Integrating Machine Learning with ADCP Data for Advanced Sediment Transport and Hydrodynamics Monitoring

Mohammad Tanvir Haque Tuhin[1,2], Christoph Mudersbach[1], Reinhard Hinkelmann[2]

[1] Bochum University of Applied Sciences, Institute of Hydraulic Engineering and Hydromechanics, Department of Civil and Environmental Engineering, Germany

[2] Technische Universität Berlin, Chair of Water Resources Management and Modeling of Hydrosystems, Institute of Civil Engineering, Germany

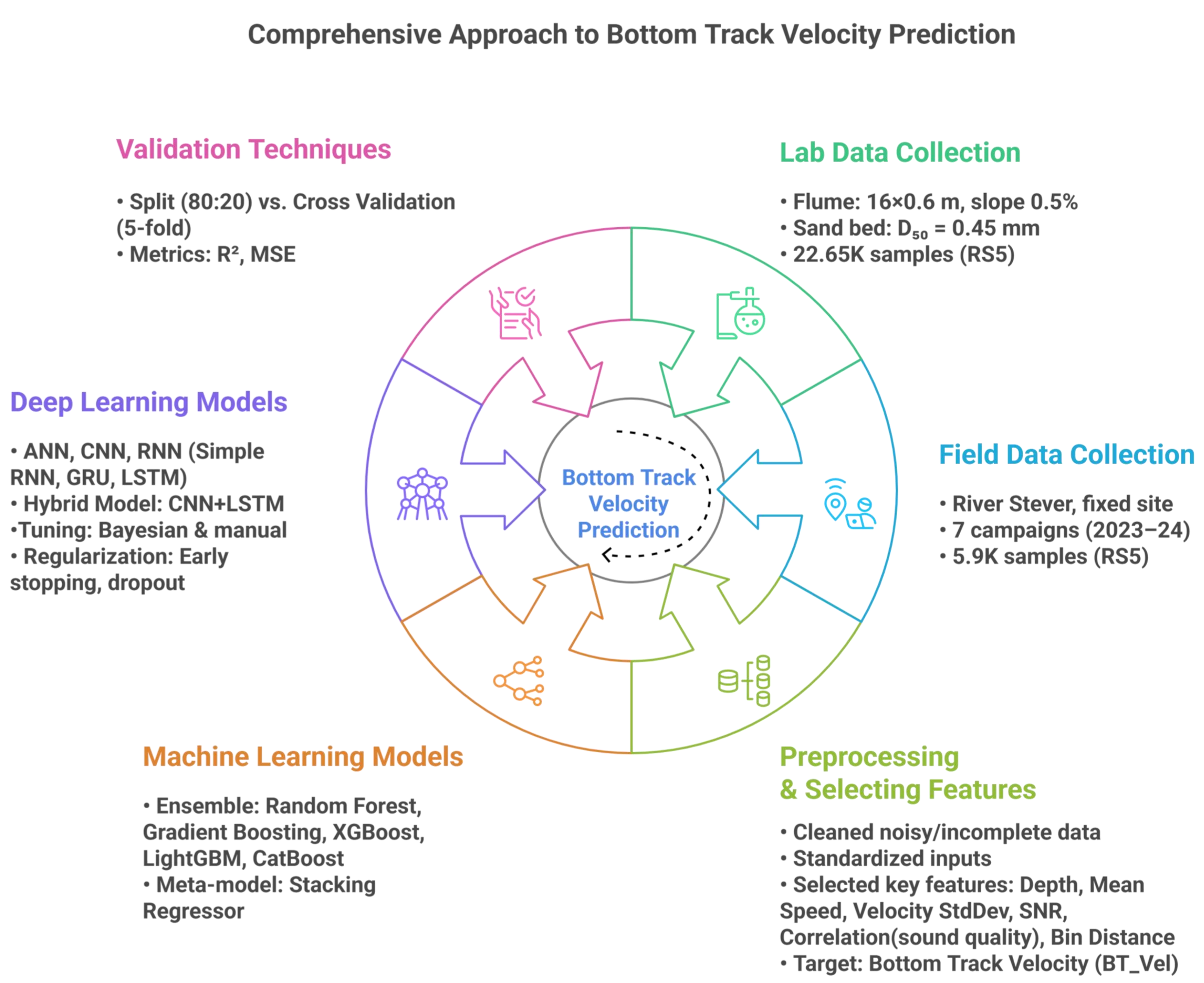
Introduction and objectives

Acoustic Doppler Current Profilers (ADCP) provide a rich yet underutilized dataset for continuous monitoring of hydrodynamics and sediment transport. Accurate prediction of sediment-related variables is essential for river engineering, morphological studies, and environmental management. Among key proxies, Bottom Track Velocity (BT_Vel) serves as a critical indicator for understanding bedload movement and near-bed sediment dynamics.

