National College of Ireland

Attention-based Deep Learning Prediction of Silica Concentrate for Quality Optimization in Mineral

MSc in Data Analytics

Fardeen Mohammad
Student ID: 24109878

School of Computing
National College of Ireland

Supervisor: Vikas Tomer

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Fardeen Mohammad
Student ID:	24109878
Programme:	MSc in Data Analytics **Year:** 2025
Module:	MSc Research Practicum Part-2
Lecturer:	Vikas Tomer
Submission Due Date:	27-01-2026
Project Title:	Attention-Based Deep Learning Prediction of Silica Concentrate
Word Count:	5573 **Page Count:** 6

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Fardeen Mohammad
Date:	27-01-2026

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Attention-based Deep Learning Prediction of Silica Concentrate for Quality Optimization in Mineral

Fardeen Mohammad
x24109878@student.ncirl.ie
MSc in Data Analytics

1. Introduction

This project develops a reproducible predictive modelling pipeline to estimate % Silica Concentrate in an iron ore flotation process using historical sensor and operating data. The task is challenging because flotation behaviour is nonlinear, noisy, and driven by interacting variables such as reagent dosage, air flow, and pulp conditions, meaning silica outcomes depend on combined effects rather than single features. The implemented system cleans and prepares the dataset, trains models from classical machine learning baselines to deep learning and attention-based interaction learning (AutoInt), and evaluates them consistently so the best approach can be identified and reused for industrial quality optimization.

2. System Specifications

Hardware requirements

- **CPU:** Modern multi-core CPU (recommended: ≥ 8 cores)
- **GPU:** Optional for deep learning; recommended NVIDIA GPU with 6 GB VRAM for faster training (CUDA-capable)
- **RAM:** Recommended 16 GB
- **Storage:** 5 GB free (dataset, notebooks, saved models, logs, figures)

Software requirements

- **Operating System:** Windows 10/11 or Linux (Ubuntu 20.04+ recommended)
- **Python version:** Python 3.9+ (compatible with modern ML/DL stacks)
- **Environment:** Conda or venv virtual environment for isolation and reproducibility
- **CUDA/cuDNN (if GPU used):** CUDA toolkit and corresponding cuDNN matching the installed PyTorch build (GPU acceleration is optional but recommended for iterative experiments and attention models)

3. Required Libraries

The implementation is based on a standard scientific Python stack for industrial regression, combined with deep learning utilities for neural training and attention-based architectures. Pandas and NumPy support tabular ingestion, numeric conversion, feature engineering, and matrix transformations. scikit-learn provides preprocessing, train-test splitting, metrics, and

systematic hyperparameter search. Gradient boosting models are provided via XGBoost and LightGBM, which serve as strong nonlinear baselines for industrial process prediction. PyTorch provides the tensor engine and training loop used for the MLP regressor and the AutoInt attention-based interaction learner. Matplotlib supports visualization of performance comparisons and training curves.

The necessary libraries can be installed using the following commands:

a) **pip install numpy==2.3.5 pandas==2.3.3 matplotlib==3.10.7 scikit-learn==1.7.2**
b) **pip install xgboost==3.1.2 lightgbm==4.6.0**
c) **pip install torch==2.9.1**
d) **pip install jupyter==1.1.1**

4. Dataset description

The data the work is based on is the measurements of industrial processes of one of the functioning iron ore flotation plants, which has a set of various variables that determine not only the quality of feed but also the final composition of the concentrate. Every record contains timed observations and process parameters in terms of percent Iron Feed, percent Silica Feed, flows of reagents (Starch Flow and Amina Flow), Ore Pulp Flow, Ore Pulp pH, and Ore Pulp Density. Moreover, the dataset also includes more specific operational data such as air flow rates and pulp levels of seven flotation columns, which represents the dynamic behaviour of the separation process. The variables of interest are the percent Iron Concentrate and the percent Silica Concentrate which are the final quality of the product. The data has common industrial properties of outliers, non-uniform data, and possible sensor noise, which is a manifestation of real-life operations. These characteristics render it appropriate to assess the strength of progressive prediction models, especially attention-based architectures with the capability to determine influential features under varying process variables. The multivariate time-series environment in general is well-developed and supports of accurate and context-sensitive silicon and impurity prediction systems in the context of mineral processing.

