Nowcasting the United States Economy

DSE3101 Technical Documentation

Jeron Tan Kang, Cai Mun Jia, Kellianne Ng, N Ashwin Kumar, Tammy Alexandra Wong

April 2025

Contents

Part I: Project Overview **3**

1 Project Overview **3**

2 Getting Started **3**

3 Overall Design **3**

4 Data and Methodology **4**

Part II: Back-End Development **4**

5 Data Pipeline and Preprocessing **4**
5.1 Data Processing 4
5.2 Data Used by Non-Linear Models 5
5.3 Data Used by Linear Models 5

6 Modeling Approach **6**
6.1 Forecasting of Monthly Indicators for Bridge Models 6
6.2 AR Benchmark 7
6.3 ADL Bridge 7
6.4 RF Benchmark 8
6.5 RF Bridge 8

7 Forecast Evaluation **10**
7.1 Algorithm for Evaluating Forecast Error 10
7.2 Model Performance 10
7.3 Diebold-Mariano Test 11

8 Challenges and Reflections **12**
8.1 Challenges, Trade-offs and Limitations 12
8.2 Future Work 12
8.3 Team Contributions 12

Part III: Front-End Development **13**

9 UI Structure and Interaction **13**

9.1 Dashboard Layout 13
9.2 Input → Backend Pipeline 13

10 Design Rationale **13**
10.1 Visual Decisions 13
10.2 UI/UX Considerations 13

11 Dashboard Implementation **14**
11.1 GDP Nowcasting 14
11.2 Time Travel 14
11.3 Model Comparison 14

12 Limitations and Reflections **14**
12.1 Known Issues 14
12.2 Future Work 15
12.3 Team Contributions 15

13 Appendix **16**

Appendix **16**
A.1 Differencing Non-Stationary Indicators 16
A.2 Identifying Months to Forecast 16
A.3 Forecasting Missing Monthly Indicators 17
A.4 Aggregating Monthly Indicators to Quarterly 19
A.5 AR Model Coefficient Estimates 20
A.6 ADL Bridge Model Coefficient Estimates 20

Part I: Project Overview

1 Project Overview

The objective of this project is to build a real-time GDP nowcasting system using a combination of statistical and machine learning models. The dashboard simulates what analysts would see in real time by only using data that would have been available at a given point in time.

End users include policy analysts, researchers, and students looking to monitor macroeconomic conditions. Key features include model comparisons, time-travel simulation, and recession-aware forecasting.

Tools used:

- Python (pandas, scikit-learn, statsmodels)
- Streamlit for hosting of frontend's dashboard
- FRED API for economic data
- GitHub for version control

Link to App: `https://nowcasting-us-gdp-ctgqbbsvayhsnvm2ejafbo.streamlit.app/`(Deployed on Streamlit Cloud)

Link to GitHub Repo: `https://github.com/JeronTanKang/Nowcasting-US-GDP`

2 Getting Started

To set up the environment and run the application locally:

1. Clone the repository: `https://github.com/JeronTanKang/Nowcasting-US-GDP`
2. Install dependencies: `pip install -r requirements.txt`
3. Add API tokens to the appropriate file (e.g., `config.py` or `.env`)
4. Run the dashboard: `streamlit run app.py`

To ensure the dashboard uses the most up-to-date macroeconomic indicators from the FRED API, the app.py file can be configured to automatically execute two R scripts (`bridge_indicators.r` and `tree_df.r`) upon startup. These scripts will pull raw data via the fredr package, preprocess it by addressing missing values, differencing non-stationary series, and generating lagged features. The resulting datasets(`bridge_df.csv` and `tree_df.csv`) are written to the Data/ directory and immediately ingested into the Streamlit dashboard. This design eliminates reliance on static CSV files and ensures that all forecasts are based on the latest available economic data. The R scripts are invoked using `subprocess.run()`, and error handling is incorporated to catch failed data pulls or processing issues.

However, for the purpose of public deployment on Streamlit Cloud, we opted to use pre-generated static CSVs to avoid exposing our FRED API keys.

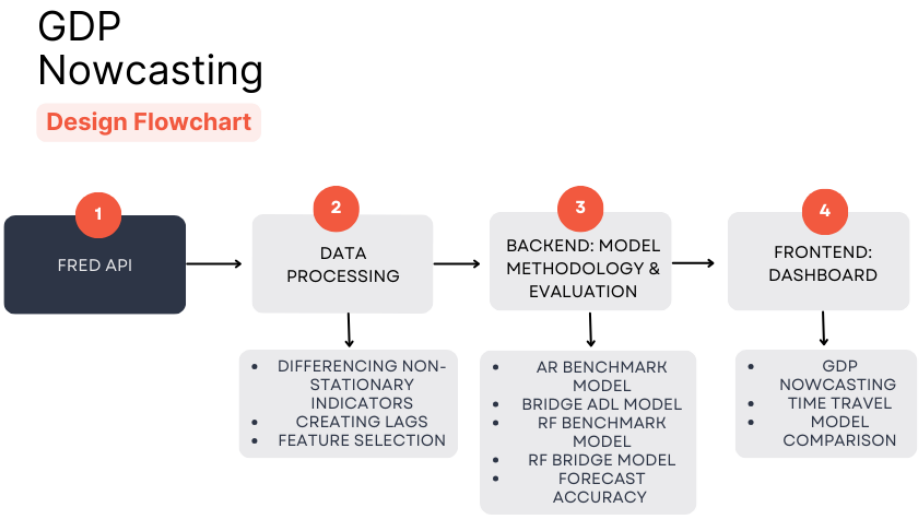
3 Overall Design

The system is structured around a modular architecture that separates data processing, modeling, and user interaction layers.

The pipeline begins with raw macroeconomic data obtained through the FRED API. These indicators are preprocessed to address stationarity, missing values, and appropriate lagging. All models (AR, RF, ADL Bridge, RF Bridge) are evaluated through a rolling window framework to ensure robust pseudo out-of-sample performance. Metrics such as RMSFE, MAFE, directional accuracy, and downturn accuracy are used to assess model quality. The Diebold-Mariano test is also implemented to test for statistically significant differences in forecasting performance.