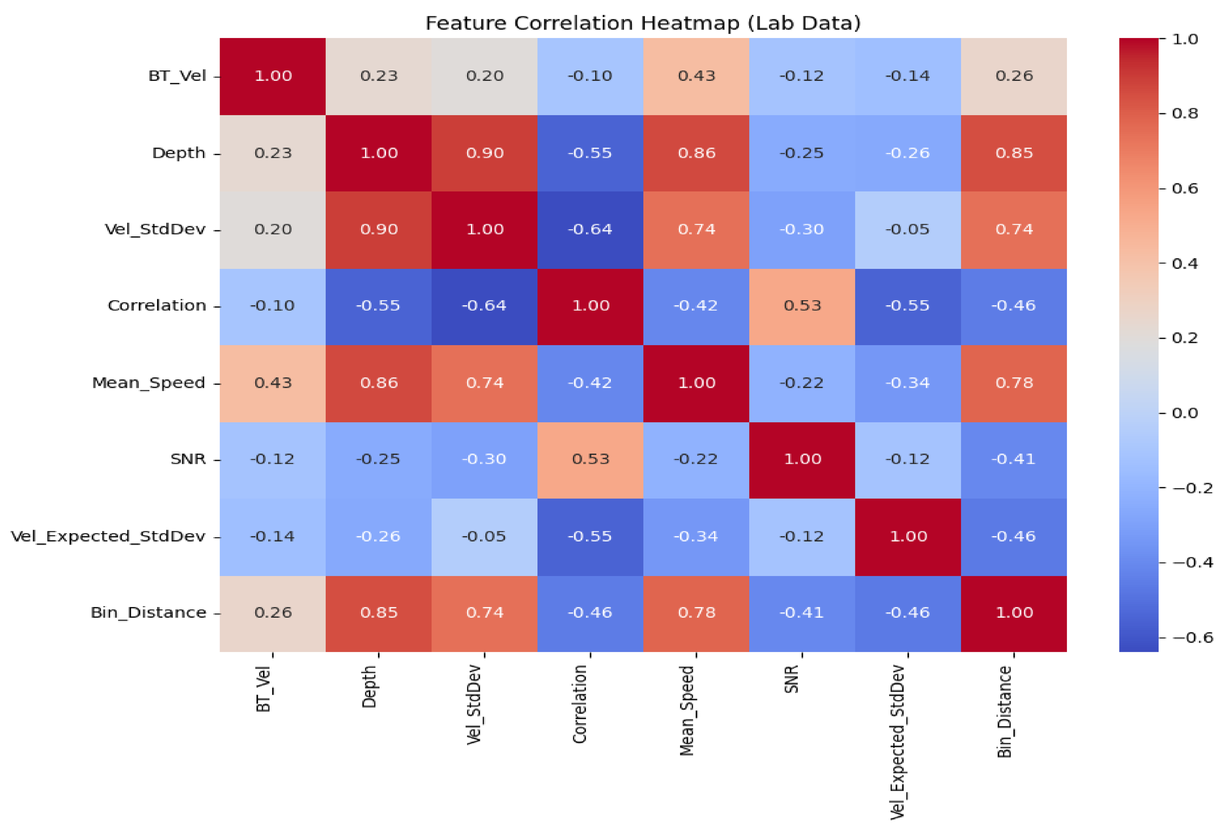
The main aims of this work are:

