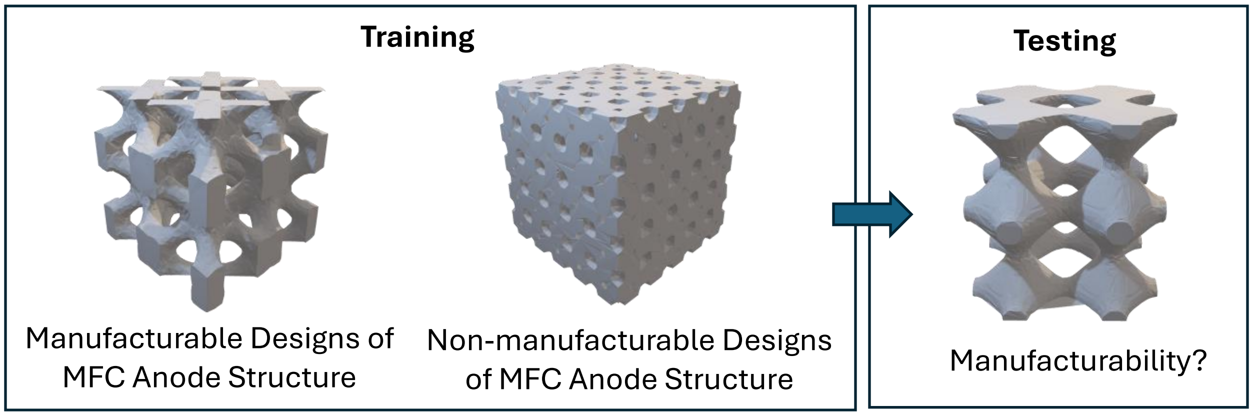
— Call for Participation —

Manufacturing AI Competition

Sponsored by Virginia Tech Academy of Data Science and
INFORMS Quality, Statistics, and Reliability Section
2024 INFORMS Annual Meeting (October 20 – 23, 2024)
Seattle, Washington, USA
Website: https://informsqsr2024challengewebsite.streamlit.app/

Problem Statement

A manufacturing designer generates feasible designs and infeasible designs of Microbial Fuel Cell Anode Structure. The design variables, generated 3D geometry data in STL files, and the manufacturability are provided, refer to the appendix for dataset structure. Please use ALL samples to train an AI model to classify the manufacturability indicator.


Training
Manufacturable Designs of MFC Anode Structure
Non-manufacturable Designs of MFC Anode Structure
Testing
Manufacturability?

There are no limitations on AI or statistic models to be used but methodological innovation is highly encouraged. Please use the 3D data from STL files to predict the binary manufacturing response. Please also discuss your feature engineering and the modeling strategies to process the 3D data. Data can be downloaded on the **website** after user registration. Sample code and the environment information can be found in the appendix. A few 3D data pre-processing packages can be found in the Appendix. Please cite [1] [2] [3] to reference the data source. Data reuse and redistribution is not allowed without written consent.

Submission Procedure

This is a team-based competition. Each team should have a maximum of 4 members from academia or industry. Students' participation is highly encouraged. Python code with required formats, the trained serialized model object (see the detailed instructions in the Appendix), and the result summary (up to 10 slides) should be submitted on the website online. Multiple submissions will be allowed for performance tests online, but only the last submission from the team will be used for competition. No email communications will be needed, however, if faced with any difficulties or have questions, please refer to the contact details of the organizers provided on the website.

Evaluation Procedure

The performance of the algorithms will be automatically evaluated on an undisclosed testing dataset generated from the same design simulation engine with the same parameter settings. The primary evaluation metric will be the F1 score. Accuracy, Type 1 error, and Type 2 error will also be considered as secondary metrics. The leaderboard will display the rankings based on the F1 score. Three finalists will be selected based on the highest F1 scores achieved on the undisclosed testing dataset. In the event of a tie in the F1 score, the secondary metrics (accuracy, Type 1 error, and Type 2 error) will be used to break the tie. If there is still a tie after considering all the metrics, the novelty of the algorithm and the quality of the submitted report will be considered to determine the finalists. The three selected finalists will be invited to present their methodology and results in an in-person session at the INFORMS 2024 Annual Meeting. After the presentations, one winning team will be selected by the judges, who are the organizers of the competition (Dr. Tom Woteki, Dr. Xiaoyu Chen, Dr. Ran Jin, Dr. Ismini Lourentzou, and Dr. Hongyue Sun). The judges will evaluate the finalists based on their presentations, the novelty and effectiveness of their algorithms, and the quality of their submitted reports. The winning team and all finalists will be recognized in the INFORMS-QSR business meeting