The frontend dashboard, built using Streamlit, provides three main views: Current Nowcast, Time Travel, and Model Comparison. These allow users to interactively simulate forecasts, view historical model behavior, and compare performance across models and time periods. Interactive

widgets such as dropdowns, sliders, and toggle buttons enable real-time exploration of forecasts, confidence intervals, and data assumptions. Backend computations are cached and pipelined efficiently to support responsive updates upon user interaction.



4 Data and Methodology

We used a set of 23 macroeconomic indicators obtained from the FRED-MD database via API. Indicators were aggregated to quarterly frequency using variable-specific rules. The target variable, annualized GDP growth, was computed from the log-difference of real GDP.

Pre-processing involved differencing non-stationary indicators, creating lags, and applying feature selection using LASSO for linear models and RFECV for Random Forest models.

We implemented four models: AR, RF, ADL Bridge, and RF Bridge. Bridge models used forecasted monthly indicators to construct quarterly inputs. The RF Bridge Model applied hyperparameter tuning. Model performance was evaluated using RMSFE, MAFE, downturn accuracy, directional accuracy and implementing the Diebold-Mariano Test.

Part II: Back-End Development

5 Data Pipeline and Preprocessing

5.1 Data Processing

We curated a set of monthly economic indicators based on economic intuition. Data on these indicators were then fetched from FREDMD using an API key, with the respective Fred Code given in the table below. After fetching these indicators, the target variable, annualised GDP growth rate, is calculated as: $\text{GDP growth}_t = 400 \times [\log(\text{GDP}_t) - \log(\text{GDP}_{t-1})]$.

Table 1: Economic Indicators and Corresponding FRED Codes

Indicator	FRED Code	Indicator	FRED Code
GDP (Real, Seasonally Adjusted)	GDPC1	CPI	CPIAUCSL
Crude Oil	DCOILWTICO	Interest Rate	FEDFUNDS
Unemployment	UNRATE	Trade Balance	BOPGSTB
Retail Sales	RSAFS	Housing Starts	HOUST
Capacity Utilization	TCU	Industrial Production	INDPRO
Nonfarm Payrolls	PAYEMS	PPI	PPIACO
Core PCE	PCEPILFE	New Orders Durable Goods	DGORDER
3-Month Treasury Yield	DTB3	Consumer Confidence Index	UMCSENT
New Home Sales	HSN1F	Business Inventories	BUSINV
Construction Spending	TTLCONS	Wholesale Inventories	WHLSLRIMSA
Personal Income	DSPIC96	Junk Bond Spread	BAA - AAA
Yield Spread (10Y - 3M)	T10Y3MM		

Indicators were checked for stationarity using the Augmented Dickey-Fuller (ADF) test. The function used is `ndiffs()` and the differencing orders for the indicators are stored in a variable `diff_orders`. Full function is provided in Appendix 13.

FRED suggests to difference `Unemployment` once even though it is stationary. `junk_bond_spread` and `yield_spread` are stationary, so they are excluded from the ADF test. These three indicators are not in `diff_orders`.

Table 2: Differencing Orders for Macroeconomic Indicators

Indicator	Order	Indicator	Order
CPI	1	Crude_Oil	1
Interest_Rate	1	Trade_Balance	1
Retail_Sales	1	Housing_Starts	1
Capacity_Utilization	0	Industrial_Production	0
Nonfarm_Payrolls	2	PPI	1
Core_PCE	1	New_Orders_Durable_Goods	1
Three_Month_Treasury_Yield	1	Consumer_Confidence_Index	1
New_Home_Sales	1	Business_Inventories	1
Construction_Spending	1	Wholesale_Inventories	1
Personal_Income	1	Unemployment	1
Junk_Bond_Spread	0	Yield_Spread	0

5.2 Data Used by Non-Linear Models

The indicators were then differenced according to Table 2. An indicator variable is also used to identify observations during Covid. These indicators, along with the target variable are packaged into a data frame, `tree_df.csv`, which will be used by the non-linear models.

5.3 Data Used by Linear Models

For the data frame that will be used by the linear models, `bridge_df.csv`, further data processing steps were carried out before running LASSO to reduce the list of indicators. In `important_indicators.R`, these data processing steps were carried out:

1. Monthly macroeconomic data spanning 30 years from the day LASSO was run (21 March 2025) was pulled from the FREDMD API and aggregated into quarterly frequency using indicator-specific rules. Stock variables are aggregated by taking their mean and flow variables are aggregated by taking the sum. For indicators that had "sum" as their aggregation rule, NA values were forward filled. These quarterly indicators are then joined with the GDP growth values according to year and quarter.

Table 3: Aggregation Rules for Monthly Indicators

Indicator	**Aggregation**	**Indicator**	**Aggregation**
CPI	mean	Crude_Oil	mean
Interest_Rate	mean	Unemployment	mean
Trade_Balance	sum	Retail_Sales	sum
Housing_Starts	sum	Capacity_Utilization	mean
Industrial_Production	mean	Nonfarm_Payrolls	sum
PPI	mean	Core_PCE	mean
New_Orders_Durable_Goods	sum	Three_Month_Treasury_Yield	mean
Consumer_Confidence_Index	mean	New_Home_Sales	sum
Business_Inventories	sum	Construction_Spending	sum
Wholesale_Inventories	sum	Personal_Income	mean
yield_spread	mean	junk_bond_spread	mean

2. Take only observations from 1995Q1 to 2019Q4 so that Covid fluctuations do not affect LASSO.
3. Difference the indicators according to `diff_orders` and scale indicators except for `gdp_growth`. Create 4 lags of each indicator including `gdp_growth` to capture any delayed effects of the indicators.
4. Run LASSO using the `hdm()` package in R, where the theoretical optimal is chosen automatically. Important indicators picked out by LASSO & economic intuition:

Table 4: Indicators selected by LASSO and economic intuition.

