## **3.0 Methodology**

**3.1.0 Materials and Methods**

3.1.1 Dataset Description:

The dataset implementation codes used for simulation and model training is provided in Appendix A: <https://github.com/muhadafa/pathlosspredictionthesis2023musah>. It encompasses various parameters relevant to wireless communication, including Frequency, Distance, Transmitter Height, Receiver Height, Antenna Gain, Environment, Obstacle Distance, Path Characteristics, Weather Conditions, Shadowing Effects, Frequency Bandwidth, and Path Loss.

The dataset exhibits a diverse range of scenarios, incorporating different environmental conditions, obstacle distances, and weather conditions. These variables contribute to the complexity of the wireless communication model, making it suitable for training and evaluating the proposed path loss prediction model.

**3.2 Proposed Approach**

In response to these challenges, we advocate for a simulation-style approach to data collection and model training. This approach capitalizes on the versatility and efficiency of the Python programming language to simulate realistic wireless communication scenarios. By varying parameters such as frequency, distance, transmitter and receiver heights, environmental conditions, and obstacle presence, we can generate a rich and diverse dataset that encompasses the complexities of modern communication environments. This simulated data serves as the foundation for training robust data-driven path loss prediction models.

**3.3 Method**

The research is built upon three key pillars:

Comprehensive Data Collection: The simulation-style approach enables us to collect data across a wider range of scenarios and frequencies compared to traditional measurement setups. This includes variations in urban, suburban, and rural environments, diverse obstacle configurations, and dynamic weather conditions.

Dimensionality Reduction: To handle the high dimensionality of the collected data, we employ dimensionality reduction techniques such as Principal Component Analysis (PCA) to identify the most relevant features for path loss prediction. This reduces computational complexity and improves model efficiency.

Neural Network Model Development: We utilize a multi-layered neural network architecture to learn complex relationships between input features (e.g., frequency, distance) and the target variable (path loss). This allows the model to capture non-linear dependencies and generalize its predictions to unseen scenarios.

**3.3.1 Implementation**

The provided algorithm is the practical implementation of the mode (all code can be found in)

*#Using TensorFlow for Building Models*

*# Step 1: Loading the required libraries and modules*

*import necessary\_libraries*

*# Step 2: Loading the data and performing basic data checks*

*load\_data\_from\_csv*

*perform\_basic\_data\_checks*

*# Step 3: Extract features and target variable*

*Assume the dataset is in a DataFrame named df*

*Split the data into features (X) and target (y)*

*extract\_features\_and\_target\_variable*

*# Step 4: Separate numerical and categorical columns*

*separate\_numerical\_and\_categorical\_columns*

*# Step 5: Use Label Encoding for categorical columns with ordinal relationship*

*label\_encode\_categorical\_columns*

*Use Label Encoding for categorical columns with ordinal relationship*

*X = pd.get\_dummies(X, columns=categorical\_cols, drop\_first=True)*

*# Step 6: Create transformers for numerical and categorical columns*

*create\_transformers\_for\_numerical\_and\_categorical\_columns*

*# Step 7: Combine transformers using ColumnTransformer*

*combine\_transformers\_using\_ColumnTransformer*

*# Step 8: Use the preprocessor in a Pipeline along with the model*

*create\_pipeline\_with\_preprocessor\_and\_regressor*

*# Step 9: Split the dataset into training and testing sets*

*split\_dataset\_into\_training\_and\_testing\_sets*

*# Step 10: Standardize the features*

*standardize\_the\_features*

*# Step 11: Build a simple neural network model*

*build\_simple\_neural\_network\_model*

*model = Sequential()*

*model.add(Dense(64, input\_dim=X\_train.shape[1], activation='relu'))*

*model.add(Dense(32, activation='relu'))*

*model.add(Dense(1, activation='linear')*

*# Step 12: Compile the model*

*compile\_the\_model*

*model.compile(optimizer='adam', loss='mean\_squared\_error')*

*# Step 13: Train the model*

*train\_the\_model*

*model.fit(X\_train\_scaled, y\_train, epochs=50, batch\_size=32, validation\_split=0.1)*

