SCHOLARLY REVIEW JOURNAL

E-ISSN: 2996-8380
Published by Leadership & Innovation Lab

SUMMER 2026

APPENDIX 4

ML model code

Retrieving weather data from OpenMeteo API

```
# ---- FETCHING OPENMETEO API DATA ----

import pickle
import os
from openmeteo_requests import Client
import openmeteo_requests
import numpy
import pandas as pd

def get_hourly(cache_file='openmeteo_cache.pkl', force_refresh=False):
    """
    Get hourly data with caching to avoid hitting API limits
    """
    # Check if cache exists and we're not forcing refresh
    if os.path.exists(cache_file) and not force_refresh:
        print(f"Loading data from cache: {cache_file}")
        with open(cache_file, 'rb') as f:
            return pickle.load(f)

    # Make API call
    print("Fetching data from Open-Meteo API...")
```

SCHOLARLY REVIEW

SCHOLARLY REVIEW JOURNAL

E-ISSN: 2996-8380
Published by Leadership & Innovation Lab

SUMMER 2026

```
    openmeteo = openmeteo_requests.Client()


    url = "https://archive-api.open-meteo.com/v1/archive"
    params = {
        "latitude": 1.3667,
        "longitude": 103.8,
        "start_date": "1970-01-01",
        "end_date": "2025-11-31",
        "hourly": [
     "temperature_2m", "relative_humidity_2m", "dew_point_2m", "rain",
     "cloud_cover", "cloud_cover_low", "cloud_cover_mid",
"cloud_cover_high",
     "vapour_pressure_deficit", "pressure_msl", "surface_pressure",
"wind_speed_10m", "wind_speed_100m"
     ],
        "timezone": "Asia/Singapore",
    }


    responses = openmeteo.weather_api(url, params=params)
    response = responses[0]


    hourly = response.Hourly()
    hourly_temperature_2m = hourly.Variables(0).ValuesAsNumpy()
    hourly_relative_humidity_2m = hourly.Variables(1).ValuesAsNumpy()
    hourly_dew_point_2m = hourly.Variables(2).ValuesAsNumpy()
    hourly_rain = hourly.Variables(3).ValuesAsNumpy()
```

SCHOLARLY REVIEW JOURNAL

E-ISSN: 2996-8380
Published by Leadership & Innovation Lab

SUMMER 2026

```
    hourly_cloud_cover = hourly.Variables(4).ValuesAsNumpy()
    hourly_cloud_cover_low = hourly.Variables(5).ValuesAsNumpy()
    hourly_cloud_cover_mid = hourly.Variables(6).ValuesAsNumpy()
    hourly_cloud_cover_high = hourly.Variables(7).ValuesAsNumpy()
    hourly_vapour_pressure_deficit =
hourly.Variables(8).ValuesAsNumpy()
    hourly_pressure_msl = hourly.Variables(9).ValuesAsNumpy()
    hourly_surface_pressure = hourly.Variables(10).ValuesAsNumpy()
    hourly_wind_speed_10m = hourly.Variables(11).ValuesAsNumpy()
    hourly_wind_speed_100m = hourly.Variables(12).ValuesAsNumpy()

    # Process response into DataFrame
    import pandas as pd

    hourly_data = {
        "date": pd.date_range(
            start=pd.to_datetime(hourly.Time(), unit="s", utc=True),
            end=pd.to_datetime(hourly.TimeEnd(), unit="s", utc=True),
            freq=pd.Timedelta(seconds=hourly.Interval()),
            inclusive="left"
        )
    }

    hourly = response.Hourly()
    hourly_data["temperature_2m"] = hourly_temperature_2m
    hourly_data["relative_humidity_2m"] = hourly_relative_humidity_2m
```

