

*COLLEGE CODE*-**5113**

**M Aravindh Kumar-(511321104006)[-bhairocky155@gmail.com](mailto:-bhairocky155@gmail.com)**

**P Mukesh-(511321104058)[-bujimukesh7@gmail.com](mailto:-bujimukesh7@gmail.com)**

**S Jayaprakash-(511321104034)[-jayaprakashkomathi642@gmail.com](mailto:-jayaprakashkomathi642@gmail.com)**

**J Jayaprakash-(511321104035)[-jayaprakash18204@gmail.com](mailto:-jayaprakash18204@gmail.com)**

**R Madeshwaran-(511321104050)[-madesh4533@gmail.com](mailto:-madesh4533@gmail.com)**

**PROJECT : Create a chatbot in Python**

PHASE 2:INNOVATION

Innovative Transformation of creating a chatbot in python : From Design to Implementation

**AI-Powered Diabetes Prediction System: Problem Understanding and Design**

***Table of Contents***

1.Introduction

2.Problem Statement Understanding

3.Solution Approach

3.1Data Collection

3.2Data Preprocessing

3.3Feature Engineering

3.4Model Selection

3.5Model Training

3.6Evaluation Metrics

4.Advanced Techniques Integration

5.Dataset

6.Conclusion

**1. Introduction**

Diabetes is a widespread chronic disease with significant health implications. Early detection and proactive management are essential in preventing complications and improving the quality of life for affected individuals. In this document, we will discuss the problem of building an AI-powered diabetes prediction system. This system aims to utilize machine learning algorithms to analyze medical data and predict the likelihood of an individual developing diabetes. By providing early risk assessments and personalized preventive measures, the system empowers individuals to take proactive actions to manage their health.

**2. Problem Statement Understanding**

To build an effective AI-powered diabetes prediction system, it is crucial to thoroughly understand the problem statement. Let's break down the key components of this problem:

***2.1 Data Availability***

We need access to a substantial dataset of medical records, including information such as patients' age, gender, family history, lifestyle factors (diet, exercise), and medical test results (e.g., glucose levels, BMI). This data should be both comprehensive and representative of a diverse population.

***2.2 Prediction Goal***

The primary goal is to predict the likelihood of an individual developing diabetes. This prediction should be binary (yes/no), indicating whether an individual is at high risk or not. Additionally, it would be beneficial to provide a probability score to quantify the risk level.

***2.3 Personalized Preventive Measures***

The system should not stop at prediction but also provide personalized recommendations to mitigate the risk of diabetes. These recommendations could include dietary changes, exercise routines, and regular health check-ups.

***2.4 Ethical and Privacy Considerations***

Handling sensitive medical data requires strict adherence to privacy regulations and ethical guidelines. Ensuring data security and patient anonymity is of utmost importance.

**3. Solution Approach**

To address the problem of diabetes prediction effectively, we will follow a structured solution approach:

***3.1 Data Collection***

Identify and collaborate with healthcare institutions or providers to obtain a diverse and comprehensive dataset of medical records.

Ensure strict adherence to data privacy regulations and obtain necessary permissions for data usage.

***3.2 Data Preprocessing***

Perform data cleaning to handle missing values, outliers, and inconsistencies in the dataset.

Normalize or scale features to ensure that they have a consistent impact on model training.

***3.3 Feature Engineering***

Extract relevant features from the raw medical data, such as BMI, cholesterol levels, and family history.

Create additional features that might provide valuable insights, such as the average glucose level over time.

***3.4 Model Selection***

Explore various machine learning algorithms suitable for binary classification, such as logistic regression, decision trees, random forests, and support vector machines.

Consider using advanced techniques like deep learning and neural networks to capture complex patterns in the data.

***3.5 Model Training***

Split the dataset into training, validation, and testing sets to assess model performance.

Train selected models on the training data while fine-tuning hyperparameters.

***3.6 Evaluation Metrics***

Use appropriate evaluation metrics, such as accuracy, precision, recall, F1-score, and AUC-ROC, to assess model performance.

Optimize models based on these metrics to achieve the desired level of prediction accuracy.

**4. Advanced Techniques Integration**

To enhance the quality of responses and recommendations, we can consider integrating pre-trained language models like GPT-3. These models can assist in:

Natural Language Processing: Understanding and generating human-readable explanations for predictions and recommendations.