*# Step 14: Evaluate the model on the test set*

*evaluate\_the\_model\_on\_test\_set*

*Calculate the correlation matrix*

*# Step 15: Visualize the predictions*

*visualize\_the\_predictions*

*Plot the correlation matrix using a heatmap*

## **Appendix A**

#Using TensorFlow for Building Models

# Step 1: Loading the required libraries and modules

import numpy as np

import pandas as pd

import seaborn as sns

from seaborn import lineplot

import matplotlib.pyplot as plt

%matplotlib inline

from pandas import read\_csv

from seaborn import lineplot

from seaborn import distplot

from seaborn import boxplot

from seaborn import scatterplot

from matplotlib import pyplot

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras import layers

tf.random.set\_seed(100)

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import r2\_score

# Import necessary libraries

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.decomposition import PCA

from sklearn.metrics import mean\_squared\_error

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

# Step 2: Loading the data and performing basic data checks

# df = sns.load\_dataset('logged\_data.csv')

Q1paper = pd.read\_csv('C:\path\_loss\_data.csv')

print(Q1paper.shape)

Q1paper.describe(include='all')

# Load the dataset from Appendix A

# Assuming your data is stored in a CSV file, adjust the file path accordingly

#To allocate 2.70 GiB for an array with shape (362703346,) and data type int64

# Step 3: Extract features and target variable

df = Q1paper

X = df.drop(columns=['Path Loss']) # Features

y = df['Path Loss'] # Target variable

import pandas as pd

from sklearn.preprocessing import StandardScaler, LabelEncoder, OneHotEncoder

from sklearn.model\_selection import train\_test\_split

# Step 4: Assume your dataset is in a DataFrame named df

# Step 5: Split the data into features (X) and target (y)

from sklearn.linear\_model import LinearRegression

from sklearn.compose import ColumnTransformer

from sklearn.impute import SimpleImputer

from sklearn.pipeline import Pipeline

from sklearn.preprocessing import StandardScaler

from sklearn.decomposition import PCA

# Step 6: Separate numerical and categorical columns

numerical\_cols = X.select\_dtypes(include=['float64']).columns

categorical\_cols = X.select\_dtypes(include=['object']).columns

# Step 7: Use Label Encoding for categorical columns with ordinal relationship

label\_encoder = LabelEncoder()

X[categorical\_cols] = X[categorical\_cols].apply(label\_encoder.fit\_transform)

# Use One-Hot Encoding for categorical columns without ordinal relationship

#X = pd.get\_dummies(X, columns=categorical\_cols, drop\_first=True)

# Step 8: Create transformers for numerical and categorical columns

numerical\_transformer = Pipeline(steps=[

('imputer', SimpleImputer(strategy='mean')), # You can change the strategy if needed

('scaler', StandardScaler()),

('pca', PCA(n\_components=2)) # Adjust n\_components as needed

])

categorical\_transformer = Pipeline(steps=[

('imputer', SimpleImputer(strategy='most\_frequent')), # You can change the strategy if needed

])

# Step 9: Combine transformers using ColumnTransformer

preprocessor = ColumnTransformer(

transformers=[

('num', numerical\_transformer, numerical\_cols),

('cat', categorical\_transformer, categorical\_cols)

])

# Step 10: Use the preprocessor in a Pipeline along with your model

pipeline = Pipeline(steps=[('preprocessor', preprocessor),

('regressor', LinearRegression())]) # Replace with your chosen regressor (LinearRegression()) such as RandomForestRegressor, GradientBoostingRegressor

from sklearn.metrics import confusion\_matrix

import seaborn as sns

import seaborn as sns

#Step 11: Combine features and target variable

df\_combined = pd.concat([X, y], axis=1)

# Step 12:Calculate the correlation matrix

correlation\_matrix = df\_combined.corr()

# Step 13:Plot the correlation matrix using a heatmap

# Ref. Path-Loss Characteristics of Urban Wireless Channels 2010 by Keith T. Herring, Jack W. Hallowy,David H. Staelin and Daniel W. Bliss

plt.figure(figsize=(12, 10))

sns.heatmap(correlation\_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5)

plt.title('Correlation Matrix')

plt.show()

import seaborn as sns

import matplotlib.pyplot as plt

# Extracting relevant columns from the original dataset

selected\_features = ['Environment', 'Path Characteristics', 'Weather Conditions', 'Frequency Bandwidth', 'Path Loss']

df\_selected = df[selected\_features]

# Plotting line plots

plt.figure(figsize=(12, 8))

# Strong positive correlation: Environment

plt.subplot(221)

sns.lineplot(x='Environment', y='Path Loss', data=df\_selected)