5. Models Implemented

The modelling process is structured in an orderly sequence of the beginning methods to the developed architectures. The first one is a classical machine learning regressor, which is the Random Forest, a powerful nonlinear baseline, then XGBoost and LightGBM, stronger gradient-boosting models, will be used to address complex tabular industrial data. It subsequently progresses to a deep-learning baseline (MLP regressor), which acquires nonlinear interactions by stacking fully connected layers. Lastly, the pipeline adopts AutoInt that is an attention-based interaction learning framework that can automatically learn feature interactions and higher-order dependency, thus being highly applicable to the flotation systems where silica concentrate is conditioned by the joint actions of numerous process variables.

6. Code snippets and plots:

1) Data info

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30000 entries, 0 to 29999
Data columns (total 24 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   date                            30000 non-null  object
 1   % Iron Feed                     30000 non-null  float64
 2   % Silica Feed                   30000 non-null  float64
 3   Starch Flow                     30000 non-null  float64
 4   Amina Flow                      30000 non-null  float64
 5   Ore Pulp Flow                   30000 non-null  float64
 6   Ore Pulp pH                     30000 non-null  float64
 7   Ore Pulp Density                30000 non-null  float64
 8   Flotation Column 01 Air Flow    30000 non-null  float64
 9   Flotation Column 02 Air Flow    30000 non-null  float64
 10  Flotation Column 03 Air Flow    30000 non-null  float64
 11  Flotation Column 04 Air Flow    30000 non-null  float64
 12  Flotation Column 05 Air Flow    30000 non-null  float64
 13  Flotation Column 06 Air Flow    30000 non-null  float64
 14  Flotation Column 07 Air Flow    30000 non-null  float64
 15  Flotation Column 01 Level       30000 non-null  float64
 16  Flotation Column 02 Level       30000 non-null  float64
 17  Flotation Column 03 Level       30000 non-null  float64
 18  Flotation Column 04 Level       30000 non-null  float64
 19  Flotation Column 05 Level       30000 non-null  float64
 20  Flotation Column 06 Level       30000 non-null  float64
 21  Flotation Column 07 Level       30000 non-null  float64
 22  % Iron Concentrate              30000 non-null  float64
 23  % Silica Concentrate            30000 non-null  float64
dtypes: float64(23), object(1)
```

2) Missing Values

```
#null values
df.isnull().sum()
```

	0
date	0
Starch Flow	0
Amina Flow	0
Ore Pulp Flow	0
Ore Pulp pH	0
Ore Pulp Density	0
Flotation Column 01 Air Flow	0
Flotation Column 02 Air Flow	0
Flotation Column 03 Air Flow	0
Flotation Column 04 Air Flow	0
Flotation Column 05 Air Flow	0
Flotation Column 06 Air Flow	0
Flotation Column 07 Air Flow	0
Flotation Column 01 Level	0
Flotation Column 02 Level	0
Flotation Column 03 Level	0
Flotation Column 04 Level	0
Flotation Column 05 Level	0
Flotation Column 06 Level	0
Flotation Column 07 Level	0
% Silica Concentrate	0

3) Duplicates

```
df.duplicated().sum()
```

```
np.int64(35)
```

```
#drop duplicates
df.drop_duplicates(inplace=True)
df.duplicated().sum()
```

```
np.int64(0)
```

4) Splitting Data

```
from sklearn.model_selection import train_test_split

# Set a seed for reproducibility
seed = 42
np.random.seed(seed)

# Define target and features
y = df_num['% Silica Concentrate']
X = df_num.drop(columns=['% Silica Concentrate'])

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=seed)

