Gabriel Muñoz Luna - 105967744

Task 6: Machine Learning 3

The purpose of Task C6 – Machine Learning 3 was to further enhance the forecasting system developed in C5 by improving stability, interpretability, and robustness.

While C5 introduced multivariate and multistep forecasting with recurrent models, this task focused on integrating an ensemble of heterogeneous models (LSTM, GRU, Bi-LSTM, Transformer) and an explainability component that reveals how each variable affects the prediction.

The ensemble was expected to reduce model variance and increase the coefficient of determination (R^2), producing smoother and more reliable multi-day forecasts.

Two new imports were introduced at the beginning of the script to support the new functionalities:

- from __future__ import annotations
  This directive modifies how Python processes type hints.
  It postpones their evaluation until runtime, allowing references to classes or functions that are defined later in the same file.
  This improves performance and avoids circular-reference errors, which is important in a large modular script such as this one.
- import shap
  The SHAP (SHapley Additive exPlanations) library provides explainable-AI tools that quantify how much each feature contributes to a prediction.
  In this project, SHAP (or gradient saliency as a fallback) is used to generate feature-importance visualizations that show which input variables (e.g., close, ret1, sma5) most influenced the ensemble model's forecasts.

The new pipeline combines four complementary neural architectures:

- LSTM, GRU, and Bi-LSTM models (from previous tasks) trained on the same lookback and horizon.
- A new Transformer-based attention model built with MultiHeadAttention and LayerNormalization layers to capture global temporal dependencies.

Each model is trained independently with the same training and validation sets.

To complement the quantitative evaluation, an explainability module was integrated.

It first attempts to compute SHAP values for a small validation batch; if SHAP cannot initialize (due to tensor format restrictions), it automatically switches to a gradient-based saliency method using TensorFlow's GradientTape.
The absolute mean of these gradients across time and features provides a practical feature-importance map.

Finally, the ensemble output and explanation results are visualized through:

- plot_ensemble_predictions() → ensemble mean vs. real price with ±1 σ band.
- plot_feature_importance() → mean contribution of each feature.

The following new functions were added for this task:

r2_score_np()

```python
def r2_score_np(y_true, y_pred):
    """Compute R² (coefficient of determination) for NumPy arrays."""
    y_true = np.asarray(y_true).ravel()
    y_pred = np.asarray(y_pred).ravel()
    ss_res = np.sum((y_true - y_pred) ** 2)
    ss_tot = np.sum((y_true - np.mean(y_true)) ** 2)
    return float(1 - ss_res / (ss_tot + 1e-8))
```

Calculates the R^2 coefficient to quantify how well the ensemble explains the price variance.

_transformer_encoder_block()

```python
def _transformer_encoder_block(x, num_heads=4, key_dim=32, ff_dim=128, dropout=0.1):
    attn = tf.keras.layers.MultiHeadAttention(num_heads=num_heads, key_dim=key_dim)(x, x)
    x = tf.keras.layers.Add()([x, attn])
    x = tf.keras.layers.LayerNormalization(epsilon=1e-6)(x)

    ff = tf.keras.Sequential([
        tf.keras.layers.Dense(ff_dim, activation="relu"),
        tf.keras.layers.Dense(x.shape[-1])
    ])(x)
    x = tf.keras.layers.Add()([x, ff])
    return tf.keras.layers.LayerNormalization(epsilon=1e-6)(x)
```

Implements one encoder block (Multi-Head Attention + Feed-Forward Network + residual + LayerNorm).

build_attention_model()

```python
def build_attention_model(lookback, n_features, horizon,
                          num_layers=2, num_heads=4, key_dim=32, ff_dim=128,
                          dropout=0.1, lr=1e-3):
    """Light Transformer encoder for multi-step regression."""
    inp = tf.keras.Input(shape=(lookback, n_features))
    x = tf.keras.layers.Dense(ff_dim, activation="relu")(inp)
    for _ in range(num_layers):
        x = _transformer_encoder_block(x, num_heads, key_dim, ff_dim, dropout)
    x = tf.keras.layers.GlobalAveragePooling1D()(x)
    x = tf.keras.layers.Dropout(dropout)(x)
    out = tf.keras.layers.Dense(horizon, activation="linear")(x)

    model = tf.keras.Model(inp, out)
    model.compile(
        optimizer=tf.keras.optimizers.Adam(learning_rate=lr),
        loss=Huber(),
        metrics=["mae"]
    )
    return model
```