Personalization: Tailoring recommendations based on user preferences and medical history.

Data Augmentation: Generating synthetic data to address potential data scarcity issues, ensuring robust model training.

Handling Unstructured Data: Processing text-based medical records or patient notes to extract valuable insights.

However, it's important to note that advanced techniques should be used judiciously and ethically, considering privacy concerns and data security.

**5.Dataset**

Link:<https://www.kaggle.com/datasets/grafstor/simple-dialogs-for-chatbot>

**Program :**

Create a chatbot in Python

import tensorflow as tf

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from tensorflow.keras.layers import TextVectorization

import re,string

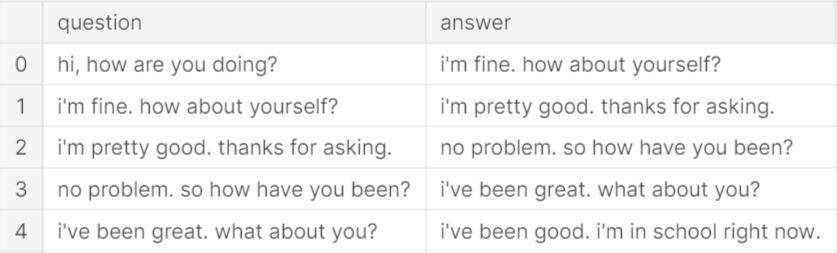
from tensorflow.keras.layers import LSTM,Dense,Embedding,Dropout,LayerNormalization

df=pd.read\_csv('/kaggle/input/simple-dialogs-for-chatbot/dialogs.txt',sep='\t',names=['question','answer'])

print(f'Dataframe size: {len(df)}')

df.head()

Output 1:

****

# Data Preprocessing

## *Data Visualization*

df['question tokens']=df['question'].apply(lambda x:len(x.split()))

df['answer tokens']=df['answer'].apply(lambda x:len(x.split()))

plt.style.use('fivethirtyeight')

fig,ax=plt.subplots(nrows=1,ncols=2,figsize=(20,5))

sns.set\_palette('Set2')

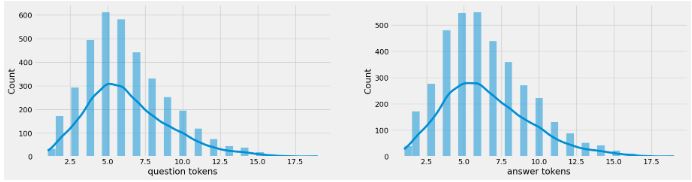
sns.histplot(x=df['question tokens'],data=df,kde=True,ax=ax[0])

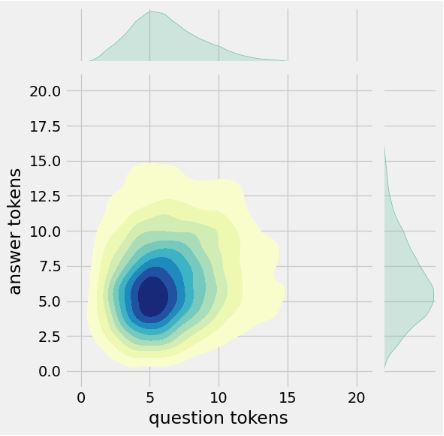
sns.histplot(x=df['answer tokens'],data=df,kde=True,ax=ax[1])

sns.jointplot(x='question tokens',y='answer tokens',data=df,kind='kde',fill=True,cmap='YlGnBu')

plt.show()

Output 2:

****

****

## **Text Cleaning**

def clean\_text(text):

text=re.sub('-',' ',text.lower())

text=re.sub('[.]',' . ',text)

text=re.sub('[1]',' 1 ',text)

text=re.sub('[2]',' 2 ',text)

text=re.sub('[3]',' 3 ',text)

text=re.sub('[4]',' 4 ',text)

text=re.sub('[5]',' 5 ',text)

text=re.sub('[6]',' 6 ',text)

text=re.sub('[7]',' 7 ',text)

text=re.sub('[8]',' 8 ',text)

text=re.sub('[9]',' 9 ',text)

text=re.sub('[0]',' 0 ',text)

text=re.sub('[,]',' , ',text)