print(f"Training set: {X_train.shape}")
print(f"Testing set: {X_test.shape}")
```

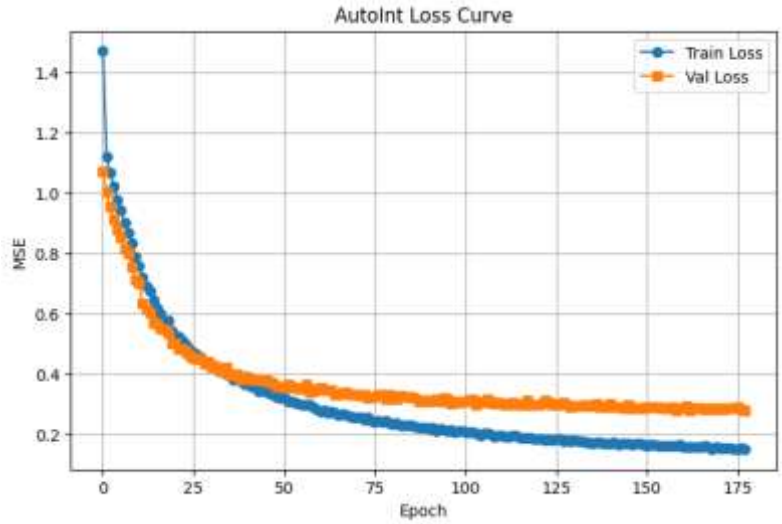
5) Standard Scaler

```
from sklearn.preprocessing import StandardScaler

# Scale the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

6) AutoInt model embeddings

```python
class AutoInt(nn.Module):
    """AutoInt Model for numerical tabular regression."""
    def __init__(self, n_features, embed_dim=16, num_heads=4, num_layers=3, mlp_hidden=[128, 64], dropout=0.1):
        super().__init__()
        self.n_features = n_features
        self.embed_dim = embed_dim

        # Learnable embeddings for each feature
        self.feature_embed = nn.Linear(1, embed_dim)

        # Interaction layers
        self.interaction_layers = nn.ModuleList([
            InteractingLayer(embed_dim, num_heads=num_heads, dropout=dropout, residual=True)
            for _ in range(num_layers)])

        # Final MLP head
        mlp = []
        input_dim = n_features * embed_dim
        for h in mlp_hidden:
            mlp += [nn.Linear(input_dim, h), nn.ReLU(), nn.Dropout(dropout)]
            input_dim = h
        mlp.append(nn.Linear(input_dim, 1))
        self.mlp = nn.Sequential(*mlp)

    def forward(self, x):
        # x: (B, F)
        B, F = x.size()
        x = x.unsqueeze(-1)
        embeds = self.feature_embed(x)
        for layer in self.interaction_layers:
            embeds = layer(embeds)
        # Flatten
        out = embeds.reshape(B, -1)
        out = self.mlp(out)
        return out
```

7) Loss plot of AutoInt



8) Comparison of evaluation metrics

Table 1. Performance Comparison of all Models

Model	MAE	MSE	RMSE	R2 Score
Random Forest	0.4604	0.3993	0.6319	0.6799
XGBoost	0.4474	0.3704	0.6086	0.7031
Light GBM	0.4571	0.3732	0.6109	0.7009
MLP	0.5001	0.4963	0.7045	0.6022
Auto Int	0.3601	0.2774	0.5267	0.7777

7. Conclusion:

The project provides an end-to-end pipeline that is both reproducible and capable of predicting % Silica Concentrate, including data preparation, model training, model tuning, and evaluation using consistent regression metrics. AutoInt is the most effective model since its attention-based interaction learning exhibits more effective extraction of complex dependencies between process variables in comparison with classical baselines and a standard MLP. This workflow can be duplicated and extended, and further enhancements in the future, including more sensors, temporal modeling, or real-time deployment to support decision-making in mineral processing quality optimisation, are possible.

References:

1. https://docs.pytorch.org/docs/stable/index.html
2. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
3. https://xgboost.readthedocs.io/en/latest/python/python_api.html
4. https://lightgbm.readthedocs.io/en/latest/Python-Intro.html
5. https://pandas.pydata.org/docs/
6. https://matplotlib.org/stable/index.html