university of groningen

faculty of science and engineering

Financial Time-Series Forecasting

Neural Networks Project

Rares Medelet(S5608902)

Bogdan Sandoiu(S5549329)

Bogdan Ivancu(S5160901)

Abstract

Stock forecasting is a very intriguing topic when it comes to machine learning. People believe that they can get rich if they can predict what a stock is going to look like in the near future. In reality, it is very hard to predict such behavior because financial markets are filled with noise (i.e., random fluctuations), but our project aims to provide some insights into this issue. Our project focused on a modular pipeline for multivariate time series decomposition and forecasting. The system integrates data preprocessing, statistical decomposition, normalization, and sequence modeling. Raw wide-format datasets are first transformed into long-format sequences and split into training, validation, and test subsets. Each time series is decomposed per identifier into trend, seasonal, and residual components using either STL, additive, or multiplicative models from the statsmodel library. Residuals are normalized on a per-series basis to ensure scale invariance and stable training. A LSTM model is trained on normalized residuals to forecast future residual patterns over an 18-step horizon, with an early-stoping strategy.

Keywords: Time Series Forecasting, STL Decomposition, Residual Normalization, LSTM

Contents

1 Introduction

For our semester project, we focus on **forecasting business and financial time series**(one of the ready-made project topic suggestions). Unlike many classical forecasting problems with stable seasonal structure, financial data are noisy, non-stationary, and influenced by exogenous shocks-policy changes, geopolitical events, and natural disasters-that do not follow regular periodicity. These shocks constrain predictability and make pattern discovery and generalization challenging for purely data-driven models.
The project was executed according to the directions in the course materials.

1. **Choose a prediction scheme**: *Jump–forward:* train a network with H outputs to predict the next H steps at once.

2. **Select the model and input representation**: *RNNs:* feed the sequence directly; first normalize the series to a bounded range (e.g., $[0,1]$ or $[-1,1]$).

3. **Estimate the trend on the training data** (smoothing / LOESS / polynomial fit / FFT+low-pass / exponential smoothing).

4. **Deseasonalize (train-only template)**: compute a per-phase seasonal template $\hat{s}_p$ from the training split and remove it:

$$y_t^{(S)} = \begin{cases} y_t - \hat{s}_{p(t)}, & \text{additive} \\ \dfrac{y_t}{\hat{s}_{p(t)}}, & \text{multiplicative} \end{cases}$$

   Use $y_t^{(S)}$ in place of y_t below.

5. **Detrend to obtain residuals**:

$$r_t = \begin{cases} y_t^{(S)} - \tau_t, & \text{additive} \\ \dfrac{y_t^{(S)}}{\tau_t}, & \text{multiplicative} \end{cases}$$

6. **Train the network** to predict the *residuals* r_t (not the raw y_t).

7. **Forecast the trend** $\hat{\tau}_{t+h}$ over the horizon using the same smoother (or an ETS/Holt–Winters continuation).

8. **Recompose the final forecast**

$$\hat{y}_{t+h} = \begin{cases} \hat{\tau}_{t+h} + \hat{r}_{t+h} + \hat{s}_{p(t+h)}, & \text{additive} \\ \hat{\tau}_{t+h}\,\hat{r}_{t+h}\,\hat{s}_{p(t+h)}, & \text{multiplicative} \end{cases}$$

9. **Evaluate vs. Arima baselines** and judge performance over aggregates; financial series are shock-driven and hard to match point-wise.

Table 1: Sample of M3 (industry–monthly) results: top entries by sMAPE.

Participant / Method	**sMAPE**	**Rank (all)**
Wildi	14.84%	1
Theta Method (Nikolopoulos)	14.89%	2
ForecastPro (Stellwagen)	15.44%	3
Theta AI (Nikolopoulos)	15.66%	4
Autobox (Reilly)	15.95%	5

Performance-wise, we aim to achieve outcomes comparable to those obtained by researchers in the original M3 competition on the chosen subset(Table 1).

To address this gap, new research provided insight into the use of neural networks. A research that we've found, Kanungo and Praggnya[1] applies an LSTM model to financial market time series, showing that deep learning methods can outperform traditional statistical approaches in capturing long-term dependencies and non-linear relationships, provided that careful feature engineering and hyper-parameter tuning are performed. The study reported that neural networks provided lower prediction error and greater adaptability under volatile conditions, particularly when temporal structure was preserved. Furthermore, Hwang et al.[2] show that an LSTM model with trainable initial hidden states achieved higher accuracy on stock-market data, effectively learning cross-series dependencies and improving generalization. These findings indicate that modern LSTM architectures are capable of modeling complex temporal behaviors.

Given these results, our study was motivated to evaluate how decomposition-based preprocessing could further enhance the forecasting capabilities of both ARIMA and LSTM models. If deep learning models outperform statistical ones in raw financial data, as Kanungo and Praggnya[1] have shown, then decomposing the data into trend, seasonal and residual components may provide a structured way to improve interpretability and stability.

2 Data Alignment

2.1 M3 Competition Dataset

We use the monthly *industry* subset of the M3 Forecasting Competition dataset. We select the monthly subset because it provides longer histories per series than the yearly or quarterly subsets. For a comprehensive description of all M3 subsets, time horizons, and counts, see Hibon and Makridakis [3]. Below, we briefly summarize the subset used in this work.

