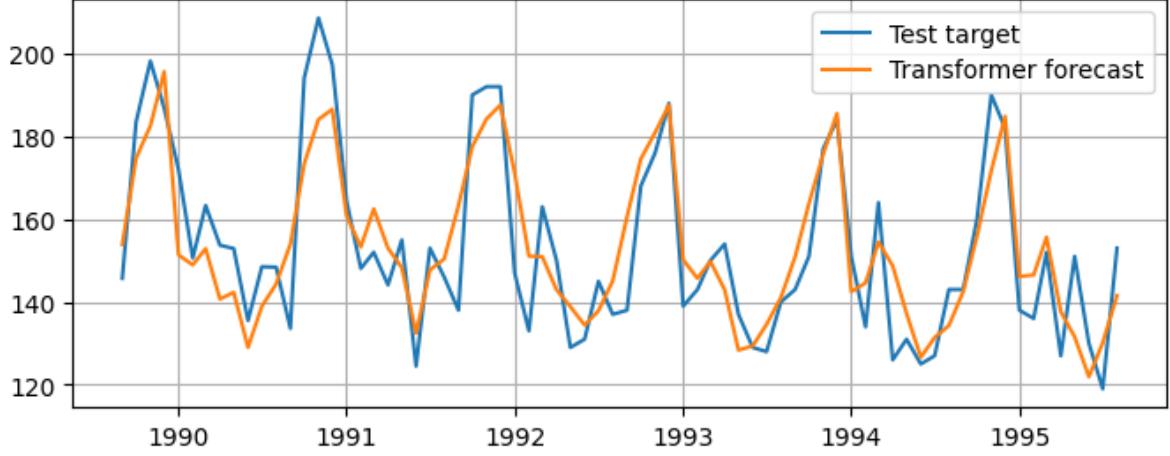


Figure 11: Transformer 1-ahead forecast.

Figure 12 presents boxplots of holdout MAE grouped by hyperparameter value. These summarize how choice of each hyperparameter affected validation MAE across the 200 random trials.

The hyperparameter analysis indicates broadly stable performance across several settings with some parameters showing modest advantages. Model dimensions between 32 and 128 perform similarly well. Batch size has little effect except for 128, which gives higher errors. Encoder depth shows minimal differences, through 3-4 layers perform best. Attention heads behave similarly overall. Feedforward dimensions show uneven variability due to interactions with model dimensions, with 256 and 1024 showing tighter, more stable error distributions than 512. The most effective dropout values fall in the lower and mid-upper ranges of the tested range. Learning rates around $0.6 - 1.8 \times 10^{-3}$ yields the lowest errors. A window size of 12 provides the lowest error rate with 24 performing comparably.

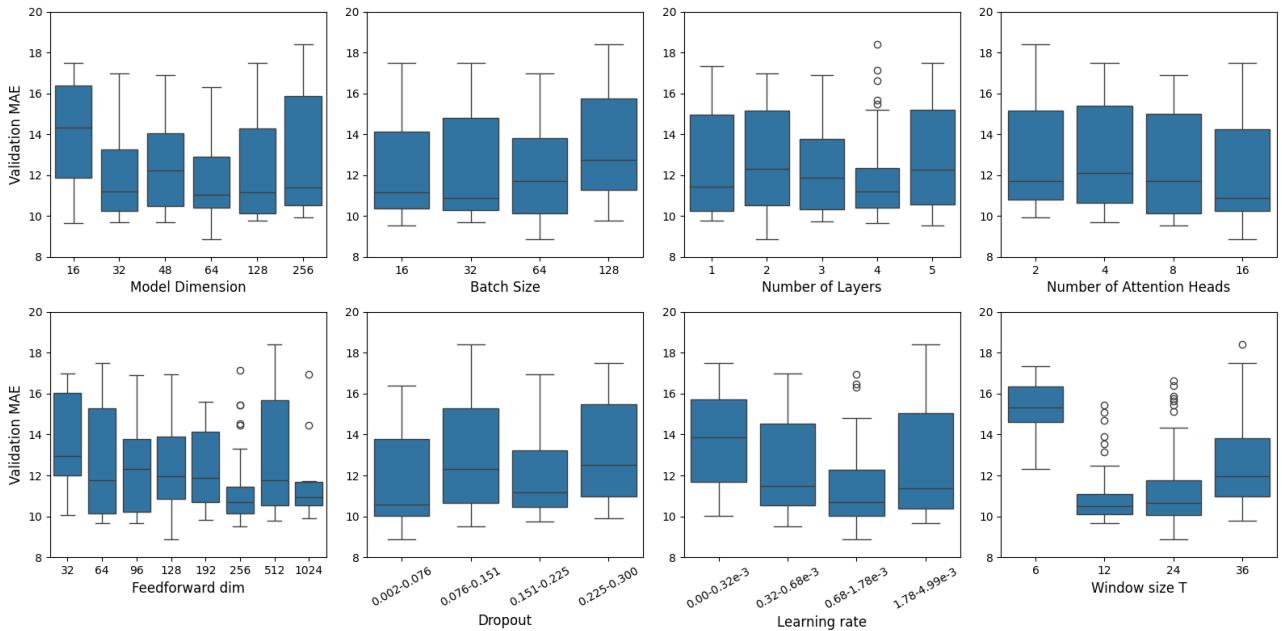


Figure 12: Validation MAE distributions across trials grouped by hyperparameter values.