Indicators Chosen by LASSO		**Indicators Chosen by Intuition**	
Housing_Starts	New_Orders_Durable_Goods	gdp_growth_lag1 to gdp_growth_lag4	yield_spread
Nonfarm_Payrolls	Interest_Rate_lag1	Trade_Balance	yield_spread_lag1 to yield_spread_lag4
Construction_Spending	junk_bond_spread_lag1	Industrial_Production_lag1	
Trade_Balance_lag1	Unemployment	junk_bond_spread	
Industrial_Production_lag3	Capacity_Utilization	junk_bond_spread_lag2 to junk_bond_spread_lag4	

5. Data on these indicators were fetched again using another R file (`bridge_indicators.R`). Indicators were made stationary according to the table above, but not scaled. The processed data, along with the Covid Dummy indicator variable was stored in `bridge_df.csv`.

6 Modeling Approach

6.1 Forecasting of Monthly Indicators for Bridge Models

Many of the monthly indicators used in our nowcasting models are only partially available at the time of forecast. To maintain time consistency and simulate real-time conditions, we use an AR(p) model to forecast missing monthly values. Indicators are then aggregated into quarterly proxies with the appropriate aggregation rule (see Table 3 in Section 5.3).

The monthly indicators are forecasted and aggregated as follows:

$$X^{\text{quarterly}}_{\text{jbs},t} \quad \text{aggregated from} \quad \begin{cases} X^{\text{quarterly}}_{\text{jbs},t} \\ X^{\text{quarterly}}_{\text{jbs},t-\frac{1}{3}} \\ X^{\text{quarterly}}_{\text{jbs},t-\frac{2}{3}} \end{cases} \quad \text{where AR(p) model is} \quad X^{\text{monthly}}_{\text{jbs},m} = \varphi_0 + \sum_{s=1}^{p} \varphi_s \cdot X^{\text{monthly}}_{\text{jbs},m-s} + u_m$$

$$X^{\text{quarterly}}_{\text{hs},t} \quad \text{aggregated from} \quad \begin{cases} X^{\text{quarterly}}_{\text{hs},t} \\ X^{\text{quarterly}}_{\text{hs},t-\frac{1}{3}} \\ X^{\text{quarterly}}_{\text{hs},t-\frac{2}{3}} \end{cases} \quad \text{where AR(p) model is} \quad X^{\text{monthly}}_{\text{hs},m} = \varphi_0 + \sum_{s=1}^{p} \varphi_s \cdot X^{\text{monthly}}_{\text{hs},m-s} + u_m$$

These bridge-aggregated indicators are then passed as inputs to both the ADL Bridge and RF Bridge models.

Three Key Functions for Forecasting Monthly Bridge Indicators

1. `record_months_to_forecast` identifies months that need forecasting for each predictor variable based on missing data. It returns a dictionary of the months to forecast for each indicator. Full function is provided in Appendix 13.
2. `forecast_indicators` generates the iterated forecasts for monthly indicators if their data is unreleased. Pre-computed optimal lags (determined using BIC) are used. If new indicators are added to the model, the optimal lags for this indicator will be computed. Full function is provided in Appendix 13.
3. `forecast_aggregate_indicators` aggregates monthly economic indicators to quarterly frequency using variable-specific rules. Stock variables are averaged, while flow variables are summed. Full function is provided in Appendix 13.

6.2 AR Benchmark

For the benchmark model, we use an Autoregressive (AR) model with two lags to forecast quarterly GDP growth. This is a direct forecast model, without any bridge features, providing a simple, interpretable, and historically reliable baseline—especially during stable economic periods.

$$\text{AR(2) Model:} \quad GDPgrowth_t = \beta_0 + \beta_1 \cdot GDPgrowth_{t-2} + \beta_2 \cdot GDPgrowth_{t-3} + \epsilon_t$$

We deliberately begin the lag structure from $t-2$ instead of $t-1$ because the most recent GDP data (i.e., previous quarter) is often unavailable at the time of forecasting. This ensures that our benchmark model is time-consistent and avoids using data not yet released in real-time settings.

Full coefficient estimates are provided in Appendix 13.

6.3 ADL Bridge

The ADL Bridge model combines lagged GDP growth with a subset of bridge-aggregated monthly indicators (as described in Section 6.1), offering a balance between interpretability and timeliness for real-time GDP nowcasting using partially available high-frequency data. This framework was chosen because it incorporates both historical GDP trends and early-released indicators, produces interpretable coefficient estimates suitable for economic analysis, and uses aggregation methods that preserve the economic meaning of stock and flow variables. The growth rate is obtained through an iterated forecast, consistent with our real-time forecasting setup.

$$\text{ADL Model:} \quad GDPgrowth_t = \beta_0 + \sum_{i=1}^{k} \beta_i X_{ti} + \delta D_t + \epsilon_t$$

where X_{ti} is the i-th selected indicator at quarter t, and

D_t is a dummy variable that captures the COVID shock period.

We also included a COVID dummy variable to account for the extreme fluctuations observed during the pandemic period and to stabilize the model's estimates.

From the initial set of 10 predictors selected using Lasso (see Section 5.3, Table 4), we applied additional adjustments based on economic reasoning and empirical performance. Specifically, we iteratively modified the indicator set to minimize both the Bayesian Information Criterion (BIC) and the Root Mean Squared Forecast Error (RMSFE). This refinement process allowed us to arrive at a final set of 11 optimal predictors for the ADL Bridge model.

Table 5: Variable Selection Adjustments and Forecast Performance for ADL Bridge Model

Indicators Removed from Lasso-Selected Indicators	**Indicators Added**	**BIC**	**RMSFE**
–	Gdp_growth_lag1	524.1	1.72
Industrial_Production_lag3, Nonfarm_Payrolls	Industrial_Production_lag1, Gdp_growth_lag1	526.7	1.57
Industrial_Production_lag3, Interest_Rates_lag1, Nonfarm_Payrolls	Industrial_Production_lag1, Gdp_growth_lag1	523.1	1.65
Interest_Rates_lag1, Nonfarm_Payrolls	Junk_bond_spread, Gdp_growth_lag1	520.9	1.61
Industrial_Production_lag3, Interest_Rates_lag1, Nonfarm_Payrolls	Junk_bond_spread, Industrial_Production_lag1, Gdp_growth_lag1	**522.0**	**1.56**

Table 6: Final 11 Variables Used in the Bridge ADL Model

gdp_growth_lag1	Industrial_Production_lag1	New_Orders_Durable_Goods
Junk_bond_spread	Construction_Spending	Housing_Starts
Covid_Dummy_Variable	Unemployment	Trade_Balance_lag1
Capacity_Utilization	Junk_bond_spread_lag1	

Full coefficient estimates are provided in Appendix 13.