2.2 Monthly Industry Subset

The M3 monthly industry subset comprises 334 independent *univariate* monthly series (e.g., N1877, N1878) - check Figure 1, with lengths between 96 and 144 observations. Each series begins at a specific year–month origin (start years 1977–1983; start months January, March, June, or December). Some series share the same start date; others start at different dates. All series are evaluated on the same task: forecasting the last 18 observations (a fixed horizon of 18 months, $NF = 18$).

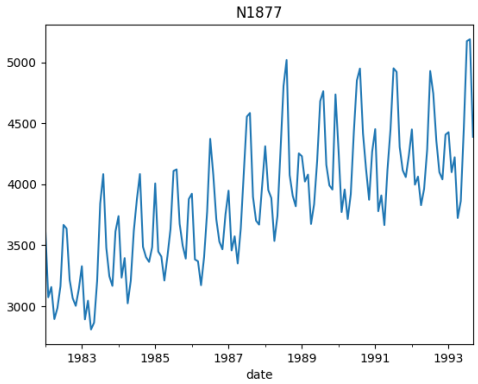

Figure 1: Example univariate monthly series (**N1877**).

The dataset is a wide CSV where each *row* is one series. The first six columns are metadata, followed by up to 144 monthly values, as in Table 2:

- **Series** - string ID (e.g., **N1877**).
- **N** - number of observed months for the series.
- **NF** - forecast horizon in months (here, 18).
- **Category** subset label (e.g., **INDUSTRY**).
- **Starting Year** - calendar year of the first observation.
- **Starting Month** - month index of the first observation (1=Jan, 3=Mar, 6=Jun, 12=Dec).
- **1**, **2**, . . . , **144** - monthly values in chronological order; only the first **N** cells are populated for each row.

Table 2: Wide-format structure of the raw file (metadata + first months), we dropped the **Category** column.

Metadata					Values (months)			
Series	N	NF	Starting Year	Starting Month	1	2	3	…
N1877	141	18	1982	1	3650.16	3072.22	3156.06	
N1878	141	18	1982	1	5379.20	5338.00	5977.00	
N1879	144	18	1977	1	9315.00	9636.00	11,657.50	
N1880	134	18	1983	1	5050.00	5025.00	5080.00	
N1881	144	18	1977	1	1955.00	2180.00	2375.00	

2.3 Preparation: Transformations and Splits

Transformations from wide to long. The raw file is in a wide format (one row per series with up to 144 value columns). We reshape it into a tidy long table with three columns: **Series**, **date**, and **value**. For each series i and column $k = 1, \ldots, N_i$, the timestamp is

$$\textbf{date} = (\textbf{Starting Year}_i, \textbf{Starting Month}_i) + (k-1) \text{ months}.$$

We stack the N_i observed values into rows and sort by **Series** and **date**, yielding contiguous blocks per ID (e.g. all months for **N1877** in order, then **N1878**, and so on), as in Table 3.

Table 3: Long-format example.

Series	date	value
N1877	1982-01-01	3650.16
N1877	1982-02-01	3072.22
N1877	1982-03-01	3156.06
	⋮	
N2209	1993-10-01	3990.00
N2209	1993-11-01	3820.00
N2209	1993-12-01	4410.00

Splits. After transforming to the long format, we split the data chronologically per series ID. For each ID i, the last 18 observations are held out as the test set (the M3 horizon); the 18 observations immediately preceding test form the validation set; all earlier observations constitute the training set. Splits are performed independently for each ID and preserve temporal order.

3 Data Preprocessing

Preprocessing policy. Preprocessing is done per series ID, with all parameters fit on the *training* split only and then applied to that ID's validation and test data, preventing any cross-ID or cross-split leakage.

3.1 Decomposition

Our preprocessing begins with a per-ID decomposition of each monthly series into trend, seasonal, and residual components using three methods: **STL**, **classical additive**, and **classical multiplicative** (explained below). We first observe clear nonstationarity, seasonal amplitude changes with the level (see Fig. 1). For clarity we illustrate with **N1877**, but the same pattern appears across all 334 series.

Log transform for STL/additive. To stabilize variance and make seasonal amplitude roughly level-invariant, we apply a log transform before STL and classical additive. This maps a multiplicative structure to an additive one on the log scale. Prior to logging, we verified that all training values are strictly positive for every ID. The transform is applied per ID, with parameters estimated on the **training** split only.

Decomposition methods. We implement three per-ID decompositions with a seasonal period of 12. The data are monthly, so a 12-month period captures the annual cycle explicitly.

STL (log scale) Figure 2. We use *odd* LOESS window sizes so the filter is centered:

- *period = 12*
- *seasonal window = 11:* close to the period *period*, allowing the seasonal component to evolve slowly over time without overfitting month-to-month noise. Smaller windows make seasonality too wiggly; much larger windows make it too rigid.
- *trend window = 21:* about ~ 2*period*, which smooths over at least one full seasonal cycle so that the trend does not "steal" seasonal variation. Shorter windows can let seasonal structure leak into the trend; longer windows risk under-tracking genuine level changes.
- *robust = True:* economic/industry series often contain shocks and outliers; robust weighting in STL mitigates their influence on both trend and seasonality.

