```python
import numpy as np
from scipy.optimize import minimize
from sklearn.model_selection import train_test_split

T = 2.0
N = 50
dt = T / (N - 1)
t = np.linspace(0, T, N)

joint_limits = [(-np.pi, np.pi), (-np.pi/2, np.pi/2)]
bounds = joint_limits * N

# intial quintic guess (didnt know this)

def quintic(q0, qf, T, t):
    a0 = q0
    a1 = 0
    a2 = 0
    a3 = 10*(qf-q0)/T**3
    a4 = -15*(qf-q0)/T**4
    a5 = 6*(qf-q0)/T**5
    return a0 + a1*t + a2*t**2 + a3*t**3 + a4*t**4 + a5*t**5

def opt_cost(q, dt, wv=0.0, wa=1.0):
    q = q.reshape(N, 2)
    q1, q2 = q[:,0], q[:,1]
    v1, v2 = np.diff(q1)/dt, np.diff(q2)/dt
    a1, a2 = np.diff(q1,2)/dt**2, np.diff(q2,2)/dt**2
    return wv*np.sum(v1**2 + v2**2) + wa*np.sum(a1**2 + a2**2)

# Datagen
NUM_SAMPLES = 250

X_data = []
Y_data = []

for i in range(NUM_SAMPLES):

    # Random start & end joint angles
    q1_start = np.random.uniform(-np.pi, np.pi)
    q2_start = np.random.uniform(-np.pi/2, np.pi/2)
    q1_end   = np.random.uniform(-np.pi, np.pi)
    q2_end   = np.random.uniform(-np.pi/2, np.pi/2)

    # ye gpt se nikala h fix
    if np.linalg.norm([q1_start-q1_end, q2_start-q2_end]) < 0.2:
        continue

    # Initial guess
    q1_poly = quintic(q1_start, q1_end, T, t)
    q2_poly = quintic(q2_start, q2_end, T, t)
    q0 = np.stack((q1_poly, q2_poly), axis=1).reshape(-1)

    #constraint
    def boundary(q):
        q = q.reshape(N, 2)
        return [
            q[0,0]  - q1_start,
            q[0,1]  - q2_start,
            q[-1,0] - q1_end,
            q[-1,1] - q2_end
        ]

    constraints = {'type': 'eq', 'fun': boundary}

    # Optimization
    res = minimize(
    opt_cost,
    q0,
    args=(dt,),
```

C:\Users\dwive\Downloads\datagen.py • - Sublime Text (UNREGISTERED)

datagen.py | Train_trajM.py | Dashboard.py

```python
        q0,
        args=(dt,),
        bounds=bounds,
        constraints=constraints,
        method='SLSQP',
        options={'ftol': 1e-6, 'maxiter': 500}
    )


    if not res.success:
        print("Optimization failed:", res.message)
        continue

    # Optimized trajectory
    q_opt = res.x.reshape(N, 2)

    # Store dataset sample
    X_data.append([q1_start, q2_start, q1_end, q2_end])
    Y_data.append(q_opt.flatten())  # length = 2N


X = np.array(X_data)
Y = np.array(Y_data)

print("Full dataset:")
print("X shape:", X.shape)
print("Y shape:", Y.shape)

# streams split krrhe hn

X_train, X_test, Y_train, Y_test = train_test_split(
    X, Y,
    test_size=0.2,
    shuffle=True,
    random_state=42
)

# Datalogging
np.save("X_train.npy", X_train)
np.save("Y_train.npy", Y_train)
np.save("X_test.npy",  X_test)
np.save("Y_test.npy",  Y_test)

print("Dataset saved successfully.")
```

```
any use case where you don't have full control of the loaded file. Please op
  model.load_state_dict(torch.load("model.pth", map_location="cpu"))
2026-01-13 16:09:14.032 WARNING streamlit.runtime.scriptrunner_utils.script_
2026-01-13 16:09:14.044
  <0x1b>[33m<0x1b>[1mWarning:<0x1b>[0m to view this Streamlit app on a brows
  command:

    streamlit run C:\Users\dwive\Downloads\New folder\Dashboard.py [ARGUMENT
2026-01-13 16:09:14.044 Thread 'MainThread': missing ScriptRunContext! This
2026-01-13 16:09:14.044 Thread 'MainThread': missing ScriptRunContext! This
2026-01-13 16:09:14.044 Thread 'MainThread': missing ScriptRunContext! This
2026-01-13 16:09:14.044 Thread 'MainThread': missing ScriptRunContext! This
2026-01-13 16:09:14.044 Thread 'MainThread': missing ScriptRunContext! This
2026-01-13 16:09:14.044 Thread 'MainThread': missing ScriptRunContext! This
2026-01-13 16:09:14.044 Thread 'MainThread': missing ScriptRunContext! This
2026-01-13 16:09:14.044 Thread 'MainThread': missing ScriptRunContext! This
2026-01-13 16:09:14.044 Thread 'MainThread': missing ScriptRunContext! This
2026-01-13 16:09:14.044 Thread 'MainThread': missing ScriptRunContext! This
2026-01-13 16:09:14.044 Thread 'MainThread': missing ScriptRunContext! This
```