SCHOLARLY REVIEW JOURNAL

E-ISSN: 2996-8380
Published by Leadership & Innovation Lab

SUMMER 2026

```
    hourly_data["dew_point_2m"] = hourly_dew_point_2m
    hourly_data["rain"] = hourly_rain
    hourly_data["cloud_cover"] = hourly_cloud_cover
    hourly_data["cloud_cover_low"] = hourly_cloud_cover_low
    hourly_data["cloud_cover_mid"] = hourly_cloud_cover_mid
    hourly_data["cloud_cover_high"] = hourly_cloud_cover_high
    hourly_data["vapour_pressure_deficit"] =
hourly_vapour_pressure_deficit
    hourly_data["pressure_msl"] = hourly_pressure_msl
    hourly_data["surface_pressure"] = hourly_surface_pressure
    hourly_data["wind_speed_10m"] = hourly_wind_speed_10m
    hourly_data["wind_speed_100m"] = hourly_wind_speed_100m


    df = pd.DataFrame(data=hourly_data)
    df.set_index('date', drop=True, inplace=True)
    # Save to cache
    with open(cache_file, 'wb') as f:
        pickle.dump(df, f)
    print(f"Data saved to cache: {cache_file}")


    return df
```

Adding features

```
import numpy as np
import pandas as pd
```

SCHOLARLY REVIEW

SCHOLARLY REVIEW JOURNAL

E-ISSN: 2996-8380
Published by Leadership & Innovation Lab

SUMMER 2026

```python
def add_lags(df, target_col, lags, drop_original=False):
    """Add lag features without copying entire dataframe"""
    if isinstance(target_col, str):
        target_cols = [target_col]
    else:
        target_cols = list(target_col)

    for col in target_cols:
        for lag in lags:
            df[f'{col}_lag_{lag}'] = df[col].shift(lag)

    if drop_original:
        df.drop(columns=target_cols, inplace=True)

    return df


def add_rolling(df, target_col, windows, stats=['mean'],
drop_original=False):
    """Add rolling features with selective statistics"""
    if isinstance(target_col, str):
        target_cols = [target_col]
    else:
        target_cols = list(target_col)

    for col in target_cols:
        shifted = df[col].shift(1)
```

SCHOLARLY REVIEW

SCHOLARLY REVIEW JOURNAL

E-ISSN: 2996-8380
Published by Leadership & Innovation Lab

SUMMER 2026

```
        for window in windows:
            roller = shifted.rolling(window)
            if 'mean' in stats:
                df[f'{col}_rolling_mean_{window}'] = roller.mean()
            if 'std' in stats:
                df[f'{col}_rolling_std_{window}'] = roller.std()
            if 'min' in stats:
                df[f'{col}_rolling_min_{window}'] = roller.min()
            if 'max' in stats:
                df[f'{col}_rolling_max_{window}'] = roller.max()

    if drop_original:
        df.drop(columns=target_cols, inplace=True)

    return df

def add_cyclic(df, features=['hour', 'month', 'day_of_year']):
    """Add selective cyclic features"""
    if not isinstance(df.index, pd.DatetimeIndex):
        df.index = pd.to_datetime(df.index)

    if 'hour' in features:
        df['hour_sin'] = np.sin(2 * np.pi * df.index.hour / 24)
        df['hour_cos'] = np.cos(2 * np.pi * df.index.hour / 24)

    if 'month' in features:
```

SCHOLARLY REVIEW

SCHOLARLY REVIEW JOURNAL

E-ISSN: 2996-8380
Published by Leadership & Innovation Lab

SUMMER 2026

```
        df['month_sin'] = np.sin(2 * np.pi * df.index.month / 12)
        df['month_cos'] = np.cos(2 * np.pi * df.index.month / 12)

    if 'day_of_year' in features:
        df['day_of_year_sin'] = np.sin(2 * np.pi * df.index.dayofyear
/ 365)
        df['day_of_year_cos'] = np.cos(2 * np.pi * df.index.dayofyear
/ 365)

    return df
```