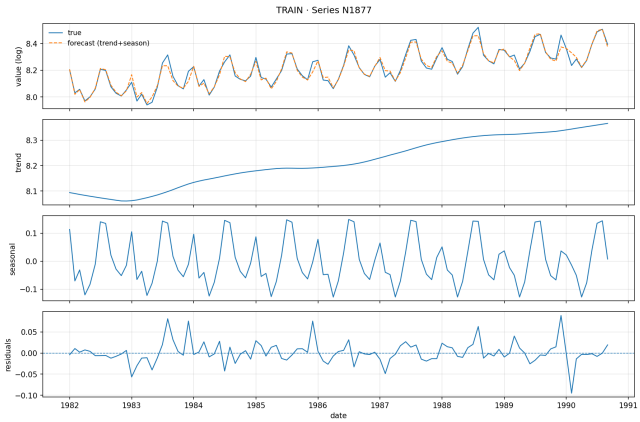

Figure 2: Example decomposed components from the training set using STL(**N1877**).

Classical additive (log scale) Figure 3.

- *model = additive, period = 12:* after a log transform, the amplitude becomes roughly level-invariant, making an additive decomposition appropriate with the same annual period.
- *two-sided centered filter (default):* yields a symmetric moving-average trend with minimal phase distortion. We *do not* extrapolate the trend at the edges, avoiding model-based bias near boundaries.
- *drop edge NA:* we discard boundary points where the centered trend is undefined, rather than forward/back-filling, to avoid introducing artificial structure.

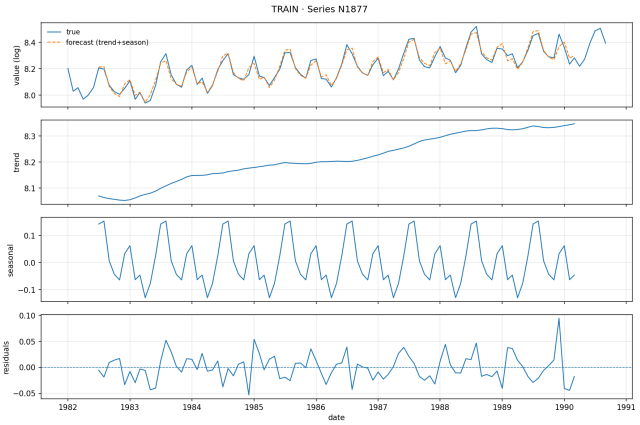

Figure 3: Example decomposed components from the training set using Additive method(**N1877**).

Classical multiplicative (original scale) Figure 4.

- *model = multiplicative, period = 12:* appropriate when seasonal swings scale with the level and values are strictly positive.
- *require positive values:* ensures mathematical validity of the multiplicative form and avoids undefined ratios.

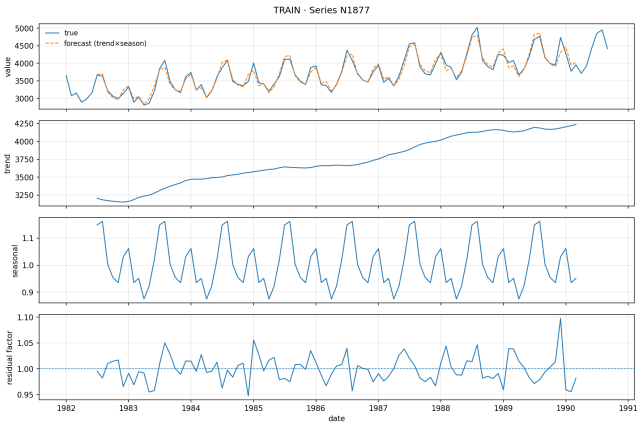

Figure 4: Example decomposed components from the training set using Multiplicative method(**N1877**).

Interpretation and comparison. The panels show the estimated trend and seasonality and the corresponding residuals. The dashed orange curve is the trend + season reconstruction plotted against the observed series (blue). All three methods recover the rising level and pronounced annual cycle, but they differ in working domain, interpretability, and flexibility.

Multiplicative (Fig. 4). Components are on the original scale with $y_t \approx T_t \times S_t \times E_t$. The trend rises from roughly 3.2k to 4.2k. The seasonal component is a *factor* around 1 (about 0.90–1.12), i.e., a $\pm 10\%$ swing relative to trend. Residuals are multiplicative (≈ 1) and highlight short-lived shocks

(e.g., late 1989–1990). This view is convenient for percent effects.

Classical additive (Fig. 3). After logging, $\log y_t = \tau_t + s_t + \varepsilon_t$. The trend is smooth and increasing (about 8.05 → 8.35 in log-units, consistent with 3.1k→4.2k on the original scale). Seasonality is roughly symmetric around 0 on the log scale (about ±0.1, i.e., ±10% back on the original scale). Residuals are zero-centered and homoscedastic; edge points are dropped because the centered moving-average trend is undefined at the boundaries.

STL (Fig. 2). Also on the log scale, but estimated via LOESS with robust weighting. The trend is very smooth yet adaptive, and the seasonal pattern can drift slowly if present. Residuals are zero-centered with slightly lower variance and less structure than in the classical additive case, indicating a cleaner separation of trend/seasonal effects and better robustness to outliers.

