CAPPA INTERNSHIP

SUMMER 2024

PROJECT: FloDX

NAME: Ashik Rahman

ID: R00212471

ONE-DRIVE:

TABLE OF CONTENTS

Introduction

The monitoring of bacterial growth is a fundamental process in microbiological research and various industrial applications. However, traditional methods for bacterial growth assessment are often laborious, time-consuming, and prone to contamination due to frequent sampling. These methods, typically involving spectrophotometric analysis, not only require significant manual intervention but also generate substantial waste, presenting both environmental and economic challenges.

FloDx, an innovative ultraportable wireless device, offers a groundbreaking solution to these issues. Designed for continuous monitoring of bacterial growth, FloDx eliminates the need for repetitive sampling, thereby reducing the risk of contamination and minimizing waste. The device's efficiency translates to significant savings in both time and cost, potentially reducing labour by up to 10,000 hours and cutting expenses by $2,000,000 annually—a tenfold reduction compared to conventional spectrophotometer-based methods.

This report details the development and implementation of a sophisticated growth model algorithm for FloDx, undertaken as part of an internship project. The primary objective is to enhance the predictive accuracy of bacterial growth monitoring using Xgboost, ensuring reliable and precise data output. Additionally, this research explores further optimization strategies for the prediction factors, aiming to maximize the operational efficiency and overall performance of the FloDx system.

Fundamentals Of Xgboost

Brief History of Xgboost

The development of advanced machine learning models has significantly enhanced the capabilities of predictive algorithms, particularly in the field of artificial intelligence. XGBoost represents a major advancement over traditional models by addressing several limitations inherent in earlier methods. The primary motivation behind XGBoost's creation was the need for a scalable and efficient boosting model capable of handling large datasets with greater speed and accuracy.

At the core of XGBoost is the decision tree, a foundational algorithm used for classification and regression tasks. Decision trees split data into smaller subsets based on input features, forming a tree-like structure where each branch represents a decision rule, and each leaf represents an outcome. Although decision trees are useful, they are often prone to overfitting, where the model performs well on training data but poorly on unseen data.

To overcome these limitations, ensemble methods like Random Forest and AdaBoost were introduced. Random Forest builds multiple decision trees using different subsets of data, aggregating their predictions to improve accuracy and reduce overfitting. AdaBoost, on the other hand, combines weak learners into a strong learner by focusing on the errors of previous models, though it uses simpler tree structures known as stumps.

Gradient Boosting further refined these concepts by using gradient descent to optimize model performance. It builds trees sequentially, where each tree corrects the errors of its predecessors. Despite its improvements, Gradient Boosting models can still suffer from overfitting if not properly tuned.

XGBoost enhances the Gradient Boosting framework by incorporating features like regularization and advanced tree pruning techniques to combat overfitting and improve model performance. These innovations make XGBoost a powerful and flexible tool for various machine learning tasks, offering significant advantages in terms of accuracy and computational efficiency. This report explores the implementation of XGBoost in the context of bacterial growth prediction, aiming to leverage its capabilities to advance monitoring technologies and improve predictive accuracy.

Mathematical Overview of Xgboost

XGBoost is an advanced ensemble learning algorithm that extends the capabilities of gradient boosting (GB). It integrates multiple weak learning models, specifically decision trees, to form a robust learner applicable to both regression and classification tasks. The algorithm iteratively refines the residuals of weak learners—defined as the discrepancies between predicted and observed values—enhancing overall model performance[1]. Notably, XGBoost is highly effective for handling sparse data, employing a sparsity-aware split-finding technique. It is distinguished by its unique objective function and the flexibility to select various loss functions. A significant advantage of XGBoost lies in its rapid and efficient processing of large datasets, facilitated by block technology. Moreover, the algorithm benefits from CPU multithreading for parallel processing, which incrementally improves accuracy through continuous algorithmic enhancements, a defining feature of XGBoost[1]. Figure 1 presents a flowchart of the XGBoost algorithm.

The outline of the XGBoost algorithm is presented in the following study[2]:

Given a dataset $\boldsymbol{D} = (\boldsymbol{x}_i, p_i)^n_{i=1}$ with n data points and m features, and a differentiable loss function, XGBoost defines the target function utilizing ensemble decision trees as follows:

$$\hat{p} = \Phi(x) \sum_{k=1}^{K} f_k (x_i) \ , f_k \in F \qquad (1)$$

Where $\boldsymbol{x}_i$ is the vector of an input feature in R^m, p_i is the actual target value in R predicted as $\hat{p}$, $F = \{fk(\boldsymbol{x}) = w_{q(\boldsymbol{x})}\}(q : \mathrm{R}^m \rightarrow T \ , w \in \mathrm{R}^T)$ is the space of decision trees in the model, and K is the total number of decision trees. fk represents a function in the functional space F.

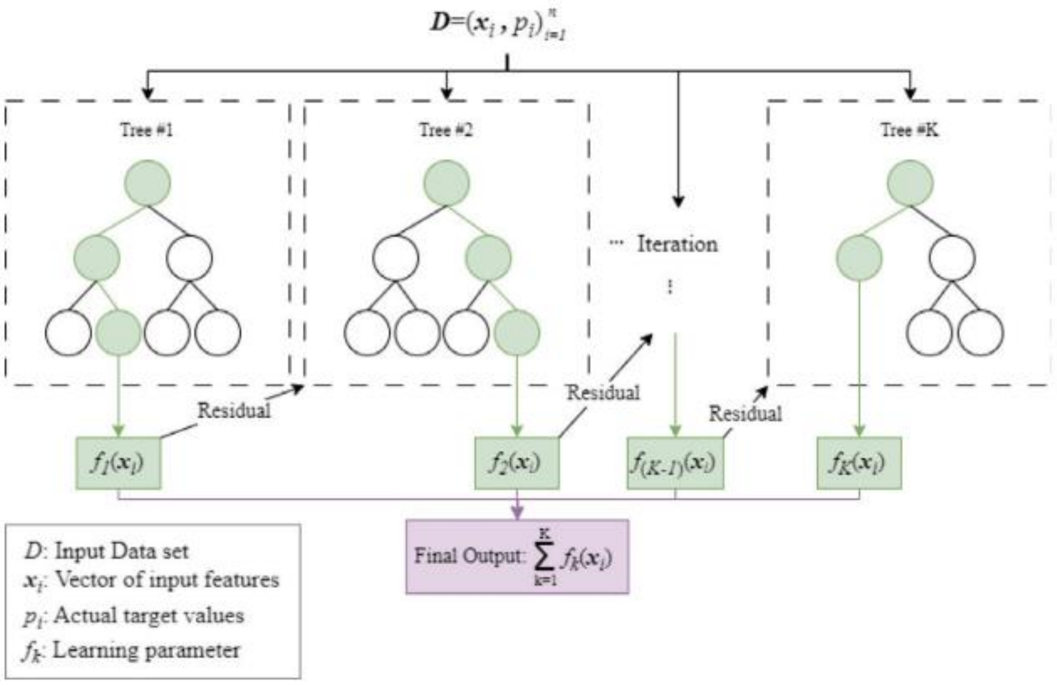


Fig. 1. XGBoost Flow Chart

In regression trees, the learning tree parameter corresponds to leaf weights *w* and an independent tree structure function *q(x*). Each tree in a regression model assigns a continuous score to each of its leaves, with *wi* representing the score on the *ith* leaf, and *T* denoting the total number of leaves. The tree structure function q(x) maps each data point to a specific leaf index.

XGBoost's unique objective function (L) comprises two primary components. The first is the training error term, which utilizes a loss function to measure the discrepancies between predicted and actual values, thereby evaluating the model's predictive accuracy. The second component is a regularization term aimed at mitigating overfitting by reducing model complexity.

$$L(\Phi) = \sum_{i=1}^{n} l(p_i\, \hat{p}_i) + \sum_{k=1}^{K} \varphi(f_k) \qquad (2)$$

$$where\ \varphi(f_k) = \gamma T + \frac{\lambda}{2}\sum_{j=1}^{T} w_j^2$$

where l is the differentiable loss function measuring the error between the predicted ($\hat{p}_i$) and actual (p_i) values for each data point, is the regularization term that adds a penalty for the complexity of the model, γ denotes the complexity of each leaf, λ is the l_2 weight regularization term that scales the regularization penalty, and (w_j) is the vector of scores on each leaf. When the regularization term $(\Phi(f_k)$ is zero, the objective function reverts to conventional gradient tree boosting.

The difference between the mathematical principles underlying regression and classification in XGBoost lies solely in the selection of the loss function. For regression, the loss function is $(\,l(p_i, \widehat{p_\iota}) = \frac{1}{2}(p_i - \widehat{p_\iota})^2\,)$, *whereas it is* $(\,l(p_i, \widehat{p_\iota}) = -[p_i \log(\widehat{p_\iota}) + (1 - p_i) \log(1 - \widehat{p_\iota})])$ for classification. Unlike conventional optimization techniques in the Euclidean space, simultaneous learning of all trees poses a considerable challenge, making it impractical to optimize the objective function.

Consequently, an alternative approach is employed where the model is trained in an additive manner. Specifically, the predicted value at step (t) is expressed as:

$$\hat{p}^{t} = \hat{p}^{(t-1)} + f_t(\boldsymbol{x_i}), \quad \text{for } t = 1,2,\dots,t \qquad (3)$$

Thus, the objective function at step (t) is formulated as:

$$\mathrm{L}^{(t)} = \sum_{i=1}^{n} l\left(p_i, \hat{p}^{(t-1)} + f_t(\boldsymbol{x_i})\right) + \sum_{k=1}^{t} \varphi(f_k) \qquad (4)$$

Utilizing the second-order Taylor series expansion, the objective function is approximated as:

$$\mathrm{L}^{(t)} \approx \sum_{i=1}^{n} \left[l\left(p_i, \hat{p}^{(t-1)}\right) + g_i w_q(\boldsymbol{x_i}) + \frac{1}{2} h_i w_q^2(\boldsymbol{x_i}) \right] + \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 + C \qquad (5)$$

Where $\left(g_i = \frac{\partial}{\partial p^{(t-1)}} l\left(p_i, \hat{p}^{(t-1)}\right)\right)$ and $\left(h_i = \frac{\partial^2}{\partial\left(\hat{p}^{(t-1)}\right)^2} l\left(p_i, \hat{p}^{(t-1)}\right)\right)$ represent the first and second derivatives of the loss function, respectively.

The algorithm aims to optimize the predicted value by minimizing the objective function. Thus, the constant terms C, $\left(p_i, \hat{p}^{(t-1)}\right)$, γT and can be omitted from Equation (5). Additionally, since each data point is associated with a specific leaf node, the loss function can be expressed as the sum of losses across all leaf nodes. Let (I_j) denote the set of data points in the jth leaf node. The objective function can then be approximated as:

$$\mathrm{L}^{(t)} \approx \sum_{j=1}^{T} \left[\sum_{i \in I_j} g_i w_j + \frac{1}{2} \left(\sum_{i \in I_j} h_i + \lambda \right) w_j^2 \right] \qquad (6)$$

For a tree with a fixed structure $(q(\boldsymbol{x}))$, setting the derivative of Equation (6) with respect to (w_j) to zero allows for the calculation of the optimal weight for the jth leaf $\left(w_j^*\right)$ and the corresponding optimal target value $\left(\mathcal{L}^t(q)\right)$ as follows:

$$w_j^* = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda} \qquad (7)$$

$$\mathcal{L}^t(q) = -\frac{1}{2} \sum_{j=1}^{T} \frac{\left(\sum_{i \in I_j} g_i\right)^2}{\sum_{i \in I_j} h_i + \lambda} + \gamma T \qquad (8)$$

Equation (8) represents the similarity score, which serves as a scoring function to evaluate the effectiveness of a tree structure in terms of quality, akin to the impurity score used in decision trees.