SCHOLARLY REVIEW JOURNAL

E-ISSN: 2996-8380
Published by Leadership & Innovation Lab

SUMMER 2026

Splitting training and testing data

```
# ---- Splitting Training and Testing Data By Year ----

train = lagged_df.loc[lagged_df.index < '01-01-2020']
test = lagged_df.loc[lagged_df.index >= '01-01-2020']

train = create_cyclic_features(train)
test = create_cyclic_features(test)

train.reset_index(drop=True)
test.reset_index(drop=True)

target_col = 'rain'
feature_cols = [col for col in train.columns if col != target_col]

X_train = train[feature_cols]
y_train = train[target_col]
X_test = test[feature_cols]
y_test = test[target_col]
```

SCHOLARLY REVIEW

SCHOLARLY REVIEW JOURNAL

E-ISSN: 2996-8380
Published by Leadership & Innovation Lab

SUMMER 2026

Random Forests

```
# Random Forests


import matplotlib.pyplot as plt

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean_squared_error, mean_absolute_error


min_samples_split = int(input("Enter minimum samples split: "))

n_samples = len(X_train)

n_bootstraps = 100

all_bootstrap_indices, all_oob_indices = [], []

np.random.seed(62)

for i in range(n_bootstraps):

    bootstrap_indices = np.random.choice(n_samples, size=5,
replace=True)

    oob_indices = list(set(range(n_samples)) - set(bootstrap_indices))

    all_bootstrap_indices.append(bootstrap_indices)

    all_oob_indices.append(oob_indices)

randomForest = RandomForestRegressor(n_estimators=100,
min_samples_split=min_samples_split, max_features='sqrt',
random_state=67)

randomForest.fit(X_train, y_train)

y_pred = randomForest.predict(X_test)

mse = mean_squared_error(y_test, y_pred)

rmse = np.sqrt(mse)

print(f"Mean Squared Error: {mse:.2f}")
```

SCHOLARLY REVIEW

SCHOLARLY REVIEW JOURNAL

E-ISSN: 2996-8380
Published by Leadership & Innovation Lab

SUMMER 2026

```
print(f"Root Mean Squared Error: {rmse:.2f}")

mae = mean_absolute_error(y_test, y_pred)

print(f'Mean Absolute Error: {mae}')
```

SCHOLARLY REVIEW JOURNAL

E-ISSN: 2996-8380
Published by Leadership & Innovation Lab

SUMMER 2026

XGBoost

Base XGBoost Model (untuned)

```
# Base XGBoost Model (untuned)


import xgboost as xgb
from sklearn.metrics import mean_squared_error, mean_absolute_error


trainReg = xgb.DMatrix(X_train, y_train, enable_categorical=False)
testReg = xgb.DMatrix(X_test, y_test, enable_categorical=False)
params = {"objective": "reg:squarederror"}
model = xgb.train(params=params, dtrain=trainReg, num_boost_round=100)
preds = model.predict(testReg)
rmse = np.sqrt(mean_squared_error(y_test, preds))
mae = mean_absolute_error(y_test, preds)
print(f"XGB RMSE (base model): {rmse:.5f}")
print(f"XGB RMSE (base model): {mae:.5f}")
print(f"XGB RMSE (trained model):")
evals = [(trainReg, "train"), (testReg, "validation")]
model = xgb.train(params=params, dtrain=trainReg, num_boost_round=100,
evals=evals, verbose_eval=5, early_stopping_rounds=10)
```

SCHOLARLY REVIEW

SCHOLARLY REVIEW JOURNAL

E-ISSN: 2996-8380
Published by Leadership & Innovation Lab

SUMMER 2026

SVM

```
# SVM Regressor


from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error, r2_score
import pandas as pd
import matplotlib.pyplot as plt

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

svm_regr= SVR(kernel="linear",C=1.0)
svm_regr.fit(X_train_scaled, y_train)

#hyperparameter tuning
param_grid = {
    'C': [1, 10, 50, 100],
    'gamma': [0.1, 0.01, 0.001, 'scale'],
    'epsilon': [0.01, 0.05, 0.1]
}
y_pred = svm_regr.predict(X_test_scaled)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)
```