6.4 RF Benchmark

The RF benchmark model (`model_RF.py`) serves as the Random Forest benchmark for nowcasting quarterly U.S. GDP growth using lagged macroeconomic indicators.

1. Monthly macroeconomic indicators are aggregated to quarterly frequency (see Section 5.3).
2. Up to 6 lags of each indicator are created. We deliberately begin the lag structure from $t-2$ instead of $t-1$ because the most recent data (i.e., previous quarter) is often unavailable at the time of forecasting. This ensures that our benchmark model is time-consistent and avoids using data not yet released in real-time settings.
3. Two separate Random Forest models are trained:
   - Model 1 (current nowcast) uses lag 2 to lag 5 indicators.
   - Model 2 (1-step ahead forecast) uses lag 3 to lag 6 indicators.
4. Model 1 is applied to data on lag 2 to lag 5 of current quarter (time t) for current nowcast. Model 2 is applied to data on lag 3 to lag 6 of next quarter (time $t+1$) for 1-step ahead forecast.
5. The benchmark RF model uses the default RF settings and no hyperparameter tuning.

6.5 RF Bridge

The `model_RF_bridge` extends the logic of the ADL Bridge model by replacing the linear OLS estimator with a Random Forest regressor. It takes the same bridge-aggregated monthly inputs

(see Section 6.1), performs an iterated forecast of GDP growth, and is capable of capturing complex non-linear interactions between predictors and the target GDP growth rate.

1. Monthly macroeconomic indicators are aggregated to a quarterly frequency (using the rules in Section 5.3).
2. Four lags were created for each indicator. 4 lags were chosen as it strikes a good balance between being able to capture delayed effects of macroeconomic indicators on GDP growth, while avoiding the inclusion of information that may be too outdated to remain relevant for accurate nowcasting.
3. Recursive feature elimination cross-validation (RFECV) was applied to identify the most relevant features to forecast GDP growth. RFECV was combined with time series-aware cross validation using `TimeSeriesSplit`. This ensured the temporal ordering of the data was preserved, avoiding data leakage and enabling realistic forecast accuracy.
   RFECV automates the feature selection process by:
   - Ranking features by importance
   - Iteratively removing the least important ones
   - Selecting the combination of features that yields the best cross-validation performance

   The scoring metric used was negative mean squared error (MSE), aligning with our objective to minimize prediction error in real-time GDP nowcasting. This process returned the following:

Table 7: Top 7 Selected Features from RFECV

Unemployment	Capacity_Utilisation
Nonfarm_Payrolls	Housing_Starts_lag2
Nonfarm_Payrolls_lag1	New_Home_Sales_lag1
New_Orders_Durable_Goods	

4. The RF Bridge Model was trained using `GridSearchCV` to identify the optimal set of hyperparameters. This tuning process is essential to control model complexity, preventing both underfitting and overfitting, while enhancing prediction accuracy and ensuring robust and consistent nowcast performance across different forecast horizons. To ensure that the tuning remained appropriate for time series data, we employed time-series aware cross-validation within `GridSearchCV`, which preserved the temporal ordering of observations and prevented information leakage from future data points.
   The final hyperparameter configuration was:

Table 8: Optimal Hyperparameter Configuration for RF Bridge Model

Hyperparameter	**Value**
Number of Trees	400
Max Features	Square Root
Maximum Depth of Each Tree	10
Minimum Samples at a Leaf Node	4
Minimum Samples Needed to Split a Node	2

5. Final tuned model was saved and reused for nowcasting GDP growth in the most recent quarters
   - Eliminates the need to retrain the model each time new data is released

7 Forecast Evaluation

7.1 Algorithm for Evaluating Forecast Error

The procedure for evaluating forecast error uses a fixed rolling-window to maintain a consistent training sample size and enable fair comparison of forecast errors over time. It is applied as follows:

1. Use a fixed-size rolling window e.g. Jan 2000 to Sep 2017.
2. The forecast period is the last 2 quarters (or 6 months) of the window. In this example, Apr 2017 (Q2) to Sep 2017 (Q3). Our methodology assumes the forecast is made on the last day of each month. This means that the forecast made for forecast period starting Apr 2017 (Q2) uses data released up to 30th April 2017.
3. Remove data from the forecast period (the last 6 months) with the exception of indicators released in the same month as the forecast target (see Table 9).
4. Estimate the model on the truncated dataset to generate GDP growth forecasts for Apr 2017 (Q2) and Jul 2017 (Q3).
5. Slide the rolling window forward by one month and repeat the above steps to include newly released indicators. This produces updated flash estimates for May 2017 (Q2) and Aug 2017 (Q3).

Table 9: Indicators typically available by end of forecast month (e.g., end of April 2017). Only 5 most recent months of data are removed.

Indicator	**Typical Release Timing**
Industrial_Production_lag1	2nd/3rd week of current month
Housing_Starts_lag2	2nd/3rd week of current month
junk_bond_spread_lag1	1st week of current month
Nonfarm_Payrolls_lag1	1st week of current month
New_Home_Sales_lag1	3rd/4th week of current month

Table 10: Indicators typically unavailable by end of forecast month. All 6 months of data are removed. For those released '1st day 2 months later', we include them in this same group for simplicity and since the actual release date is closer to the end of the previous month than the end of the next month.