Finally, considering all possible tree structures is typically impractical. Therefore, XGBoost employs a greedy algorithm that starts with a single leaf and incrementally adds branches to build the tree. Let $(I = I_L \cup I_R)$, where (I_L) and (I_R) are the sets of instances for the left and right nodes resulting from a split, respectively. The objective function after the split can be expressed as:

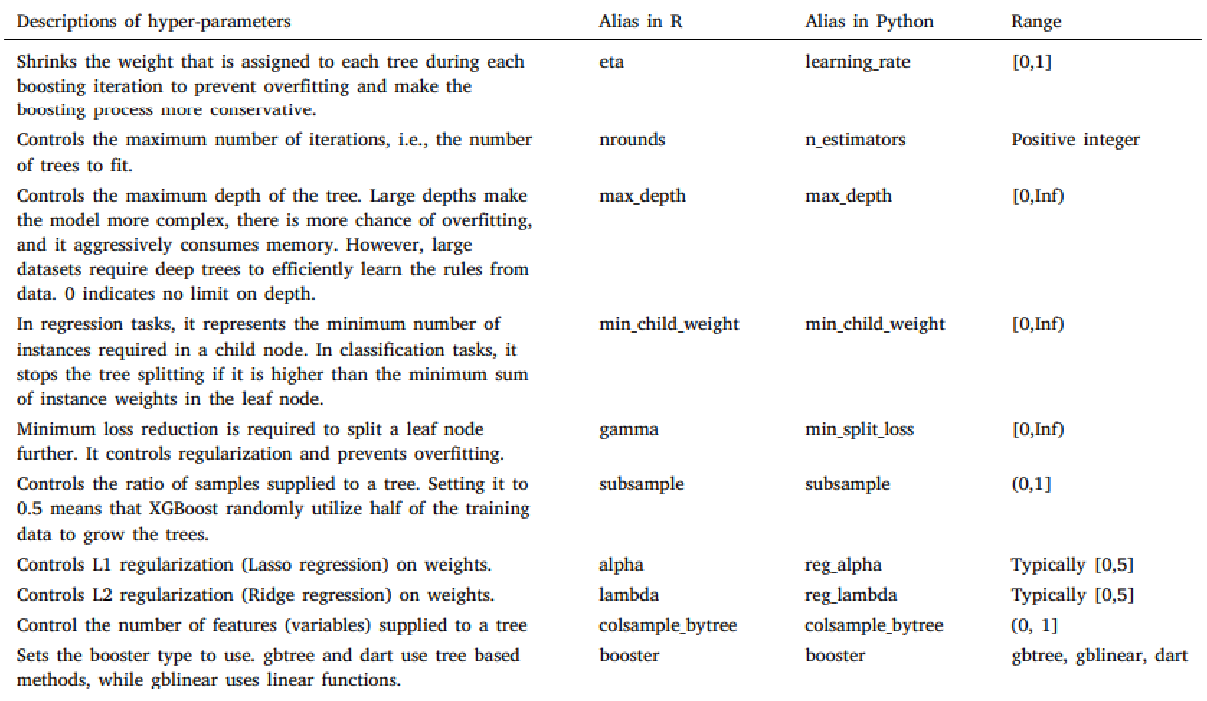
$$L_{\text{split}} = \frac{1}{2}\left[\frac{\left(\sum_{i\in I_L} g_i\right)^2}{\sum_{i\in I_L} h_i + \lambda} + \frac{\left(\sum_{i\in I_R} g_i\right)^2}{\sum_{i\in I_R} h_i + \lambda} - \frac{\left(\sum_{i\in I} g_i\right)^2}{\sum_{i\in I} h_i + \lambda}\right] + \gamma T \qquad (9)$$

Implementation

The XGBoost algorithm is primarily implemented in C++, providing a robust core for the algorithm. It offers bindings or interfaces for various programming languages, enabling users to interact with XGBoost using their preferred language. While the underlying algorithm remains consistent across different languages, there may be variations in how the Application Programming Interface (API) is utilized, depending on the language.

For example, training a model, setting parameters, or making predictions may differ between the Python API and those for R or Java. Some language-specific bindings may offer additional features or integrations. The Python API, for instance, might include extra utilities or compatibility with Python libraries, whereas the R API may feature functions designed to align with R's conventions.

Although XGBoost's performance is generally uniform across different languages, the efficiency of certain operations, such as data manipulation and input/output tasks, may vary depending on the language's performance characteristics and associated libraries. Additionally, the level of community support, availability of resources, tutorials, and documentation can differ across programming languages.

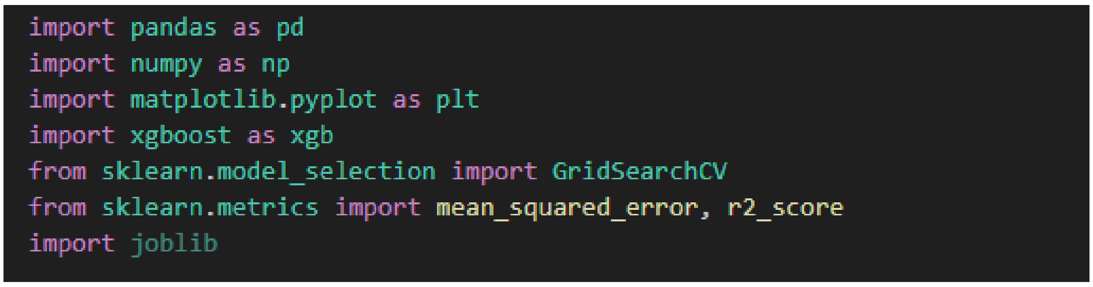


Descriptions of hyper-parameters	Alias in R	Alias in Python	Range
Shrinks the weight that is assigned to each tree during each boosting iteration to prevent overfitting and make the boosting process more conservative.	eta	learning_rate	[0,1]
Controls the maximum number of iterations, i.e., the number of trees to fit.	nrounds	n_estimators	Positive integer
Controls the maximum depth of the tree. Large depths make the model more complex, there is more chance of overfitting, and it aggressively consumes memory. However, large datasets require deep trees to efficiently learn the rules from data. 0 indicates no limit on depth.	max_depth	max_depth	[0,Inf)
In regression tasks, it represents the minimum number of instances required in a child node. In classification tasks, it stops the tree splitting if it is higher than the minimum sum of instance weights in the leaf node.	min_child_weight	min_child_weight	[0,Inf)
Minimum loss reduction is required to split a leaf node further. It controls regularization and prevents overfitting.	gamma	min_split_loss	[0,Inf)
Controls the ratio of samples supplied to a tree. Setting it to 0.5 means that XGBoost randomly utilize half of the training data to grow the trees.	subsample	subsample	(0,1]
Controls L1 regularization (Lasso regression) on weights.	alpha	reg_alpha	Typically [0,5]
Controls L2 regularization (Ridge regression) on weights.	lambda	reg_lambda	Typically [0,5]
Control the number of features (variables) supplied to a tree	colsample_bytree	colsample_bytree	(0, 1]
Sets the booster type to use. gbtree and dart use tree based methods, while gblinear uses linear functions.	booster	booster	gbtree, gblinear, dart

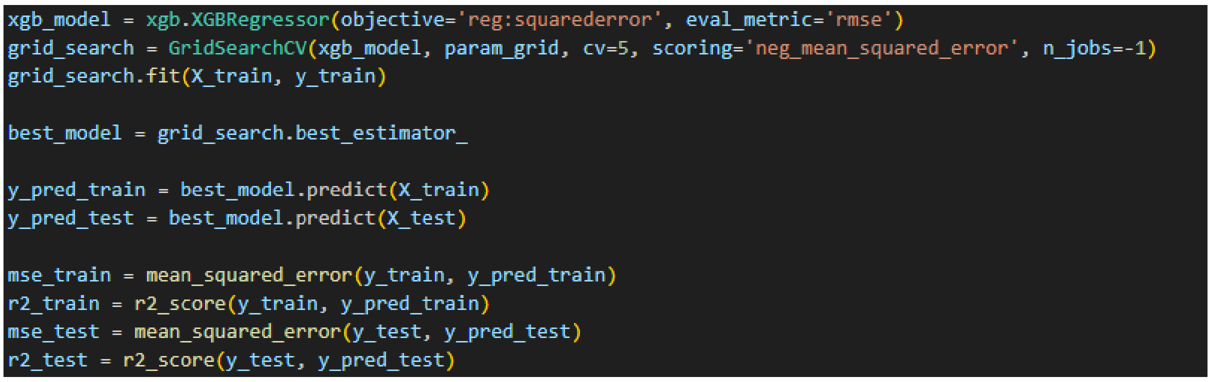
Fig. 2. Brief Outline of common hyper-parameters used to tune XGBoost

Python and R are two widely used programming languages for implementing XGBoost in water studies. Both languages leverage the XGBoost library (or package) available at [XGBoost Documentation]. In addition, XGBoost offers the `xgboost.sklearn` API in Python, which is compatible with scikit-learn [scikit-learn]. This API facilitates the integration of XGBoost into existing scikit-learn pipelines and workflows.

From the Xgboost Model File here’s an example of the implementation of the code:



```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import xgboost as xgb
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error, r2_score
import joblib
```



```python
xgb_model = xgb.XGBRegressor(objective='reg:squarederror', eval_metric='rmse')
grid_search = GridSearchCV(xgb_model, param_grid, cv=5, scoring='neg_mean_squared_error', n_jobs=-1)
grid_search.fit(X_train, y_train)

best_model = grid_search.best_estimator_

y_pred_train = best_model.predict(X_train)
y_pred_test = best_model.predict(X_test)

mse_train = mean_squared_error(y_train, y_pred_train)
r2_train = r2_score(y_train, y_pred_train)
mse_test = mean_squared_error(y_test, y_pred_test)
r2_test = r2_score(y_test, y_pred_test)
```

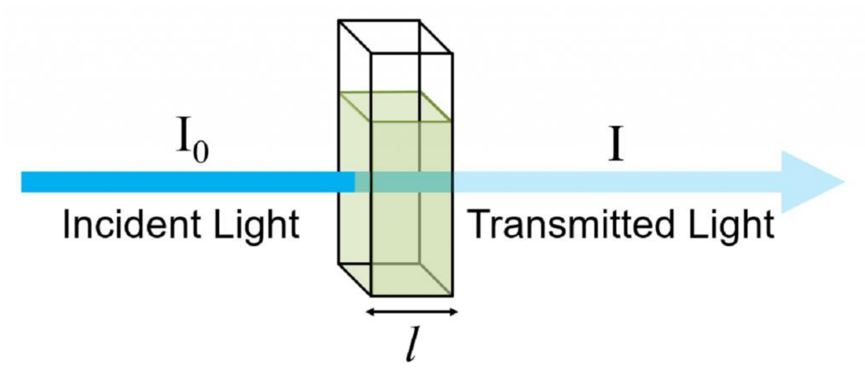
To enhance the performance of XGBoost for a particular task, it is crucial to fine-tune several key hyperparameters. Figure 2 provides a summary of commonly adjusted hyperparameters used in water-related studies. Tuning other hyperparameters of XGBoost not covered in this report generally does not have a substantial impact on the overall results.