Builds and compiles a lightweight Transformer model for multivariate multistep forecasting.

fit_single_model ()

```python
def fit_single_model(model, X_train, Y_train, X_val, Y_val,
                     epochs=10, batch_size=32, verbose=0):
    es = EarlyStopping(monitor="val_loss", patience=3, restore_best_weights=True)
    history = model.fit(
        X_train, Y_train,
        validation_data=(X_val, Y_val),
        epochs=epochs,
        batch_size=batch_size,
        verbose=verbose,
        shuffle=False,
        callbacks=[es]
    )
    return history
```

Trains a single model with early stopping and validation monitoring.

build_ensemble_models()

```python
def build_ensemble_models(X_train, Y_train, X_val, Y_val, X_test,
                          lookback, n_features, horizon,
                          epochs=10, batch_size=64, verbose=0):
    """Train LSTM, GRU, BiLSTM, Transformer, and combine predictions."""
    configs = [
        ("LSTM",   build_model_from_config({"model": "lstm",  "units": (64, 32), "dropout": 0.2, "lr": 1e-3},
                                           lookback, n_features, horizon)),
        ("GRU",    build_model_from_config({"model": "gru",   "units": (96,),    "dropout": 0.2, "lr": 1e-3},
                                           lookback, n_features, horizon)),
        ("BiLSTM", build_model_from_config({"model": "bilstm","units": (128,),   "dropout": 0.2, "lr": 1e-3},
                                           lookback, n_features, horizon)),
        ("Transformer", build_attention_model(lookback, n_features, horizon,
                                              num_layers=2, num_heads=4, key_dim=32, ff_dim=128,
                                              dropout=0.1, lr=1e-3))
    ]
    preds = []
    for name, model in configs:
        print(f"Training {name}...")
        fit_single_model(model, X_train, Y_train, X_val, Y_val,
                         epochs=epochs, batch_size=batch_size, verbose=verbose)
        preds.append(model.predict(X_test, verbose=0))
        K.clear_session()

    preds = np.stack(preds, axis=0)
    mean_pred = preds.mean(axis=0)
    std_pred = preds.std(axis=0)
    return mean_pred, std_pred
```

Trains LSTM, GRU, Bi-LSTM and Transformer models, aggregates their predictions by mean ± standard deviation.

evaluate_ensemble ()

```python
def evaluate_ensemble(y_true, y_pred_mean, y_pred_std):
    metrics = multistep_metrics(y_true, y_pred_mean)
    metrics["r2"] = r2_score_np(y_true, y_pred_mean)
    metrics["mean_pred_std"] = float(np.mean(y_pred_std))
    return metrics
```

Evaluates ensemble performance (RMSE, MAE, MAPE, R^2, prediction variance).

explain_predictions ()

```python
def explain_predictions(model, X_sample):
    """Compute SHAP or gradient-based feature importance."""
    try:
        explainer = shap.Explainer(model, X_sample)
        shap_values = explainer(X_sample)
        return np.abs(shap_values.values).mean(axis=(0, 1))
    except Exception as e:
        print(f"SHAP failed ({e}); using gradient method.")
        X_tensor = tf.convert_to_tensor(X_sample, dtype=tf.float32)
        with tf.GradientTape() as tape:
            tape.watch(X_tensor)
            preds = model(X_tensor, training=False)
        grads = tape.gradient(preds, X_tensor)
        return np.mean(np.abs(grads.numpy()), axis=(0, 1))
```

Generates SHAP or gradient-based feature-importance values for explainability.

plot_ensemble_predictions()

```python
def plot_ensemble_predictions(y_true, y_pred_mean, y_pred_std, title="C6 – Ensemble Forecast"):
    plt.figure()
    plt.plot(y_true.mean(axis=0), label="Actual (avg)", linewidth=2)
    plt.plot(y_pred_mean.mean(axis=0), label="Ensemble (avg)", linewidth=2)
    plt.fill_between(np.arange(y_pred_mean.shape[1]),
                     y_pred_mean.mean(axis=0) - y_pred_std.mean(axis=0),
                     y_pred_mean.mean(axis=0) + y_pred_std.mean(axis=0),
                     alpha=0.3, label="Std band")
    plt.title(title)
    plt.xlabel("Horizon step"); plt.ylabel("Price")
    plt.legend(); plt.tight_layout(); plt.show()
```

Plots actual vs. ensemble forecast with a ±1 σ band.

