Experiment - 1

Basic python code

1) Find the square root of number

```
import math
n= int(input())
print(math.sqrt(n))
```

→ 4
2.0

2) Swap two numbers

```
a= int(input("enter 1st number"))
b= int(input("enter 2nd number:"))
a,b = b,a
print(a,b)
```

→ enter 1st number :: 2
enter 2nd number : 3
3 2

3) Check number is positive (or) negative

```
n= int(input())
if n>0:
    print("positive")
else:
    print("Negative")
```

→ 4
positive

4) Check num is even (or) odd

```
n= int(input())
if (n%2 == 0)
    print("even")
```

Scanned with OKEN Scanner

```
else:
    print ("Odd")
```

→ 5
Odd

5) Find the Factorial of a number

```
import math
n = int (input ())
print (math.factorial (n))
```

→ 3
6

6) Check wherther the number is palindrome

```
n = input()
if (n == n[::-1]):
    print ("palindrome")
else:
    print ("Not palindrome")
```

→ 323
palindrome

7) Use pandas and create a dataset, print age > 25

```
import pandas as pd
data = {"Name" : ["Alice", "Bob", "charlie"]
        "Age" : [21, 32, 29],
        "city" : ["Blore", "Hydrabed", pune]}

df = pd.DataFrame (data)
print (df)
```

Scanned with OKEN Scanner

→	Name	Age	City
0	Alice	21	Blre
1	Bob	22	Hydrabed
2	Charlie	29	pune

df["Age"]

→ Age

0 21
1 32
2 29

df[df["Age"]>25

→	Name	Age	City
1	Bob	32	Hydrabad
2	Charlie	29	pune

o/p verified
14/10/2025

Scanned with OKEN Scanner

Experiement 2(a)

```
From trasformers impact pipeline
Sentiment - analyzer = pipeline ("Sentiment - analysis")
text = "I am Falling happy"
Sentiment - result = Sentiment - analyzer (text)
print ("Sentiment analysis")
print ("F" Sentiment : { sentiment - result (a) ['ab
        (score : (sentiment - result[0] ['score" : 2f
```

Output:

```
Sentiment analysis
sentiment : positive (score : 1.00)
```

```
From trasformers import pipeline
transation - result = traslation pipeline - en - to - fr (text)
print("Traslation :")
print (f" Translated text : { traslation - result [0]
["translation - text'] }\n")
```

output:

```
config . json          1.42 K/?
py. torch - model . bin   : 100%
tokenezed - config . json : 100%
generation - config . json : 100%


Source Spam : 100%
target . Spam : 100%
  vocab . json
```

Scanned with OKEN Scanner

Translation :

Translated text : le Me Sendy teurem.

Objective
18/10/2025

Scanned with OKEN Scanner

Experiement - 3

Varrational Auto encoder

```
import numpy as np
import tr Flow as tf
import keras
From keras import layers
```

Algorithm

1. Define Sampling layer
   -> Take mean & log variance From encoder output
   -> Sample = using epslim * exp (0.5 * log - Var) + m

2. Build encoder
   -> input (28, 28, 1) image
   -> apply conv2D layers to exact Features
   -> Flatten and use dense layers
   -> output : mean, log -Var & Sampled latent Vectors

3. Build Decoder
   -> input latent vectors (latent - dim)
   -> dense - reshape to (7, 7, 64)
   -> Convo2D transpare layers to upsample back to (28, 28, 1) image

4. Define VAE model
   -> combine encoder & decoder
   -> train the model apply gradient to update weights.

Scanned with OKEN Scanner

5. Train VAE

→ load and normalize Fashion - MNIST data

→ fit model for desired epaches

6. plot latent Space

→ create a 2D grid of latent points

→ For each (x_i, y_i) decode to generate an image

→ place image to correct spot in large figure grid

→ label axes as "1[0]" & "2[1]

→ Show the generated grid image.

Op verified
17/10/2025

Scanned with OKEN Scanner

Experiment 4

Generative Adverasial network

Algorithm

1. Load and prepare data:
   got MNIST images, scale them, add a channel & create a batched dataset

2. Build genarator
   Create a model that takes noise and generate images

3. Build discriminator : create a model that takes an image and outputs if its real or fake

4. Set up loss and optimizes
   use binary cross entropy for loss define generator and discriminator loss function, use Adam optimizer

5. Define training Step:
   Create a function to generate fake images, evaluate real/fake with discriminator Calculate losses compute gradient and update model weigulk.