The best model found in the 200-trial search used 64 model dimensions, 16 attention heads, 2 encoder layer, 128 feedforward dimensions, dropout of roughly 0.003, learning rate 1.58×10^{-3} , window size of 24 and a batch size of 64, Achieving these test results $MAE = 9.27$ and $RMSE = 11.21$.

Across the top three models, the hyperparameters vary noticeably, suggesting that multiple configurations can achieve similar performance. This variance may arise because several of the hyperparameters interact, allowing different combinations to compensate for one another, and because the optimization contains several nearby minima rather than a single clear optimum.

5. Conclusions

The transformer model performed best on the held-out test set with test $RMSE = 11.21$ and test $MAE = 9.27$. In comparison, the LSTM obtained a test $RMSE = 16.53$ and a test $MAE = 14.32$, while the autoregressive baseline showed substantially higher errors: test $RMSE = 22.90$ and test $MAE = 18.09$. Hyperparameter variability is large: several distinct parameter combinations produced similar top performance for both the transformer and the LSTM, which suggests interactions between settings and multiple nearby optima rather than a single optimum.

For future refinement, a reasonable next step is to retrain the best configuration of each model using multiple random seeds and to report mean and standard deviation of the evaluation metrics. A denser local search around the most promising window lengths and learning rates may also help improve the performance. Additionally, ensembling or model averaging of the top-scoring configurations could reduce variance and stabilize forecasts. Given the present results, the transformer appears to be the most promising candidate for further tuning and development.

References

- Bandara, K., Bergmeir, C. & Smyl, S. (2020), “Forecasting across time series databases using recurrent neural networks on groups of similar series: A clustering approach”, *Expert Systems with Applications*, 140, pp. 112896.
- Bianchi, F.M., Livi, L., Rizzi, A. & Sadeghian, A. (2022), *Recurrent Neural Networks for Short-Term Load Forecasting*, Springer.
- Fawaz, H.I., Forestier, G., Weber, J., Idoumghar, L. & Muller, P.A. (2019), “Deep learning for time series classification: A review”, *Data Mining and Knowledge Discovery*, 33(4), pp. 917–963.
- Hewamalage, H., Bergmeir, C. & Bandara, K. (2021), “Recurrent neural networks for time series forecasting: Current status and future directions”, *International Journal of Forecasting*, 37(1), pp. 388–427.
- Hyndman, R. & Athanasopoulos, G. (2021), *Forecasting: Principles and Practice*, 3rd edn., OTexts.
- Lemke, C. & Gabrys, B. (2019), “Meta-learning for time series forecasting and forecast combination”, *Neurocomputing*, 353, pp. 1–16.
- Pranolo, A., Setyaputri, F.U., Paramarta, A.K.I., Triono, A.P.P., Fadhillah, A.F., Akbari, A.K.G., Putra Utama, A.B., Prasetya Wibawa, A. & Uriu, W. (2024) ‘Enhanced Multivariate Time Series Analysis Using LSTM: A Comparative Study of Min-Max and Z-Score Normalization Techniques’, *ILKOM Jurnal Ilmiah*, 16(2), pp. 210-220. DOI: 10.33096/ilkom.v16i2.2333.210-220.
- Hochreiter, S. & Schmidhuber, J. (1997) ‘Long Short-Term Memory’, *Neural Computation*, 9(8), pp. 1735–1780.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł. & Polosukhin, I. (2017) ‘Attention is All You Need’, *Advances in Neural Information Processing Systems*, 30.
- Lim, B., Arik, S.Ö., Loeff, N. & Pfister, T. (2021) ‘Temporal Fusion Transformers for Interpretable Multi-Horizon Time Series Forecasting’, *Advances in Neural Information Processing Systems*, 34, pp. 1121–1131.
- Luong, M., Pham, H. & Manning, C.D. (2015) ‘Effective Approaches to Attention-Based Neural Machine Translation’, *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*.

Bengio, Y., Simard, P. & Frasconi, P. (1994) ‘Learning Long-Term Dependencies with Gradient Descent is Difficult’, *IEEE Transactions on Neural Networks*, 5(2), pp. 157–166.

Zerveas, G., Jayaraman, S., Patel, D., Bhamidipaty, A. & Eickhoff, C. (2021) ‘A Transformer-based Framework for Multivariate Time Series Representation Learning’, *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pp. 2114–2124.