Indicator	**Typical Release Timing**
junk_bond_spread	1st week of next month
Trade_Balance_lag1	1st week of next month
Construction_Spending	1st day 2 months later
Housing_Starts	2nd/3rd week of next month
Capacity_Utilization	2nd/3rd week of current month
Unemployment	1st week of next month
New_Orders_Durable_Goods	1st day 2 months later

7.2 Model Performance

The table below summarizes the forecasting performance of the four models based on RMSFE, MAFE, and their accuracy in predicting the direction of growth and downturns. RMSFE and MAFE are reported for both the current quarter (Nowcast) and the subsequent quarter (Next Quarter Forecast). Pseudo out of sample forecasts from 2012:Q3 to 2024:Q3 are used unless otherwise stated.

Table 11: Forecast Performance Across Models

Model	**RMSFE**		**MAFE**		**Downturn Accuracy**	**Directional Accuracy**
	Nowcast	Next Quarter	Nowcast	Next Quarter		
AR Benchmark	7.50	7.50	3.39	3.33	9%	96%
ADL Bridge	4.45	7.77	2.17	3.27	45%	92%
RF Benchmark	7.81	7.62	2.87	2.94	27%	88%
RF Bridge	5.38	7.02	2.19	2.74	45%	93%

Note: RMSFE and MAFE are expressed as annualized growth rates (in %). Downturn Accuracy refers to correct predictions of negative GDP growth. Directional Accuracy is the proportion of forecasts matching the sign of actual GDP growth. For Downturn and Directional Accuracy, data from 2006:Q3 to 2024:Q3 is used.

Table 12: Forecast Accuracy by Model Excluding Covid Period

Model	**RMSFE**		**MAFE**	
	Nowcast	Next Quarter	Nowcast	Next Quarter
AR Benchmark	2.79	3.28	1.74	1.80
ADL Bridge	1.56	2.06	1.28	1.60
RF Benchmark	3.41	3.66	2.16	2.55
RF Bridge	1.46	1.71	1.08	1.17

Note: Forecasts for 2020 and 2021:Q1 are excluded from the calculations in this table due to significant deviations caused by the COVID-19 pandemic. These periods are removed to focus on forecast errors under typical economic conditions.

7.3 Diebold-Mariano Test

In `dm_test.py`, the `dm_test_hac_regression` function implements the Diebold-Mariano (DM) test to statistically compare the forecasting accuracy between benchmark models (AR, RF) and bridge models (ADL Bridge, RF Bridge).

The test evaluates whether the difference in predictive performance is significant by computing a loss differential series based on either Mean Squared Error (MSE) or Mean Absolute Difference (MAD). To ensure the validity of the test under potential autocorrelation and heteroskedasticity, HAC (Heteroskedasticity and Autocorrelation Consistent) standard errors are used to compute the test statistic.

Before the DM test, the DF-GLS test is run on the loss differential series to assess stationarity.

The `run_dm_test` function takes in `row_error.csv` from `forecast_evaluation.py` and returns a table of comparisons, DM statistics, p-values, and significance flags.

H_0: Both models have the same performance; H_1: Bridge model performs better

The loss function is defined as:

$$\begin{aligned} &\text{(MSE)} \quad d = (\text{error of benchmark model})^2 - (\text{error of bridge model})^2 \\ &\text{(MAD)} \quad d = |\text{error of benchmark model}| - |\text{error of bridge model}| \end{aligned}$$

The DM statistic is defined as:

$$DM = \frac{\bar{d}}{\hat{\sigma}_d/\sqrt{P}} \sim N(0,1)$$

where $\hat{\sigma}_d/\sqrt{P}$ is a HAC estimate of the standard deviation.

8 Challenges and Reflections

8.1 Challenges, Trade-offs and Limitations

1. Before running LASSO in `important_indicators.py`, indicators were made stationary using the ADF test in R for convenience. However, a better approach to this would be to study the economic indicators and manually determine the stationarity of the indicators.
2. Currently, `important_indicators.py` is running on fixed data. This is because we wanted a quick way to reduce the number of indicators. However, LASSO may identify other indicators in the future. Hence, a better approach will be to allow LASSO to run on live data and have a function that pulls the indicators identified by LASSO automatically.
3. ADL Bridge has an edge at predicting directional changes but is often over-sensitive, especially when making the first/second flash estimate of the quarter. An example is April 2020 when the actual gdp growth was -33% but our second flash estimate by ADL model was -407% (but a thankfully only 12/436 residuals for ADL bridge have absolute value greater than 5%). This is likely due to the linear nature of the model being unable to adapt to the sudden shocks during the pandemic. Furthermore, the predictions from the AR model could have amplified the shocks when predicting the unreleased data.
4. For the benchmark RF (`model_RF.py`) model, we wanted to implement a benchmark that required little to no tuning. Firstly, this may result in overfitting. Secondly, the tree is hard to interpret as no limit on the number of splits was implemented.
5. The RF Bridge Model (`model_RF_bridge.py`) uses a fixed list of selected features that were identified using RFECV during an initial training phase. Although this approach optimises runtime, it assumes that the predictive power of the selected features remains stable over time. However, in reality, the importance of different indicators can shift with changing economic conditions. Therefore, the hard-coded features may limit the model's adaptability to the changing economic conditions, potentially leading to reduced forecasting accuracy.

8.2 Future Work

1. **Research Optimal Retraining Schedule**
   - LASSO runs on fixed data and may identify more important indicators in the future
   - Coefficient estimates of the indicators may change in the future
2. **Exploring Ensemble Model**
   - Combining Bridge ADL model and RF Bridge model to leverage strengths of both models
3. **Higher Frequency Nowcasting**
   - Increase frequency of nowcast from monthly to daily/weekly

8.3 Team Contributions

Table 13: Summary of Backend Team Contributions

Team Member	**Contributions**
Cai Mun Jia	Data Processing, Model Development, Forecast Evaluation, Feature Selection
Jeron Tan Kang	Forecast Evaluation, Integration, Model Development
Kellianne Ng	Model Development, Feature Selection

Part III: Front-End Development

9 UI Structure and Interaction

9.1 Dashboard Layout

The dashboard is structured around three core interactive views accessible via a sidebar menu: Current Nowcast, Time Travel, and Model Comparison. Each view is implemented as a dedicated function in `dashboard_layout.py` and rendered based on user navigation input from the sidebar using `streamlit_option_menu`.