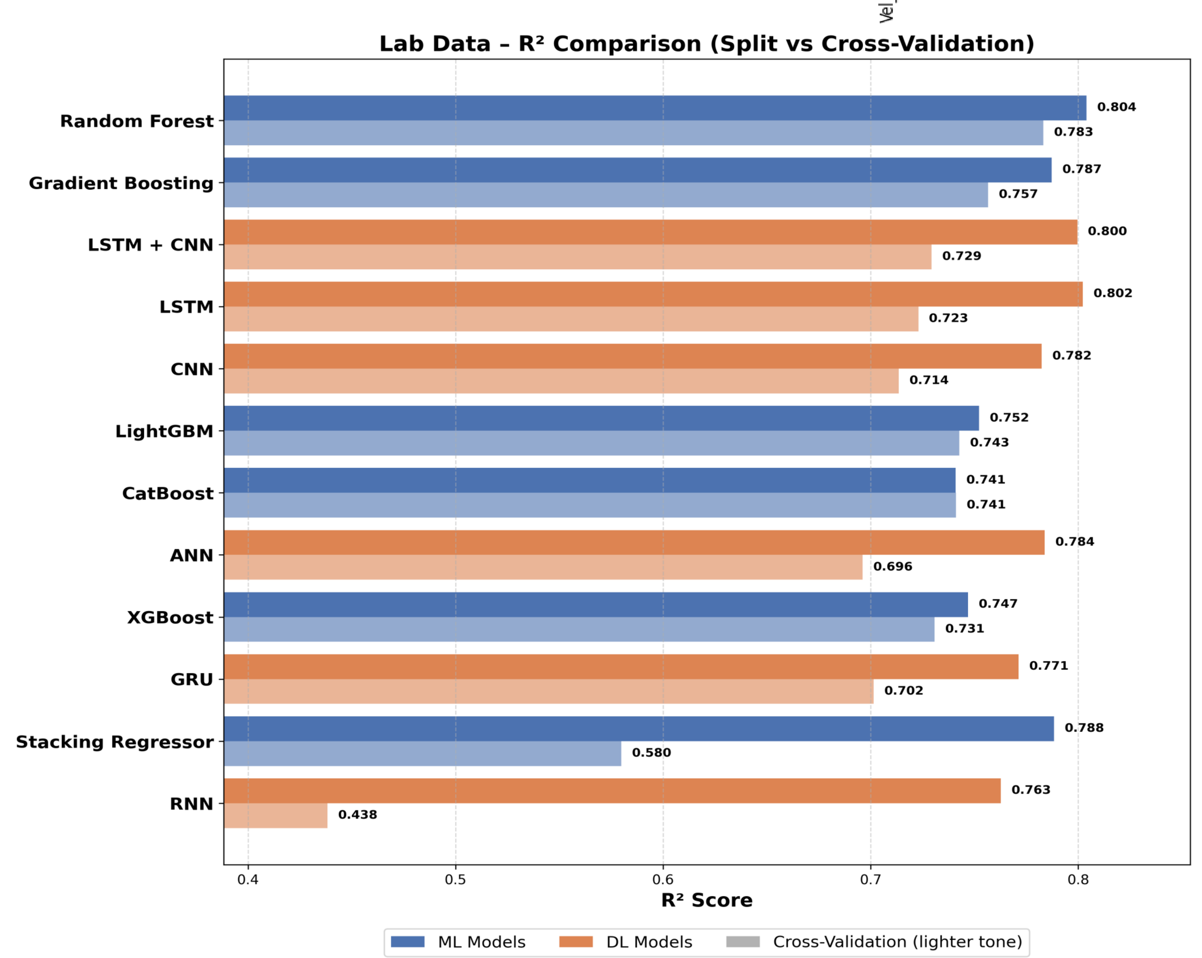
- Bridge physical sensing (ADCP) with machine learning (ML) and deep learning (DL) for enhanced sediment and flow monitoring.
- Evaluate a broad set of ML & DL models for predicting BT_Vel.
- Assess model performance on both a large-scale lab samples (22,650) and real-world field samples (5,900).
- Compare Split vs. Cross-Validation (CV) to examine model reliability and generalization.