FloDx

Background

Monitoring microbial growth patterns generally requires ongoing observation to ensure that populations stay within predefined limits or are maintained at desired levels. Traditionally, methods such as plate counting, direct counting, biomass measurement, and light scattering have been used to gauge bacterial growth. However, optical density (OD) measurements are now regarded as the most efficient technique for quantifying bacterial growth.

Despite its effectiveness, optical density measurement presents its own set of challenges. A key consideration is determining the optimal wavelength of light for accurate sample assessment. According to Beer-Lambert's Law, there is a linear relationship between the absorbance of a substance and its concentration, which underscores the importance of selecting the appropriate wavelength for precise measurements.



As the incident ray of light passes through the solution, a portion of it is absorbed, resulting in reduced light intensity exiting the opposite side. This phenomenon reflects the material's capacity to attenuate light, which is related to how effectively a beam of photons is absorbed as it penetrates the solution.

In the context of bacterial growth, the attenuation factor exhibits exponential growth due to the dynamic nature of bacterial proliferation. To address this, we solve the corresponding ordinary differential equation using an integrating factor, which yields a general solution to accurately model the attenuation behaviour.

$$d\Phi_z = -\mu_z \Phi_z dz \quad (10)$$

$$\mu = \frac{1}{\Phi_e}\frac{d\Phi}{dz} = attenuation\ coefficent$$

$$Integrating\ Factor: e^{\int_0^z \mu z' dz'}$$

$$\frac{d\Phi_{ez}}{dz} e^{\int_0^z \mu z\prime dz\prime} + \mu_z \Phi_z e^{\int_0^z \mu z\prime dz\prime}$$

$$\frac{d}{dz}\left[\Phi(z)^{\int_0^z \mu z\prime dz\prime}\right] = 0$$

$$\Phi_T = \Phi_I e^{-\int_0^l \mu z\prime dz\prime}$$

$$T = \frac{\Phi_T}{\Phi_I} = e^{-\int_0^l \mu(z) dz} \quad (11)$$

Since our focus is on cross-sectional attenuation for FloDx, we are interested in the total reduction in intensity as the signal passes through the entire length of the test tube.

$$\sigma_i = \frac{\mu_z(z)}{n_i(z)} = \frac{attenuation\ coefficent}{number\ densities}$$

$$\mu_{10} = \frac{\mu}{Log10} = exponential\ folding\ volume$$

$$\therefore T = \frac{\Phi_T}{\Phi_I} = e^{-\sum_{i=1}^{n} \sigma_i \int_0^l n_i(z) dz} \quad (12)$$

By applying Beer-Lambert's Law to account for both the concentration of bacteria within the solution and the cross-sectional length of the test tube, we obtain:

$$A = \log\frac{\Phi_T}{\Phi_I} = \varepsilon cl$$

OR

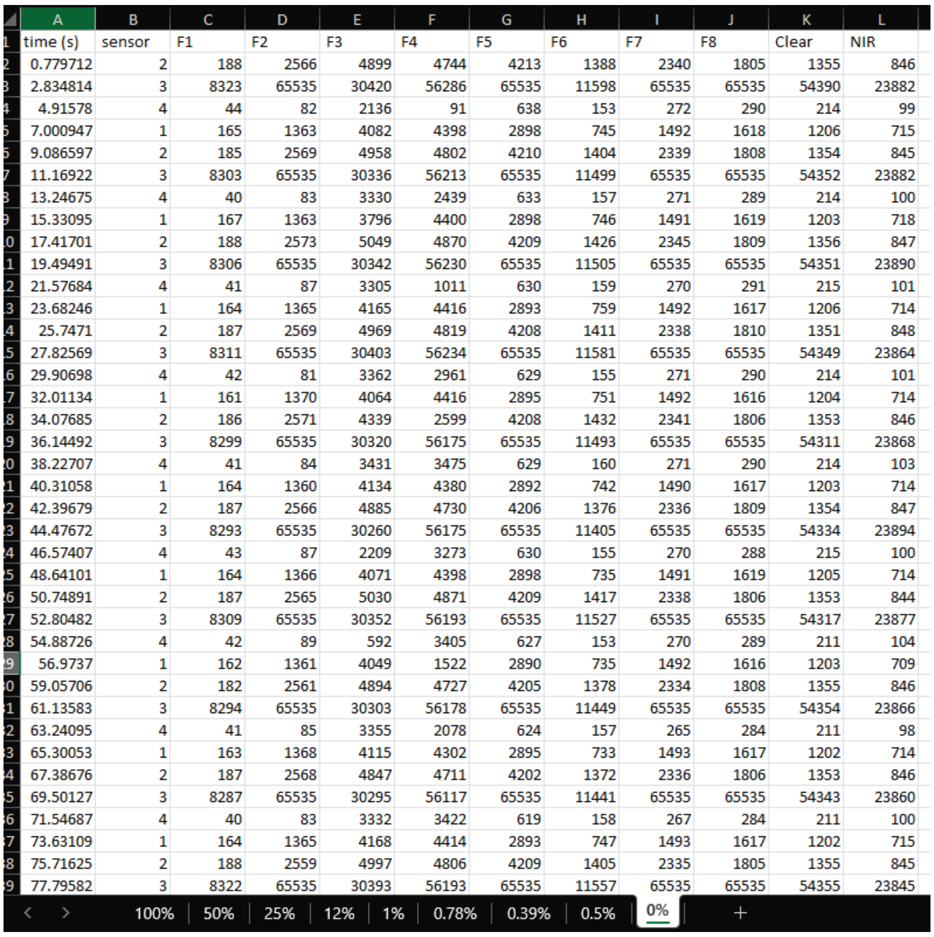
$$OD = -\log 10 \frac{\Phi_T}{\Phi_I} \quad (13)$$

Calculating Optimal Wavelength – Task 1

Research Method

Using Equation 13, we can process data from two FloDX devices, each equipped with four sensors. These sensors measure white light across eight different frequencies. This setup enhances the variability and accuracy of optical density (OD) measurements. Our objective is to identify the optimal wavelengths detected by the photoelectric sensors and determine which wavelength is most effective for detecting bacterial growth.

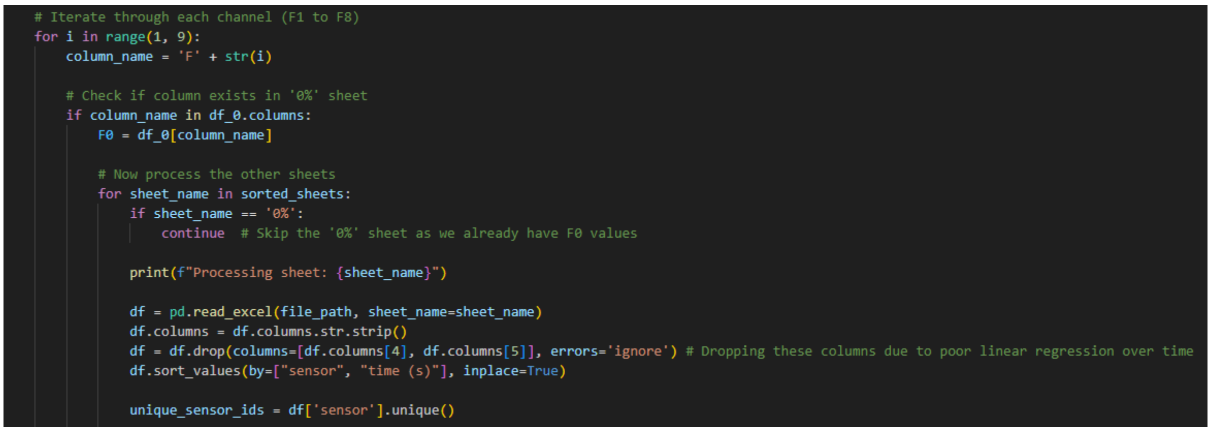
For our data analysis, we were provided with two data files containing different concentrations and arbitrary readings representing optical density (OD). Given the substantial volume of data and the XLSX format used by the devices, a specialized sorting software was developed. This program efficiently visualizes key data by calibrating OD for each column and sorting through the data for each concentration, ensuring the information remains easily readable for future reference.



time (s)	sensor	F1	F2	F3	F4	F5	F6	F7	F8	Clear	NIR
0.779712	2	188	2566	4899	4744	4213	1388	2340	1805	1355	846
2.834814	3	8323	65535	30420	56286	65535	11598	65535	65535	54390	23882
4.91578	4	44	82	2136	91	638	153	272	290	214	99
7.000947	1	165	1363	4082	4398	2898	745	1492	1618	1206	715
9.086597	2	185	2569	4958	4802	4210	1404	2339	1808	1354	845
11.16922	3	8303	65535	30336	56213	65535	11499	65535	65535	54352	23882
13.24675	4	40	83	3330	2439	633	157	271	289	214	100
15.33095	1	167	1363	3796	4400	2898	746	1491	1619	1203	718
17.41701	2	188	2573	5049	4870	4209	1426	2345	1809	1356	847
19.49491	3	8306	65535	30342	56230	65535	11505	65535	65535	54351	23890
21.57684	4	41	87	3305	1011	630	159	270	291	215	101
23.68246	1	164	1365	4165	4416	2893	759	1492	1617	1206	714
25.7471	2	187	2569	4969	4819	4208	1411	2338	1810	1351	848
27.82569	3	8311	65535	30403	56234	65535	11581	65535	65535	54349	23864
29.90698	4	42	81	3362	2961	629	155	271	290	214	101
32.01134	1	161	1370	4064	4416	2895	751	1492	1616	1204	714
34.07685	2	186	2571	4339	2599	4208	1432	2341	1806	1353	846
36.14492	3	8299	65535	30320	56175	65535	11493	65535	65535	54311	23868
38.22707	4	41	84	3431	3475	629	160	271	290	214	103
40.31058	1	164	1360	4134	4380	2892	742	1490	1617	1203	714
42.39679	2	187	2566	4885	4730	4206	1376	2336	1809	1354	847
44.47672	3	8293	65535	30260	56175	65535	11405	65535	65535	54334	23894
46.57407	4	43	87	2209	3273	630	155	270	288	215	100
48.64101	1	164	1366	4071	4398	2898	735	1491	1619	1205	714
50.74891	2	187	2565	5030	4871	4209	1417	2338	1806	1353	844
52.80482	3	8309	65535	30352	56193	65535	11527	65535	65535	54317	23877
54.88726	4	42	89	592	3405	627	153	270	289	211	104
56.9737	1	162	1361	4049	1522	2890	735	1492	1616	1203	709
59.05706	2	182	2561	4894	4727	4205	1378	2334	1808	1355	846
61.13583	3	8294	65535	30303	56178	65535	11449	65535	65535	54354	23866
63.24095	4	41	85	3355	2078	624	157	265	284	211	98
65.30053	1	163	1368	4115	4302	2895	733	1493	1617	1202	714
67.38676	2	187	2568	4847	4711	4202	1372	2336	1806	1353	846
69.50127	3	8287	65535	30295	56117	65535	11441	65535	65535	54343	23860
71.54687	4	40	83	3332	3422	619	158	267	284	211	100
73.63109	1	164	1365	4168	4414	2893	747	1493	1617	1202	715
75.71625	2	188	2559	4997	4806	4209	1405	2335	1805	1355	845
77.79582	3	8322	65535	30393	56193	65535	11557	65535	65535	54355	23845

Fig. 3. Unsorted Device 1 data

To facilitate data processing, we initially standardized the column headings by removing any extraneous whitespace. Subsequently, we utilized the Pandas library to systematically sort the data by sensor for each column. This approach ensures consistency and ease of processing for future datasets.