6. Run training loop:
   iterate through epochs and dataset batches, calling the training step generate & save images after each epoch.

Scanned with OKEN Scanner

6. Run training loop :-

iterate through epochs and dataset batches, calling the training step generate and save images after each epoch.

7. start training :

Executes the training function

Code :

```
!pip install diffusers.
import pin ArcAlpha Pipeline
!pip install diffusers accelerate transformers safe
    tensors torch torchvision -- upgrade.

model_id = "PixArt-alpha/PixArt-XL-2-512x512"
dtype = torch.float16 if torch.cuda.is available()
else torch.float32

pipe = PixArtAlphapipeline.from_pretrained(model_id,
 torch_dtype = dtype)

device = "cuda" if torch.cuda is available()
else "cpu"

pipe = pipe.to(device)

if device == "cuda"
    Pipe.enable_vae_slicing()

prompt = ("Ultra-realistic Close-up portrait
of a young adult human, neutral expression,"
photographic quality")

image.save("face.png")
print("saved to face.png)
image.show()
```

Scanned with OKEN Scanner

Experiment - 05

Human Face Generation

```
pip install ftfy
```

→ # import libraries
```
import torch
from diffusers import PixArtAlphaPipeline
```

→ Loading Transformers based T2I model
```
model_id = "PixArt-alpha/PixArt-XL-2-512x512"
dtype = torch.float16 if torch.cuda.is_available()
else torch.float32
pipe = PixArtAlphaPipeline.from_pretrained(
        model_id,
        torch_dtype=dtype,
)
```

→ write a ~~progra~~ prompt for generating Human face
```
prompt = (
    "Ultra-realistic close-up portrait of a young
     adult human, neutral expression,"
)
negative_prompt = (
    "blurry, low quality, extra fingers,
     text, watermark, logo"
```

→ Image generation
```
generator = torch.Generate(device = device)
image = pipe(
    prompt = prompt,
    negative_prompt = negative_prompt,
    num_inference_steps = 24,
    guidance_scale = 4.5,
    height = 512,
    width = 512,
    generator = generator
) image.]
```

Scanned with OKEN Scanner

50544519
Class_id
Enrollment_id
12345

```
image.save("face.png")
print("Saved to face.png")
```

Output:

```
Setting 'clean_caption' to False...
100% 24/24
Saved to face.png.
```

O/P verified
14/10/2025

Experiment - 06
Conditional GAN

Step 1: import libraries
```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
```

Step 2: Hyper parameters
```
batch_size = 128
lr = 0.0002
z_dim = 100
epochs = 3
num_classes = 10
img_size = 28 * 28
```

Step 3: Generator.
```
class generator(nn.module):
    def_init_(self, z_dim, num_classes, img_size):
        super(Generator, self).__init__()
        self.label_emb = nn.Embedding(num_classes
        self.model = nn.Sequential(
            nn.Linear(z_dim+num_classes, 256),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Linear(256, 512)
            nn.LeakyReLU(0.2, inplace=True),
            nn.Tanh()
        )
```