Important Dates

- Submission Deadline: August 19, 2024
- Notification of finalists: September 2, 2024
- Presentations as a session in INFORMS annual meeting: October 20 – 23, 2024

Organizers

Dr. Tom Woteki, Virgnia Tech, drwo@vt.edu
Dr. Xiaoyu Chen, University at Buffalo, xchen325@buffalo.edu
Dr. Ran Jin, Virginia Tech, jran5@vt.edu (Contact Person)
Dr. Ismini Lourentzou, University of Illinois Urbana-Champaign, lourent2@illinois.edu
Dr. Hongyue Sun, University of Georgia, hongyuesun@uga.edu
Technical Support: Mr. Premith Chilukuri, Virginia Tech, cpremithkumar@vt.edu

Email to noreply.qsr24@gmail.com if you have any questions or concerns.

Acknowledgment

This competition is partially supported by NSF projects CMMI-2208864 and CMMI-2331985.

References:

1. Zeng, Y., Chilukuri, PK., Zhou, X., Lourentzou, I., & Jin, R. (2024). Performance-Oriented Representation Learning in Directed Acyclic Graph Neural Network for High-quality Dataset Sharing. In Manuscript.
2. Zeng, Y. (2024) Data Exchange for Artificial Intelligence Incubation in Manufacturing Industrial Internet (Doctoral dissertation, Virginia Tech)
3. P. K. Chilukuri, B. Song, S. Kang, and R. Jin, Generating Optimized 3D Designs for Manufacturing Using a Guided Voxel Diffusion Model, in Proc. ASME 2024 Int. Manuf. Sci. Eng. Conf., MSEC2024, Knoxville, TN, USA, Jun. 17-21, 2024, MSEC2024-125075

Appendix

Programming Language:

ONLY Python3 programming language will be accepted by the online evaluation platform. The operating system is Ubuntu >= 20.04.6 LTS.

Deep Learning Frameworks:

We will support PyTorch==latest (>= 2.1.0) primarily, if other libraries are used, then please write a through and efficient config.yaml, so that we can setup a virtual environment to automatically evaluate your submitted model (more details below). For details on how to create a config.yaml, please refer to this link: https://saturncloud.io/blog/how-to-install-packages-from-yaml-file-in-conda-a-guide/ . Please check your config file, make sure that it's error prone. Also, **Only CPU versions of the deep learning libraries will be used for evaluation**.

Please submit a requirements.txt or **config.yaml (preferred)** document to specify the modules and the corresponding versions you would install to run the model. The platform will install these automatically. In the config or requirements file, please do not forget to include the libraries (along with corresponding versions) used for 3D data processing, transformation, and loading.

The test module in the script should contain the relevant 3D data formatting, transformation, and loading logic which was used during the model training. Please also include the code for loading the trained model. This test script will be used to evaluate your model performance, so make sure the test script is fool proof.

Note: Please include comments wherever it is necessary. If you are planning to transform the given 3D data into a different representation for training the model, the same data transformation logic should be implemented in the test script part of template.py

Warning: installing incompatible modules may crash the virtual environment.

Note:

In some cases, if the trained weight file is larger than 200MB, then the participants should upload their weights to a google drive and then use the "gdown" library in the test script to automatically download the weights and load them into the model.

Sample Script (to download online weights and then load them into the model) to help you get started:

```
import os
import gdown
import torch

# URL of the weights file on Google Drive
```

```python
weights_url = 'https://drive.google.com/uc?id=XXXXXXXXXXXXXXXXXXXXXXXXX'

# Destination path to save the weights file
destination = 'model_weights.pth'

# Download the weights file from Google Drive
gdown.download(weights_url, destination, quiet=False)

# Load the PyTorch model architecture
model = torch.load('model_architecture.pth')

# Load the downloaded weights into the model
model.load_state_dict(torch.load(destination))

# Set the model to evaluation mode
model.eval()

print("Weights loaded successfully!")
```

Python Script Format:
Please submit your Python module with one entrance script following the below mentioned template.py template, make sure you submit the training code and a dedicated testing code separately. The test data set structure is detailed below, make sure that your code follows this structure so that it can successfully load the test data and the trained weights.