Methodology



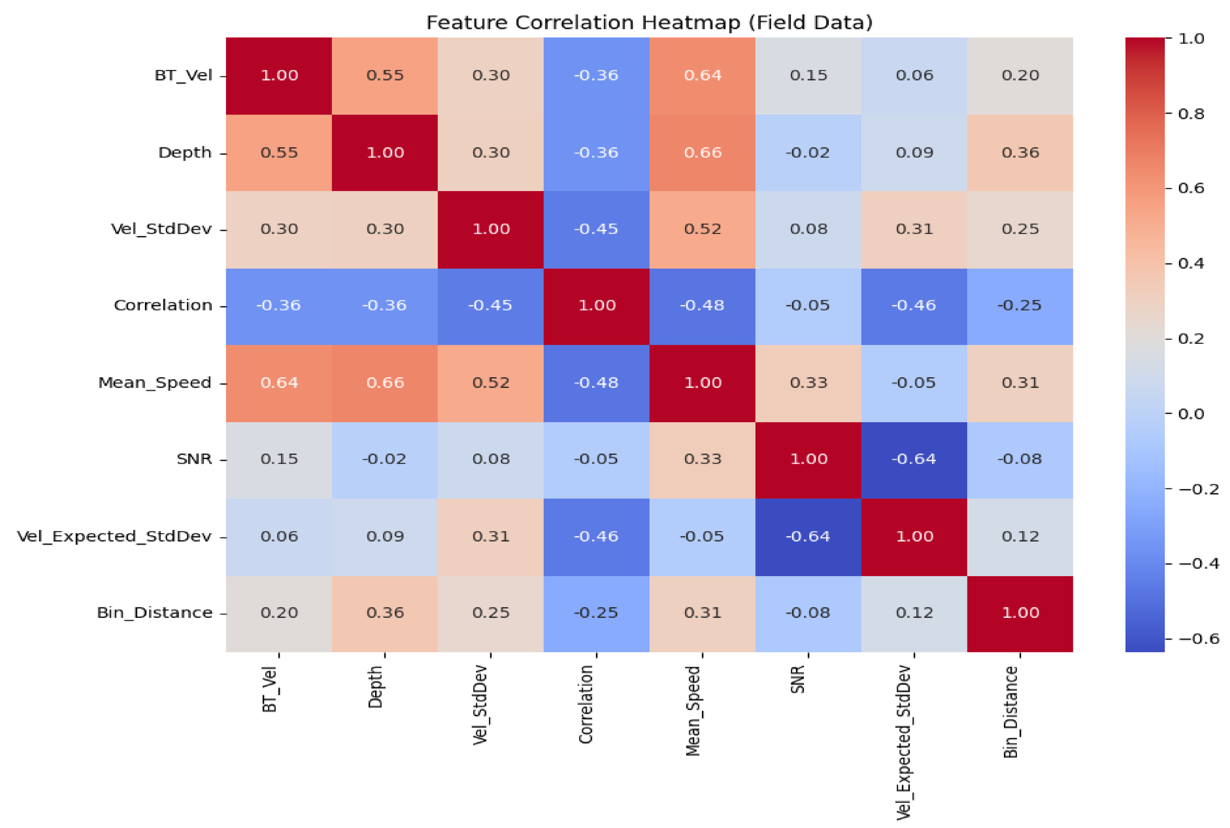
Results (Lab)