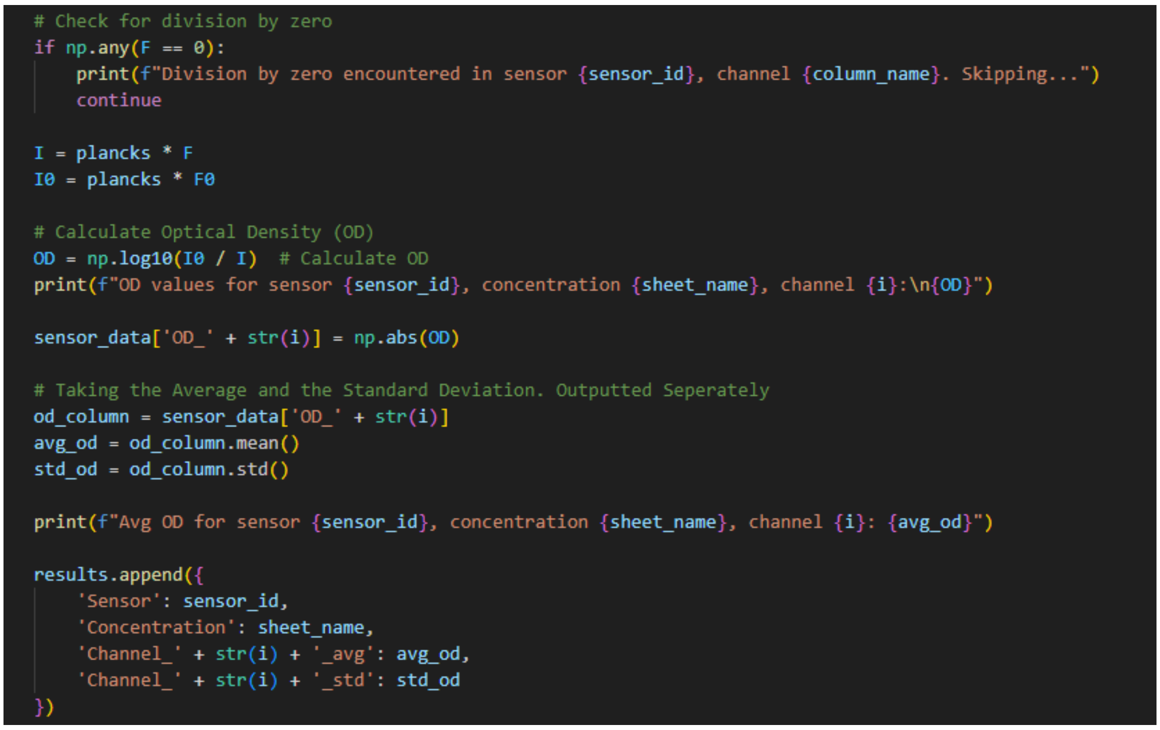


```python
# Iterate through each channel (F1 to F8)
for i in range(1, 9):
    column_name = 'F' + str(i)

    # Check if column exists in '0%' sheet
    if column_name in df_0.columns:
        F0 = df_0[column_name]

        # Now process the other sheets
        for sheet_name in sorted_sheets:
            if sheet_name == '0%':
                continue  # Skip the '0%' sheet as we already have F0 values

            print(f"Processing sheet: {sheet_name}")

            df = pd.read_excel(file_path, sheet_name=sheet_name)
            df.columns = df.columns.str.strip()
            df = df.drop(columns=[df.columns[4], df.columns[5]], errors='ignore') # Dropping these columns due to poor linear regression over time
            df.sort_values(by=["sensor", "time (s)"], inplace=True)

            unique_sensor_ids = df['sensor'].unique()
```

Upon standardizing the column headings and sorting the data by sensor, we applied Equation 13 to each data point within the DataFrame. This transformation converts the previously arbitrary data points into precise optical density (OD) values, thereby enhancing the accuracy and reliability of the dataset.

By obtaining true OD values, we can now generate multiple plots that illustrate the OD against time for each concentration across all sensors. These plots are instrumental in visualizing the dynamics of bacterial growth over time and across varying concentrations. Through this detailed visualization, we can identify trends and anomalies that might otherwise go unnoticed.

Furthermore, this comprehensive approach allows us to determine the optimal wavelength for detecting bacterial growth. By analysing the OD values across different wavelengths, we can identify which wavelengths yield the most accurate and consistent measurements. This is crucial for improving the sensitivity and accuracy of our measurements, ultimately contributing to more reliable and robust experimental outcomes.



```
# Check for division by zero
if np.any(F == 0):
    print(f"Division by zero encountered in sensor {sensor_id}, channel {column_name}. Skipping...")
    continue

I = plancks * F
I0 = plancks * F0

# Calculate Optical Density (OD)
OD = np.log10(I0 / I)  # Calculate OD
print(f"OD values for sensor {sensor_id}, concentration {sheet_name}, channel {i}:\n{OD}")

sensor_data['OD_' + str(i)] = np.abs(OD)

# Taking the Average and the Standard Deviation. Outputted Seperately
od_column = sensor_data['OD_' + str(i)]
avg_od = od_column.mean()
std_od = od_column.std()

print(f"Avg OD for sensor {sensor_id}, concentration {sheet_name}, channel {i}: {avg_od}")

results.append({
    'Sensor': sensor_id,
    'Concentration': sheet_name,
    'Channel_' + str(i) + '_avg': avg_od,
    'Channel_' + str(i) + '_std': std_od
})
```

Results

The software was executed on both datasets, and upon generating the plots, a significant error was identified within the graphs. As an example, for 0.5% concentration for SensorID_4, it was evident that channels 3 and 4 exhibited irregular behaviour. These channels displayed consistent dips in their readings, not only at the 0.5% concentration but across all concentrations examined. This anomaly suggests potential issues with the sensors or the data collection process for these specific channels.

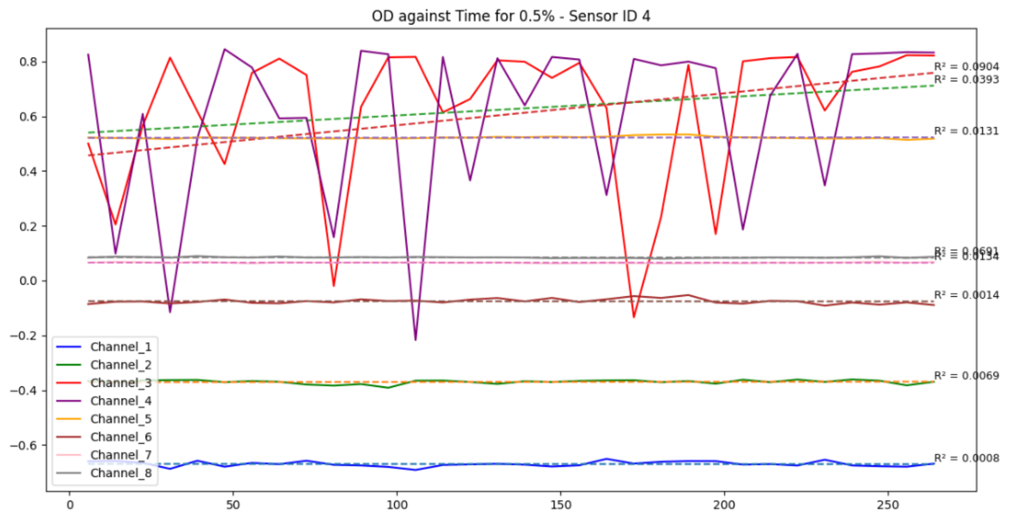


Fig. 4. 0.5% concentration SensorID_4

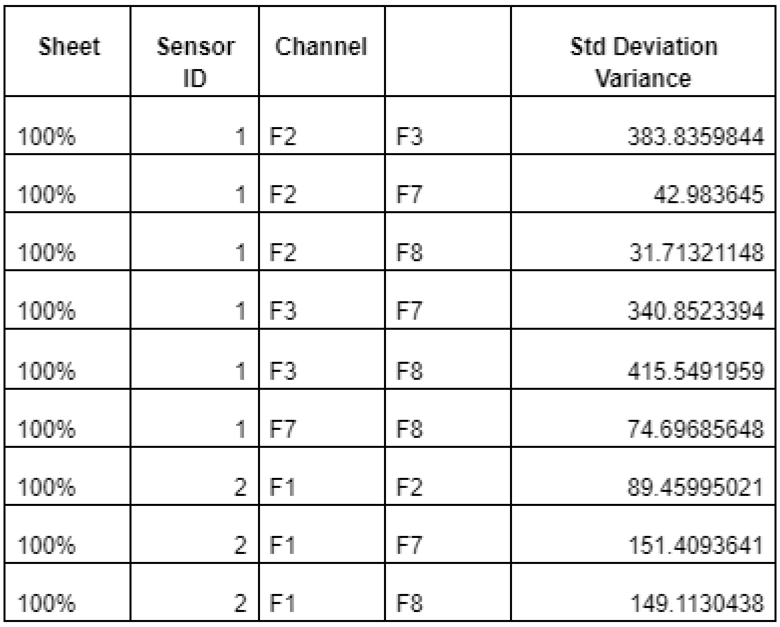
Further investigation revealed that channels 3 and 4 were experiencing mutual interference. The program logic employed a blinking algorithm, where alternating blinks of white LEDs cycled through each channel. Residual effects from the timing of these blinks were detected by channels 3 and 4, leading to the observed irregularities. Specifically, whenever one channel spiked, the other followed within less than a second. To ensure the accuracy and reliability of the data collected, channels 3 and 4 will be excluded from further use.

After adjusting the code to exclusively read data from channels 1, 2, and 5 through 8, we addressed the variance between each permutation of channels by calculating the standard deviation. This statistical approach allowed us to accurately measure and correct for any inconsistencies across the different channel readings, ensuring a more reliable and precise dataset.

	Channel 3 (hz)	Channel 4 (hz)
Sensor 1	3544.896552	4075.827586
Sensor 2	4880.965517	4439.448276
Sensor 3	30333.55172	56173.89655
Sensor 4	2938.137931	2954.586207

Fig. 5. Average Frequency Across All Concentrations

The dataset sample (Fig. 6.) revealed significant deviation across each channel. To address this issue, a data sorting algorithm was developed to perform several key tasks. First, values not directly associated with the dataset, such as '%', and '()', were stripped and sorted. Next, the data frame was filtered to isolate the initial concentration (e.g., 0%). The optical density (OD) was then calculated against various concentrations. Finally, the calculated data was stored in a new pivoted data frame and saved as an .xlsx file for further analysis.



Sheet	Sensor ID	Channel		Std Deviation Variance
100%	1	F2	F3	383.8359844
100%	1	F2	F7	42.983645
100%	1	F2	F8	31.71321148
100%	1	F3	F7	340.8523394
100%	1	F3	F8	415.5491959
100%	1	F7	F8	74.69685648
100%	2	F1	F2	89.45995021
100%	2	F1	F7	151.4093641
100%	2	F1	F8	149.1130438

Fig. 6. Data(Head) of Set Sample

While the initial graphs appeared promising, subsequent plots exhibited a downward trend, indicating that OD was decreasing as concentration increased. This outcome directly contradicts Beer-Lambert's Law, necessitating a review and correction of the sorting methodology.