# Strong positive correlation: Path Characteristics

plt.subplot(222)

sns.lineplot(x='Path Characteristics', y='Path Loss', data=df\_selected)

# Weak correlation: Weather Conditions

plt.subplot(223)

sns.lineplot(x='Weather Conditions', y='Path Loss', data=df\_selected)

# Weak correlation: Frequency Bandwidth

plt.subplot(224)

sns.lineplot(x='Frequency Bandwidth', y='Path Loss', data=df\_selected)

plt.tight\_layout()

plt.show()

# Separate features (X) and target variable (y)

X = df.drop(columns=['Path Loss']) # Features

y = df['Path Loss'] # Target variable

# Check column names

print(X.columns)

# Check if 'Path Loss' is in y

#print('Path Loss' in y.columns)

# Verify the structure of X and y before concatenating

print(X.head())

print(y.head())

# Combine features and target variable

df\_combined = pd.concat([X, y], axis=1)

# Standardize the features

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

# Apply StandardScaler to numerical columns only

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train[numerical\_cols])

X\_test\_scaled = scaler.transform(X\_test[numerical\_cols])

# Combine the scaled numerical features with the encoded categorical features

X\_train\_scaled = pd.concat([pd.DataFrame(X\_train\_scaled, columns=numerical\_cols), X\_train.drop(columns=numerical\_cols)], axis=1)

X\_test\_scaled = pd.concat([pd.DataFrame(X\_test\_scaled, columns=numerical\_cols), X\_test.drop(columns=numerical\_cols)], axis=1)

# Apply dimension reduction using PCA

#pca = PCA(n\_components=10) # You can adjust the number of components based on your needs

# Fit the pipeline to your training data

# pipeline.fit(X\_train, y\_train)

# Apply dimension reduction using PCA

#pca = PCA(n\_components=10) # You can adjust the number of components based on your needs

# Fit and transform the PCA on the training data

#X\_train\_pca = pca.fit\_transform(X\_train\_scaled)

# Transform the test data using the same PCA

# X\_test\_pca = pca.transform(X\_test\_scaled)

#-----------------------------------------------------------------------------------------------

from sklearn.impute import SimpleImputer

# Create a numerical imputer

numerical\_imputer = SimpleImputer(strategy='mean')

# Fit and transform the imputer on the training data

X\_train\_imputed = numerical\_imputer.fit\_transform(X\_train)

# Transform the test data using the same imputer

X\_test\_imputed = numerical\_imputer.transform(X\_test)

# Apply standard scaling to numerical columns

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train\_imputed)

X\_test\_scaled = scaler.transform(X\_test\_imputed)

# Apply dimension reduction using PCA

pca = PCA(n\_components=10) # You can adjust the number of components based on your needs

# Fit and transform the PCA on the scaled training data

X\_train\_pca = pca.fit\_transform(X\_train\_scaled)

# Transform the scaled test data using the same PCA

X\_test\_pca = pca.transform(X\_test\_scaled)

# Build a simple neural network model

model = Sequential()

model.add(Dense(11, input\_dim= 11, activation='relu'))

model.add(Dense(3, activation='relu'))

model.add(Dense(1, activation='linear'))

# Compile the model

model.compile(optimizer='adam', loss='mean\_squared\_error')

# Train the model

model.fit(X\_train, y\_train, epochs= 150, batch\_size= 100, validation\_split= 0.1)

import matplotlib.pyplot as plt

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

from sklearn.model\_selection import train\_test\_split

# Visualize the training history

# Plot training loss

plt.figure(figsize=(10, 6))

plt.plot(history.history['loss'], label='Training Loss', color='blue')

# Plot validation loss

plt.plot(history.history['val\_loss'], label='Validation Loss', color='orange')

plt.title('Training and Validation Loss Over Epochs', fontsize=16)

plt.xlabel('Epochs', fontsize=14)

plt.ylabel('Loss', fontsize=14)

plt.legend(fontsize=12)

plt.grid(True)

plt.show()

import matplotlib.pyplot as plt

# Data

models = ['Neural Network', 'Okumura-Hata', 'Costa 231', 'Egli']

mae\_values = [323.22, 304.41, 164.44, 203.17]

mse\_values = [105142.72, 92662.66, 27040.91, 41276.93]

rmse\_values = [324.26, 304.41, 164.44, 203.17]

