Configuration Manual

MSc Research Project
Data Analytics

Michael Ward
Student ID: x20190212

School of Computing
National College of Ireland

Supervisor: Zahid Iqbal

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Michael Ward		
Student ID:	x20190212		
Programme:	MSc in Data Analytics	**Year:**	2022
Module:	MSc Research Project		
Supervisor:	Zahid Iqbal		
Submission Due Date:	15/12/22		
Project Title:	Automated Detection of Semi-Conductor Wafer Map Defects Using Machine Learning Techniques		
Word Count:	………………………………	**Page Count:**	……………………………………………

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: ……

Date: ……

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):	

Configuration Manual

Michael Ward
Student ID: x20190212

1 Introduction

This configuration manual covers the specifications needed to successfully replicate the project. Each is laid out in a step-by-step manner that will enable implementation of the research project “Automated Detection of Semi-Conductor Wafer Map Defects Using Machine Learning Techniques”.

2 System configuration

2.1 Software Specification

- Windows 10 operating system
- Spyder Integrated Development Environment (IDE)

2.2 Hardware Specification

- HP ZBOOK PC
- Intel(R) Core(TM) i7-8850H CPU @ 2.60GHz 2.59 GHz
- 48.0 GB Ram
- Nvidia Quadro P1000

2.3 Prerequisites[1]

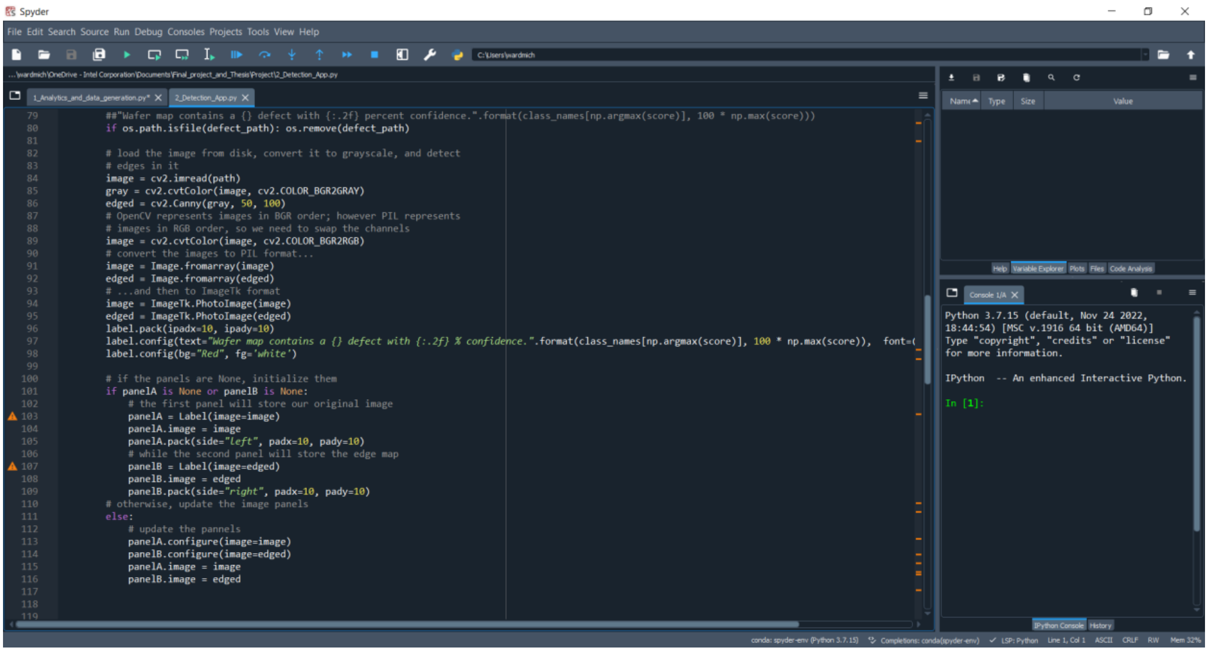
- NVIDIA® GPU card with CUDA® architectures 3.5, 5.0, 6.0, 7.0, 7.5, 8.0 and higher
- Goto https://developer.nvidia.com/cuda-gpus to check if your card is compatible
- Python 3.7–3.10
- pip version 19.0 or higher for Linux (requires manylinux2010 support) and Windows. pip version 20.3 or higher for macOS.
- Windows Native Requires Microsoft Visual C++ Redistributable for Visual Studio 2015, 2017 and 2019
- NVIDIA® GPU drivers version 450.80.02 or higher.
- CUDA® Toolkit 11.2.
- cuDNN SDK 8.1.0.
- (Optional) TensorRT to improve latency and throughput for inference.