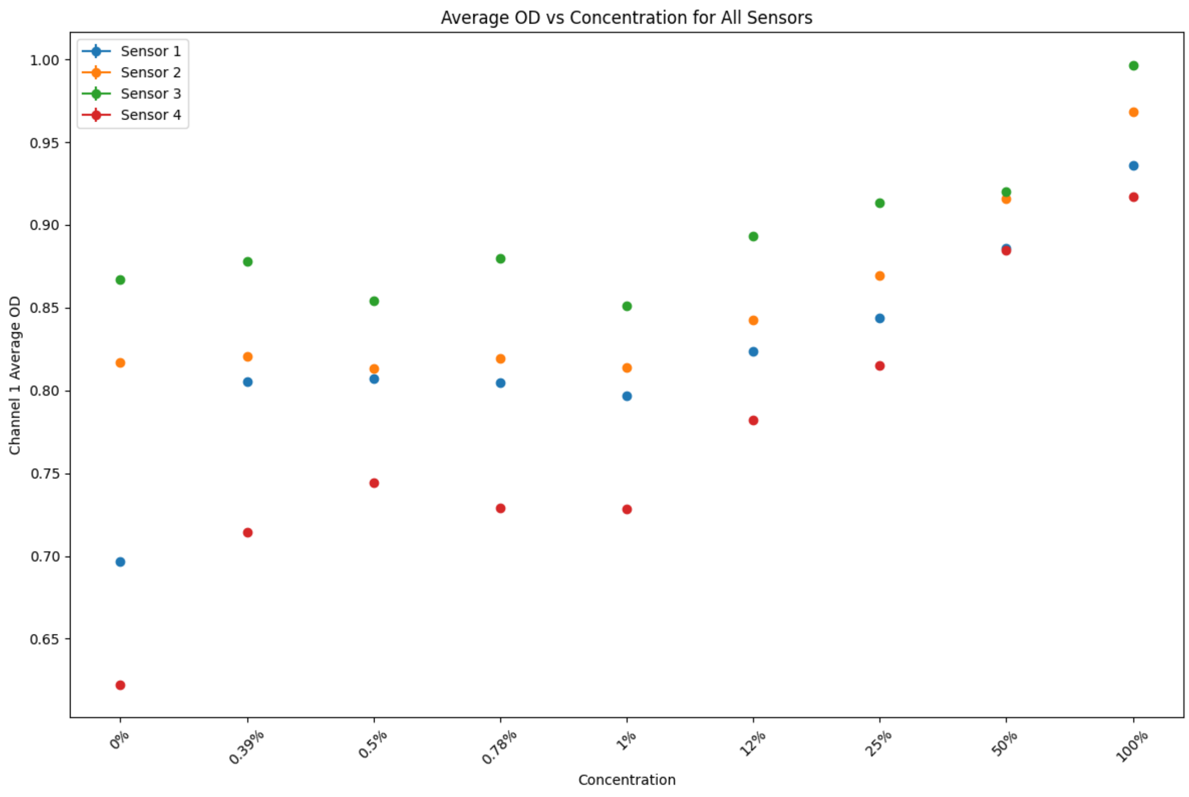
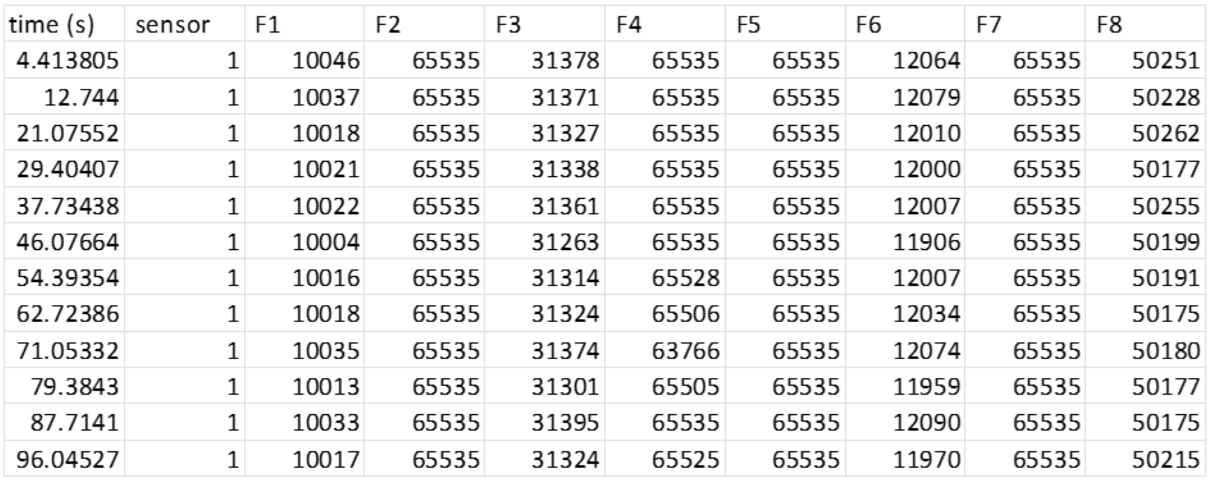


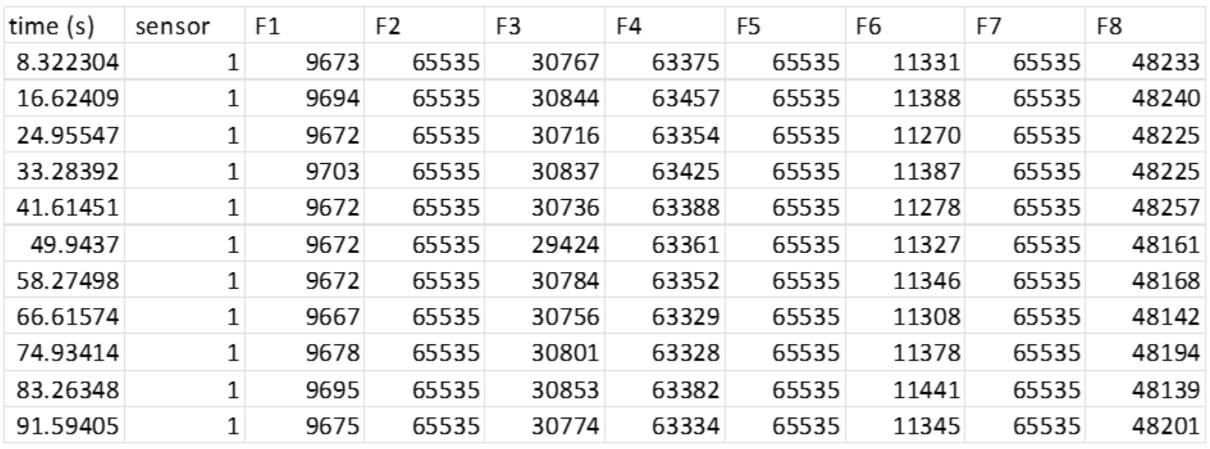
Fig. 7. Device 2 Failed OD vs Concentration Channel 1

After a thorough inspection of the sorting algorithm and the raw dataset, a key source of error was identified. It was observed that channels 2, 4, 5, and 7 within the raw dataset were contributing significantly to this error. Detailed analysis revealed that these channels exhibited irregularities that were not present in the other channels, leading to inconsistencies in the data processing and subsequent calculations.



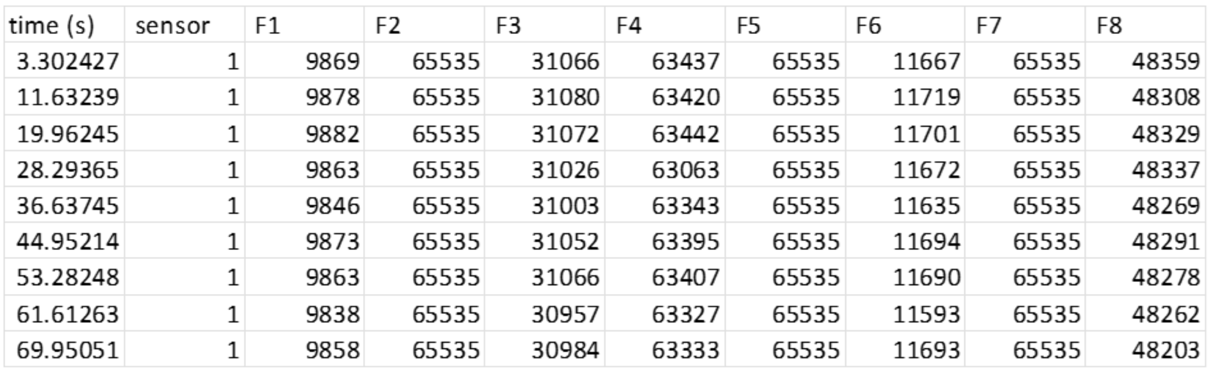
time (s)	sensor	F1	F2	F3	F4	F5	F6	F7	F8
4.413805	1	10046	65535	31378	65535	65535	12064	65535	50251
12.744	1	10037	65535	31371	65535	65535	12079	65535	50228
21.07552	1	10018	65535	31327	65535	65535	12010	65535	50262
29.40407	1	10021	65535	31338	65535	65535	12000	65535	50177
37.73438	1	10022	65535	31361	65535	65535	12007	65535	50255
46.07664	1	10004	65535	31263	65535	65535	11906	65535	50199
54.39354	1	10016	65535	31314	65528	65535	12007	65535	50191
62.72386	1	10018	65535	31324	65506	65535	12034	65535	50175
71.05332	1	10035	65535	31374	63766	65535	12074	65535	50180
79.3843	1	10013	65535	31301	65505	65535	11959	65535	50177
87.7141	1	10033	65535	31395	65535	65535	12090	65535	50175
96.04527	1	10017	65535	31324	65525	65535	11970	65535	50215

Fig. 8. Device 2- Sorted, Concentration 0%



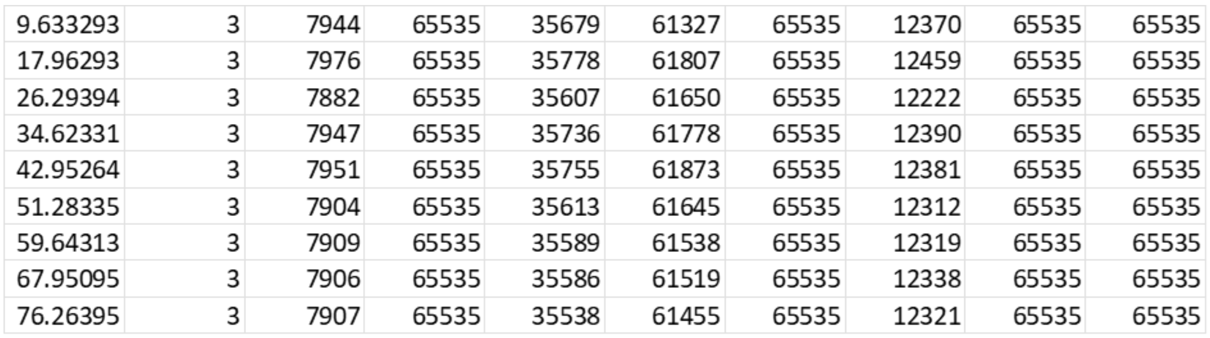
time (s)	sensor	F1	F2	F3	F4	F5	F6	F7	F8
8.322304	1	9673	65535	30767	63375	65535	11331	65535	48233
16.62409	1	9694	65535	30844	63457	65535	11388	65535	48240
24.95547	1	9672	65535	30716	63354	65535	11270	65535	48225
33.28392	1	9703	65535	30837	63425	65535	11387	65535	48225
41.61451	1	9672	65535	30736	63388	65535	11278	65535	48257
49.9437	1	9672	65535	29424	63361	65535	11327	65535	48161
58.27498	1	9672	65535	30784	63352	65535	11346	65535	48168
66.61574	1	9667	65535	30756	63329	65535	11308	65535	48142
74.93414	1	9678	65535	30801	63328	65535	11378	65535	48194
83.26348	1	9695	65535	30853	63382	65535	11441	65535	48139
91.59405	1	9675	65535	30774	63334	65535	11345	65535	48201