SCHOLARLY REVIEW

SCHOLARLY REVIEW JOURNAL

E-ISSN: 2996-8380
Published by Leadership & Innovation Lab

SUMMER 2026

```
print(f"RMSE: {rmse:.2f}")
print(f"R² Score: {r2:.2f}")
```

SCHOLARLY REVIEW JOURNAL

E-ISSN: 2996-8380
Published by Leadership & Innovation Lab

SUMMER 2026

LSTM

```
# LSTM using PyTorch

import torch
import torch.nn as nn
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error
import matplotlib.pyplot as plt

print(f"PyTorch version: {torch.__version__}")
print(f"CUDA available: {torch.cuda.is_available()}")

# Scale the data
scaler_X = StandardScaler()
scaler_y = StandardScaler()

X_train_scaled = scaler_X.fit_transform(X_train)
X_test_scaled = scaler_X.transform(X_test)

y_train_scaled = scaler_y.fit_transform(y_train.values.reshape(-1, 1))
y_test_scaled = scaler_y.transform(y_test.values.reshape(-1, 1))

# Convert to PyTorch tensors
```

SCHOLARLY REVIEW

SCHOLARLY REVIEW JOURNAL

E-ISSN: 2996-8380
Published by Leadership & Innovation Lab

SUMMER 2026

```
X_train_tensor = torch.FloatTensor(X_train_scaled).reshape(-1, 1,
X_train_scaled.shape[1])

y_train_tensor = torch.FloatTensor(y_train_scaled)

X_test_tensor = torch.FloatTensor(X_test_scaled).reshape(-1, 1,
X_test_scaled.shape[1])


print(f"X_train shape: {X_train_tensor.shape}")

print(f"X_test shape: {X_test_tensor.shape}")


# Define LSTM Model

class LSTMModel(nn.Module):

    def __init__(self, input_size, hidden_size=64):

        super(LSTMModel, self).__init__()

        self.lstm = nn.LSTM(input_size, hidden_size, batch_first=True)

        self.dropout = nn.Dropout(0.2)

        self.fc1 = nn.Linear(hidden_size, 32)

        self.fc2 = nn.Linear(32, 1)

        self.relu = nn.ReLU()


    def forward(self, x):

        lstm_out, _ = self.lstm(x)

        lstm_out = lstm_out[:, -1, :]  # Get last timestep output

        x = self.dropout(lstm_out)

        x = self.relu(self.fc1(x))

        x = self.fc2(x)

        return x
```

SCHOLARLY REVIEW

SCHOLARLY REVIEW JOURNAL

E-ISSN: 2996-8380
Published by Leadership & Innovation Lab

SUMMER 2026

```
# Initialize model
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f"Using device: {device}")

model = LSTMModel(input_size=X_train_scaled.shape[1]).to(device)
criterion = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

# Train the model
batch_size = 128
n_epochs = 30

print("\nTraining LSTM model...")
train_losses = []

for epoch in range(n_epochs):
    model.train()
    epoch_loss = 0
    n_batches = 0

    # Mini-batch training
    for i in range(0, len(X_train_tensor), batch_size):
        batch_X = X_train_tensor[i:i+batch_size].to(device)
        batch_y = y_train_tensor[i:i+batch_size].to(device)
```

SCHOLARLY REVIEW

SCHOLARLY REVIEW JOURNAL

E-ISSN: 2996-8380
Published by Leadership & Innovation Lab

SUMMER 2026

```
        # Forward pass
        optimizer.zero_grad()
        outputs = model(batch_X)
        loss = criterion(outputs, batch_y)

        # Backward pass
        loss.backward()
        optimizer.step()

        epoch_loss += loss.item()
        n_batches += 1

    avg_loss = epoch_loss / n_batches
    train_losses.append(avg_loss)

    if (epoch + 1) % 5 == 0:
        print(f'Epoch [{epoch+1}/{n_epochs}], Loss: {avg_loss:.6f}')

# Make predictions
print("\nMaking predictions...")
model.eval()
with torch.no_grad():
    y_train_pred_scaled =
model(X_train_tensor.to(device)).cpu().numpy()
    y_test_pred_scaled = model(X_test_tensor.to(device)).cpu().numpy()
```