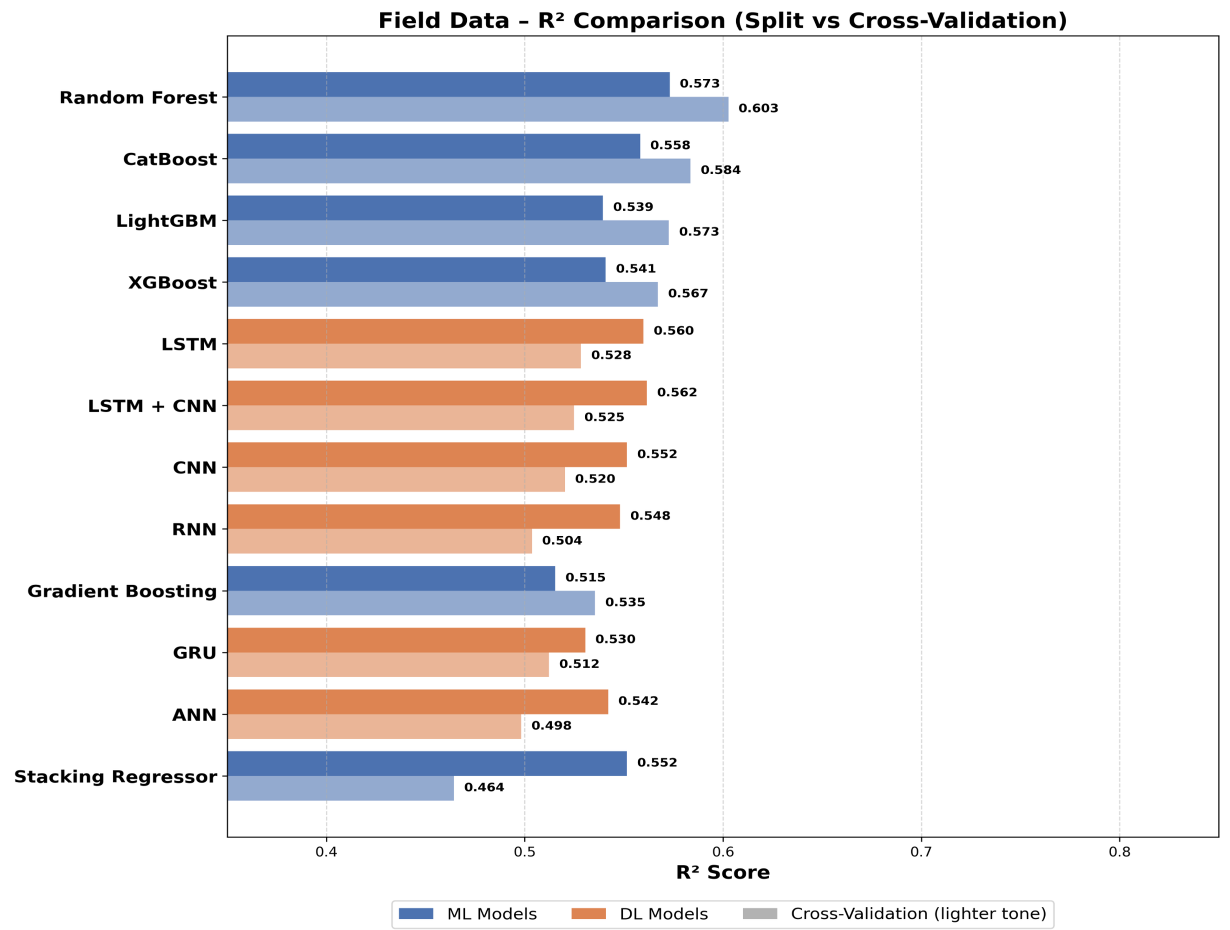




- **Random Forest** and **Gradient Boosting** achieved the highest and most consistent R² scores across both Split (darker tone) and Cross-Validation, showing strong generalization. LightGBM, CatBoost, and XGBoost followed with slightly lower but stable performance.
- **LSTM** and **LSTM+CNN** matched top ML models in Split testing and remained competitive during CV; other DL models (CNN, GRU, ANN) performed well in Split but dropped in CV, indicating sensitivity to data variation and potential overfitting.
- Despite strong Split-set scores, the **Stacking Regressor** and **RNN** showed sharp R² drops in CV; RNN ranked lowest, likely due to limitations of simple recurrent architectures in structured lab setting.

Results (Field)





- Tree-based ML models performed robustly under real-river, noisy conditions; **Random Forest** led, followed by CatBoost, LightGBM, and XGBoost, all showing strong generalizability across Split and CV. **Gradient Boosting** was competitive but slightly less consistent.
- DL models like **LSTM, LSTM+CNN,** and **CNN** showed solid R² in Split, with slight drops in CV due to data sparsity and fold variability. **RNN** surprisingly outperformed ANN and GRU in field data, making it a viable lightweight DL option under limited-data conditions.
- The **Stacking Regressor**, despite moderate Split performance, showed a marked R² decline in CV, suggesting overfitting and low resilience to fold variation—consistent with its lab result.

Note: The comparison was further validated using **MSE** values. **Bayesian Optimization** did not improve DL performance.