Fig. 9. Device 2- Sorted, Concentration 0.39%



time (s)	sensor	F1	F2	F3	F4	F5	F6	F7	F8
3.302427	1	9869	65535	31066	63437	65535	11667	65535	48359
11.63239	1	9878	65535	31080	63420	65535	11719	65535	48308
19.96245	1	9882	65535	31072	63442	65535	11701	65535	48329
28.29365	1	9863	65535	31026	63063	65535	11672	65535	48337
36.63745	1	9846	65535	31003	63343	65535	11635	65535	48269
44.95214	1	9873	65535	31052	63395	65535	11694	65535	48291
53.28248	1	9863	65535	31066	63407	65535	11690	65535	48278
61.61263	1	9838	65535	30957	63327	65535	11593	65535	48262
69.95051	1	9858	65535	30984	63333	65535	11693	65535	48203

Fig. 10. Device 2- Sorted, Concentration 1%



9.633293	3	7944	65535	35679	61327	65535	12370	65535	65535
17.96293	3	7976	65535	35778	61807	65535	12459	65535	65535
26.29394	3	7882	65535	35607	61650	65535	12222	65535	65535
34.62331	3	7947	65535	35736	61778	65535	12390	65535	65535
42.95264	3	7951	65535	35755	61873	65535	12381	65535	65535
51.28335	3	7904	65535	35613	61645	65535	12312	65535	65535
59.64313	3	7909	65535	35589	61538	65535	12319	65535	65535
67.95095	3	7906	65535	35586	61519	65535	12338	65535	65535
76.26395	3	7907	65535	35538	61455	65535	12321	65535	65535

Fig. 11. Device 2- Sorted, Concentration 50%

Although these are small snippets of data, they clearly demonstrate that the raw output data of the sensor consistently saturates at a certain point. This saturation behaviour is not unique to one device but is also observed in Device 1. Interestingly, Device 1 shows a pattern where the data, instead of remaining continuously saturated, experiences a drop before returning to its saturation point. This fluctuation suggests a potential issue with the sensor's response dynamics or its interaction with the measured environment.

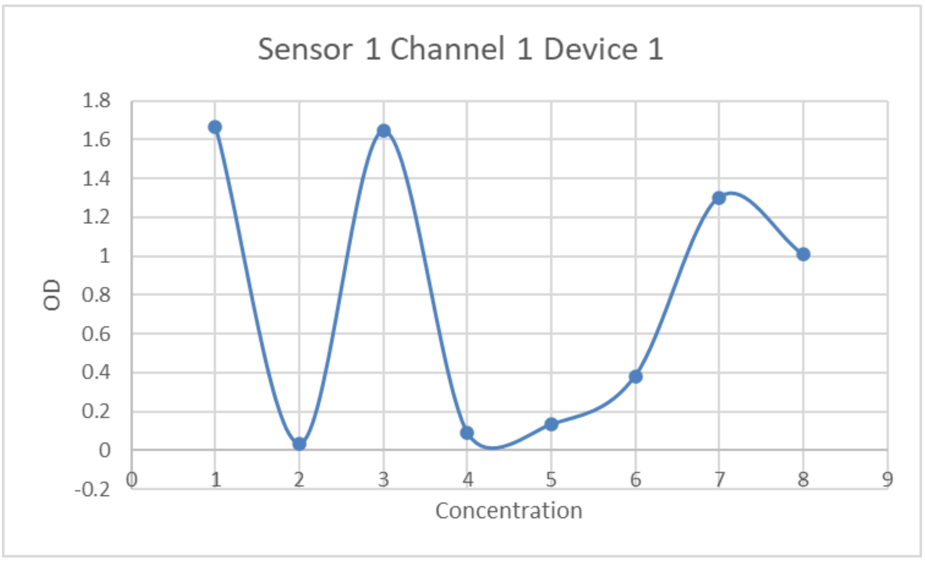
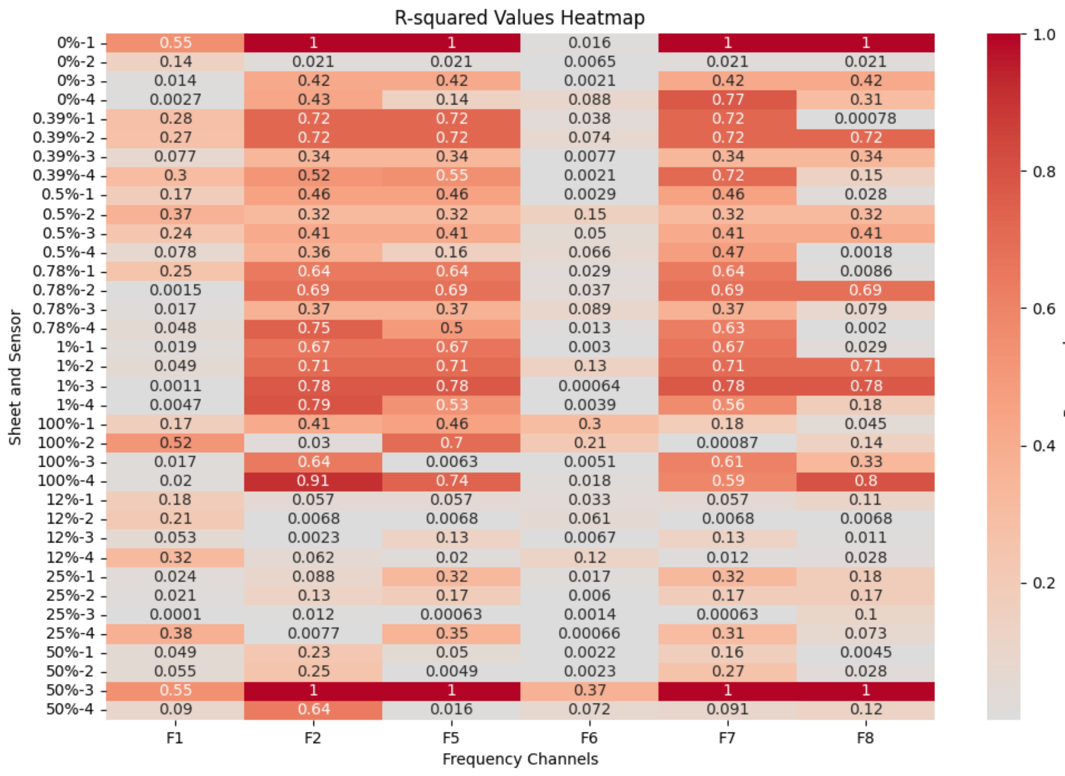


Fig. 12. Observed Fluctuations of OD across all Concentrations

Understanding this saturation and drop patterns is critical for interpreting sensor behaviour and ensuring accurate data collection in future models. Further investigation into the underlying causes of these anomalies is essential for improving sensor performance and reliability. These findings indicate that all sensors are failing to some degree, highlighting the need for comprehensive diagnostics and possible recalibration or redesign of the sensors to mitigate these issues.



Sheet and Sensor	F1	F2	F5	F6	F7	F8
0%-1	0.55	1	1	0.016	1	1
0%-2	0.14	0.021	0.021	0.0065	0.021	0.021
0%-3	0.014	0.42	0.42	0.0021	0.42	0.42
0%-4	0.0027	0.43	0.14	0.088	0.77	0.31
0.39%-1	0.28	0.72	0.72	0.038	0.72	0.00078
0.39%-2	0.27	0.72	0.72	0.074	0.72	0.72
0.39%-3	0.077	0.34	0.34	0.0077	0.34	0.34
0.39%-4	0.3	0.52	0.55	0.0021	0.72	0.15
0.5%-1	0.17	0.46	0.46	0.0029	0.46	0.028
0.5%-2	0.37	0.32	0.32	0.15	0.32	0.32
0.5%-3	0.24	0.41	0.41	0.05	0.41	0.41
0.5%-4	0.078	0.36	0.16	0.066	0.47	0.0018
0.78%-1	0.25	0.64	0.64	0.029	0.64	0.0086
0.78%-2	0.0015	0.69	0.69	0.037	0.69	0.69
0.78%-3	0.017	0.37	0.37	0.089	0.37	0.079
0.78%-4	0.048	0.75	0.5	0.013	0.63	0.002
1%-1	0.019	0.67	0.67	0.003	0.67	0.029
1%-2	0.049	0.71	0.71	0.13	0.71	0.71
1%-3	0.0011	0.78	0.78	0.00064	0.78	0.78
1%-4	0.0047	0.79	0.53	0.0039	0.56	0.18
100%-1	0.17	0.41	0.46	0.3	0.18	0.045
100%-2	0.52	0.03	0.7	0.21	0.00087	0.14
100%-3	0.017	0.64	0.0063	0.0051	0.61	0.33
100%-4	0.02	0.91	0.74	0.018	0.59	0.8
12%-1	0.18	0.057	0.057	0.033	0.057	0.11
12%-2	0.21	0.0068	0.0068	0.061	0.0068	0.0068
12%-3	0.053	0.0023	0.13	0.0067	0.13	0.011
12%-4	0.32	0.062	0.02	0.12	0.012	0.028
25%-1	0.024	0.088	0.32	0.017	0.32	0.18
25%-2	0.021	0.13	0.17	0.006	0.17	0.17
25%-3	0.0001	0.012	0.00063	0.0014	0.00063	0.1
25%-4	0.38	0.0077	0.35	0.00066	0.31	0.073
50%-1	0.049	0.23	0.05	0.0022	0.16	0.0045
50%-2	0.055	0.25	0.0049	0.0023	0.27	0.028
50%-3	0.55	1	1	0.37	1	1
50%-4	0.09	0.64	0.016	0.072	0.091	0.12

Fig. 13. R^2 Heatmap

The observations presented in Figure 12 further support this conclusion. Despite the errors present in the raw data, the results from the functioning channels confirm that the software is operating correctly. Figure 12 offers a comprehensive view of performance across various concentrations and frequencies, enabling the identification of the most effective combinations.

Specifically, Figure 12 indicates that channels 1 and 6 exhibited optimal performance throughout the experiment. This finding is consistent with the trends observed in Figures 8 through 11, which show that channels 1 and 6 consistently outperformed others. The data suggests that, for this instance of the project, the ideal channel frequency range is between 11,000 and 7,000 units, which appears to maximize performance under the given conditions.

Discussion and Conclusion

The insights derived from the current study are crucial for advancing the experimental process and optimizing sensor settings in future prototypes. The analysis indicates that channels 1 and 6 performed optimally, with the ideal frequency range falling between 11,000 and 7,000 units. This information provides a valuable starting point for enhancing the accuracy and effectiveness of experimental setups, particularly in refining sensor calibration and improving data collection methods.

However, several limitations must be acknowledged. The current analysis is based on only two available channels, which restricts the ability to generalize the findings across a broader range of channel-frequency combinations. Additionally, the presence of varying issues and inconsistencies within the datasets prevents the establishment of a definitive, universally optimal channel-frequency combination. These inconsistencies highlight the need for more extensive testing and data collection to better understand the sensor performance and address the underlying problems affecting data reliability.