[1] https://www.tensorflow.org/install/pip

3 Implementation

3.1 Code Set Up

The code was developed in the Spyder IDE. It allowed for a more robust development experience for python.



Follow the below steps set up the project.

1. Open Research_project.zip
2. The folder contains 5 items.
3. Extract the contents to any directory as seen in Figure 1.
4. Download the dataset from

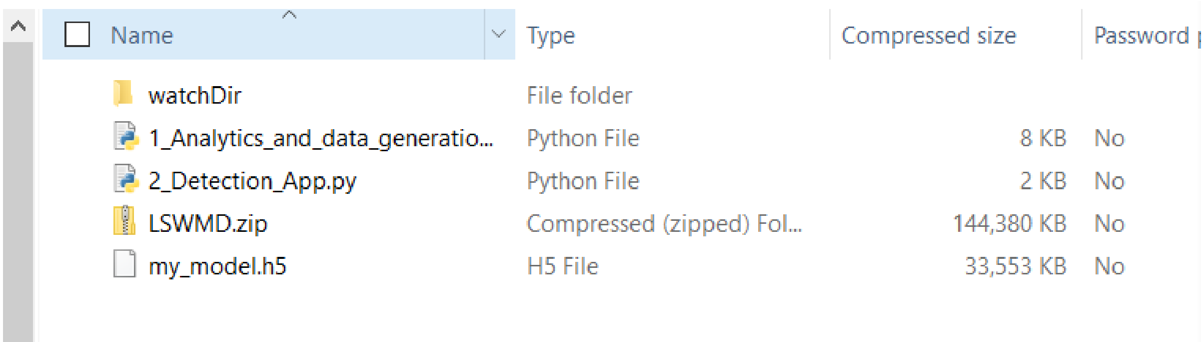


Figure 1 project zip contents

5. Download the dataset from here
6. Place the zip file in the same directory as the python files
7. Once extracted you have two options. You can open the full project and run the analytical and data generation code. Or you can open the app and load the pre saved model. The zip also contains test maps for quick use with the second python file.

3.2 WatchDir

This is a directory where the automated checking of the app will constantly look for a test.png file.

3.3 1_Analytics_and_data_generation.py

The file contains a total of 16 functions

- def find_and_read_file()
- def perform_analytics()
- def data_cleaning()
- def wafer_generation()
- def cal_den()
- def change_val()
- def cubic_inter_mean()
- def cubic_inter_mean()
- def find_regions()
- def cal_dist()
- def fea_geom()
- def plot_confusion_matrix()

Running this file will automatically do the analytics and generate and manipulate the data set.