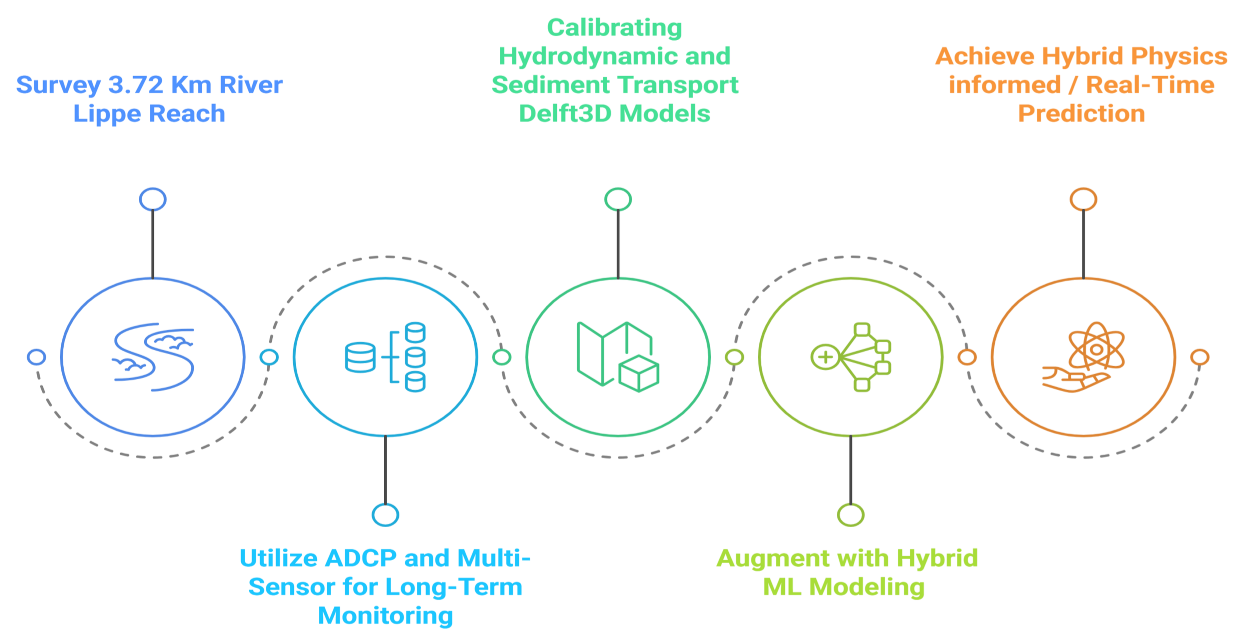
Discussion

- **ADCP-derived features** (e.g., bottom track velocity) effectively predicted sediment transport dynamics, with high model performance in lab conditions (large, clean, consistent data) and lower generalizability in field conditions (smaller, variable, temporally sparse data).
- **Cross-Validation** exposed overfitting risks more clearly than Split-Validation, especially for deep learning and ensemble models.
- **Tree-based models**, especially **Random Forest**, consistently outperformed others across both datasets, demonstrating strong resilience to noise and real-river variability.
- While **DL models** like **LSTM** performed well in lab settings, their sensitivity to field data limitations (e.g., volume, noise) led to CV drops—underscoring the need for regularization and data augmentation in real-world deployments.
- DL models showed smaller CV **performance drops** on field data compared to lab, likely due to lower variance and simpler structure in available data

Conclusion and Future Outlook

This study presents a robust framework integrating **ADCP-derived data** with ML/DL models for predicting **Bottom Track Velocity**, enhancing sediment transport analysis. **Tree-based models** (e.g., Random Forest) demonstrated high accuracy and stability, while DL models showed potential with further tuning.

Building on these findings, we plan long-term monitoring of a 3.72 km River Lippe reach, integrating ADCP proxies with **Delft3D and ML** for hybrid, physics-informed hydrodynamic and sediment transport prediction.



Bochum University of Applied Sciences
TECHNOLOGY BUSINESS HEALTH
TECHNISCHE UNIVERSITÄT BERLIN
DFG Deutsche Forschungsgemeinschaft
Project-ID 528740160 – FIP 61
Scan for Abstract & Further Data
EGU General Assembly 2025

Contact:
Mohammd Tanvir Haque Tuhin
mohammd.tuhin@hs-bochum.de

Supplementary Script - Example of code pipeline

This notebook provides a part of reproducible pipeline used in the study:

"Integrating Machine Learning with ADCP Data for Advanced Sediment Transport and Hydrodynamics Monitoring"

It includes:

- Scalar feature extraction from `.mat` files
- Preprocessing and feature engineering (e.g., Relative ABS)
- Machine learning model training (Split & Stratified CV)
- Deep learning model training (LSTM+CNN with batch size tuning)
- A visual summary (SHAP-based feature importance for ML models on the lab dataset)

All code is based on the actual implementation used in the poster:

EGU25-8296 | Mohammad Tanvir Haque Tuhin, Bochum University of Applied Sciences

Import All Required Libraries and Set Random Seed

```
# Core packages for data manipulation and reproducibility
import numpy as np
import pandas as pd
import random, os

# ML libraries
from sklearn.model_selection import train_test_split, StratifiedKFold
from sklearn.preprocessing import StandardScaler, KBinsDiscretizer
from sklearn.pipeline import make_pipeline
from sklearn.base import clone
from sklearn.metrics import mean_squared_error, r2_score

# ML models
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor, StackingRegressor
from xgboost import XGBRegressor
from lightgbm import LGBMRegressor
from catboost import CatBoostRegressor

# DL libraries
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.optimizers import Adam

# For loading MATLAB (.mat) files
from scipy.io import loadmat

# Set random seeds for reproducible results
seed_value = 42
np.random.seed(seed_value)
tf.random.set_seed(seed_value)
random.seed(seed_value)
os.environ["TF_DETERMINISTIC_OPS"] = "1"
```