In conclusion, while the study provides important insights into the performance of channels 1 and 6 and identifies a promising frequency range, it is essential to recognize the limitations of the current analysis. The inability to recommend a definitive channel-frequency combination underscores the need for further research. Future studies should focus on expanding the range of channels and frequencies tested, addressing dataset inconsistencies, and refining the experimental methodologies. By doing so, more robust guidelines can be established, ultimately leading to improved regression performance and more accurate experimental results.

XGBoost Model

Model Design

Designing an initial model for FloDx presents several significant challenges that must be addressed before the model can be effectively developed. The primary challenge lies in defining the parameters and variables that will drive the predictive capabilities of the model. To begin with, it is crucial to clarify the specific objectives of the prediction. For instance, if a user introduces a sample containing an unknown bacterium in an unknown medium, the goal is to predict both the growth rate of the bacterium and the optimal time at which it will reach a measurable growth stage. This task is complicated by the vast diversity of bacterial species—over 10,000 distinct species and many more strains—alongside a similarly extensive range of growth mediums.

Given the complexity of predicting bacterial growth in such a scenario, a more manageable approach involves simplifying the problem. Instead of dealing with two unknowns, we can modify the scope to predict the growth of an unknown bacterium in a known medium. This approach allows us to reduce the number of variables in our model, as the medium's parameters are already defined. Consequently, we only need to focus on a single variable—the growth rate of the bacterium in this specific medium.

By adopting this simplified model, we can parameterize the problem more effectively. We can input the known medium's characteristics into our objective function, as specified in Equation 2, and use this data to construct a training set (p_i). This refined approach provides a more practical framework for developing and validating the model, enabling us to generate more accurate predictions for bacterial growth under defined conditions. The training set, derived from known mediums and various bacterial growth rates, will serve as the foundation for calibrating the model and improving its predictive accuracy.

The following flow chart illustrates the proposed design for integrating all the necessary features to ensure accurate predictions.

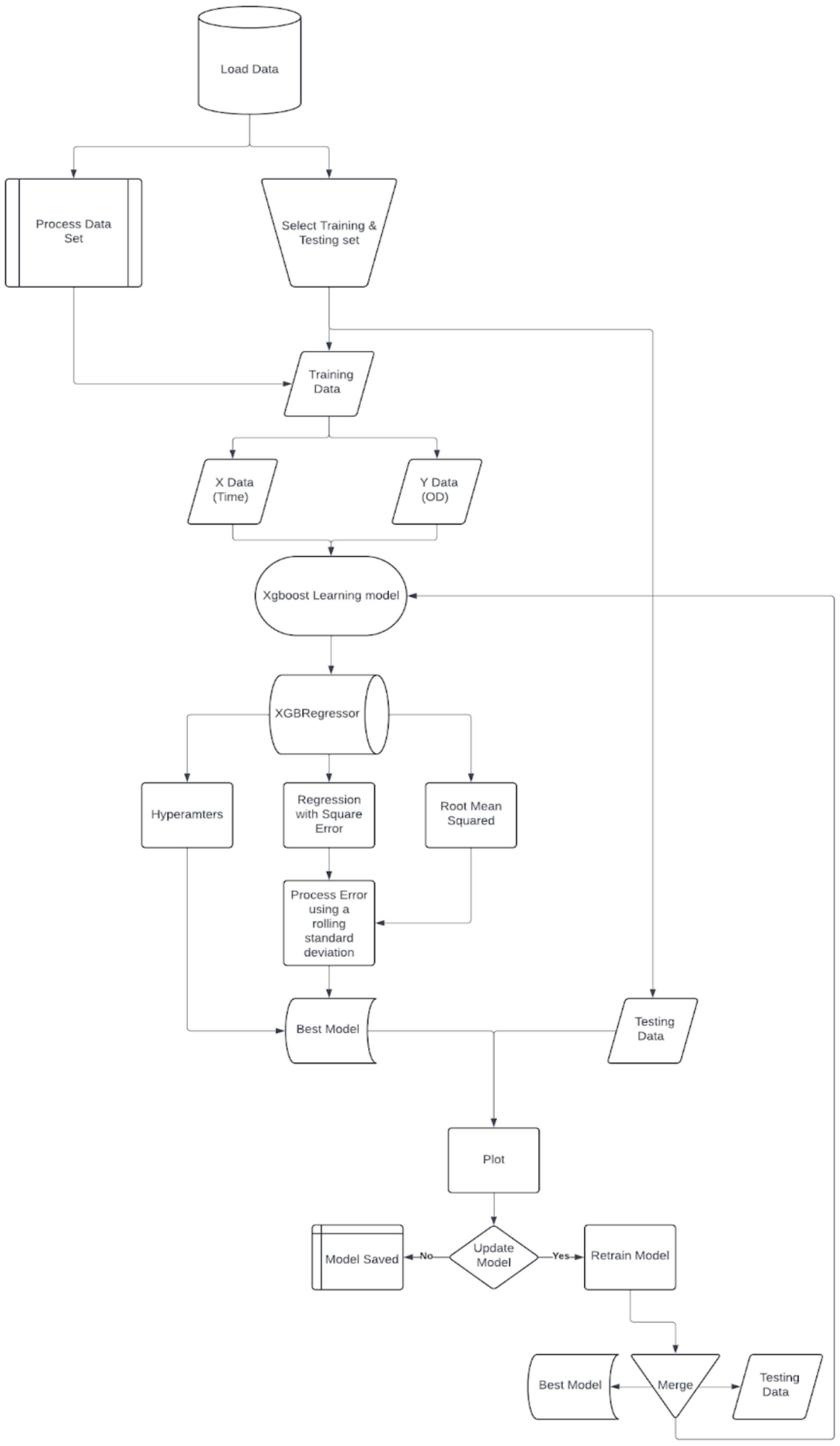


Fig. 14. Flow Chart of Proposed Model

Once the optimal model has been established using the current dataset, we can proceed to visualize its performance using Matplotlib. This visualization step is crucial as it allows us to comprehensively evaluate the model's accuracy and performance when compared against a chosen dataset. Through these visual assessments, we can identify any discrepancies or areas where the model may require further refinement.

Following this evaluation, FlowDx provides the user with two distinct options. The first option is to retain the calibrated model in its current state for recording future data. This approach is suitable when the model's performance is deemed satisfactory, and its predictions align closely with the observed data.

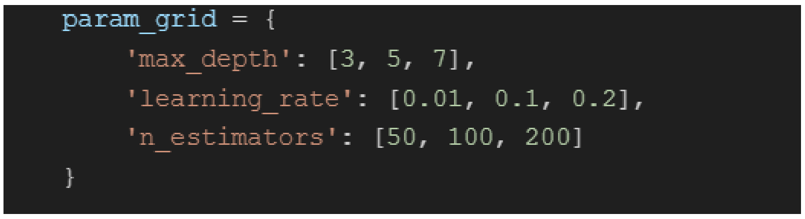
The second option involves retraining the model using the newly designated control dataset. This process entails merging the original dataset with the new data, thereby creating a more comprehensive dataset that can enhance the model's predictive capabilities. By incorporating additional data, the model can be further calibrated to improve its accuracy and reliability in predicting the growth of the user's selected bacterium.

The decision to retain the current model or proceed with retraining is significant, as it directly impacts the model's future performance and its ability to provide accurate predictions. Opting for retraining and merging datasets can lead to a more robust and finely tuned model, which is better equipped to handle variations and new data points. Ultimately, this iterative process of evaluation, calibration, and retraining is essential for developing a reliable predictive model that meets the specific needs and requirements of the user and thereby giving the user full calibration control of the device.

Creating Test Models using experimental data

E Coli Growth (Reduced Genomes) in 3 different Media

Test data were collected from multiple sources to ensure a comprehensive analysis, focusing on Escherichia coli (E. coli) K-12 strains with reduced genomes cultivated in three distinct media: LB, MAA, and M63 where each round of testing is separated by number of sheets within each file. Despite the genomic reductions in these strains, our analysis concentrated exclusively on their growth curves, thereby excluding the genomic details from consideration. To handle this extensive dataset effectively, specific parameters for the machine learning model were determined, with the chosen parameters being max_depth and learning_rate with the number of estimations per data point to run from 50-200. These parameters were selected to optimize the learning process, enabling accurate modelling and prediction of the growth dynamics of the E. coli strains across the different media.



```
param_grid = {
    'max_depth': [3, 5, 7],
    'learning_rate': [0.01, 0.1, 0.2],
    'n_estimators': [50, 100, 200]
}
```

Note: Although adding more learning parameters could potentially enhance the accuracy of the XGBoost model, it would also increase the computational load, leading to longer run times. This could result in the program either crashing or taking an extended period to generate a plot.

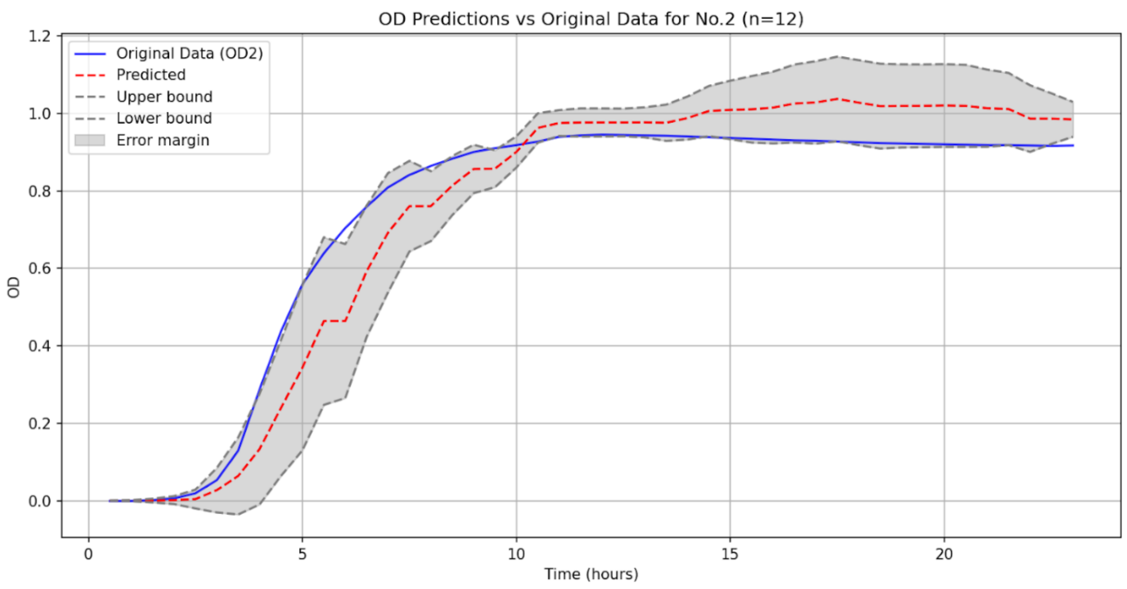


Fig.15. Initial Predictions (Medium: LB)

Figure 15 illustrates the intial predicted growth response based on a single dataset (Sheet 1). The graph subsequently plots the error margin of the XGBoost model, calculated using Equation 9, against a separate dataset (Sheet 2). This comparison specifically involves column 1, highlighting the model's performance when applied to a different strain of E. coli grown in the same medium (LB). Although the medium remains consistent, the growth patterns differ due to the variability between the two E. coli strains.

Although the test data performed within XGBoost's expectations, the model failed to produce accurate predictions in some areas of the graph, indicating it is not yet "perfect." To improve accuracy, we reran the program, incorporating both the test data and the original model data into the algorithm.