3.3.1 Packages

Below in figure 2 is a list of all packages used in the Analytics_and_data_generation.py.

```python
#%% Importing packages
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from tensorflow import keras
import warnings
from pathlib import Path
import zipfile
from skimage import measure
from skimage.transform import radon
from scipy import interpolate
from scipy import stats
import theano
from theano import tensor as T
from keras.utils import np_utils
from matplotlib import gridspec
import itertools
from sklearn.metrics import classification_report
import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
```

Figure 2 imported packages

3.3.2 Functions()

Function looks to see if the dataset exists, if not then it will extract it from the zip file and add it to a data frame as seen from figure 3 below.

```python
def find_and_read_file():
    my_file = Path("LSWMD.pkl")
    if my_file.is_file():
        print("Dataset exists in current working directory")
    # file exists
    else:
        print("unzipping dataset")
        with zipfile.ZipFile("LSWMD.zip","r") as zip_ref:
            zip_ref.extractall("")
        if my_file.is_file():
            print("dataset extraction succesful")
        # file exists
    warnings.filterwarnings("ignore")
    print("Reading dataset, please wait....")
    df=pd.read_pickle("LSWMD.pkl")
    print("Dataframe created")
    df.info()
    df.head()
    df.tail()

    return(df)
```

Figure 3 def find_and_read_file()

This function plots a bar chart of the wafer frequency as seen in figure 4.

```python
def perform_analytics(df):
    uni_Index=np.unique(df.waferIndex, return_counts=True)
    plt.bar(uni_Index[0],uni_Index[1], color='blue', align='center', alpha=0
    plt.xlabel("Number of wafers")
    plt.ylabel("frequency")
    plt.xlim(0,26)
    plt.ylim(30000,35000)
    plt.show()

perform_analytics(df)
```

Figure 4 perform_analytics()

runs data cleaning and finds the dimensions of the wafers.

```python
def data_cleaning(df):
    df = df.drop(['waferIndex'], axis = 1)

    def find_dim(x):
        dim0=np.size(x,axis=0)
        dim1=np.size(x,axis=1)
        return dim0,dim1
    df['wmDim']=df.waferMap.apply(find_dim)
    print(df.sample(5))
    print("Max Dim: ",max(df.wmDim), " Min Dim:", min(df.wm
    uni_waferDim=np.unique(df.wmDim, return_counts=True)
    uni_waferDim[0].shape[0]
    return(df)
```

Figure 5 data cleaning

This function is to big to fit into a figure so only partial code is show in figure 6.

```python
def wafer_generation(df):
    #-----------------------------------------------------------
    df_Center = df[(df.failureType == "Center")]
    df_Donut = df[(df.failureType == "Donut")]
    df_Edge_loc = df[(df.failureType == "Edge-Loc")]
    df_Edge_Ring = df[(df.failureType == "Edge-Ring")]
    df_loc = df[(df.failureType == "Loc")]
    df_Random = df[(df.failureType == "Random")]
    df_Scratch = df[(df.failureType == "Scratch")]
    df_Near_Full = df[(df.failureType == "Near-full")]
    df_None = df[(df.failureType == "None")]

    df_Scratch.head()

    df_Scratch.tail()

    df_Scratch.info()

    #adding Dim columns to scratch data frame
    def find_dim(x):
        dim0=np.size(x,axis=0)
        dim1=np.size(x,axis=1)
        return dim0,dim1



    #adding Dim columns to df_Center data frame
    df_Center['wmDim']=df_Center.waferMap.apply(find_dim)
    df_Center.sample(5)

    #adding Dim columns to df_Donut data frame
    df_Donut['wmDim']=df_Donut.waferMap.apply(find_dim)
```

Figure 6 wafer_generation()

The cal_den() function helps calculate the wafer density as shown in figure 7.

```
def cal_den(x):
    return 100*(np.sum(x==2)/np.size(x))
```

Figure 7 cal_den()