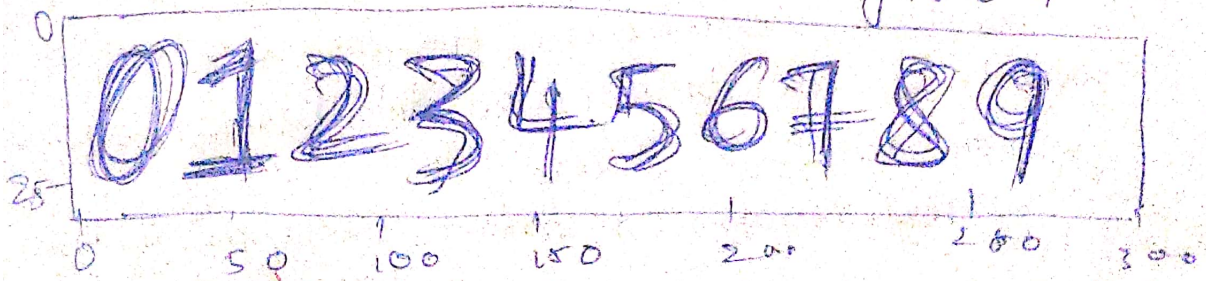
Step 4: Discriminator
```
class Discriminator(nn.module):
    def_init_(self, num_classes, img_size):
        super. label_emb=nn.Embedding(num_classes
            num_classes)
        self.model = nn.Sequential(
            nn.LeakyReLU(0.2, inplace=True),
            nn.Linear(0.2, inplace=True),
            nn.Linear(512, 256)
            nn.Sigmoid()
```

```
for epoch in range(epochs):
    for i,(image , labels) in enumerate(train_load)
        batch_size_curr = images.size(0)
        imgs = imgs.view(batch_size_curr, -1).to(device)
        labels = labels.to device
print(f" Epoch [{epoch+1}/{epochs}] D-Loss:{d_loss.
        item(): .4f} | G-loss:{g_loss.item(): .4f}")
if (epoch +1) % 3 ==0:
    z = torch.randn (10, z_dim).to (device)
grid = torch.cat([gen_imgs[i]) for in range(10)]
        dim=2).squeeze()
plt.imshow(grid, cmap = "grey")
plt.title ("Generated Digits 0-9")
plt.show()
```

output

```
Epoch [1/3] | D_Loss: 0.7418 | G_loss : 2.8354
Epoch [2/3] | D-loss: 0.7406 | G_loss : 2.0088
Epoch [3/3] | D_loss: 0.8861 | G_loss : 1.7418
```

Generated Digits 0-9



o/p saved
14/10/2025

29/04/2026

Experiment - 07

Aim:

LLM output parsing using long chain LLM

Code:

Step 1: Install libraries

```
! pip install langchain transformers huggingface_hub
                pydantic
from langchain.llms import HuggingFaceHub
from langchain.pydantic import PromptTemplate
```

Step 2: Define pydantic Schema.

```
class SummaryData(BaseModel):
    title: str = Field(description: "Title of the text")
    summary: str = Field(description = "Concise summary
                of the text")
```

Step 3: Define the prompt

```
prompt = PromptTemplate(
    template = "summarize the following text and
provide a title. \n\n {text} \n\n {format_instructions}
    input_variables = ["text"],
    partial_variables = {"format_instructions": parser.get_
                format_instructions()}
```

Step 4: Load the hugging face LLM

```
llm = HuggingFaceHub(
    repo_id = "facebook/bart-large-cnn")
)
```

Step 5: Run the Chain

```
temp_input = "Langchain is a powerful framework
for developing llm_output = llm(chain_input to str)
print(parsed_output)
```

output:

```
{"title": "LangChain Framework overview"
```

Scanned with OKEN Scanner

10/09/2025 Experiment - 08

Deep Convolutional GAN

Algorithm:

→ Load the Dataset (MNIST)

→ Import libraries : torch, torch_vision, torchinn, torch options, numpy, matplotlib

→ Download or load CelebA dataset

Apply image transformation

Resize image to 64x64 pixels

Convert to tensors & normalize pixel values to range [-1,1].

→ Define Generator network

Input: random noise vector z

Use transposed convolutional layers to upsample noise into images.

each layer followed by Batch normalization, ReLU activation.

output: 3 channel image with tanh

→ Define Discriminator network.

Input: an image (64x64x3)

Use convolutional layer to downsample.

Each layer followed by Batch Normalization, Leaky ReLU activation.

Output: single scalar with sigmoid.

→ Initialize model and optimization:

Initialize weights using normal distribution.

Define loss function Binary Cross entropy loss (BCE)

Define optimizer Adam for both G and D with learning rate = 0.002

→ Initial models and optimizes

Initialize weights using normalization

Scanned with OKEN Scanner

→ Training loop:

for each batch of real images
from discriminator, train generator
Save and visualize results

→ Generates final synthetic faces

Result: Implement deep convolutional Generative adversarial network and synthetic faces.