# Plotting Mean Absolute Error (MAE)

plt.figure(figsize=(10, 6))

plt.bar(models, mae\_values, color=['blue', 'orange', 'green', 'red'])

plt.xlabel('Models')

plt.ylabel('MAE Values')

plt.title('Comparison of Mean Absolute Error (MAE) Across Models')

plt.show()

# Plotting Mean Squared Error (MSE)

plt.figure(figsize=(10, 6))

plt.bar(models, mse\_values, color=['blue', 'orange', 'green', 'red'])

plt.xlabel('Models')

plt.ylabel('MSE Values')

plt.title('Comparison of Mean Squared Error (MSE) Across Models')

plt.show()

# Plotting Root Mean Squared Error (RMSE)

plt.figure(figsize=(10, 6))

plt.bar(models, rmse\_values, color=['blue', 'orange', 'green', 'red'])

plt.xlabel('Models')

plt.ylabel('RMSE Values')

plt.title('Comparison of Root Mean Squared Error (RMSE) Across Models')

plt.show()

## **Appendix B**

Epoch 1/150

72/72 [==============================] - 4s 7ms/step - loss: 627345875558989824.0000 - val\_loss: 267557306986659840.0000

Epoch 2/150

72/72 [==============================] - 0s 3ms/step - loss: 142261961227239424.0000 - val\_loss: 62083374306557952.0000

Epoch 3/150

72/72 [==============================] - 0s 3ms/step - loss: 30237570491219968.0000 - val\_loss: 10917189464883200.0000

Epoch 4/150

72/72 [==============================] - 0s 3ms/step - loss: 4700193702281216.0000 - val\_loss: 1321789875552256.0000

Epoch 5/150

72/72 [==============================] - 0s 3ms/step - loss: 497777515692032.0000 - val\_loss: 106528635879424.0000

Epoch 6/150

72/72 [==============================] - 0s 3ms/step - loss: 38127229992960.0000 - val\_loss: 8335313076224.0000

Epoch 7/150

72/72 [==============================] - 0s 3ms/step - loss: 3710015766528.0000 - val\_loss: 1489187110912.0000

Epoch 8/150

72/72 [==============================] - 0s 4ms/step - loss: 861205299200.0000 - val\_loss: 502693462016.0000

Epoch 9/150

72/72 [==============================] - 0s 3ms/step - loss: 320834502656.0000 - val\_loss: 222357651456.0000

Epoch 10/150

72/72 [==============================] - 0s 3ms/step - loss: 148672462848.0000 - val\_loss: 114572328960.0000

Epoch 11/150

72/72 [==============================] - 0s 3ms/step - loss: 78464794624.0000 - val\_loss: 66427981824.0000

Epoch 12/150

72/72 [==============================] - 0s 3ms/step - loss: 45741649920.0000 - val\_loss: 40657977344.0000

Epoch 13/150

72/72 [==============================] - 0s 3ms/step - loss: 28258600960.0000 - val\_loss: 26727516160.0000

Epoch 14/150

72/72 [==============================] - 0s 3ms/step - loss: 18376648704.0000 - val\_loss: 18233038848.0000

Epoch 15/150

72/72 [==============================] - 0s 3ms/step - loss: 12402024448.0000 - val\_loss: 12871159808.0000

Epoch 16/150

72/72 [==============================] - 0s 3ms/step - loss: 8613060608.0000 - val\_loss: 9467026432.0000

Epoch 17/150

72/72 [==============================] - 0s 3ms/step - loss: 6187203584.0000 - val\_loss: 7146542080.0000

Epoch 18/150

72/72 [==============================] - 0s 3ms/step - loss: 4574246400.0000 - val\_loss: 5470049280.0000

Epoch 19/150

72/72 [==============================] - 0s 3ms/step - loss: 3457024000.0000 - val\_loss: 4323700736.0000

Epoch 20/150

72/72 [==============================] - 0s 3ms/step - loss: 2676867072.0000 - val\_loss: 3444050944.0000

Epoch 21/150

72/72 [==============================] - 0s 4ms/step - loss: 2091538176.0000 - val\_loss: 2791732224.0000

Epoch 22/150

72/72 [==============================] - 0s 3ms/step - loss: 1667225600.0000 - val\_loss: 2280669696.0000

Epoch 23/150

72/72 [==============================] - 0s 3ms/step - loss: 1346320640.0000 - val\_loss: 1885910656.0000