Below function calculates the regions on the wafer as shown in figure 8.

```
def find_regions(x):
    rows=np.size(x,axis=0)
    cols=np.size(x,axis=1)
    ind1=np.arange(0,rows,rows//5)
    ind2=np.arange(0,cols,cols//5)

    reg1=x[ind1[0]:ind1[1],:]
    reg3=x[ind1[4]:,:]
    reg4=x[:,ind2[0]:ind2[1]]
    reg2=x[:,ind2[4]:]

    reg5=x[ind1[1]:ind1[2],ind2[1]:ind2[2]]
    reg6=x[ind1[1]:ind1[2],ind2[2]:ind2[3]]
    reg7=x[ind1[1]:ind1[2],ind2[3]:ind2[4]]
    reg8=x[ind1[2]:ind1[3],ind2[1]:ind2[2]]
    reg9=x[ind1[2]:ind1[3],ind2[2]:ind2[3]]
    reg10=x[ind1[2]:ind1[3],ind2[3]:ind2[4]]
    reg11=x[ind1[3]:ind1[4],ind2[1]:ind2[2]]
    reg12=x[ind1[3]:ind1[4],ind2[2]:ind2[3]]
    reg13=x[ind1[3]:ind1[4],ind2[3]:ind2[4]]

    fea_reg_den = []
    fea_reg_den = [cal_den(reg1),cal_den(reg2),
    return fea_reg_den
```

Figure 8 find regions

Below functions are used for the calculation of the radon transforms.

```
def cubic_inter_mean(img):
    theta = np.linspace(0., 180., max(img.shape), endpoint=False)
    sinogram = radon(img, theta=theta)
    xMean_Row = np.mean(sinogram, axis = 1)
    x = np.linspace(1, xMean_Row.size, xMean_Row.size)
    y = xMean_Row
    f = interpolate.interp1d(x, y, kind = 'cubic')
    xnew = np.linspace(1, xMean_Row.size, 20)
    ynew = f(xnew)/100   # use interpolation function returned by `interp1d`
    return ynew

def cubic_inter_std(img):
    theta = np.linspace(0., 180., max(img.shape), endpoint=False)
    sinogram = radon(img, theta=theta)
    xStd_Row = np.std(sinogram, axis=1)
    x = np.linspace(1, xStd_Row.size, xStd_Row.size)
    y = xStd_Row
    f = interpolate.interp1d(x, y, kind = 'cubic')
    xnew = np.linspace(1, xStd_Row.size, 20)
    ynew = f(xnew)/100   # use interpolation function returned by `interp1d`
    return ynew
```

Figure 9 cubic inter mean

Below code is used for the calculation and extraction of densirt based features.

```
def fea_geom(img):
    norm_area=img.shape[0]*img.shape[1]
    norm_perimeter=np.sqrt((img.shape[0])**2+(img.shape[1])**2)

    img_labels = measure.label(img,  connectivity=1, background=0)

    if img_labels.max()==0:
        img_labels[img_labels==0]=1
        no_region = 0
    else:
        info_region = stats.mode(img_labels[img_labels>0], axis = None)
        no_region = info_region[0][0]-1

    prop = measure.regionprops(img_labels)
    prop_area = prop[no_region].area/norm_area
    prop_perimeter = prop[no_region].perimeter/norm_perimeter

    prop_cent = prop[no_region].local_centroid
    prop_cent = cal_dist(img,prop_cent[0],prop_cent[1])

    prop_majaxis = prop[no_region].major_axis_length/norm_perimeter
    prop_minaxis = prop[no_region].minor_axis_length/norm_perimeter
    prop_ecc = prop[no_region].eccentricity
    prop_solidity = prop[no_region].solidity

    return prop_area,prop_perimeter,prop_majaxis,prop_minaxis,prop_ecc,prop_solidity
```