C:\Users\dwive\Downloads\Train_trajM.py - Sublime Text (UNREGISTERED)

Maxwater PS | import.py | Rev_INT.cpp | MCS_shrt.t

```python
import numpy as np
import torch
import torch.nn as nn
import matplotlib.pyplot as plt

# Load dataset
X_train = np.load("X_train.npy")
Y_train = np.load("Y_train.npy")
X_test  = np.load("X_test.npy")
Y_test  = np.load("Y_test.npy")

print("Train shapes:", X_train.shape, Y_train.shape)
print("Test shapes :", X_test.shape, Y_test.shape)

X_mean, X_std = X_train.mean(axis=0), X_train.std(axis=0)
Y_mean, Y_std = Y_train.mean(axis=0), Y_train.std(axis=0)

X_train_n = (X_train - X_mean) / X_std
X_test_n  = (X_test  - X_mean) / X_std

Y_train_n = (Y_train - Y_mean) / Y_std
Y_test_n  = (Y_test  - Y_mean) / Y_std

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Using device:", device)

Xtr = torch.tensor(X_train_n, dtype=torch.float32).to(device)
Ytr = torch.tensor(Y_train_n, dtype=torch.float32).to(device)
Xte = torch.tensor(X_test_n,  dtype=torch.float32).to(device)
Yte = torch.tensor(Y_test_n,  dtype=torch.float32).to(device)

class TrajectoryMLP(nn.Module):
    def __init__(self):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(4, 64),
            nn.ReLU(),
            nn.Linear(64, 128),
            nn.ReLU(),
            nn.Linear(128, 100)
        )

    def forward(self, x):
        return self.net(x)

model = TrajectoryMLP().to(device)

criterion = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=1e-3)

EPOCHS = 500

for epoch in range(EPOCHS):
    optimizer.zero_grad()

    Y_pred = model(Xtr)
    loss = criterion(Y_pred, Ytr)

    loss.backward()
    optimizer.step()

    if epoch % 50 == 0:
        print(f"Epoch {epoch:4d} | Train MSE: {loss.item():.6f}")

with torch.no_grad():
    Y_test_pred = model(Xte)
    test_loss = criterion(Y_test_pred, Yte)

print("Test MSE:", test_loss.item())

```

```python
print("Test MSE:", test_loss.item())

# Pick random test example
i = np.random.randint(len(X_test))

# De-normalize
pred = Y_test_pred[i].cpu().numpy() * Y_std + Y_mean
true = Y_test[i]

N = 50
T = 2.0
t = np.linspace(0, T, N)

pred_q = pred.reshape(N, 2)
true_q = true.reshape(N, 2)

plt.figure()
plt.plot(t, true_q[:,0], label="q1 optimized")
plt.plot(t, pred_q[:,0], "--", label="q1 predicted")
plt.plot(t, true_q[:,1], label="q2 optimized")
plt.plot(t, pred_q[:,1], "--", label="q2 predicted")
plt.xlabel("Time (s)")
plt.ylabel("Joint Angle (rad)")
plt.legend()
plt.grid()
plt.show()

torch.save(model.state_dict(), "trajectory_mlp.pth")

```

```
Train shapes: (197, 4) (197, 100)
Test shapes : (50, 4) (50, 100)
Using device: cuda
Epoch    0 | Train MSE: 1.010643
Epoch   50 | Train MSE: 0.043073
Epoch  100 | Train MSE: 0.004391
Epoch  150 | Train MSE: 0.002317
Epoch  200 | Train MSE: 0.001559
Epoch  250 | Train MSE: 0.001167
Epoch  300 | Train MSE: 0.000927
Epoch  350 | Train MSE: 0.000764
Epoch  400 | Train MSE: 0.000645
Epoch  450 | Train MSE: 0.000556
Test MSE: 0.0011485472787171602
[Finished in 156.1s]
```