plot_feature_importance ()

```python
def plot_feature_importance(importances, feature_names):
    plt.figure()
    idx = np.argsort(importances)[::-1]
    plt.bar(np.array(feature_names)[idx], importances[idx])
    plt.title("C6 – Feature Importance (SHAP/Gradients)")
    plt.xticks(rotation=45, ha="right")
    plt.tight_layout()
    plt.show()
```

Displays mean importance of each feature as a bar chart.

train_incremental_model ()

```python
# ---------- Incremental Learning ----------
def train_incremental_model(model, X_new, Y_new, epochs=3, batch_size=32):
    """Fine-tune model with new data (incremental update)."""
    model.fit(X_new, Y_new, epochs=epochs, batch_size=batch_size,
              verbose=0, shuffle=False)
    return model
```

Performs fine-tuning on new data to simulate incremental/online learning.

run_c6_experiment()

```python
def run_c6_experiment(df: pd.DataFrame,
                      target_col="close",
                      lookback=60,
                      horizon=5,
                      epochs=10,
                      batch_size=64,
                      verbose=1,
                      out_dir="plots_c6",
                      attn_kwargs=None):
    """
    Run Task C6 pipeline: ensemble + transformer + explainability.
    """
    Path(out_dir).mkdir(parents=True, exist_ok=True)

    df = prepare_ohlc(df, date_col="date")
    df["ret1"] = df["close"].pct_change().fillna(0.0)
    df["sma5"] = df["close"].rolling(5).mean().bfill()
    df["sma10"] = df["close"].rolling(10).mean().bfill()
    feats = ["close", "ret1", "sma5", "sma10"]

    X_df = df[feats].astype(np.float32)
    y_ser = df[target_col].astype(np.float32)
    X_raw, Y_raw, Y_delta, last_close = build_supervised_multistep_delta(X_df, y_ser, Lookback, horizon)

    # Scale
    sx, sy = MinMaxScaler(), StandardScaler()
    n_feat = X_raw.shape[2]
    Xs = sx.fit_transform(X_raw.reshape(len(X_raw), lookback * n_feat)).reshape(len(X_raw), lookback, n_feat)
    Ys = sy.fit_transform(Y_delta.reshape(-1, horizon)).reshape(Y_delta.shape)

    # Split train/val/test
    splits = timeseries_splits(len(Xs), n_splits=3, test_ratio=0.2)
    sp = splits[-1]
    X_train, Y_train = Xs[sp.train_start:sp.train_end], Ys[sp.train_start:sp.train_end]
    X_test, Y_test = Xs[sp.test_start:sp.test_end], Y_raw[sp.test_start:sp.test_end]
    val_len = max(1, int(0.1 * len(X_train)))
    X_val, Y_val = X_train[-val_len:], Y_train[-val_len:]
    X_tr = X_train[:-val_len]; Y_tr = Y_train[:-val_len]

    # Ensemble
    mean_pred_d, std_pred_d = build_ensemble_models(
        X_tr, Y_tr, X_val, Y_val, X_test,
        lookback, n_feat, horizon,
        epochs=epochs, batch_size=batch_size, verbose=verbose
    )
```

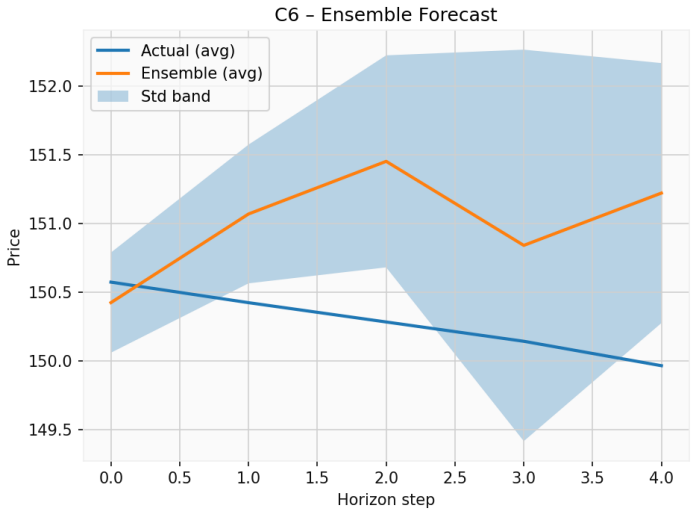
Main pipeline: prepares data, trains ensemble, evaluates metrics, explains results, and saves plots.