3.2 Holdout Decomposition (Validation & Test)

We do *not* re-fit a decomposition on VAL/TEST. Instead, for each ID we re-use train-only components and continue them forward at the holdout dates, thus avoiding leakage.

Seasonal template from train. From the train decomposition we compute a per-ID mean seasonal value for each month-of-year phase $p \in \{1,\dots,12\}$:

$$\hat{s}_{i,p} = \text{mean}\{\, s_{i,t} \;:\; p(t) = p,\ t \in \text{train}\,\}.$$

For STL/additive (log-domain) we center $\hat{s}_{i,p}$ to have zero mean over the 12 phases; for multiplicative we keep it as a factor near 1.

Trend continuation at holdout dates. We forecast the train trend forward with a damped Holt–Winters model (Exponential Smoothing, additive trend, no season):

$$\hat{\tau}_{i,T+h} = \text{ETS}(\tau_{i,1:T}), \qquad h = 1,\dots,H_i,$$

where T is the last train timestamp for ID i. For the test split we offset the forecast by the length of that ID's validation window so dates align exactly. In the multiplicative path we clip $\hat{\tau}_{i,t} > 0$.

Compose holdout components and residuals. Let $y_{i,t}$ be the observed value at holdout date t with phase $p(t)$.

- **STL/additive (log scale)**: require $y_{i,t} > 0$.

$$\log y_{i,t} = \hat{s}_{i,p(t)} + \hat{\tau}_{i,t} + \varepsilon_{i,t}, \quad \varepsilon_{i,t} = \log y_{i,t} - \hat{s}_{i,p(t)} - \hat{\tau}_{i,t}.$$

  We also report $\log y_{i,t} - \hat{s}_{i,p(t)}$ as the deseasoned value.
- **Multiplicative (original scale)**:

$$y_{i,t} = \hat{\tau}_{i,t}\,\hat{s}_{i,p(t)}\,e_{i,t}, \quad e_{i,t} = \frac{y_{i,t}}{\hat{\tau}_{i,t}\,\hat{s}_{i,p(t)}}.$$

  Deseasoned values are $y_{i,t}/\hat{s}_{i,p(t)}$.

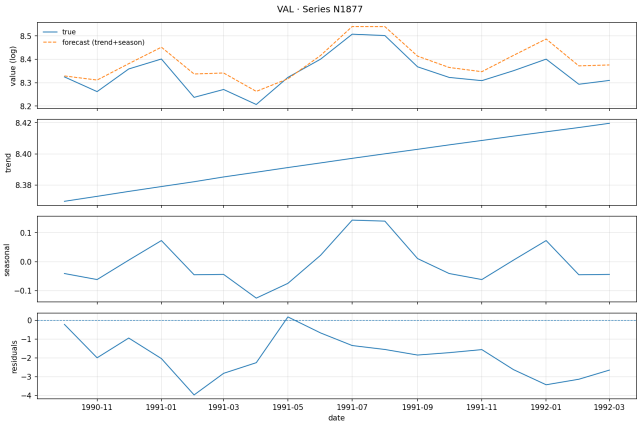

Figure 5: Example decomposed components from the val set using STL predicted components(**N1877**).

This setup prevents leakage and yields an out-of-sample continuation of trend and seasonality for the validation/test horizons (see Figs. 5 and 6). Subtracting these components gives the holdout residuals; we train the model on the *training* residuals and evaluate it by predicting residuals on validation/test. As the figures indicate, the trend continuation is already strong, so the main difficulty lies in modeling the residuals.

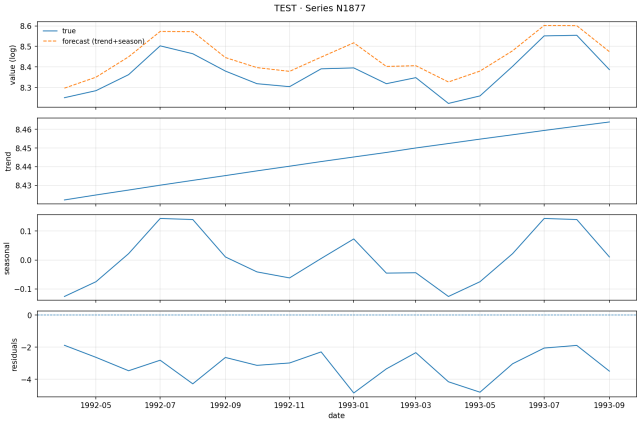

Figure 6: Example decomposed components from the test set using STL predicted components(**N1877**).

3.3 Residuals normalization.

After decomposition we standardize the residuals independently for each series ID. Let $r_{i,t}$ denote the residual for ID i at time t. We compute the training statistics $\mu_i = \text{mean}\{r_{i,t}\}_{t\in\text{train}}$ and $\sigma_i = \text{std}\{r_{i,t}\}_{t\in\text{train}}$, and define the normalized residual $\tilde{r}_{i,t} = (r_{i,t} - \mu_i)/\sigma_i$ for train/val/test using the same train-only (μ_i, σ_i). This removes scale differences across IDs and prevents cross-split leakage. For STL/additive (log domain) the residuals are additive; for the multiplicative model the residuals are factors around 1, and standardization centers them near 0 with unit variance.