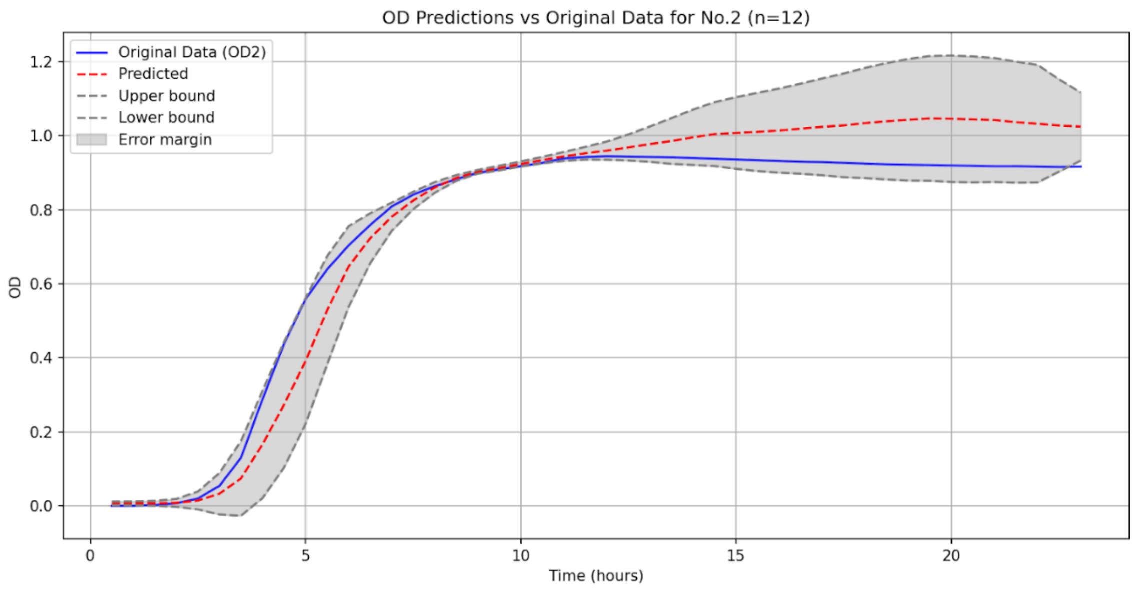


Fig.16 New Predictions

The improved model, as shown in Figure 16, reveals several important observations. While the accuracy of the predictions has increased, the model exhibits a significant contraction in the 6 to 10-hour growth region. This suggests that if the experiment were repeated, there would be a 95% confidence that the growth of E. coli would fall within this narrowed range. This represents a notable limitation of the current model.

It is essential to recognize that not all data used in model training is necessarily beneficial. In this case, the integration of test data with the original model resulted in a heavily skewed graph, indicating inaccuracies in the predicted growth of E. coli. This discrepancy may stem from various factors, such as sensor malfunctions or issues during incubation.

To address this issue in future work, it is important to carefully select a suitable control dataset for XGBoost and consider whether a hyper-tuned model, like the one depicted in Figure 16, is appropriate. The specific use cases will influence these decisions, making it a valuable consideration for future experiments.

Initial Observations

The overall performance of the model for this dataset can be considered a success. The model effectively ingests the data and performs according to the user-defined specifications, demonstrating its ability to meet the intended objectives. This performance is evidenced by its predictions, which align well with the expected outcomes based on the provided data.

When compared to predictions (Figure 17) and results presented in graphs (Figure 16) from other research papers, the model's outputs are notably promising. The consistency and accuracy of the model's predictions not only validate its reliability but also place it on par with, or even exceed, the standards observed in existing literature. This comparison highlights the model's robustness and its potential as a valuable tool for similar analyses. The alignment with established research underscores the model's effectiveness and provides confidence in its applicability for future studies and practical applications.

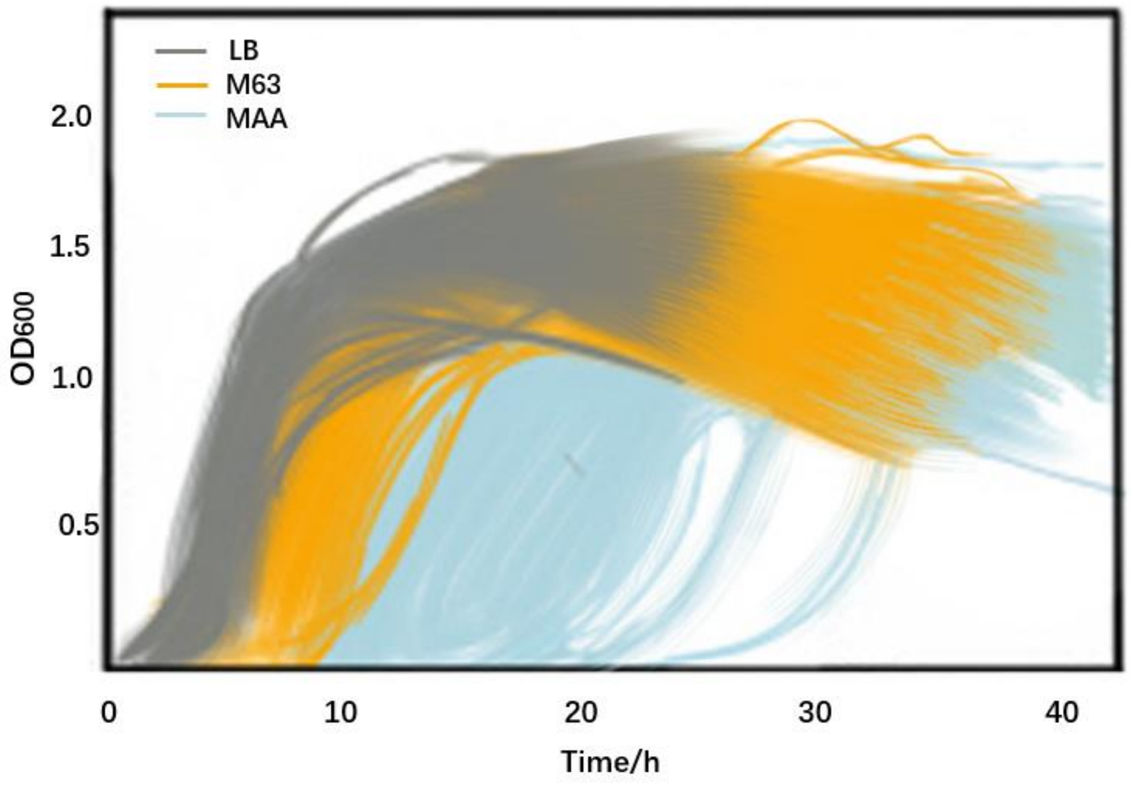


Fig.16 Graph of E-Coli Growth From JLU-China 2022

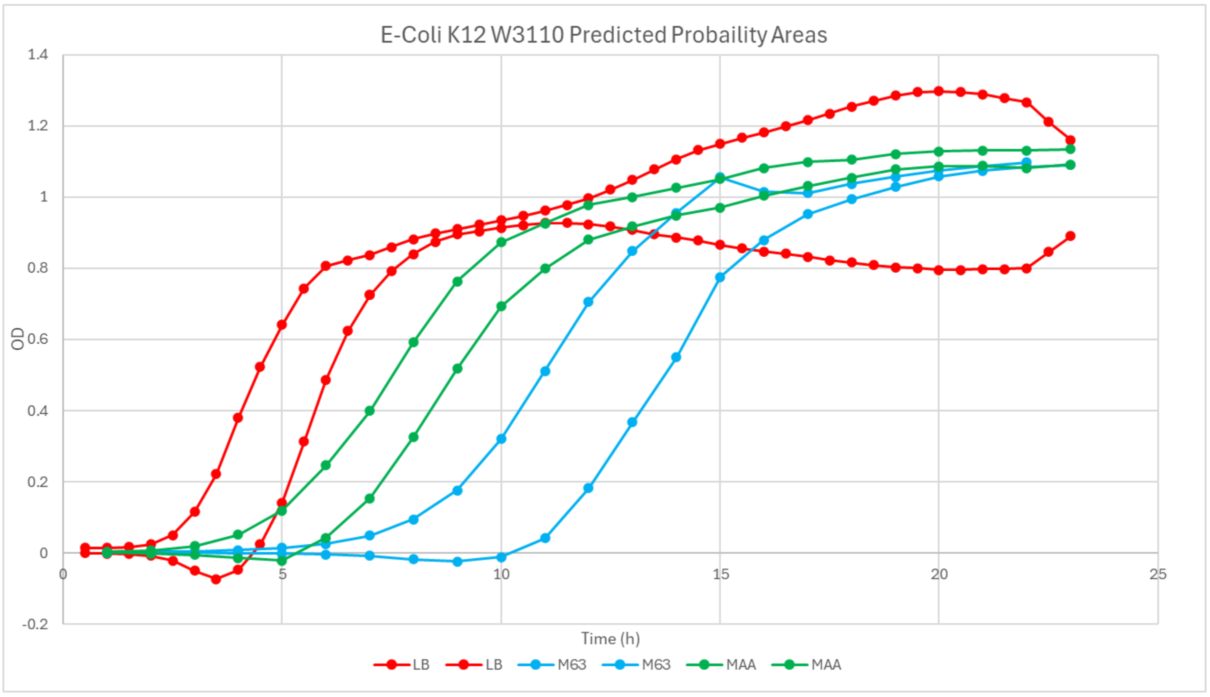


Fig. 17. Prediction area of all strains (reduced genome) in 3 different media

Comparing Models

Traditional (Mathematical) Modelling

Traditional modelling techniques have long been a cornerstone of research across various scientific disciplines. These established methods offer a foundational approach to understanding complex systems and phenomena by applying well-defined mathematical frameworks and analytical tools. Researchers frequently rely on these techniques for their robustness, interpretability, and the extensive body of knowledge that supports their application. Despite their proven efficacy, traditional modelling methods often face limitations in handling the increasing complexity and volume of modern datasets.

From several papers E-coli is by far the most tested and researched bacterium for prediction with this we can derive the data sets to contrast the existing Xgboost algorithm. The following models were tested using a 2 data sets (1 large 1 small):

- Logistic Regression
- Gompertz

Logistic Regression

Logistic regression is a widely utilized statistical method for modelling binary outcomes and is particularly valued for its simplicity and interpretability. It is a type of regression analysis used when the dependent variable is categorical and specifically binary, such as the presence or absence of a condition, success or failure, or yes or no responses.

The fundamental principle behind logistic regression is to model the probability of a certain class or event occurring based on one or more predictor variables. Unlike linear regression, which predicts a continuous outcome, logistic regression estimates the probability of a binary outcome using the logistic function. This function, also known as the sigmoid function, maps predicted values to a range between 0 and 1, making it suitable for binary classification tasks.

The logistic function is defined as:

$$[p = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n)}}] \qquad (10)$$

where (p) is the probability of the dependent event occurring, (β_0) is the intercept, $(\beta_1, \beta_2, \dots, \beta_n)$ are the coefficients of the predictor variables $(x_1, x_2, \dots, x_n)$.

To model bacterial growth more effectively, we will use a modified version of this equation that better represents our specific variables:

Logistic regression is particularly advantageous due to its ability to provide clear probabilistic interpretations of predictions. It also facilitates the assessment of the relationships between predictor variables and the probability of the outcome event. The coefficients obtained from logistic regression can be interpreted in terms of odds ratios, which represent the change in odds of the outcome occurring for a one-unit change in a predictor variable, holding all other variables constant.

Additionally, logistic regression is relatively straightforward to implement and computationally efficient, making it a popular choice for many practical applications.