Group ID: 88

Group Members Name with Student ID:

1. Student 1: PEYALA SAMARASIMHA REDDY ID: 2023AA05072
2. Student 2: PEGALLAPATI SAI MAHARSHI ID: 2023AA05924
3. Student 3: CHADALAWADA VISWANATH HEMANTH ID: 2023AA05195
4. Student 4: SIGINAM SIVASAI ID: 2023AA05371

Problem Statement

The objective of the problem is to implement an Actor-Critic reinforcement learning algorithm to optimize energy consumption in a building. The agent should learn to adjust the temperature settings dynamically to minimize energy usage while maintaining comfortable indoor conditions.

Dataset Details

Dataset: https://archive.ics.uci.edu/dataset/374/appliances+energy+prediction

This dataset contains energy consumption data for a residential building, along with various environmental and operational factors.

Data Dictionary:

- Appliances: Energy use in Wh
- lights: Energy use of light fixtures in the house in Wh
- T1 - T9: Temperatures in various rooms and outside
- RH_1 to RH_9: Humidity measurements in various rooms and outside
- Visibility: Visibility in km
- Tdewpoint: Dew point temperature
- Pressure_mm_hgg: Pressure in mm Hg
- Windspeed: Wind speed in m/s

Environment Details

State Space: The state space consists of various features from the dataset that impact energy consumption and comfort levels.

- Current Temperature (T1 to T9): Temperatures in various rooms and outside.
- Current Humidity (RH_1 to RH_9): Humidity measurements in different locations.
- Visibility (Visibility): Visibility in meters.
- Dew Point (Tdewpoint): Dew point temperature.
- Pressure (Press_mm_hg): Atmospheric pressure in mm Hg.
- Windspeed (Windspeed): Wind speed in m/s.

Total State Vector Dimension: Number of features = 9 (temperature) + 9 (humidity) + 1 (visibility) + 1 (dew point) + 1 (pressure) + 1 (windspeed) = 22 features

Target Variable: Appliances (energy consumption in Wh).

Action Space: The action space consists of discrete temperature adjustments:

- Action 0: Decrease temperature by 1°C
- Action 1: Maintain current temperature
- Action 2: Increase temperature by 1°C

- If the action is to decrease the temperature by 1°C, you'll adjust each temperature feature (T1 to T9) down by 1°C.
- If the action is to increase the temperature by 1°C, you'll adjust each temperature feature (T1 to T9) up by 1°C.
- Other features remain unchanged.

Policy (Actor): A neural network that outputs a probability distribution over possible temperature adjustments.

Value function (Critic): A neural network that estimates the expected cumulative reward (energy savings) from a given state.

Reward function: The reward function should reflect the overall comfort and energy efficiency based on all temperature readings. i.e., balance between minimising temperature deviations and minimizing energy consumption.

- Calculate the penalty based on the deviation of each temperature from the target temperature and then aggregate these penalties.
- Measure the change in energy consumption before and after applying the RL action.
- Combine the comfort penalty and energy savings to get the final reward.

Example:

Target temperature=22°C

Initial Temperatures: T1=23, T2=22, T3=21, T4=23, T5=22, T6=21, T7=24, T8=22, T9=23

Action Taken: Decrease temperature by 1°C for each room

Resulting Temperatures: T1 = 22, T2 = 21, T3 = 20, T4 = 22, T5 = 21, T6 = 20, T7 = 23, T8 = 21, T9 = 22

Energy Consumption: 50 Wh (before RL adjustment) and 48 Wh (after RL adjustment)

- Energy Before (50 Wh): Use the energy consumption from the dataset at the current time step.
- Energy After (48 Wh): Use the energy consumption from the dataset at the next time step (if available).

Consider only temperature features for deviation calculation.

Deviation = abs (Ti− Ttarget)

Deviations=[abs(22−22), abs(21−22), abs(20−22), abs(22−22), abs(21−22), abs(20−22), abs(23−22), abs(21−22), abs(22−22)]

Deviations = [0, 1, 2, 0, 1, 2, 1, 1, 0], Sum of deviations = 8

Energy Savings = Energy Before−Energy After = 50 – 48 = 2Wh

Reward= −Sum of Deviations + Energy Savings = -8+6 = -2

Expected Outcomes

1. Pre-process the dataset to handle any missing values and create training and testing sets.
2. Implement the Actor-Critic algorithm using TensorFlow.
3. Train the model over 500 episodes to minimize energy consumption while maintaining an indoor temperature of 22°C.
4. Plot the total reward obtained in each episode to evaluate the learning progress.
5. Evaluate the performance of the model on test set to measure its performance
6. Provide graphs showing the convergence of the Actor and Critic losses.
7. Plot the learned policy by showing the action probabilities across different state values (e.g., temperature settings).
8. Provide an analysis on a comparison of the energy consumption before and after applying the reinforcement learning algorithm.