9.2 Input → Backend Pipeline

The dashboard follows a clear input-processing-output pipeline that ensures a seamless interaction between frontend components and backend model logic:

- **User Input Captures** - Inputs such as selected models, forecast dates, and confidence interval settings are captured through interactive widgets including `selectbox`, `multiselect`, `radio`, and `button`. These inputs are stored in `st.session_state` to persist user selections across re-runs.
- **Data Filtering and Forecast Generation** - Based on the user's selected forecast date, the relevant datasets (`bridge_df.csv, tree_df.csv`) and models (`model_AR.py, model_RF.py`) are then triggered through wrapper functions like `generate_oos_forecast`, `model_ADL_bridge`, and `model_RF_bridge`.
- **Performance and Caching** - To ensure responsiveness, the dashboard caches data using `@st.cache_data`, stores intermediate results in session state, and avoids redundant computation.

10 Design Rationale

10.1 Visual Decisions

In designing the nowcasting dashboard, several visual decisions were made to ensure clarity, usability, and effective communication of model outputs:

- **Line plots** were chosen over bar charts or tables to present GDP growth trends over time. This format allows users to easily identify patterns, turning points, and the direction of economic activity, which is especially important when tracking real-time changes.
- **Interactive tooltips** were implemented to allow users to hover over data points and view exact values. This provides precision without overwhelming the visual space with labels.
- A **vertical marker** was added to represent the current forecast quarter, making it visually clear where the transition from actual to forecast data begins. This helps orient users when interpreting the timeline.
- **Shaded areas** represent forecast uncertainty, providing a visual cue for model confidence and helping users understand the reliability of predictions over different time horizons. The shaded area in the Model Comparison tab highlights the impact of excluding COVID periods across models.

10.2 UI/UX Considerations

- **Progressive disclosure** Advanced information (e.g. individual model output, forecast intervals) is hidden behind toggles or collapsible panels so that new users are not overwhelmed at first glance.
- **Loading states and error messages** When model results are being fetched or compiled, a status message is displayed. If data is not available, the dashboard provides a clear and user-friendly explanation instead of going blank.

- **Minimal cognitive load** The interface is designed with clean layouts, intuitive controls, and minimal clutter to help users focus on key insights without being overwhelmed by excessive information or visual noise.
- **Consistent controls across tabs** Interactive elements such as drop-downs, sliders, and buttons are standardized across all dashboard views to ensure a familiar and predictable user experience, reducing the learning curve for new users.

11 Dashboard Implementation

11.1 GDP Nowcasting

The Current Nowcast tab estimates real GDP growth for the ongoing quarter using all currently available economic indicators. It simulates real-time conditions based on available economic data for the current quarter and forecasts of explanatory variables for the remaining term. Key components include:

1. A line chart showing nowcasted GDP growth estimates along with confidence intervals.
2. A model comparison table summarizing nowcast values (SAAR) for the current quarter across all models.
3. A variable-level forecast table displaying monthly release schedules for each model's predictors, highlighting how the forecast evolves throughout the quarter.

A key feature of the nowcast estimates is the inclusion of forecast intervals, which communicate the models' uncertainty around each prediction. These intervals are constructed using model-specific RMSFE values adjusted via the Cornish-Fisher expansion:

$$\text{Adjusted Quantile} = z + \frac{1}{6}(z^2 - 1)\gamma + \frac{1}{24}(z^3 - 3z)\kappa - \frac{1}{36}(2z^3 - 5z)\gamma^2$$

where z is the standard normal quantile, γ is skewness, and κ is excess kurtosis. This accounts for non-normal forecast error distributions when constructing asymmetric confidence bounds. Users can toggle between two forecast interval settings, *Exclude COVID Period* and *Normal (All Years)* to compare forecast uncertainty under typical versus volatile macroeconomic conditions.

11.2 Time Travel

The Time Travel tab enables users to simulate past forecast environments. It represents what a nowcast would have looked like at a selected point in time, based only on data available then. This supports historical backtesting and understanding model behavior during key economic events like the 2008 Global Financial Crisis and/or the COVID-19 recession.

11.3 Model Comparison

The Model Comparison tab provides a visual benchmark of each model's forecasting performance against actual GDP growth, based on the models selected by the user. It consists of three interactive plots:

1. Displays only the actual historical GDP series to establish a baseline trend.
2. Overlays predictions from the models selected by the user, alongside actual GDP-across the full sample, including the volatile COVID-19 period.
3. Mirrors the 2nd but omits data from COVID-19 quarters to isolate the models' performance under normal economic conditions.

12 Limitations and Reflections

12.1 Known Issues

- **Limited mobile usability:** While the layout is responsive, the dashboard is currently optimized for desktop users. On smaller screens, plot labels and controls may be cramped or

hard to interact with.

- **No forecast customization yet** (e.g., simulate scenarios) : Model parameters and lag structures are preset. Users cannot adjust assumptions (e.g., change lag length, drop variables) directly from the frontend, limiting experimentation or scenario analysis.
- **No user personalization or bookmarking:** The dashboard does not save user settings (e.g., selected model, date range) across sessions. Each visit resets to default views, which may disrupt workflow for returning users.

12.2 Future Work

- **Enable user-driven model customization:** Future iterations of the dashboard could allow users to adjust model assumptions—such as selecting lag lengths, or changing forecast horizons—directly from the frontend. Users can also perform variable selection, and tuning of hyperparameters in the random forest model. This would support more flexible experimentation and enable users to conduct scenario analysis tailored to specific economic questions.
- **Integrate macroeconomic sentiment indicators via NLP:** Future versions of the dashboard could include an interactive sentiment index derived from macroeconomic news using Natural Language Processing (NLP). This would allow users to visualize non-quantitative shocks—such as political unrest, climate events, or geopolitical tensions—alongside traditional economic indicators, enriching the interpretability of forecasts during turbulent periods.