C:\Users\dwive\Downloads\New folder\Dashboard.py - Sublime Text (UNREGISTERED)

datagen.py | Train_trajM.py | Dashboard.py | MO

```python
import streamlit as st
import numpy as np
import torch
import matplotlib.pyplot as plt
from scipy.optimize import minimize

T = 2.0
N = 50
dt = T / (N - 1)
t = np.linspace(0, T, N)

# ------------------------------
L1, L2 = 1.0, 1.0

def fk(q1, q2):
    x = L1*np.cos(q1) + L2*np.cos(q1 + q2)
    y = L1*np.sin(q1) + L2*np.sin(q1 + q2)
    return x, y

def quintic(q0, qf, T, t):
    a0 = q0
    a1 = 0
    a2 = 0
    a3 = 10*(qf-q0)/T**3
    a4 = -15*(qf-q0)/T**4
    a5 = 6*(qf-q0)/T**5
    return a0 + a1*t + a2*t**2 + a3*t**3 + a4*t**4 + a5*t**5

# Optimization cost
def opt_cost(q, dt):
    q = q.reshape(N, 2)
    v = np.diff(q, axis=0)/dt
    a = np.diff(v, axis=0)/dt
    return np.sum(a**2)

def boundary(q, qs, qe):
    q = q.reshape(N, 2)
    return [
        q[0,0]-qs[0], q[0,1]-qs[1],
        q[-1,0]-qe[0], q[-1,1]-qe[1]
    ]

# trained NN + normalization

class MLP(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.net = torch.nn.Sequential(
            torch.nn.Linear(4, 64),
            torch.nn.ReLU(),
            torch.nn.Linear(64, 128),
            torch.nn.ReLU(),
            torch.nn.Linear(128, 100)
        )

    def forward(self, x):
        return self.net(x)

model = MLP()
model.load_state_dict(torch.load("model.pth", map_location="cpu"))
model.eval()

stats = np.load("norm_stats.npz")
X_mean, X_std = stats["X_mean"], stats["X_std"]
Y_mean, Y_std = stats["Y_mean"], stats["Y_std"]

# ------------------------------
```

C:\Users\dwive\Downloads\New folder\Dashboard.py • - Sublime Text (UNREGISTERED)

datagen.py | Train_trajM.py | Dashboard.py | MCS_shrt.

```python
#UI

st.title("Learning-Based Trajectory Prediction")

st.sidebar.header("Joint Angles (rad)")

q1_start = st.sidebar.slider("q1 start", -1.5, 1.5, 0.0)
q2_start = st.sidebar.slider("q2 start", -1.0, 1.0, 0.0)
q1_end   = st.sidebar.slider("q1 end",   -1.5, 1.5, 1.0)
q2_end   = st.sidebar.slider("q2 end",   -1.0, 1.0, 0.5)

qs = np.array([q1_start, q2_start])
qe = np.array([q1_end, q2_end])


# Optimized trajectory

q1_init = quintic(q1_start, q1_end, T, t)
q2_init = quintic(q2_start, q2_end, T, t)
q0 = np.stack((q1_init, q2_init), axis=1).reshape(-1)

res = minimize(
    opt_cost,
    q0,
    args=(dt,),
    constraints={'type':'eq','fun': lambda q: boundary(q, qs, qe)},
    method='SLSQP'
)

q_opt = res.x.reshape(N, 2)

# NN prediction

X = np.array([q1_start, q2_start, q1_end, q2_end])
Xn = (X - X_mean)/X_std

with torch.no_grad():
    Yn = model(torch.tensor(Xn, dtype=torch.float32)).numpy()

Y = Yn*Y_std + Y_mean
q_pred = Y.reshape(N, 2)


fig1, ax1 = plt.subplots()
ax1.plot(t, q_opt[:,0], label="q1 optimized")
ax1.plot(t, q_pred[:,0], "--", label="q1 predicted")
ax1.plot(t, q_opt[:,1], label="q2 optimized")
ax1.plot(t, q_pred[:,1], "--", label="q2 predicted")
ax1.set_xlabel("Time (s)")
ax1.set_ylabel("Joint Angle (rad)")
ax1.legend()
ax1.grid(True)

st.pyplot(fig1)


# Plot end-effector paths

x_opt, y_opt = fk(q_opt[:,0], q_opt[:,1])
x_pred, y_pred = fk(q_pred[:,0], q_pred[:,1])

fig2, ax2 = plt.subplots()
ax2.plot(x_opt, y_opt, label="Optimized path")
ax2.plot(x_pred, y_pred, "--", label="Predicted path")
ax2.set_aspect("equal")
ax2.set_xlabel("x")
ax2.set_ylabel("y")
ax2.legend()
ax2.grid(True)
```