Dataset Format:
Training Data: A total of 1,000 designs with 70% feasible ones and 30% infeasible ones. Design variables and generated 3D geometry data (in stl format) are provided. Folder and file structure are as follows:

```
./Training Data/
        ./Feasible_Designs/
                1.stl
                2.stl ........
        ./Infeasible_Designs/
                35.stl
                42.stl ......
        Training_MetaData.csv
```

Test Data: To format your test script accordingly, the structure of the test data is provided. Save your model predictions in the "Save_Predictions.csv" file.

```
./Test Data/
        ./Test_designs/
                100.stl
                200.stl ......
        Save_Predictions.csv
```

Format of "Training_MetaData.csv":

Sample ID	STL FileName	X1	X2	X3	X4	X5	X6	Feasibility Class
1	305.stl	D	2	0.1	0.1	0	0	0
2	2.stl	B	1	0.8	0.1	1.576	0	1

Input: STL FileName (Main Predictor, encouraged to use this 3D data predictor only), X1, X2, X3, X4, X5, X6 (Design Variables: secondary Predictors, encouraged not to use them, but can be used to analyze the feasibility).
Output: Feasibility Class (Binary class)

Please upload a Python file named test.py containing the inference code for your trained model. The test.py file should adhere to the following guidelines:

- The code should automatically load the trained model's weight file using the relative path of the saved weight file.
- The code should read all the .stl files present in the test directory ("./Test Data/") and perform inference on each file.
- The model should perform binary classification on the test data and generate predictions.
- The binary classification results should be saved in a file named "Save_Predictions.csv". The format of the saved predictions should match the data frame format provided in the appendix of the competition flyer.

Please ensure that your test.py file follows these requirements and can be executed successfully to generate the desired predictions. The organizers will use this file to evaluate your model's performance on the undisclosed test dataset. F1 Score, accuracy %, Type 1 error, and Type 2 error are the evaluation metrics used to measure the model performance.

Format of "Save_Predictions.csv": A skeleton placeholder code logic should be implemented in the test script logic so that when the test directory path is given the code should be able to **load the model** and **navigate** to the test data directory and then make **predictions** on all .stl files present in the directory and save the prediction output to

“Save_Predictions.csv”. As the test data is not provided to you, you will have to implement the mentioned logic and during the evaluation phase we will run the test script to evaluate the model performance.

STL FileName	Predicted Class
713.stl	0
405.stl	1

Reference material to help you process the STL files.

- Please review the “numpy-stl” python package to process, access, modify, or save the .STL files. Sample codes related to “numpy-stl” can be found in this Link.
- Please refer to Open3D, which is another modern library for 3D data processing. Using this library, you can render, process, modify, and save the STL data, or transform them into voxel or point-cloud representations. Link
- The stl-to-voxel python package helps you to turn STL files into voxels, images, and videos.

Template.py:

```python
'''Import your modules here, e.g.,

import numpy as np
import tensorflow as tf
'''


'''
Fill in the TODOs of the following class.
Please feel free to add any customized functions/classes
'''
class Model():
    def __init__(self, path2mdl, trainData, testData, modelOptions):
        self.mdl = None
        self.mdlpath = path2mdl # path to the serialized model
        self.prediction = None
        self.testData = testData
        self.trainData = trainData
        self.options = modelOptions # contains the settings for model training

    def loadModel(self):
        '''
        TODO: Load your model according to self.mdlpath here
        '''
        self.mdl = None # replace None by the model you may load from self.mdlpath

    def saveModel(self):
        '''
        TODO: Save your model to file
        '''
        self.path2mdl = None # replace None by the path you saved your model

    def train(self):
        '''
        TODO: use self.trainData and self.options to train your model
        '''
        self.mdl = None # replace None by the model you trained

    def test(self):
        '''
        TODO: use self.testData and self.mdl to test your model
        '''
        self.prediction = None # replace None by the model prediction you received
        return self.prediction
```