12.3 Team Contributions

Team Member	**Contributions**
N Ashwin Kumar	UI/UX Design, Data Processing and Visualization
Tammy Alexandra Wong	UI/UX Design, Data Processing and Visualization

Table 14: Summary of Frontend Team Contributions

13 Appendix

A.1 Function to Difference Non-Stationary Indicators

The function below determines the number of differences needed to stationarize each macroeconomic indicator using the Augmented Dickey-Fuller (ADF) test. It stores the differencing orders and applies them individually to each indicator in the dataset.

```
# Create an empty list to store differencing orders
diff_orders <- c()

# Making data stationary (ndiffs uses KPSS test by default)
# Loop through each column excluding date
for (col in colnames(df_not_stationary)) {
  if (col != "year_quarter") {

    # Find the number of differences needed for each indicator
    order <- ndiffs(df_not_stationary[[col]], test = "adf")  # use adf test

    # Store the differencing order
    diff_orders[col] <- order
  }
}

# Create a copy of the dataset
data_stationary <- df_not_stationary

# Apply differencing column by column
for (col in colnames(df_not_stationary)) {
  if (col != "year_quarter" && col %in% names(diff_orders)) {  # Skip date column

    # Get the differencing order for this column
    order <- diff_orders[col]

    # Apply differencing only if needed
    if (order > 0) {
      data_stationary[[col]] <- c(rep(NA, order), diff(df_not_stationary[[col]],
          differences = order))
    }
  }
}
```

Listing 1: Appendix A.2: R function to compute and apply differencing orders using the ADF test

A.2 Function that Identifies Months to Forecast

The function below identifies months that need forecasting for each predictor variable based on missing data. It returns a dictionary of the months to forecast for each indicator.

```
def record_months_to_forecast(df, predictors):
    """
    Identifies months that need forecasting for each predictor variable based on
        missing data.

    This function checks for missing data in the time series for each predictor
        variable and identifies
    which months need to be forecasted. The months are determined by finding the
        last known date
    for each predictor and then identifying the subsequent missing months.

    Args:
    - df (pd.DataFrame): DataFrame with dates as index and predictors as columns.
    - predictors (list): List of predictor variable names to check for missing
        data.
```

```
    Returns:
    - dict: A dictionary where the keys are predictor names and values are lists
        of months with
            missing data that need forecasting.
    """
    if not isinstance(df.index, pd.DatetimeIndex):
        df.index = pd.to_datetime(df.index)

    months_to_forecast = {}

    for col in predictors:
        months_to_forecast[col] = []

        last_known_index = df[col].dropna().index.max() if df[col].dropna().size >
            0 else None

        if last_known_index:
            last_known_date = pd.to_datetime(last_known_index)
            next_date = last_known_date + pd.DateOffset(months=1)

            while next_date in df.index and pd.isna(df.loc[next_date, col]):
                months_to_forecast[col].append(next_date)
                next_date += pd.DateOffset(months=1)

    return months_to_forecast
```

Listing 2: Appendix A.1: Function to identify months to forecast

A.3 Function that Forecasts Missing Monthly Indicators

The function below generates the iterated forecasts for monthly indicators if their data is unreleased. Pre-computed optimal lags (determined using BIC) are used. If new indicators are added to the model, the optimal lags for this indicator will be computed.

```
def forecast_indicators(df,
    exclude=["date", "GDP", "gdp_growth", "gdp_growth_lag1", "gdp_growth_lag2", "
        gdp_growth_lag3", "gdp_growth_lag4"],
    lag_dict={
        'Capacity_Utilization': 4,
        'Construction_Spending': 4,
        'Housing_Starts': 4,
        'Industrial_Production_lag1': 4,
        'Industrial_Production_lag3': 4,
        'Interest_Rate_lag1': 3,
        'New_Orders_Durable_Goods': 4,
        'Nonfarm_Payrolls': 4,
        'Trade_Balance': 4,
        'Trade_Balance_lag1': 4,
        'Unemployment': 4,
        'junk_bond_spread': 2,
        'junk_bond_spread_lag1': 2,
        'junk_bond_spread_lag2': 2,
        'junk_bond_spread_lag3': 2,
        'junk_bond_spread_lag4': 2,
        'yield_spread': 4,
        'yield_spread_lag1': 4,
        'yield_spread_lag2': 4,
        'yield_spread_lag3': 4,
        'yield_spread_lag4': 4,
        'Business_Inventories': 4,
        'CPI': 4,
        'Consumer_Confidence_Index': 4,
```

```
        'Core_PCE': 4,
        'Crude_Oil': 4,
        'Industrial_Production': 4,
        'Interest_Rate': 3,
        'New_Home_Sales': 4,
        'PPI': 4,
        'Personal_Income': 4,
        'Retail_Sales': 4,
        'Three_Month_Treasury_Yield': 4,
        'Wholesale_Inventories': 4
    }):

    """
    Forecasts missing values for predictor variables using AutoRegressive models.
    Uses provided lags if lag_dict is given; otherwise, selects optimal lags using
        AIC.
    """
    df = df.sort_values(by="date").reset_index(drop=True).set_index("date")
    predictors = df.columns.difference(exclude)
    months_to_forecast = record_months_to_forecast(df, predictors)
    final_lag_dict = {} if lag_dict is None else lag_dict.copy()

    for col in predictors:
        if col == "dummy":
            df[col].fillna(0, inplace=True)
            continue

        if col in months_to_forecast and months_to_forecast[col]:
            for forecast_date in months_to_forecast[col]:
                data_before_forecast = df.loc[df.index <= forecast_date, col].
                    dropna()

                if len(data_before_forecast) < 10:
                    continue

                try:
                    if col in final_lag_dict:
                        best_lag = final_lag_dict[col]
                    else:
                        best_aic, best_lag, max_lags = float("inf"), 1, 4
                        for lag in range(1, max_lags + 1):
                            try:
                                model = AutoReg(data_before_forecast, lags=lag,
                                    old_names=False).fit()
                                if model.aic < best_aic:
                                    best_aic, best_lag = model.aic, lag
                            except:
                                continue
                        final_lag_dict[col] = best_lag

                    model = AutoReg(data_before_forecast, lags=best_lag, old_names
                        =False).fit()
                    df.loc[forecast_date, col] = model.predict(
                        start=len(data_before_forecast), end=len(
                            data_before_forecast)
                    ).iloc[0]

                except Exception:
                    pass

    df = df.reset_index().sort_values(by="date").reset_index(drop=True)
    return df
```