text=re.sub('[?]',' ? ',text)

text=re.sub('[!]',' ! ',text)

text=re.sub('[$]',' $ ',text)

text=re.sub('[&]',' & ',text)

text=re.sub('[/]',' / ',text)

text=re.sub('[:]',' : ',text)

text=re.sub('[;]',' ; ',text)

text=re.sub('[\*]',' \* ',text)

text=re.sub('[\']',' \' ',text)

text=re.sub('[\"]',' \" ',text)

text=re.sub('\t',' ',text)

return text

df.drop(columns=['answer tokens','question tokens'],axis=1,inplace=True)

df['encoder\_inputs']=df['question'].apply(clean\_text)

df['decoder\_targets']=df['answer'].apply(clean\_text)+' <end>'

df['decoder\_inputs']='<start> '+df['answer'].apply(clean\_text)+' <end>'

df.head(10)

Output 3:

****

df['encoder input tokens']=df['encoder\_inputs'].apply(lambda x:len(x.split()))

df['decoder input tokens']=df['decoder\_inputs'].apply(lambda x:len(x.split()))

df['decoder target tokens']=df['decoder\_targets'].apply(lambda x:len(x.split()))

plt.style.use('fivethirtyeight')

fig,ax=plt.subplots(nrows=1,ncols=3,figsize=(20,5))

sns.set\_palette('Set2')

sns.histplot(x=df['encoder input tokens'],data=df,kde=True,ax=ax[0])

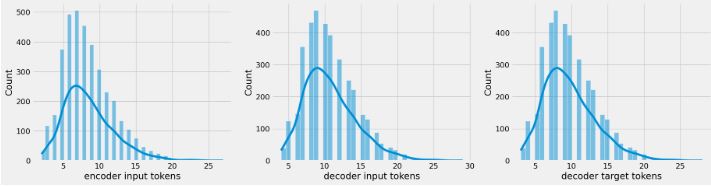
sns.histplot(x=df['decoder input tokens'],data=df,kde=True,ax=ax[1])

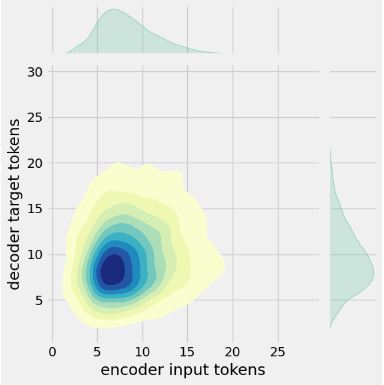
sns.histplot(x=df['decoder target tokens'],data=df,kde=True,ax=ax[2])

sns.jointplot(x='encoder input tokens',y='decoder target tokens',data=df,kind='kde',fill=True,cmap='YlGnBu')

plt.show()

Output 4:





print(f"After preprocessing: {' '.join(df[df['encoder input tokens'].max()==df['encoder input tokens']]['encoder\_inputs'].values.tolist())}")

print(f"Max encoder input length: {df['encoder input tokens'].max()}")

print(f"Max decoder input length: {df['decoder input tokens'].max()}")

print(f"Max decoder target length: {df['decoder target tokens'].max()}")

df.drop(columns=['question','answer','encoder input tokens','decoder input tokens','decoder target tokens'],axis=1,inplace=True)

params={

"vocab\_size":2500,

"max\_sequence\_length":30,

"learning\_rate":0.008,

"batch\_size":149,

"lstm\_cells":256,

"embedding\_dim":256,

"buffer\_size":10000

}

learning\_rate=params['learning\_rate']

batch\_size=params['batch\_size']

embedding\_dim=params['embedding\_dim']

lstm\_cells=params['lstm\_cells']

vocab\_size=params['vocab\_size']

buffer\_size=params['buffer\_size']

max\_sequence\_length=params['max\_sequence\_length']

df.head(10)

Output 5:

**After preprocessing: for example , if your birth date is january 1 2 , 1 9 8 7 , write 0 1 / 1 2 / 8 7 .**

**Max encoder input length: 27**

**Max decoder input length: 29**

**Max decoder target length: 28**



## Tokenization

vectorize\_layer=TextVectorization(

max\_tokens=vocab\_size,

standardize=None,

output\_mode='int',

output\_sequence\_length=max\_sequence\_length

)

vectorize\_layer.adapt(df['encoder\_inputs']+' '+df['decoder\_targets']+' <start> <end>')

vocab\_size=len(vectorize\_layer.get\_vocabulary())

print(f'Vocab size: {len(vectorize\_layer.get\_vocabulary())}')

print(f'{vectorize\_layer.get\_vocabulary()[:12]}')

def sequences2ids(sequence):

return vectorize\_layer(sequence)

def ids2sequences(ids):

decode=''

if type(ids)==int:

ids=[ids]

for id in ids:

decode+=vectorize\_layer.get\_vocabulary()[id]+' '

return decode

x=sequences2ids(df['encoder\_inputs'])

yd=sequences2ids(df['decoder\_inputs'])

y=sequences2ids(df['decoder\_targets'])

print(f'Question sentence: hi , how are you ?')

print(f'Question to tokens: {sequences2ids("hi , how are you ?")[:10]}')

print(f'Encoder input shape: {x.shape}')

print(f'Decoder input shape: {yd.shape}')

print(f'Decoder target shape: {y.shape}')

Output 6:

**Question sentence: hi , how are you ?**

**Question to tokens: [1971 9 45 24 8 7 0 0 0 0]**

**Encoder input shape: (3725, 30)**

**Decoder input shape: (3725, 30)**

**Decoder target shape: (3725, 30)**

# Visualize Metrics

fig,ax=plt.subplots(nrows=1,ncols=2,figsize=(20,5))

ax[0].plot(history.history['loss'],label='loss',c='red')

ax[0].plot(history.history['val\_loss'],label='val\_loss',c = 'blue')

ax[0].set\_xlabel('Epochs')

ax[1].set\_xlabel('Epochs')

ax[0].set\_ylabel('Loss')

ax[1].set\_ylabel('Accuracy')

ax[0].set\_title('Loss Metrics')

ax[1].set\_title('Accuracy Metrics')

ax[1].plot(history.history['accuracy'],label='accuracy')

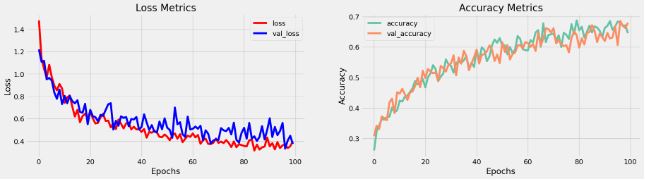
ax[1].plot(history.history['val\_accuracy'],label='val\_accuracy')

ax[0].legend()

ax[1].legend()

plt.show()

Output 7:



# Save Model

model.load\_weights('ckpt')

model.save('models',save\_format='tf')

for idx,i in enumerate(model.layers):

print('Encoder layers:' if idx==0 else 'Decoder layers: ')

for j in i.layers:

print(j)

print('---------------------')

# Create Inference Model

class ChatBot(tf.keras.models.Model):

def \_\_init\_\_(self,base\_encoder,base\_decoder,\*args,\*\*kwargs):

super().\_\_init\_\_(\*args,\*\*kwargs)

self.encoder,self.decoder=self.build\_inference\_model(base\_encoder,base\_decoder)

def build\_inference\_model(self,base\_encoder,base\_decoder):

encoder\_inputs=tf.keras.Input(shape=(None,))

x=base\_encoder.layers[0](encoder\_inputs)

x=base\_encoder.layers[1](x)

x,encoder\_state\_h,encoder\_state\_c=base\_encoder.layers[2](x)

encoder=tf.keras.models.Model(inputs=encoder\_inputs,outputs=[encoder\_state\_h,encoder\_state\_c],name='chatbot\_encoder')

decoder\_input\_state\_h=tf.keras.Input(shape=(lstm\_cells,))

decoder\_input\_state\_c=tf.keras.Input(shape=(lstm\_cells,))

decoder\_inputs=tf.keras.Input(shape=(None,))

x=base\_decoder.layers[0](decoder\_inputs)

x=base\_encoder.layers[1](x)

x,decoder\_state\_h,decoder\_state\_c=base\_decoder.layers[2](x,initial\_state=[decoder\_input\_state\_h,decoder\_input\_state\_c])

decoder\_outputs=base\_decoder.layers[-1](x)

decoder=tf.keras.models.Model(

inputs=[decoder\_inputs,[decoder\_input\_state\_h,decoder\_input\_state\_c]],

outputs=[decoder\_outputs,[decoder\_state\_h,decoder\_state\_c]],name='chatbot\_decoder'