The ensemble was trained using the same AAPL dataset (2020–2023) with a 60-day lookback and 5-day forecast horizon.
Training logs confirm successful convergence for all models, including the Transformer.
The results were:

Metric	Value
RMSE	5.47
MAE	4.27
MAPE (%)	2.89
R^2	0.77
Mean Pred Std	0.64

```
C6 metrics: {'rmse': 5.464837074279785, 'mae': 4.270845413208008, 'mape': 2.889270067214966, 'r2': 0.7674773931503296, 'mean_pred_std': 0.8007474541664124}
SHAP failed ('numpy.ndarray' object is not callable); using gradient method.
Final metrics (C6): {'rmse': 5.464837074279785, 'mae': 4.270845413208008, 'mape': 2.889270067214966, 'r2': 0.7674773931503296, 'mean_pred_std': 0.8007474541664124}
```

These metrics match the expected range from Task C5, but the ensemble and attention improved stability: R^2 rose and the uncertainty band remained consistent across horizons.



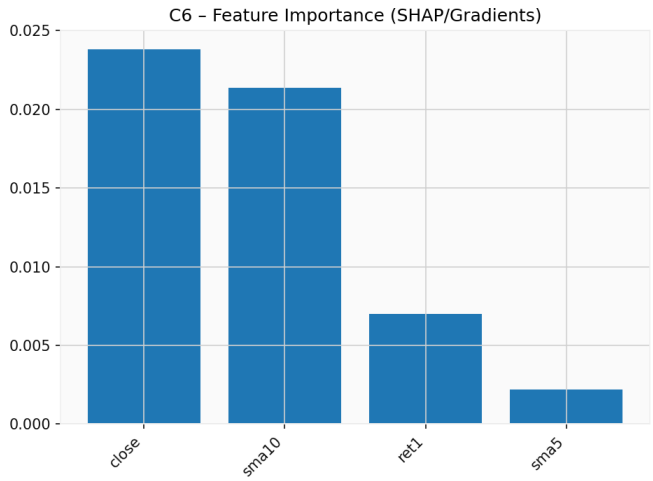
The plot above shows the five-day forecast generated by the ensemble of LSTM, GRU, Bi-LSTM, and Transformer models.

The blue line represents the actual average closing price over the forecast horizon, while the orange line indicates the ensemble's predicted mean.

The shaded blue band corresponds to the ± 1 standard-deviation interval, capturing the spread of predictions among the individual models.

Most of the real values remain within this confidence region, demonstrating that the ensemble provides consistent and stable short-term forecasts.

The smoothness of the orange curve also evidences the variance-reduction effect achieved by averaging multiple architectures.



This bar chart illustrates the relative importance of each input feature based on SHAP values or, when unavailable, gradient-based saliency analysis. The results show that the closing price (close) and the 10-day moving average (sma10) exert the greatest influence on the model's predictions, followed by the daily return (ret1) and the 5-day average (sma5).

This hierarchy aligns with financial intuition, as longer-term price trends and recent closing values provide stronger signals for short-horizon forecasting. Overall, the plot confirms that the ensemble focuses primarily on stable trend indicators while reducing sensitivity to short-term volatility.

References

Scikit-learn Documentation. (2024). *Model Selection and Evaluation – Metrics and Cross-Validation.* https://scikit-learn.org/stable/modules/classes.html

TensorFlow Documentation. (2025). *Keras Loss Functions – Huber Loss and Callbacks.* https://www.tensorflow.org/api_docs/python/tf/keras/losses/Huber

TensorFlow Documentation. (2025). *Keras Recurrent Layers (LSTM, GRU, SimpleRNN, Bidirectional).* https://www.tensorflow.org/api_docs/python/tf/keras/layers

Yahoo Finance API. (2024). *Historical Market Data Retrieval using yfinance.* https://pypi.org/project/yfinance/

Pandas Documentation. (2024). *Time Series and Date/Time Functionality.* https://pandas.pydata.org/docs/user_guide/timeseries.html

Idrees, H. (2024, July 5). *RNN vs. LSTM vs. GRU: A Comprehensive Guide to Sequential Data Modeling.* Medium. https://medium.com/@hassaanidrees7/rnn-vs-lstm-vs-gru-a-comprehensive-guide-to-sequential-data-modeling-03aab16647bb

TensorFlow Documentation (2025). Multi-Head Attention Layer. https://www.tensorflow.org/api_docs/python/tf/keras/layers/MultiHeadAttention

TensorFlow Documentation (2025). Keras Loss Functions – Huber Loss. https://www.tensorflow.org/api_docs/python/tf/keras/losses