Epoch 24/150

72/72 [==============================] - 0s 3ms/step - loss: 1112266624.0000 - val\_loss: 1579922944.0000

Epoch 25/150

72/72 [==============================] - 0s 3ms/step - loss: 927978560.0000 - val\_loss: 1326099456.0000

Epoch 26/150

72/72 [==============================] - 0s 3ms/step - loss: 779938304.0000 - val\_loss: 1120064768.0000

Epoch 27/150

72/72 [==============================] - 0s 3ms/step - loss: 658885184.0000 - val\_loss: 942807680.0000

Epoch 28/150

72/72 [==============================] - 0s 4ms/step - loss: 559551744.0000 - val\_loss: 796660288.0000

Epoch 29/150

72/72 [==============================] - 0s 3ms/step - loss: 478022432.0000 - val\_loss: 671430848.0000

Epoch 30/150

72/72 [==============================] - 0s 3ms/step - loss: 410963264.0000 - val\_loss: 567191168.0000

Epoch 31/150

72/72 [==============================] - 0s 3ms/step - loss: 355631104.0000 - val\_loss: 482114368.0000

Epoch 32/150

72/72 [==============================] - 0s 4ms/step - loss: 314023424.0000 - val\_loss: 419421024.0000

Epoch 33/150

72/72 [==============================] - 0s 3ms/step - loss: 281224416.0000 - val\_loss: 362690304.0000

Epoch 34/150

72/72 [==============================] - 0s 3ms/step - loss: 254887904.0000 - val\_loss: 316383072.0000

Epoch 35/150

72/72 [==============================] - 0s 3ms/step - loss: 233118400.0000 - val\_loss: 282883264.0000

Epoch 36/150

72/72 [==============================] - 0s 3ms/step - loss: 214985664.0000 - val\_loss: 251275024.0000

Epoch 37/150

72/72 [==============================] - 0s 3ms/step - loss: 199298512.0000 - val\_loss: 226399584.0000

Epoch 38/150

72/72 [==============================] - 0s 3ms/step - loss: 187141872.0000 - val\_loss: 206154336.0000

Epoch 39/150

72/72 [==============================] - 0s 3ms/step - loss: 177159424.0000 - val\_loss: 188600688.0000

Epoch 40/150

72/72 [==============================] - 0s 3ms/step - loss: 169453552.0000 - val\_loss: 176116080.0000

Epoch 41/150

72/72 [==============================] - 0s 3ms/step - loss: 161744112.0000 - val\_loss: 161818688.0000

Epoch 42/150

72/72 [==============================] - 0s 3ms/step - loss: 155736080.0000 - val\_loss: 151098128.0000

Epoch 43/150

72/72 [==============================] - 0s 3ms/step - loss: 150010880.0000 - val\_loss: 138311520.0000

Epoch 44/150

72/72 [==============================] - 0s 3ms/step - loss: 144840016.0000 - val\_loss: 127441672.0000

Epoch 45/150

72/72 [==============================] - 0s 4ms/step - loss: 140556640.0000 - val\_loss: 120748664.0000

Epoch 46/150

72/72 [==============================] - 0s 3ms/step - loss: 135459104.0000 - val\_loss: 109448864.0000

Epoch 47/150

72/72 [==============================] - 0s 3ms/step - loss: 130491760.0000 - val\_loss: 100404384.0000

Epoch 48/150

72/72 [==============================] - 0s 3ms/step - loss: 127706960.0000 - val\_loss: 94928744.0000

Epoch 49/150

72/72 [==============================] - 0s 3ms/step - loss: 124891816.0000 - val\_loss: 88258808.0000

Epoch 50/150

72/72 [==============================] - 0s 3ms/step - loss: 121833864.0000 - val\_loss: 82021312.0000

Epoch 51/150

72/72 [==============================] - 0s 3ms/step - loss: 119119848.0000 - val\_loss: 77850360.0000

Epoch 52/150

72/72 [==============================] - 0s 3ms/step - loss: 116978728.0000 - val\_loss: 72091232.0000

Epoch 53/150

72/72 [==============================] - 0s 4ms/step - loss: 114728408.0000 - val\_loss: 67697744.0000

Epoch 54/150

72/72 [==============================] - 0s 3ms/step - loss: 112715376.0000 - val\_loss: 63429480.0000