)

return encoder,decoder

def summary(self):

self.encoder.summary()

self.decoder.summary()

def softmax(self,z):

return np.exp(z)/sum(np.exp(z))

def sample(self,conditional\_probability,temperature=0.5):

conditional\_probability = np.asarray(conditional\_probability).astype("float64")

conditional\_probability = np.log(conditional\_probability) / temperature

reweighted\_conditional\_probability = self.softmax(conditional\_probability)

probas = np.random.multinomial(1, reweighted\_conditional\_probability, 1)

return np.argmax(probas)

def preprocess(self,text):

text=clean\_text(text)

seq=np.zeros((1,max\_sequence\_length),dtype=np.int32)

for i,word in enumerate(text.split()):

seq[:,i]=sequences2ids(word).numpy()[0]

return seq

def postprocess(self,text):

text=re.sub(' - ','-',text.lower())

text=re.sub(' [.] ','. ',text)

text=re.sub(' [1] ','1',text)

text=re.sub(' [2] ','2',text)

text=re.sub(' [3] ','3',text)

text=re.sub(' [4] ','4',text)

text=re.sub(' [5] ','5',text)

text=re.sub(' [6] ','6',text)

text=re.sub(' [7] ','7',text)

text=re.sub(' [8] ','8',text)

text=re.sub(' [9] ','9',text)

text=re.sub(' [0] ','0',text)

text=re.sub(' [,] ',', ',text)

text=re.sub(' [?] ','? ',text)

text=re.sub(' [!] ','! ',text)

text=re.sub(' [$] ','$ ',text)

text=re.sub(' [&] ','& ',text)

text=re.sub(' [/] ','/ ',text)

text=re.sub(' [:] ',': ',text)

text=re.sub(' [;] ','; ',text)

text=re.sub(' [\*] ','\* ',text)

text=re.sub(' [\'] ','\'',text)

text=re.sub(' [\"] ','\"',text)

return text

def call(self,text,config=None):

input\_seq=self.preprocess(text)

states=self.encoder(input\_seq,training=False)

target\_seq=np.zeros((1,1))

target\_seq[:,:]=sequences2ids(['<start>']).numpy()[0][0]

stop\_condition=False

decoded=[]

while not stop\_condition:

decoder\_outputs,new\_states=self.decoder([target\_seq,states],training=False)

# index=tf.argmax(decoder\_outputs[:,-1,:],axis=-1).numpy().item()

index=self.sample(decoder\_outputs[0,0,:]).item()

word=ids2sequences([index])

if word=='<end> ' or len(decoded)>=max\_sequence\_length:

stop\_condition=True

else:

decoded.append(index)

target\_seq=np.zeros((1,1))

target\_seq[:,:]=index

states=new\_states

return self.postprocess(ids2sequences(decoded))

chatbot=ChatBot(model.encoder,model.decoder,name='chatbot')

chatbot.summary()

Model: "chatbot\_encoder"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

input\_1 (InputLayer) [(None, None)] 0

encoder\_embedding (Embeddin (None, None, 256) 625408

g)

layer\_normalization (LayerN (None, None, 256) 512

ormalization)

encoder\_lstm (LSTM) [(None, None, 256), 525312

(None, 256),

(None, 256)]

=================================================================

Total params: 1,151,232

Trainable params: 1,151,232

Non-trainable params: 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Model: "chatbot\_decoder"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param # Connected to

==================================================================================================

input\_4 (InputLayer) [(None, None)] 0 []

decoder\_embedding (Embedding) (None, None, 256) 625408 ['input\_4[0][0]']

layer\_normalization (LayerNorm (None, None, 256) 512 ['decoder\_embedding[0][0]']

alization)

input\_2 (InputLayer) [(None, 256)] 0 []

input\_3 (InputLayer) [(None, 256)] 0 []

decoder\_lstm (LSTM) [(None, None, 256), 525312 ['layer\_normalization[1][0]',

(None, 256), 'input\_2[0][0]',

(None, 256)] 'input\_3[0][0]']

decoder\_dense (Dense) (None, None, 2443) 627851 ['decoder\_lstm[0][0]']