Joint Angles (rad)

q1 start

0.00

q2 start

0.00

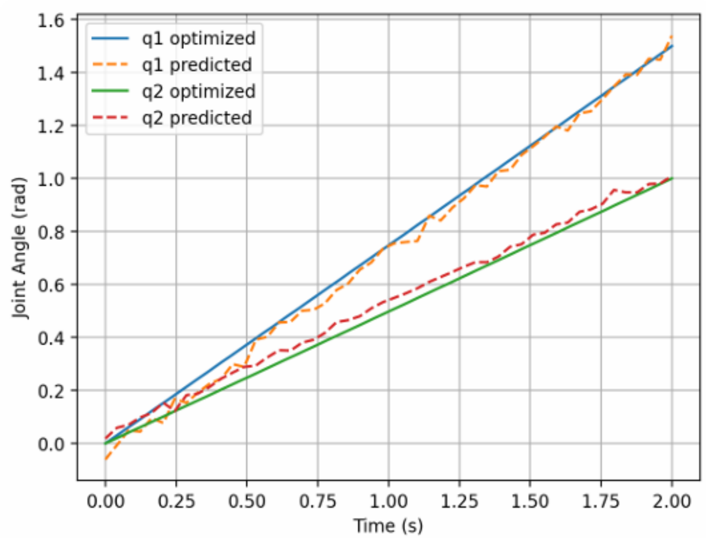
q1 end

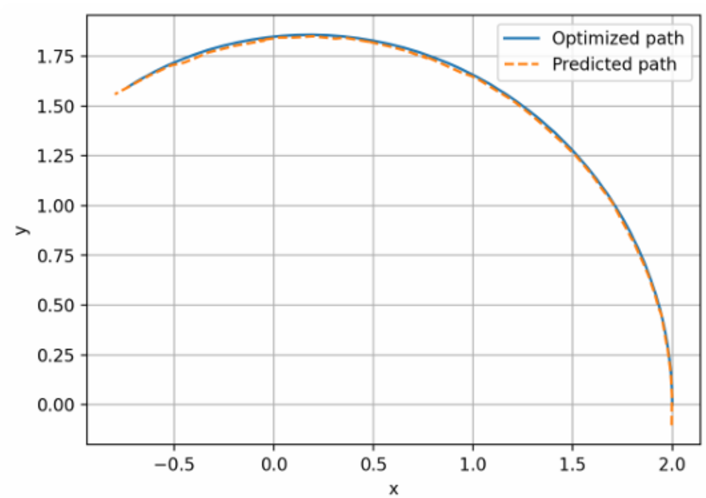
1.50

q2 end

1.00

Learning-Based Trajectory Prediction





The optimized trajectories generated using numerical optimization are smooth by construction and satisfy the boundary conditions, as the cost function penalizes high joint accelerations. In the learned trajectories produced by the neural network approximate the optimized solutions and closely follow their overall shape, but may exhibit small deviations or minor oscillations because smoothness is not explicitly enforced during training. Despite these differences, the learned trajectories remain accurate within the training distribution and produce end-effector paths that are visually similar to those obtained from optimization.

The learning-based model performs well when the start and end joint configurations lie within the range of the training data and when the motion does not involve extreme joint excursions. In such cases, the prediction error is small and the neural network provides smooth and reliable trajectories. However, the learned model performs poorly when inputs fall outside the training distribution since i generaated the dataset in limited angle range to make it more practical or when the motion requires behavior not represented in the dataset, leading to increased deviation from the optimized trajectory.

A key trade-off between the two approaches lies in computation time versus accuracy. Numerical optimization produces highly accurate trajectories but is computationally expensive and unsuitable for real-time use due to computation time. In contrast, the learning-based approach slightly sacrifices accuracy but enables trajectory prediction in milliseconds, making it suitable for real-time and repeated trajectory generation. Therefore, learning-based methods are preferable in time-critical applications with known task distributions, while direct optimization remains more appropriate when exact optimality and strict constraint satisfaction are required.