Figure 10 fea geom

```python
# ---multicalss classification ---#
# One-Vs-One
from sklearn.svm import LinearSVC
from sklearn.multiclass import OneVsOneClassifier
clf2 = OneVsOneClassifier(LinearSVC(random_state = RANDOM_STATE)).fit(X_train, y_train)
print(X_train)
y_train_pred = clf2.predict(X_train)
y_test_pred = clf2.predict(X_test)
train_accuracy2 = np.sum(y_train == y_train_pred, axis=0, dtype='float') / X_train.shape[0]
print(train_accuracy2)
test_accuracy2 = np.sum(y_test == y_test_pred, axis=0, dtype='float') / X_test.shape[0]
print('One-Vs-One Training accuracy: {}'.format(X_train*100))
print('One-Vs-One Testing accuracy: {}'.format(X_test*100))
print("y_train_pred[:100]: ", y_train_pred[:100])
print ("y_train[:100]: ", y_train[:100])


print(X_test)

def plot_confusion_matrix(cm, normalize=False, title='Confusion matrix', cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    #print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True Defect')
    plt.xlabel('Predicted Defect')
```

Figure 11 SVM classification

Above code is use the create the SVM mode and plot the confusion matrix.

```python
# Compute confusion matrix
cnf_matrix = confusion_matrix(y_test, y_test_pred)
np.set_printoptions(precision=2)

from matplotlib import gridspec
fig = plt.figure(figsize=(15, 8))
gs = gridspec.GridSpec(1, 2, width_ratios=[1, 1])

## Plot non-normalized confusion matrix
plt.subplot(gs[0])
plot_confusion_matrix(cnf_matrix, title='Confusion matrix')

# Plot normalized confusion matrix
plt.subplot(gs[1])
plot_confusion_matrix(cnf_matrix, normalize=True, title='Normalized confusion matrix')

plt.show()

y_true = y_train
y_pred = y_train_pred
target_names =  ['Center','Donut','Edge-Loc','Edge-Ring','Loc','Random','Scratch','Near-full']
print(classification_report(y_true, y_pred, target_names=target_names))
```

Figure 12 confusion matrix

3.3.3 Sequential Neural Network

The next part of the code is in relation to the sequential neural network. Figure 13 below shows a partial amount of the code. It imports a local dataset that is generated in a previous function. It will import that data and perform a data analysis and train the neural network.

```python
#---------------------------------------------- Neural Network------------

data_dir = os.path.abspath('C:/Users/wardmich/OneDrive - Intel Corporation/Documents/Final_project_and_Thesis/Project/data')

print(data_dir)

batch_size = 32
img_height = 180
img_width = 180

train_ds = tf.keras.utils.image_dataset_from_directory(
  data_dir,
  validation_split=0.2,
  subset="training",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)

val_ds = tf.keras.utils.image_dataset_from_directory(
  data_dir,
  validation_split=0.2,
  subset="validation",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)

class_names = train_ds.class_names
print(class_names)

import matplotlib.pyplot as plt

plt.figure(figsize=(10, 10))
for images, labels in train_ds.take(1):
  for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(images[i].numpy().astype("uint8"))
    plt.title(class_names[labels[i]])
    plt.axis("off")


for image_batch, labels_batch in train_ds:
  print(image_batch.shape)
  print(labels_batch.shape)
  break
```

Figure 13 sequential model

3.4 2_Detection_App.py

This is the second python file. It contains the developed application. It uses the Tkinter package to make the create the graphical user interface. This file can be ran independently of the first python file as it will import a presaved model named “My_Model”. Below figure.

```python
from tkinter import *
from PIL import Image
from PIL import ImageTk
from tkinter import filedialog
import cv2
import tensorflow as tf
import os
import threading
import time
import numpy as np

batch_size = 32
img_height = 180
img_width = 180


#%%
cd = os.getcwd()
data_dir = os.path.abspath(cd+'/data')
train_ds = tf.keras.utils.image_dataset_from_directory(
  data_dir,
  validation_split=0.2,
  subset="training",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)

val_ds = tf.keras.utils.image_dataset_from_directory(
  data_dir,
  validation_split=0.2,
  subset="validation",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)



class_names = train_ds.class_names
print(class_names)
imported_model = tf.keras.models.load_model('data/my_model.h5')

# Show the model architecture
imported_model.summary()
```

Figure 14

Once the code is executed, the GUI will pop up. From here, the user has two options as seen below in figure 15. When the user clicks the Manual wafer map inspection button, they call browse to wherever they have wafer maps stored. Click on the wafer map and import it into the detection app.