Epoch 55/150

72/72 [==============================] - 0s 3ms/step - loss: 110800496.0000 - val\_loss: 60055392.0000

Epoch 56/150

72/72 [==============================] - 0s 3ms/step - loss: 109557736.0000 - val\_loss: 57375324.0000

Epoch 57/150

72/72 [==============================] - 0s 3ms/step - loss: 108214992.0000 - val\_loss: 53635236.0000

Epoch 58/150

72/72 [==============================] - 0s 3ms/step - loss: 106711152.0000 - val\_loss: 51398932.0000

Epoch 59/150

72/72 [==============================] - 0s 3ms/step - loss: 105542560.0000 - val\_loss: 48457364.0000

Epoch 60/150

72/72 [==============================] - 0s 3ms/step - loss: 104385104.0000 - val\_loss: 45345976.0000

Epoch 61/150

72/72 [==============================] - 0s 4ms/step - loss: 103166368.0000 - val\_loss: 42367616.0000

Epoch 62/150

72/72 [==============================] - 0s 4ms/step - loss: 101854136.0000 - val\_loss: 39943012.0000

Epoch 63/150

72/72 [==============================] - 0s 4ms/step - loss: 101004448.0000 - val\_loss: 37690024.0000

Epoch 64/150

72/72 [==============================] - 0s 3ms/step - loss: 100148720.0000 - val\_loss: 35306088.0000

Epoch 65/150

72/72 [==============================] - 0s 3ms/step - loss: 99121744.0000 - val\_loss: 33431034.0000

Epoch 66/150

72/72 [==============================] - 0s 3ms/step - loss: 98209776.0000 - val\_loss: 30519944.0000

Epoch 67/150

72/72 [==============================] - 0s 3ms/step - loss: 97130520.0000 - val\_loss: 28244086.0000

Epoch 68/150

72/72 [==============================] - 0s 3ms/step - loss: 96202824.0000 - val\_loss: 26690156.0000

Epoch 69/150

72/72 [==============================] - 0s 3ms/step - loss: 95555184.0000 - val\_loss: 25084798.0000

Epoch 70/150

72/72 [==============================] - 0s 3ms/step - loss: 94942504.0000 - val\_loss: 23888994.0000

Epoch 71/150

72/72 [==============================] - 0s 3ms/step - loss: 94336032.0000 - val\_loss: 22179126.0000

Epoch 72/150

72/72 [==============================] - 0s 3ms/step - loss: 93612176.0000 - val\_loss: 20573990.0000

Epoch 73/150

72/72 [==============================] - 0s 3ms/step - loss: 92901616.0000 - val\_loss: 19155718.0000

Epoch 74/150

72/72 [==============================] - 0s 3ms/step - loss: 92388864.0000 - val\_loss: 17804830.0000

Epoch 75/150

72/72 [==============================] - 0s 3ms/step - loss: 91852864.0000 - val\_loss: 16872186.0000

Epoch 76/150

72/72 [==============================] - 0s 3ms/step - loss: 91460680.0000 - val\_loss: 16002917.0000

Epoch 77/150

72/72 [==============================] - 0s 3ms/step - loss: 91070952.0000 - val\_loss: 15033586.0000

Epoch 78/150

72/72 [==============================] - 0s 3ms/step - loss: 90593456.0000 - val\_loss: 14262884.0000

Epoch 79/150

72/72 [==============================] - 0s 3ms/step - loss: 90116208.0000 - val\_loss: 13127288.0000

Epoch 80/150

72/72 [==============================] - 0s 3ms/step - loss: 89658576.0000 - val\_loss: 12207572.0000

Epoch 81/150

72/72 [==============================] - 0s 3ms/step - loss: 89210704.0000 - val\_loss: 11010934.0000

Epoch 82/150

72/72 [==============================] - 0s 3ms/step - loss: 88703040.0000 - val\_loss: 10425123.0000

Epoch 83/150

72/72 [==============================] - 0s 3ms/step - loss: 88409344.0000 - val\_loss: 9582501.0000

Epoch 84/150

72/72 [==============================] - 0s 3ms/step - loss: 87994352.0000 - val\_loss: 8589167.0000

Epoch 85/150

72/72 [==============================] - 0s 3ms/step - loss: 87492288.0000 - val\_loss: 7579160.5000

Epoch 86/150

72/72 [==============================] - 0s 3ms/step - loss: 87069904.0000 - val\_loss: 6587794.0000