Convert .mat Files → Scalar Features + Relative_ABS(Accoustic Backscatter Strength)

```
# Function to extract and process a specific field from a .mat file
def process_flow_rate(file_path, feature_name, field_name, matrix_field=True, max_time_steps=1620, preprocess=None):
    matlab_data = loadmat(file_path, struct_as_record=False, squeeze_me=True)
    struct_data = matlab_data[feature_name]
    field_data = getattr(struct_data, field_name)
    if preprocess is not None:
        field_data = preprocess(field_data)
    return np.nanmean(field_data, axis=(0, 1)) if matrix_field else field_data

# Clean bottom track velocity: remove negatives and average across beams
def preprocess_bottom_track(data):
    data[data < 0] = 0
```

```
    return np.nanmean(data, axis=1)

# Prepare feature list from .mat files into rows for tabular analysis
alpha = 0.07  # Acoustic attenuation factor for Relative ABS calculation
final_data = []

# NOTE: This assumes 'processed_data' is already filled from your .mat extraction
for label, features in processed_data.items():
    for i in range(1618):  # 1618 time steps
        row = {"Flow Rate": label}
        for key, values in features.items():
            if values is not None and len(values) > i:
                row[key.split('_', 1)[1]] = values[i]

        # Feature Engineering: calculate Bin_Distance and Relative ABS
        if ('System_Cell_Start' in features and 'System_Cell_Size' in features and 'System_SNR' in features):
            bin_distance = features['System_Cell_Start'][i] + features['System_Cell_Size'][i] / 2
            row['Bin_Distance'] = bin_distance
            snr = features['System_SNR'][i]
            row['Relative_ABS'] = snr + 20 * np.log10(bin_distance) + 2 * alpha * bin_distance

        final_data.append(row)

# Create the final structured DataFrame
df = pd.DataFrame(final_data)
features = ['Depth', 'Vel_StdDev', 'Correlation', 'Mean_Speed', 'SNR', 'Vel_Expected_StdDev', 'Bin_Distance', 'Relative_ABS']
target = 'BT_Vel'
df = df.dropna(subset=features + [target])  # Drop rows with missing values
X = df[features].values
y = df[target].values
```

Train & Evaluate ML Models using Stratified Cross-Validation

```
# Normalize feature scales
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Define ML models for benchmarking
models = {
   # "Linear Regression": LinearRegression(),
    #"Decision Tree": DecisionTreeRegressor(random_state=42),
    "Random Forest": RandomForestRegressor(random_state=42, n_estimators=100),
    "Gradient Boosting": GradientBoostingRegressor(random_state=42, n_estimators=100),
    "XGBoost": XGBRegressor(random_state=42, n_estimators=100, learning_rate=0.1),
    "LightGBM": LGBMRegressor(random_state=42, n_estimators=100, learning_rate=0.1),
    "CatBoost": CatBoostRegressor(random_state=42, verbose=0),
    "Stacking Regressor": StackingRegressor(
        estimators=[
            ('rf', RandomForestRegressor(random_state=42)),
            ('gb', GradientBoostingRegressor(random_state=42))
        ],
        final_estimator=GradientBoostingRegressor(random_state=42)
    )
}

# Stratify continuous target using quantile binning
binner = KBinsDiscretizer(n_bins=10, encode='ordinal', strategy='quantile')
y_binned = binner.fit_transform(y.reshape(-1, 1)).ravel()
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Run cross-validation
ml_cv_results = []
for name, model in models.items():
    r2_scores, mse_scores = [], []
    for train_idx, test_idx in skf.split(X_scaled, y_binned):
        X_tr, X_val = X_scaled[train_idx], X_scaled[test_idx]
        y_tr, y_val = y[train_idx], y[test_idx]
        pipeline = make_pipeline(StandardScaler(), clone(model))
        pipeline.fit(X_tr, y_tr)
        preds = pipeline.predict(X_val)
        r2_scores.append(r2_score(y_val, preds))
        mse_scores.append(mean_squared_error(y_val, preds))
    ml_cv_results.append({
        "Model": name,
        "Mean R²": np.mean(r2_scores),
        #"Std R²": np.std(r2_scores),
        "Mean MSE": np.mean(mse_scores),
```

```
        #"Std MSE": np.std(mse_scores)
    })
```

Train & Evaluate ML Models using Split Validation (80/20)

```
# Simple 80/20 split validation for comparison
X_train_split, X_test_split, y_train_split, y_test_split = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

ml_split_results = []
for name, model in models.items():
    model.fit(X_train_split, y_train_split)
    y_pred = model.predict(X_test_split)
    ml_split_results.append({
        "Model": name,
        "MSE": mean_squared_error(y_test_split, y_pred),
        "R²": r2_score(y_test_split, y_pred)
    })
```