==================================================================================================

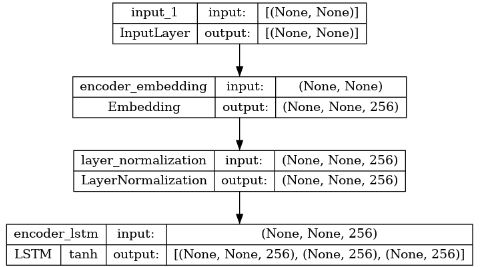
Total params: 1,779,083

Trainable params: 1,779,083

Non-trainable params: 0

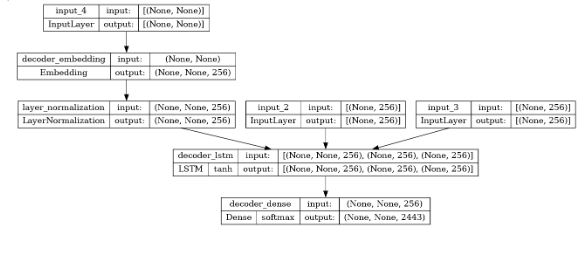
tf.keras.utils.plot\_model(chatbot.encoder,to\_file='encoder.png',show\_shapes=True,show\_layer\_activations=True)

Output 8:



tf.keras.utils.plot\_model(chatbot.decoder,to\_file='decoder.png',show\_shapes=True,show\_layer\_activations=True)

Output 9:



# Time to Chat

def print\_conversation(texts):

for text in texts:

print(f'You: {text}')

print(f'Bot: {chatbot(text)}')

print('========================')

print\_conversation([

'hi',

'do yo know me?',

'what is your name?',

'you are bot?',

'hi, how are you doing?',

"i'm pretty good. thanks for asking.",

"Don't ever be in a hurry",

'''I'm gonna put some dirt in your eye ''',

'''You're trash ''',

'''I've read all your research on nano-technology ''',

'''You want forgiveness? Get religion''',

'''While you're using the bathroom, i'll order some food.''',

'''Wow! that's terrible.''',

'''We'll be here forever.''',

'''I need something that's reliable.''',

'''A speeding car ran a red light, killing the girl.''',

'''Tomorrow we'll have rice and fish for lunch.''',

'''I like this restaurant because they give you free bread.'''

])

Output 10:

You: hi

Bot: i have to go to the bathroom.

========================

You: do yo know me?

Bot: yes, it's too close to the other.

========================

You: what is your name?

Bot: i have to walk the house.

========================

You: you are bot?

Bot: no, i have. all my life.

========================

You: hi, how are you doing?

Bot: i'm going to be a teacher.

========================

You: i'm pretty good. thanks for asking.

Bot: no problem. i'll have to give you the english assignments from my mind.

========================

You: Don't ever be in a hurry

Bot: it's not a great.

========================

You: I'm gonna put some dirt in your eye

Bot: that's a good idea.

========================

You: You're trash

Bot: the tv news is reporting a bank robbery.

========================

You: I've read all your research on nano-technology

Bot: it's the weather. i've gone around the world.

========================

You: You want forgiveness? Get religion

Bot: no, i'll be my.

========================

You: While you're using the bathroom, i'll order some food.

Bot: don't order for me. i've been a cheater.

========================

You: Wow! that's terrible.

Bot: never park your car under the house.

========================

You: We'll be here forever.

Bot: we'll be there in half an hour.

========================

You: I need something that's reliable.

Bot: you need a car with low mileage.

========================

You: A speeding car ran a red light, killing the girl.

Bot: what happened?

========================

You: Tomorrow we'll have rice and fish for lunch.

Bot: i'll make a sandwich.

========================

You: I like this restaurant because they give you free bread.

Bot: well, i think that's a good idea.

========================

6.Conclusion

Building an AI-powered diabetes prediction system is a multifaceted task that requires a thorough understanding of the problem statement and a structured approach to data collection, preprocessing, model selection, training, and evaluation. Furthermore, integrating advanced techniques like pre-trained language models can significantly enhance the system's quality and user experience.

By following this approach, we can develop a robust and effective system that empowers individuals with early risk assessments and personalized preventive measures, ultimately contributing to better health outcomes in the fight against diabetes.