Figure 15 option selection

The user will then be presented with the screen as shown below in figure 16. It shows the wafer map that the user loaded. The salient version of the wafer map and the classification of the wafer map with an accuracy percentage displayed at the top of the app window. Next, if the user clicks the Automated wafer map inspection, it will scan the WatchDir folder for new test.png wafer maps. An alert will display under the results label that Auto Watch is active, as seen from figure 17 below.

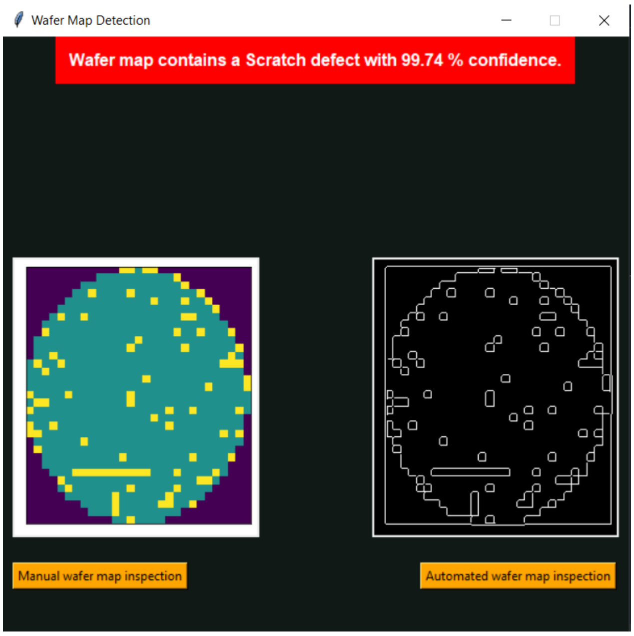


Figure 16 Detection app

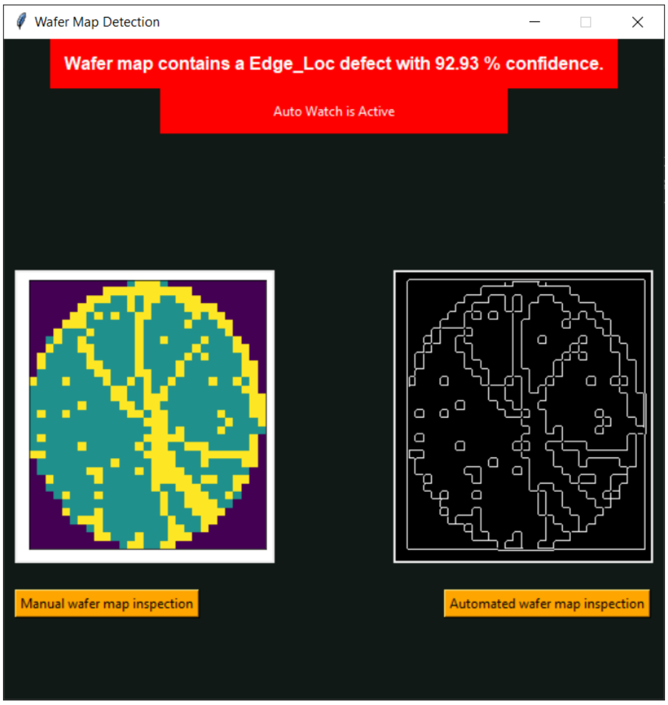


Figure 17 Auto watch enabled

3.5 LSWMD.zip

This zip file contains a compressed version of the dataset at a size of 157,513KB. When extracted it contains the LSWMD.pkl dataset. The decompressed size of this dataset is 2,046,393KB.

3.6 my_model.h5

This is the sequential model saved in a .h5 format. The model can be easily reloaded for use with the app.