O/P verified
14/10/2025

15/09/2025

Experiment - 09

Customizing LLM on local machine

→ Install anaconda navigator
→ open anaconda prompt
→ check for the environments
Conda env list
base env was present
→ Create new environment
Create -- name : gena1
python = 3.9
→ Activate the new environment
Conda activate gena1
→ The new environment is created in the name of gena1
→ Import requirements.txt
* Created new folder named gena1 and paste requirements.
txt file in it.
* Copy the path of the folder and then run.

Scanned with OKEN Scanner

Experiment - 10

Gradio App (Customizing UIs)

→ Text input and output: Function takes a name and returns a greeting.

Input : Textbook

Output : Text label

→ Slider Input : Function multiplies slides value by 2

Input : Slider (1-100)

Output : Number

→ checkbox input: Function returns checked or not checked.

Input : Checkbox

output : Text

→ Radio buttons: Functions returns which option the user selected

Input : radio

output : Text

→ Dropdown : Function returns which item is chosen

Input : Dropdown (Item : 1,2,3)

output : Text

→ Image upload : Displays uploaded image back to back users

Input : Image

output : Image

→ video uploaded : Displays uploaded video back to user

Input : video

output : video

Scanned with OKEN Scanner

→ Audio upload: plays back recorded / uploaded audio.

Input: Audio
output: Audio.

→ Ison Input: Function parses a JSON string and displays structured data.
Input :- JSON Text
Output :- Dictionary

Result:

The experiment shows how gradio can be used to quickly build user friendly interfaces without need web development skills.

O/P verified
14/10/2025

Scanned with OKEN Scanner

06/10/25

Experiment - 11A

Deploying chat based Application using FastAPI

1. What is asyncio?
   Asyncio is a python library that provides tools to write asynchronous, concurrent programs using the asyncio and await.

2. Define FastAPI?
   It is a modern, high-performance web framework for building API's with python, based on asynchronous programming using asyncio.

3. Define ngrok?
   It is a tool that create a secure tunnel from your machine to the internet.

Code:

Step 1: Install libraries and import

```
→ !pip install fastAPI uvicorn nest-asyncio
                pyngrok
  import nest-asyncio
  from pyngrok import ngrok
  from fastAPI import FastAPI
  import uvicorn
```

Step 2: Create FastAPI App

```
→ app = FastAPI [title = "FASTAPI Chat App(colab-safe)
  chat-history = [ ]
  Class Message [Base Model]:
      Sender : str
      text : str
  @app.get ("/history", response-model = list(Message))
  def get_history():
      return chat-history
  ngrok URL
```

Step 3 Start ngrok tunnel

```
port = 8000
public_url = ngrok.connect(port).public_url
print(f"public url: {public_url}/docs:
```

Step 4: Configure uvicorn without asyncio.run()

```
config: uvicorn.Config(app=app, host="0.0.0.0",
port = port, log_level ("info")
server = uvicorn.Server(config)
```

Step 5. Run inside the existing event loop.

```
loop = asyncio.get_event_loop()
loop.create_task(server.serve())
```

O/P verified

14/10/2025

Scanned with OKEN Scanner

13/10/2025

Experiment - 11B

Retrieval - Augumented Generation

Aim : To develop a Retrieval - Augumented generation system the efficiency integrates and retrieves relevant information.

Python program:

```
→ !pip install faiss-cpu
  import warnings
→ warnings.filterwarnings('ignore')

→ from sentence-transformers import SentenceTransformer.
  import faiss
  import numpy as np
```

Step 1: Define documents as a list

```
docs = [
    "The Taj Mahal is located in Agra",
    "The effiel Tower is in paris"
    "The Greate wall of china can be found in China.
]
```

Step 2:

```
Model = SentenceTransformer('all-MiniLM-L6-v2')
doc_embeddings = model.encode(docs)
```

Step 3:

```
dimension = doc_embeddings.shape[1]
index = faiss.IndexFlatL2(dimension)
index.add(np.array(doc_embeddings).astype('float32'))
```

Scanned with OKEN Scanner

Step 4:

```
def retrieve (query, top-k=3)
    query embedding = model.encode([query])
    return [docs[i] for in indices[0]]
```

Step5:

```
Question= "where is the Taj Mahal located?"
Result= retrieve (query)
print ("Question:" query)
print ("Retrieved documents")
for results in results :
        print (result)
```

O/P verified
14/10/2025