3.4 Fourier regressors

To capture mild seasonal drift that may remain after templating, we augment the *residuals* with Fourier harmonics of the month-of-year phase. For period $P = 12$ and harmonics $k = 1, 2, 3$, the features are

$$\{\sin(2\pi k\, p/12),\ \cos(2\pi k\, p/12)\}_{k=1}^{3},$$

where $p \in \{1, \ldots, 12\}$ is the month index of each timestamp. We compute these deterministically from the calendar phase (no leakage) and attach them to the **training**, **validation**, and **test** residual tables, using the same period and number of harmonics across splits. This keeps the model's input space consistent and helps explain residual periodicity beyond the fixed per-ID seasonal template.

4 Methods

In this section, we detail how we build per-ID sliding windows (for the LSTM only) for the train/validation/test splits, and we describe the ARIMA and LSTM forecasting architectures.

4.1 ARIMA: explanation and hyperparameters

An ARIMA model predicts a time series by combining three simple ideas: **(i) Autoregression** (use recent past values), **(ii) Differencing** (remove slow drift so the series is stable), and **(iii) Moving average** (use recent past forecast errors). Its seasonal extension, SARIMA, adds the same three ideas but at a fixed seasonal lag (e.g., 12 for monthly data).

What it does . At time t, the forecast is a weighted sum of: recent values $(r_{t-1}, r_{t-2}, \ldots)$, recent shocks $(\varepsilon_{t-1}, \varepsilon_{t-2}, \ldots)$, and optionally their seasonal counterparts $(r_{t-12}, r_{t-24}, \ldots;$ $\varepsilon_{t-12}, \varepsilon_{t-24}, \ldots)$. Differencing steps ensure the series behaves stably around a constant level.

Why it fits our pipeline. Because we first remove trend and seasonality and then z–score the residuals, the remaining signal is close to stationary. SARIMA is therefore a good, interpretable baseline to capture the short-run autocorrelation left in the *residuals*. A seasonal ARIMA is written $\mathrm{SARIMA}(p,d,q) \times (P,D,Q)_s$:

- p **(AR order):** how many recent past values influence the forecast $(r_{t-1}, \ldots, r_{t-p})$. Larger p captures richer short-term memory.
- d **(nonseasonal differencing):** how many times we difference the series to remove slow drift. After decomposition, we typically set $d = 0$.
- q **(MA order):** how many recent past errors we use $(\varepsilon_{t-1}, \ldots, \varepsilon_{t-q})$.
- P **(seasonal AR order):** AR lags at the seasonal step (e.g. r_{t-12} for monthly). Captures similarity across the same month in different years.
- D **(seasonal differencing):** seasonal differencing to remove seasonal level shifts. After deseasonalizing upstream, we typically set $D = 0$.
- Q **(seasonal MA order):** MA terms at the seasonal step (e.g. ε_{t-12}).
- s **(seasonal period):** the seasonal length; here $s = 12$ (monthly).

4.2 ARIMA baseline on residuals

We fit one SARIMA model *per ID* on the **training residuals** (already z–scored per ID). No re-fitting is done on VAL/TEST.

Model class. For each series i we consider a seasonal ARIMA:

$$\mathrm{SARIMA}(p,d,q) \times (P,D,Q)_s, \qquad s = 12,$$

estimated with `statsmodels`' `SARIMAX` (stationarity and invertibility enforced).

Specification search. We perform a small AIC–guided grid search:

$$(p,d,q) \in \{(0,0,0), (1,0,0), (0,0,1), (1,0,1), (2,0,1)\},$$

$$(P,D,Q,12) \in \{(0,0,0,12), (1,0,0,12), (0,0,1,12), (1,0,1,12)\}$$

For each candidate we fit the model (max. 200 iterations), compute AIC, and keep the specification with the lowest AIC. The fitted model and its spec are cached per ID.

Forecasting on holdout. Given a template of holdout timestamps for ID i (VAL/TEST), we forecast h steps ahead, producing $\widehat{r}_{i,t}$ aligned to those dates.

Evaluation. Predictions are compared to the true residuals (still in z–score space) using

$$\mathrm{MAE} = \frac{1}{n}\sum |r_{i,t} - \widehat{r}_{i,t}|, \quad \mathrm{RMSE} = \sqrt{\frac{1}{n}\sum (r_{i,t} - \widehat{r}_{i,t})^2},$$

$$\mathrm{sMAPE} = \frac{1}{n}\sum \frac{|r_{i,t} - \widehat{r}_{i,t}|}{|r_{i,t}| + \varepsilon},$$

reported per ID and with a global aggregate. Because we model residuals only, final reconstructions on the original scale are obtained by combining the held-out trend/season continuation with the predicted residuals.

4.3 Windowing for the LSTM (per ID)

To train the LSTM, each per–ID residual series is converted into supervised windows. Because M3 monthly series are short (about 96–144 points) and we hold out 18 months for validation and 18 for test, the holdout segments are too small to form (36 lags + 18 labels) windows on their own.

- **Training (sliding windows inside TRAIN only).** For each ID, build sliding windows where the input length is $T = 36$ and the direct horizon is $H = 18$: each window uses months $t-35{:}t$ to predict $t+1{:}t+18$, stopping before the train boundary. Windows are created for ID A, then ID B, etc., and concatenated *without* shuffling across IDs.
- **Validation (one forecast per ID).** For each ID, take the *last 36 train months* as context, feed a single $(1 \times 36 \times F)$ block to the model, and align its 18-step output to the 18 VAL timestamps.
- **Test (one forecast per ID).** For each ID, concatenate TRAIN and VAL, take the *last 36 months* as context, and align the model's 18-step output to the 18 TEST timestamps.