SCHOLARLY REVIEW

SCHOLARLY REVIEW JOURNAL

E-ISSN: 2996-8380
Published by Leadership & Innovation Lab

SUMMER 2026

```
# Inverse transform
y_train_pred = scaler_y.inverse_transform(y_train_pred_scaled)
y_test_pred = scaler_y.inverse_transform(y_test_pred_scaled)

y_train_actual = scaler_y.inverse_transform(y_train_scaled)
y_test_actual = scaler_y.inverse_transform(y_test_scaled)

# Evaluate MAE, RMSE
train_rmse = np.sqrt(mean_squared_error(y_train_actual, y_train_pred))
test_rmse = np.sqrt(mean_squared_error(y_test_actual, y_test_pred))
train_mae = mean_absolute_error(y_train_actual, y_train_pred)
test_mae = mean_absolute_error(y_test_actual, y_test_pred)



print("\n" + "="*60)
print("LSTM MODEL PERFORMANCE")
print("="*60)
print(f"Train RMSE: {train_rmse:.4f}")
print(f"Test RMSE:  {test_rmse:.4f}")
print(f"Train MAE: {train_mae:.4f}")
print(f"Test MAE:  {test_mae:.4f}")
gap = test_rmse - train_rmse
gap_pct = (gap / train_rmse) * 100
print(f"Gap:        {gap:.4f} ({gap_pct:.1f}%)")
print("="*60)
```

SCHOLARLY REVIEW JOURNAL

E-ISSN: 2996-8380
Published by Leadership & Innovation Lab

SUMMER 2026

```
# Plot training loss
plt.figure(figsize=(10, 5))
plt.plot(train_losses)
plt.xlabel('Epoch')
plt.ylabel('Loss (MSE)')
plt.title('LSTM Training Loss')
plt.grid(True, alpha=0.3)
plt.show()
```

GLM

```
import optuna
from sklearn.linear_model import TweedieRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error
import numpy as np
import pandas as pd
import warnings
from sklearn.exceptions import ConvergenceWarning


def train_tweedie_glm(X_train, y_train, X_test=None, y_test=None,
                      n_trials=50, verbose=True,
                      suppress_warnings=True):
    """
```

SCHOLARLY REVIEW

SCHOLARLY REVIEW JOURNAL

E-ISSN: 2996-8380
Published by Leadership & Innovation Lab

SUMMER 2026

```
    Train a Tweedie Regressor GLM model with Optuna hyperparameter
optimization.


    Parameters:
    -----------
    X_train : pd.DataFrame or np.array
        Training features
    y_train : pd.Series or np.array
        Training targets (non-negative values)
    X_test : pd.DataFrame or np.array, optional
        Test features for validation during optimization
    y_test : pd.Series or np.array, optional
        Test targets for validation during optimization
    n_trials : int
        Number of Optuna trials for optimization
    verbose : bool
        Print training info
    suppress_warnings : bool
        Suppress convergence warnings during optimization


    Returns:
    --------
    Trained TweedieRegressor model with study results
    """
    # Ensure non-negative values
    y_train_tweedie = y_train.copy()
```