Code Execution

In [1]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Load the dataset
file_path = 'energydata_complete.csv'  # Replace with your dataset path
data = pd.read_csv(file_path)

# Check the columns to identify non-numeric columns
print("Columns in the dataset:", data.columns)

# Drop non-numeric columns (for example, a date-time column if present)
# You can modify this line based on your dataset's column names
data = data.select_dtypes(include=[float, int])  # Only keep numeric columns for now

# Check for missing values and replace them (if any) with column mean
data.fillna(data.mean(), inplace=True)

# Define the features and target variables
features = ['T1', 'T2', 'T3', 'T4', 'T5', 'T6', 'T7', 'T8', 'T9',  # Temperatures
            'RH_1', 'RH_2', 'RH_3', 'RH_4', 'RH_5', 'RH_6', 'RH_7', 'RH_8', 'RH_9',
            'Visibility', 'Tdewpoint', 'Press_mm_hg', 'Windspeed']  # Other features
target = ['Appliances']  # Energy consumption

# Make sure that only selected features exist in the dataset
X = data[features]
y = data[target]

# Normalize features using StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split the dataset into training and testing sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, rando

print("Training set size:", X_train.shape)
print("Test set size:", X_test.shape)
```

```
Columns in the dataset: Index(['date', 'Appliances', 'lights', 'T1', 'RH_1', 'T2', 'RH
_2', 'T3',
       'RH_3', 'T4', 'RH_4', 'T5', 'RH_5', 'T6', 'RH_6', 'T7', 'RH_7', 'T8',
       'RH_8', 'T9', 'RH_9', 'T_out', 'Press_mm_hg', 'RH_out', 'Windspeed',
       'Visibility', 'Tdewpoint', 'rv1', 'rv2'],
      dtype='object')
Training set size: (15788, 22)
Test set size: (3947, 22)
```

Create an EnergyConsumption Environment (1 M)

In [2]:
```
### write your code below this line
import numpy as np
import random

class EnergyConsumptionEnv:
    def __init__(self):
        self.state = np.random.randint(-10, 30, size=(21,))
        self.temperature_min = -10
        self.temperature_max = 30
        self.target_temperature = 22

    def reset(self):
        self.state = np.random.randint(self.temperature_min, self.temperature_max, si
        return self.state

    def step(self, action):
        if action == 0:  # Decrease temperature
            self.state[:9] = np.clip(self.state[:9] - 1, self.temperature_min, self.t
        elif action == 1:  # Maintain temperature
            pass
        elif action == 2:  # Increase temperature
            self.state[:9] = np.clip(self.state[:9] + 1, self.temperature_min, self.t

        deviation = np.abs(self.state[:9] - self.target_temperature).sum()
        energy_before = np.random.uniform(30, 60)
        energy_after = np.random.uniform(20, 55)
        energy_savings = energy_before - energy_after
        reward = -deviation + energy_savings

        done = False  # No termination condition

        return self.state, reward, done
```

Print state space and action space (0.5 M)

In [3]:
```
### write your code below this line
env = EnergyConsumptionEnv()
state = env.reset()
print("Initial State:", state)
print("State Space Dimension:", len(state))

action_space = [0, 1, 2]
print("Action Space:", action_space)
```

```
Initial State: [11 -1 -9 15  3 29  7 18  7 22 26  5 -8 26 10 -9 28 17 -6  7  5]
State Space Dimension: 21
Action Space: [0, 1, 2]
```

Clearly define the parameters used for training an AI agent. (1 M)

- Number of episodes
- Max capacity of replay memory
- Batch size
- Period of Q target network updates
- Discount factor for future rewards
- Initial value for epsilon of the e-greedy
- Final value for epsilon of the e-greedy
- Learning rate of ADAM optimizer, and etc.

In [4]:
```
### write your code below this line
### Define the Parameters for Training the AI Agent
params = {
    "episodes": 500,                # Number of episodes to run
    "memory_capacity": 100000,      # Max capacity of the replay memory buffer
    "batch_size": 32,               # Batch size for experience replay
    "target_update_period": 1000,   # Frequency to update the target Q-network
    "discount_factor": 0.99,        # Discount factor for future rewards (gamma)
    "epsilon_start": 1.0,           # Initial value for epsilon in the e-greedy polic
    "epsilon_end": 0.01,            # Final value for epsilon (minimum exploration)
    "epsilon_decay": 0.995,         # Rate of decay for epsilon over episodes
    "learning_rate": 0.001          # Learning rate for the Adam optimizer
}