Listing 3: Appendix A.2: Function to forecast missing values

A.4 Function to Aggregate Monthly Indicators to Quarterly

The `aggregate_indicators` function transforms monthly economic indicators into quarterly frequency using variable-specific aggregation rules. Flow variables (e.g., counts and totals) are summed, while stock variables (e.g., rates and percentages) are averaged. GDP is extracted as the last available value each quarter to preserve its official release.

```
def aggregate_indicators(df):
    """
    Aggregates monthly frequency indicators to quarterly frequency.

    The function takes in a DataFrame with monthly frequency indicators and GDP,
        and converts them to quarterly frequency using
    predefined aggregation rules for each column. GDP remains unchanged, taking
        the only available value per quarter.

    Args:
        df (pd.DataFrame): DataFrame with monthly frequency data, including 'GDP'
            and other economic indicators.

    Returns:
        pd.DataFrame: DataFrame with quarterly frequency data, including
            aggregated indicators and GDP.
    """


    df = df.set_index('date')

    aggregation_rule = {
        "GDP": "sum",  # GDP should be aggregated using mean
        "gdp_growth": "sum",  # GDP growth, as a rate, should be averaged
        "gdp_growth_lag1": "sum",  # Lagged GDP growth, average
        "gdp_growth_lag2": "sum",  # Lagged GDP growth, average
        "gdp_growth_lag3": "sum",  # Lagged GDP growth, average
        "gdp_growth_lag4": "sum",  # Lagged GDP growth, average
        "Nonfarm_Payrolls": "sum",  # Sum of non-farm payrolls
        "Nonfarm_Payrolls_lag1": "sum", # Non-farm payrolls, lag 1(sum)
        "Construction_Spending": "sum",  # Sum of construction spending
        "Trade_Balance_lag1": "sum",  # Trade balance, lag1 (sum)
        "Industrial_Production_lag1": "mean",  # Industrial production, lag1 (
            average)
        "Industrial_Production_lag3": "mean",  # Industrial production, lag3 (
            average)
        "New_Home_Sales_lag1": "sum", # New Home Sales, lag1, sum
        "Housing_Starts": "sum",  # Housing starts, sum
        "Housing_Starts_lag2": "sum", # Housing starts, lag 2 (sum)
        "Capacity_Utilization": "mean",  # Capacity utilization, average
        "New_Orders_Durable_Goods": "sum",  # New orders for durable goods, sum
        "Interest_Rate_lag1": "mean",  # Interest rate, lag1 (average)
        "Unemployment": "mean",  # Unemployment rate, average
        "junk_bond_spread": "mean",  # Junk bond spread, average
        "junk_bond_spread_lag1": "mean",  # Junk bond spread, lag1 (average)
        "junk_bond_spread_lag2": "mean",  # Junk bond spread, lag2 (average)
        "junk_bond_spread_lag3": "mean",  # Junk bond spread, lag3 (average)
        "junk_bond_spread_lag4": "mean",  # Junk bond spread, lag4 (average)
        "dummy": "mean"  # Dummy variable, sum
    }

    # Extract the last available GDP value each quarter
    gdp_data = df[['GDP']].resample('QS').last()

    indicators_data = pd.DataFrame()

    # Remove columns that should not be aggregated
    if "date" in df.columns:
        df_indicators = df.drop(columns=["date", "GDP"], errors="ignore")
```

```
    else:
        df_indicators = df.drop(columns=["GDP"], errors="ignore")

    for col in df_indicators.columns:
        if col in aggregation_rule:
            method = aggregation_rule[col]
            if method == "mean":
                indicators_data[col] = df_indicators[col].resample('QS').mean()
            elif method == "sum":
                indicators_data[col] = df_indicators[col].resample('QS').sum()
        else:
            # Default to mean if there are columns not listed in aggregation_rule
            indicators_data[col] = df_indicators[col].resample('QS').mean()

    quarterly_df = gdp_data.merge(indicators_data, left_index=True, right_index=
        True, how='left')
    quarterly_df = quarterly_df.reset_index()
    quarterly_df["date"] = pd.to_datetime(quarterly_df["date"], format='%Y-%m')

    # Run line below to check output
    #print("output from aggregate_indicators", quarterly_df)
    return quarterly_df
```

Listing 4: Appendix A.2: Function to forecast missing values

A.5 AR Model Coefficient Estimates

The AR model is designed to be dynamically re-estimated as new data becomes available. The coefficient estimates reported below are obtained from a historical estimation using quarterly data from **1995Q2 to 2024Q4** (*N = 119 observations*):

- $\hat{\beta}_0$ (Intercept): 2.475
- $\hat{\beta}_1$ (Lag 2): -0.0176
- $\hat{\beta}_2$ (Lag 3): -0.0005

A.6 ADL Bridge Model Coefficient Estimates

The ADL model is designed to be dynamically re-estimated as new data becomes available. The coefficient estimates reported below are obtained from a historical estimation using quarterly data from **1995Q2 to 2024Q4** (*N = 119 observations*):

- $\hat{\beta}_{\text{gdp\_growth\_lag1}}$: -0.1834
- $\hat{\beta}_{\text{Construction\_Spending}}$: 3.185×10^{-6}
- $\hat{\beta}_{\text{Trade\_Balance\_lag1}}$: 6.58×10^{-5}
- $\hat{\beta}_{\text{Industrial\_Production\_lag1}}$: -0.0408
- $\hat{\beta}_{\text{Housing\_Starts}}$: 0.0026
- $\hat{\beta}_{\text{Capacity\_Utilization}}$: 0.0928
- $\hat{\beta}_{\text{New\_Orders\_Durable\_Goods}}$: 3.183×10^{-5}

- $\hat{\beta}_{\text{Unemployment}}$: -15.7340
- $\hat{\beta}_{\text{junk\_bond\_spread}}$: -5.3268
- $\hat{\beta}_{\text{junk\_bond\_spread\_lag1}}$: 4.6373
- $\hat{\beta}_{\text{dummy}}$: 1.9781