We choose $T = 36$ (about three seasonal cycles) to provide sufficient seasonal context while leaving enough training examples within the TRAIN split.

4.4 LSTM model (global residual forecaster)

We train a single LSTM on *all* IDs to predict residuals directly over an H-step horizon. Each sample is a window of length T with F features (e.g., residual r_t and Fourier terms), so the input tensor has shape (T, F) and the model outputs a vector in R^H.

Input/Output.
- **Input:** window $(x_{t-T+1}, \ldots, x_t) \in R^{T \times F}$.
- **Output:** direct multi–step forecast $(\hat{r}_{t+1}, \ldots, \hat{r}_{t+H}) \in R^H$.

Architecture.
- LSTM(128, `return_sequences=True`), $L_2 = 10^{-4}$, Dropout 0.1
- LSTM(64, `return_sequences=False`), $L_2 = 10^{-4}$, recurrent Dropout 0.1, Dropout 0.1
- Dense(128, ReLU, $L_2 = 10^{-4}$), Dropout 0.2
- Dense(H) linear head `residuals_hat`

This is a *direct* forecaster (the head emits all H steps at once, not autoregressively).

Why these choices for *our* data.

- **Keep it simple to avoid overfitting.** Per–ID histories are short and the global window set is modest; deeper/wider stacks quickly overfit. Two LSTM layers (128→64) with a small dense head gave the best val loss vs. capacity trade-off.
- **Regularization tuned for "just enough" constraint.** Light L_2=10^{-4} on recurrent and kernel weights plus small dropout (0.1–0.2) stabilizes training without erasing the weak residual signal.
- **Huber loss to resist mean–collapse.** Residuals are z-scored ($\approx [-1, 1]$) and the naïve optimum is near zero. With MSE we observed faster drift toward the mean and high sensitivity to shocks; Huber keeps strong curvature near zero (learning beyond the mean) but is linear in the tails, limiting the influence of outliers.

Training setup.
- **Loss:** Huber with $\delta = 1.0$:

$$\mathcal{L}_\delta(e) = \begin{cases} \frac{1}{2}e^2, & |e| \le \delta \\ \delta(|e| - \frac{1}{2}\delta), & |e| > \delta \end{cases}$$

- **Optimizer:** Adam, learning rate 3×10^{-4}, gradient clipping (`clipnorm`=1.0) to prevent exploding gradients.
- **Batching/epochs:** batch size 32, up to 100 epochs, **no shuffling** to preserve temporal order.
- **Callbacks:** *EarlyStopping* on `val_loss` (patience 20, restore best weights) used instead of k-fold CV to keep the temporal split intact and avoid excessive compute; *ReduceLROnPlateau* (factor 0.5, min LR 10^{-6}) drops the LR once the model hits a plateau, typically after it has learned the near-zero baseline, so it can pick up smaller residual patterns.
- **Metric reported during training:** MAE.

Data interface. Training uses the prebuilt windows: $X_{\text{train}} \in R^{N \times T \times F}$, $y_{\text{train}} \in R^{N \times H}$. Validation/test predictions are produced from a *single* history block per ID (last T rows from TRAIN for VAL; last T from TRAIN+VAL for TEST) and aligned to the holdout timestamps.

Evaluation on arrays. Given X and ground truth y (both shaped as above), we report MAE and RMSE.
Across multiple architectures, training consistently converged to the zero-mean baseline(Figure 7).

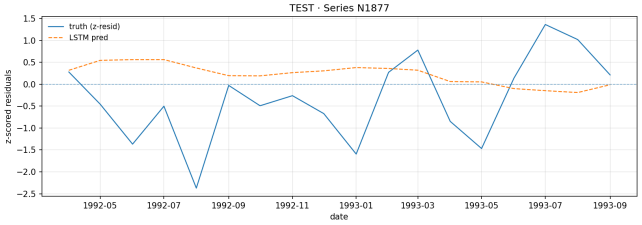

Figure 7: Example predictions of the residuals using LSTM(**N1877**).

4.5 Forecast Reconstruction

After obtaining residual forecasts from either the LSTM or ARIMA model, the final predictions were reconstructed by combining the forecasted residuals with the extrapolated trend and seasonal components:

For additive and STL decomposition:

$$\hat{Y}_{i,t} = \exp(\hat{T}_{i,t} + \hat{S}_{i,t} + \hat{R}_{i,t} \cdot \sigma_i + \mu_i)$$

For multiplicative decomposition:

$$\hat{Y}_{i,t} = \hat{T}_{i,t} \times \hat{S}_{i,t} \times (\hat{R}_{i,t} \cdot \sigma_i + \mu_i)$$

Where $\hat{T}_{i,t}$ and $\hat{S}_{i,t}$ are the forecasted trend and seasonal components, $\hat{R}_{i,t}$ is the predicted normalized residual, and μ_i, σ_i are the normalization parameters from training.