Train & Tune Deep Learning Model (LSTM + CNN) with Batch Size

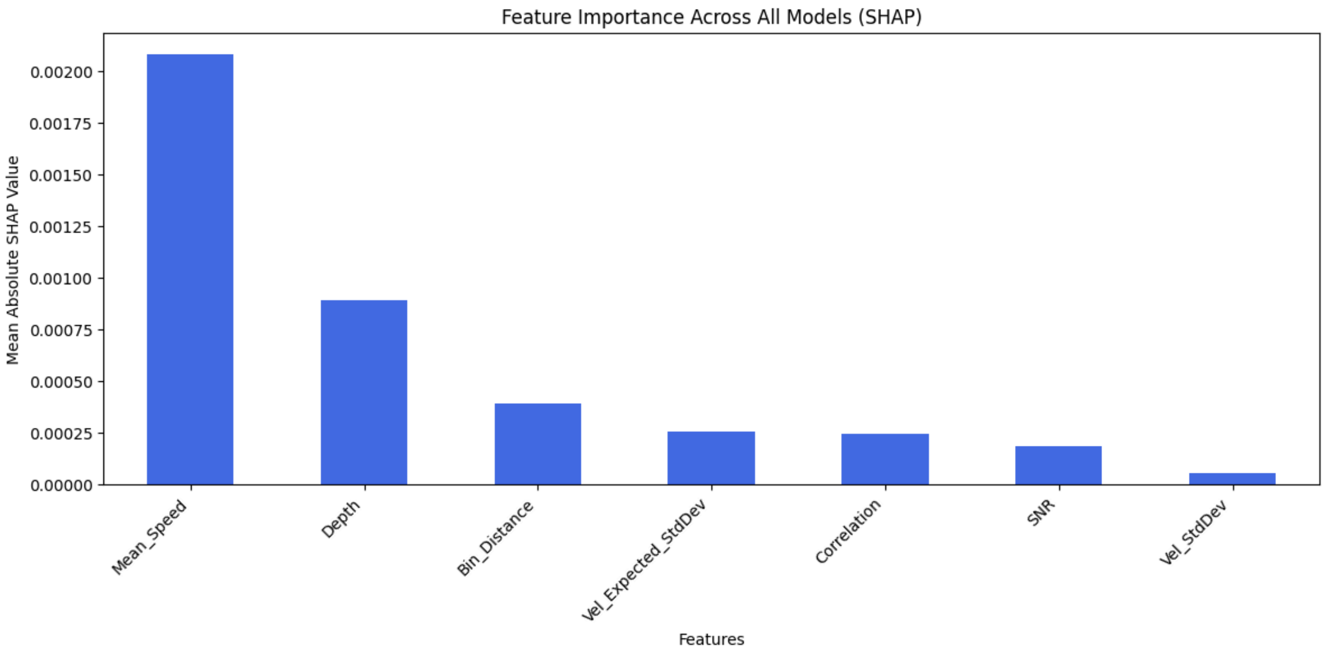
```
# Reshape input for CNN+LSTM (samples, time_steps, features)
X_train_dl, X_test_dl, y_train_dl, y_test_dl = train_test_split(X_scaled, y, test_size=0.2, random_state=seed_value)
X_train_dl = X_train_dl.reshape((X_train_dl.shape[0], 1, X_train_dl.shape[1]))
X_test_dl = X_test_dl.reshape((X_test_dl.shape[0], 1, X_test_dl.shape[1]))

# Define the hybrid model
def create_cnn_lstm_model():
    model = keras.Sequential([
        layers.Conv1D(64, 1, activation='relu', input_shape=(1, X_train.shape[1])),
        layers.Conv1D(32, 1, activation='relu'),
        layers.Flatten(),
        layers.Reshape((1, -1)),
        layers.LSTM(64, return_sequences=True),
        layers.LSTM(32),
        layers.Dropout(0.2),
        layers.Dense(32, activation='relu'),
        layers.Dense(1)
    ])
    model.compile(optimizer=Adam(learning_rate=0.001), loss='mse', metrics=['mse'])
    return model

# Evaluate across multiple batch sizes
batch_sizes = [6, 8, 10, 12, 16, 20, 24, 32, 40, 64]
dl_results = []

for bs in batch_sizes:
    model = create_cnn_lstm_model()
    early_stop = keras.callbacks.EarlyStopping(monitor='val_loss', patience=7, restore_best_weights=True)
    model.fit(X_train_dl, y_train_dl, validation_data=(X_test_dl, y_test_dl),
              batch_size=bs, epochs=50, callbacks=[early_stop], verbose=0)
    y_pred = model.predict(X_test_dl).flatten()
    dl_results.append({
        "Batch Size": bs,
        "MSE": mean_squared_error(y_test_dl, y_pred),
        "R² Score": r2_score(y_test_dl, y_pred)
    })
```

Feature importance for Bottom Track Velocity prediction based on SHAP values across all trained machine learning models (lab dataset).



Feature Importance Across All Models (SHAP)
Mean Absolute SHAP Value
0.00200
0.00175
0.00150
0.00125
0.00100
0.00075
0.00050
0.00025
0.00000
Mean_Speed
Depth
Bin_Distance
Vel_Expected_StdDev
Correlation
SNR
Vel_StdDev
Features