SCHOLARLY REVIEW JOURNAL

E-ISSN: 2996-8380
Published by Leadership & Innovation Lab

SUMMER 2026

```
    if isinstance(y_train_tweedie, pd.Series):
        y_train_tweedie = y_train_tweedie.values

    # Tweedie requires y >= 0
    y_train_tweedie = np.maximum(y_train_tweedie, 0)

    # Handle test set if provided
    if X_test is not None and y_test is not None:
        y_test_tweedie = y_test.copy()
        if isinstance(y_test_tweedie, pd.Series):
            y_test_tweedie = y_test_tweedie.values
        y_test_tweedie = np.maximum(y_test_tweedie, 0)

    def objective(trial):
        # Suggest hyperparameters
        power = trial.suggest_float('power', 1.0, 2.0)
        alpha = trial.suggest_float('alpha', 1e-4, 100.0, log=True)
        max_iter = trial.suggest_int('max_iter', 100, 2000, step=100)

        # Train model with warning suppression
        if suppress_warnings:
            with warnings.catch_warnings():
                warnings.filterwarnings('ignore',
category=ConvergenceWarning)
                model = TweedieRegressor(
                    power=power,
```

SCHOLARLY REVIEW

SCHOLARLY REVIEW JOURNAL

E-ISSN: 2996-8380
Published by Leadership & Innovation Lab

SUMMER 2026

```
                alpha=alpha,
                max_iter=max_iter,
                link='log'  # log link is standard for Tweedie
            )
            model.fit(X_train, y_train_tweedie)
    else:
        model = TweedieRegressor(
            power=power,
            alpha=alpha,
            max_iter=max_iter,
            link='log'
        )
        model.fit(X_train, y_train_tweedie)

    # Evaluate on test set if provided, otherwise on train set
    if X_test is not None and y_test is not None:
        y_pred = model.predict(X_test)
        score = mean_squared_error(y_test_tweedie, y_pred)
    else:
        y_pred = model.predict(X_train)
        score = mean_squared_error(y_train_tweedie, y_pred)

    return score

# Create study
if verbose:
```

SCHOLARLY REVIEW

SCHOLARLY REVIEW JOURNAL

E-ISSN: 2996-8380
Published by Leadership & Innovation Lab

SUMMER 2026

```
        optuna.logging.set_verbosity(optuna.logging.INFO)
    else:
        optuna.logging.set_verbosity(optuna.logging.WARNING)


    study = optuna.create_study(direction='minimize',
study_name='tweedie_glm_optimization')
    study.optimize(objective, n_trials=n_trials,
show_progress_bar=verbose)


    # Get best parameters
    best_params = study.best_params


    if verbose:
        print("\n" + "="*60)
        print("BEST PARAMETERS FOUND:")
        print("="*60)
        print(f"Power: {best_params['power']:.4f}")
        print(f"Alpha: {best_params['alpha']:.6f}")
        print(f"Max Iterations: {best_params['max_iter']}")
        print(f"Best MSE: {study.best_value:.6f}")
        print(f"Best RMSE: {np.sqrt(study.best_value):.6f}")


    # Train final model with best parameters
    final_power = best_params.get('power')


    if suppress_warnings and not verbose:
```

SCHOLARLY REVIEW

SCHOLARLY REVIEW JOURNAL

E-ISSN: 2996-8380
Published by Leadership & Innovation Lab

SUMMER 2026

```
        with warnings.catch_warnings():
            warnings.filterwarnings('ignore',
category=ConvergenceWarning)
            model = TweedieRegressor(
                power=final_power,
                alpha=best_params['alpha'],
                max_iter=best_params['max_iter'],
                link='log'
            )
            model.fit(X_train, y_train_tweedie)
    else:
        model = TweedieRegressor(
            power=final_power,
            alpha=best_params['alpha'],
            max_iter=best_params['max_iter'],
            link='log'
        )
        model.fit(X_train, y_train_tweedie)

    # Store optimization results
    model.study_ = study
    model.best_params_ = best_params

    # Create results DataFrame
    trials_df = study.trials_dataframe()
    model.cv_results_ = trials_df
```

SCHOLARLY REVIEW

SCHOLARLY REVIEW JOURNAL

E-ISSN: 2996-8380
Published by Leadership & Innovation Lab

SUMMER 2026

```
    return model

# Usage example
model = train_tweedie_glm(
    X_train, y_train,
    X_test=X_test, y_test=y_test,
    n_trials=10,
    verbose=True,
    suppress_warnings=True
)
```