Epoch 87/150

72/72 [==============================] - 0s 3ms/step - loss: 86621880.0000 - val\_loss: 6189611.0000

Epoch 88/150

72/72 [==============================] - 0s 3ms/step - loss: 86405544.0000 - val\_loss: 5863196.0000

Epoch 89/150

72/72 [==============================] - 0s 3ms/step - loss: 86211400.0000 - val\_loss: 5452313.5000

Epoch 90/150

72/72 [==============================] - 0s 3ms/step - loss: 85919784.0000 - val\_loss: 4788399.5000

Epoch 91/150

72/72 [==============================] - 0s 3ms/step - loss: 85648280.0000 - val\_loss: 4454097.5000

Epoch 92/150

72/72 [==============================] - 0s 3ms/step - loss: 85433296.0000 - val\_loss: 4076755.5000

Epoch 93/150

72/72 [==============================] - 0s 3ms/step - loss: 85204912.0000 - val\_loss: 3609808.2500

Epoch 94/150

72/72 [==============================] - 0s 3ms/step - loss: 84922648.0000 - val\_loss: 3127519.7500

Epoch 95/150

72/72 [==============================] - 0s 3ms/step - loss: 84671976.0000 - val\_loss: 2700583.0000

Epoch 96/150

72/72 [==============================] - 0s 3ms/step - loss: 84392568.0000 - val\_loss: 2474445.0000

Epoch 97/150

72/72 [==============================] - 0s 3ms/step - loss: 84159320.0000 - val\_loss: 2040386.2500

Epoch 98/150

72/72 [==============================] - 0s 3ms/step - loss: 83906208.0000 - val\_loss: 1734307.0000

Epoch 99/150

72/72 [==============================] - 0s 3ms/step - loss: 83642672.0000 - val\_loss: 1499641.5000

Epoch 100/150

72/72 [==============================] - 0s 3ms/step - loss: 83375824.0000 - val\_loss: 1219477.0000

Epoch 101/150

72/72 [==============================] - 0s 3ms/step - loss: 83145696.0000 - val\_loss: 1049215.3750

Epoch 102/150

72/72 [==============================] - 0s 3ms/step - loss: 82877808.0000 - val\_loss: 733560.5625

Epoch 103/150

72/72 [==============================] - 0s 4ms/step - loss: 82542056.0000 - val\_loss: 603985.7500

Epoch 104/150

72/72 [==============================] - 0s 3ms/step - loss: 82276176.0000 - val\_loss: 389924.4375

Epoch 105/150

72/72 [==============================] - 0s 3ms/step - loss: 81921176.0000 - val\_loss: 304936.6562

Epoch 106/150

72/72 [==============================] - 0s 3ms/step - loss: 81672328.0000 - val\_loss: 223821.9062

Epoch 107/150

72/72 [==============================] - 0s 4ms/step - loss: 81338624.0000 - val\_loss: 151503.5625

Epoch 108/150

72/72 [==============================] - 0s 3ms/step - loss: 80994920.0000 - val\_loss: 112137.9297

Epoch 109/150

72/72 [==============================] - 0s 3ms/step - loss: 80668632.0000 - val\_loss: 105530.6328

Epoch 110/150

72/72 [==============================] - 0s 3ms/step - loss: 80328552.0000 - val\_loss: 105530.6328

Epoch 111/150

72/72 [==============================] - 0s 3ms/step - loss: 79908392.0000 - val\_loss: 105530.6328

Epoch 112/150

72/72 [==============================] - 0s 3ms/step - loss: 79547272.0000 - val\_loss: 105530.6328

Epoch 113/150

72/72 [==============================] - 0s 3ms/step - loss: 79147624.0000 - val\_loss: 105530.6328

Epoch 114/150

72/72 [==============================] - 0s 3ms/step - loss: 78672688.0000 - val\_loss: 105530.6328

Epoch 115/150

72/72 [==============================] - 0s 3ms/step - loss: 78313800.0000 - val\_loss: 105530.6328

Epoch 116/150

72/72 [==============================] - 0s 3ms/step - loss: 77803520.0000 - val\_loss: 105530.6328

Epoch 117/150

72/72 [==============================] - 0s 4ms/step - loss: 77312640.0000 - val\_loss: 105530.6328

Epoch 118/150

72/72 [==============================] - 0s 3ms/step - loss: 76849488.0000 - val\_loss: 105530.6328