4.6 Evaluation Metrics

Three standard forecasting metrics were computed to evaluate model performance:

Mean Absolute Error(MAE):

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

Root Mean Squared Error (RMSE):

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

Symmetric Mean Absolute Percentage Error (sMAPE):

$$\text{sMAPE} = \frac{1}{N}\sum_{i=1}^{N}\frac{|y_i - \hat{y}_i|}{|y_i| + \varepsilon}$$

Where $\varepsilon = 10^{-8}$ is a small constant to prevent division by zero.

Metrics were computed both at the per-series level (aggregated within each time series) and globally (aggregated across all series and time points). The global metrics provide an overall assessment of model performance across the entire dataset.

5 Results

We evaluate two residual forecasters-**ARIMA** and **LSTM**-under three decomposition pipelines: *additive*, *multiplicative*, and *STL*. Both models are trained on per-ID, z-scored *residuals*; trend and seasonal components are *extrapolated from train only* and used at the end to reconstruct predictions on the original scale. We report MAE, MSE, and sMAPE on the test sets.

5.1 Model Performance

Across the *additive* and *multiplicative* pipelines the two models behave similarly, while the *STL* path consistently yields higher errors. Since both models are fit on normalized residuals, performance differences across decompositions largely appear after *reconstruction* (combining the predicted residuals with each method's trend/season continuation).

ARIMA. Because the ARIMA baseline is trained on per-ID z-scored residuals, it does not directly "favor" any decomposition in residual space. After reconstruction to the original scale, however, the **multiplicative** path gives the best aggregate accuracy (MAE = 889.67, RMSE = 1825.71, sMAPE = 0.1827), with the **additive** path nearly indistinguishable. STL underperforms both, consistent with its smoother trend/season carrying larger reconstruction error at turning points.

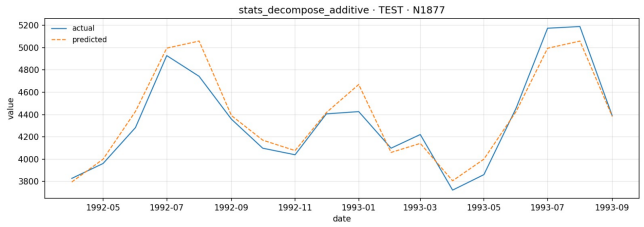

Figure 8: ARIMA on residuals with *additive* reconstruction (**N1877**).

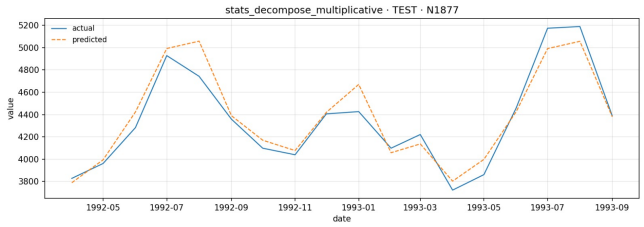

Figure 9: ARIMA on residuals with *multiplicative* reconstruction (**N1877**).

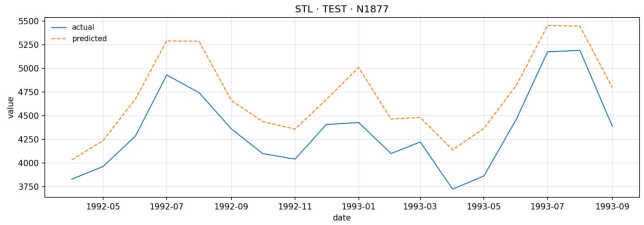

Figure 10: ARIMA on residuals with *STL* reconstruction (**N1877**).

LSTM. The LSTM is likewise trained on normalized residual sequences and evaluated after reconstruction. Under this setup, the **multiplicative** pipeline again gives the strongest results (lowest TEST sMAPE = 0.1936), with the **additive** pipeline close behind. This indicates that pairing a global residual model with a scale-consistent (multiplicative) reconstruction is beneficial, even when the model itself never sees the original scale.

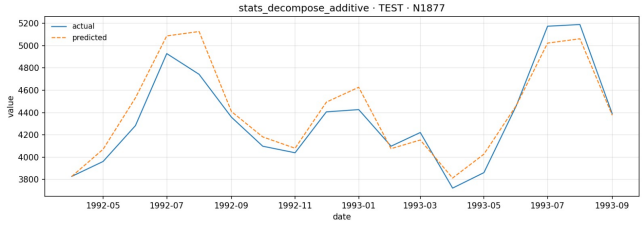

Figure 11: LSTM on residuals with *additive* reconstruction (**N1877**), test.

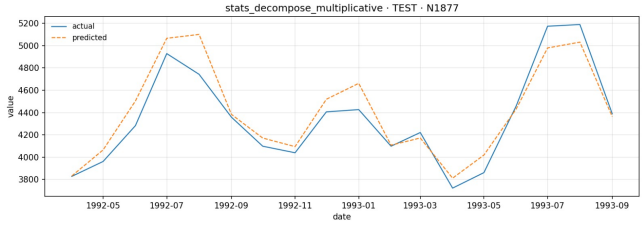

Figure 12: LSTM on residuals with *multiplicative* reconstruction (**N1877**), test.