Epoch 119/150

72/72 [==============================] - 0s 3ms/step - loss: 76367192.0000 - val\_loss: 105530.6016

Epoch 120/150

72/72 [==============================] - 0s 3ms/step - loss: 75886824.0000 - val\_loss: 105530.6016

Epoch 121/150

72/72 [==============================] - 0s 3ms/step - loss: 75305600.0000 - val\_loss: 105530.6016

Epoch 122/150

72/72 [==============================] - 0s 3ms/step - loss: 74734912.0000 - val\_loss: 105530.6016

Epoch 123/150

72/72 [==============================] - 0s 3ms/step - loss: 74130880.0000 - val\_loss: 105530.6016

Epoch 124/150

72/72 [==============================] - 0s 3ms/step - loss: 73494576.0000 - val\_loss: 105530.6016

Epoch 125/150

72/72 [==============================] - 0s 3ms/step - loss: 73016800.0000 - val\_loss: 105530.6016

Epoch 126/150

72/72 [==============================] - 0s 3ms/step - loss: 72278752.0000 - val\_loss: 105530.6016

Epoch 127/150

72/72 [==============================] - 0s 3ms/step - loss: 71596720.0000 - val\_loss: 105530.5781

Epoch 128/150

72/72 [==============================] - 0s 3ms/step - loss: 70891544.0000 - val\_loss: 105530.5781

Epoch 129/150

72/72 [==============================] - 0s 3ms/step - loss: 70151776.0000 - val\_loss: 105530.5781

Epoch 130/150

72/72 [==============================] - 0s 3ms/step - loss: 69447592.0000 - val\_loss: 105530.5781

Epoch 131/150

72/72 [==============================] - 0s 3ms/step - loss: 68670128.0000 - val\_loss: 105530.5781

Epoch 132/150

72/72 [==============================] - 0s 3ms/step - loss: 67956224.0000 - val\_loss: 105530.5781

Epoch 133/150

72/72 [==============================] - 0s 3ms/step - loss: 67329168.0000 - val\_loss: 105530.5703

Epoch 134/150

72/72 [==============================] - 0s 3ms/step - loss: 66368836.0000 - val\_loss: 105530.5703

Epoch 135/150

72/72 [==============================] - 0s 4ms/step - loss: 65592724.0000 - val\_loss: 105530.5703

Epoch 136/150

72/72 [==============================] - 0s 3ms/step - loss: 64703200.0000 - val\_loss: 105530.5703

Epoch 137/150

72/72 [==============================] - 0s 3ms/step - loss: 63879008.0000 - val\_loss: 105530.5703

Epoch 138/150

72/72 [==============================] - 0s 3ms/step - loss: 62952220.0000 - val\_loss: 105530.5469

Epoch 139/150

72/72 [==============================] - 0s 3ms/step - loss: 62072272.0000 - val\_loss: 105530.5469

Epoch 140/150

72/72 [==============================] - 0s 3ms/step - loss: 61228524.0000 - val\_loss: 105530.5469

Epoch 141/150

72/72 [==============================] - 0s 3ms/step - loss: 60251628.0000 - val\_loss: 105530.5469

Epoch 142/150

72/72 [==============================] - 0s 4ms/step - loss: 59312336.0000 - val\_loss: 105530.5234

Epoch 143/150

72/72 [==============================] - 0s 4ms/step - loss: 58348632.0000 - val\_loss: 105530.5234

Epoch 144/150

72/72 [==============================] - 0s 3ms/step - loss: 57418592.0000 - val\_loss: 105530.5234

Epoch 145/150

72/72 [==============================] - 0s 3ms/step - loss: 56408612.0000 - val\_loss: 105530.5000

Epoch 146/150

72/72 [==============================] - 0s 3ms/step - loss: 55490168.0000 - val\_loss: 105530.5000

Epoch 147/150

72/72 [==============================] - 0s 3ms/step - loss: 54492456.0000 - val\_loss: 105530.5000

Epoch 148/150

72/72 [==============================] - 0s 3ms/step - loss: 53582928.0000 - val\_loss: 105530.4922

Epoch 149/150

72/72 [==============================] - 0s 3ms/step - loss: 52319024.0000 - val\_loss: 105530.4922

Epoch 150/150

72/72 [==============================] - 0s 3ms/step - loss: 51345316.0000 - val\_loss: 105530.4766