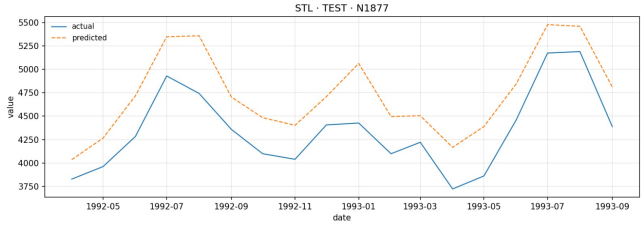

Figure 13: LSTM on residuals with *STL* reconstruction (**N1877**), test.

STL observations. STL-based results are substantially worse (e.g. test RMSE $\approx 7.5 \times 10^4$ on the original scale). Two factors contribute: (i) STL's LOESS trend/season is very smooth and can under-track turning points, increasing reconstruction error, and (ii) the residuals left for the sequence model become harder (less predictable) after aggressive smoothing upstream. The same pattern appears for ARIMA, suggesting the issue lies in the reconstruction path rather than a specific residual learner.

5.2 ARIMA vs. LSTM: comparative analysis

On validation, LSTM typically edges ARIMA (especially under multiplicative and additive pipelines), but the gap narrows on test, where ARIMA often generalizes more steadily. This reflects the trade-off between *flexibility* (LSTM can fit subtle residual structure but may overfit) and *simplicity* (ARIMA captures short-run linear dependence with stable out-of-sample behavior). Since both models operate on residuals, the decomposition mainly affects (a) how hard the residuals are to forecast and (b) how errors rescale during reconstruction. In our data, the multiplicative path provides the most reliable end-to-end accuracy.

6 Discussion

6.1 Overview of findings

Our results confirm that decomposing financial time series *can* improve downstream forecasting by isolating trend and seasonality and giving models a simpler, more stationary target (the residuals). Across setups, the **multiplicative** pipeline yields the most reliable reconstructions on the original scale; **additive** is close, while **STL** underperforms. However, the realized gains depend critically on how well residuals are *calibrated* across splits (train/val/test).

6.2 Residual calibration diagnostics (train vs. holdout)

This is the main problem that we could not solve and affected our LSTM and ARIMA models. After per-ID normalization (z-scoring with train-only μ_i, σ_i), we expected residuals to be centered near 0 with unit variance on *all* splits. Instead, we observe large departures:

Table 4: Residual calibration diagnostics (z-scored residuals, per ID).

Split	mean / std	IDs with $\lvert\text{mean}\rvert > 0.10$	IDs with std $\notin [0.8, 1.2]$
TRAIN	0.0003 / 0.1174	0 / 334	333 / 334
VAL	0.1329 / 2.4695	309 / 334	215 / 334
TEST	1.0008 / 4.2557	324 / 334	231 / 334

Primary driver: train-based continuation of components. The inflated holdout means and variances arise mainly from *forecasting the trend and templating the seasonality from TRAIN into VAL/TEST*. Concretely: (i) even a mild level bias in the 18-month trend forecast shifts all holdout residuals away from zero; (ii) seasonal drift not captured by the train template leaves phase-specific offsets and inflates variance;
Likely causes. (a) Trend continuation offset errors (VAL/TEST not aligned to the end of TRAIN); (b) seasonal template not centered (log/additive) or too close to zero (multiplicative); (c) missing/invalid merges of (μ_i, σ_i) for some IDs; (d) no lower bounds on $\hat{\tau}$ or $\hat{s}$, leading to divisions by very small numbers.

6.3 Interpretation

When residuals are well calibrated, ARIMA provides a transparent, stable baseline for short-run linear dependence, while the LSTM can capture weak nonlinear and cyclic leftovers. In our runs, the poor calibration on holdout (and even on train) likely violated SARIMA's stationarity assumption and gave the LSTM a moving/ill-scaled target, encouraging mean collapse and plateauing.

6.4 Limitations and potential improvements

- **Residual miscalibration** (above) is the primary limitation; fixing it is prerequisite to fair model comparison.
- **Manual LSTM hyperparameters.** We chose a deliberately small network to limit overfitting.
- **STL smoothing.** STL likely over-smooths turning points, inflating reconstruction error.
- **No exogenous signals.** Incorporating macro/market covariates could improve residual predictability.

6.5 Practical implications and future work

Decomposition remains useful, but its benefits materialize only with *consistent* residual scaling across splits. Next steps: repair residual calibration, re-run diagnostics, then revisit (i) richer global models (e.g., Transformers), (ii) hybrid pipelines that jointly learn trend/season + residuals.

6.6 Conclusion

A decomposition-first pipeline is promising for financial forecasting, but its success hinges on correct holdout continuation and residual normalization. Once residuals are properly calibrated, SARIMA offers steady baselines and LSTM can add nonlinear gains; without that, both will underperform or learn trivial solutions.

Bibliography

[1] P. Kanungo, "Time series forecasting in financial markets using deep learning models," *World Journal of Advanced Engineering Technology and Sciences*, vol. 15, April 2025. PDF available.

[2] J. Hwang, S. Yoon, S. Kim, and J. Choi, "Modeling financial time series using lstm with trainable initial hidden states," *arXiv preprint arXiv:2007.06848*, 2020.

[3] M. Hibon and S. Makridakis, "The m3-competition: results, conclusions and implications," *International Journal of Forecasting*, vol. 16, no. 4, pp. 451–476, 2000.