# Print the training parameters
for key, value in params.items():
    print(f"{key}: {value}")
```

```
episodes: 500
memory_capacity: 100000
batch_size: 32
target_update_period: 1000
discount_factor: 0.99
epsilon_start: 1.0
epsilon_end: 0.01
epsilon_decay: 0.995
learning_rate: 0.001
```

Define the separate functions for DecreaseTemperature, IncreaseTemperature and MaintainCurrentTemperature actions. (1.5 M)

In [5]:
```
### write your code below this line
def decrease_temperature(state):
    state[:9] = np.clip(state[:9] - 1, -10, 30)
    return state

def maintain_temperature(state):
    return state

def increase_temperature(state):
    state[:9] = np.clip(state[:9] + 1, -10, 30)
    return state

print("Example state before action:", state)
state = increase_temperature(state)
print("State after increasing temperature:", state)
```

```
Example state before action: [11 -1 -9 15  3 29  7 18  7 22 26  5 -8 26 10 -9 28 17 -6
7  5]
State after increasing temperature: [12  0 -8 16  4 30  8 19  8 22 26  5 -8 26 10 -9 2
8 17 -6  7  5]
```

Implement a replay buffer for storing the experiences. (0.5 M)

In [6]:
```
### write your code below this line
from collections import deque

class ReplayBuffer:
    def __init__(self, max_size=100000):
        self.buffer = deque(maxlen=max_size)

    def store(self, experience):
        self.buffer.append(experience)

    def sample_batch(self, batch_size):
```

```
        return random.sample(self.buffer, batch_size)

    def size(self):
        return len(self.buffer)

replay_buffer = ReplayBuffer()
```

Design network DQN (0.5 M)

In [7]:
```
import tensorflow as tf
from tensorflow.keras import layers

def build_model(input_shape, output_shape):
    model = tf.keras.Sequential()
    model.add(layers.Input(shape=(input_shape,)))  # Define input shape with Input la
    model.add(layers.Dense(128, activation='relu'))
    model.add(layers.Dense(128, activation='relu'))
    model.add(layers.Dense(output_shape, activation='linear'))
    model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001), loss='mse'
    return model

state_size = 21  # Number of input features
action_size = 3  # Number of output actions
dqn_model = build_model(state_size, action_size)
print(dqn_model.summary())
```

```
2024-09-19 18:08:12.153138: I tensorflow/core/util/port.cc:153] oneDNN custom operatio
ns are on. You may see slightly different numerical results due to floating-point roun
d-off errors from different computation orders. To turn them off, set the environment
variable `TF_ENABLE_ONEDNN_OPTS=0`.
2024-09-19 18:08:12.163808: E external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:
485] Unable to register cuFFT factory: Attempting to register factory for plugin cuFFT
when one has already been registered
2024-09-19 18:08:12.175293: E external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:
8454] Unable to register cuDNN factory: Attempting to register factory for plugin cuDN
N when one has already been registered
2024-09-19 18:08:12.178624: E external/local_xla/xla/stream_executor/cuda/cuda_blas.c
c:1452] Unable to register cuBLAS factory: Attempting to register factory for plugin c
uBLAS when one has already been registered
2024-09-19 18:08:12.187365: I tensorflow/core/platform/cpu_feature_guard.cc:210] This
TensorFlow binary is optimized to use available CPU instructions in performance-critic
al operations.
To enable the following instructions: AVX2 AVX_VNNI FMA, in other operations, rebuild
TensorFlow with the appropriate compiler flags.
2024-09-19 18:08:12.755128: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF
-TRT Warning: Could not find TensorRT
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
I0000 00:00:1726749493.029426 3643744 cuda_executor.cc:1015] successful NUMA node read
from SysFS had negative value (-1), but there must be at least one NUMA node, so retur
ning NUMA node zero. See more at https://github.com/torvalds/linux/blob/v6.0/Documenta
tion/ABI/testing/sysfs-bus-pci#L344-L355
I0000 00:00:1726749493.050896 3643744 cuda_executor.cc:1015] successful NUMA node read
from SysFS had negative value (-1), but there must be at least one NUMA node, so retur
ning NUMA node zero. See more at https://github.com/torvalds/linux/blob/v6.0/Documenta
tion/ABI/testing/sysfs-bus-pci#L344-L355
I0000 00:00:1726749493.052050 3643744 cuda_executor.cc:1015] successful NUMA node read
from SysFS had negative value (-1), but there must be at least one NUMA node, so retur
ning NUMA node zero. See more at https://github.com/torvalds/linux/blob/v6.0/Documenta
tion/ABI/testing/sysfs-bus-pci#L344-L355
I0000 00:00:1726749493.054908 3643744 cuda_executor.cc:1015] successful NUMA node read
from SysFS had negative value (-1), but there must be at least one NUMA node, so retur
ning NUMA node zero. See more at https://github.com/torvalds/linux/blob/v6.0/Documenta
tion/ABI/testing/sysfs-bus-pci#L344-L355
I0000 00:00:1726749493.056001 3643744 cuda_executor.cc:1015] successful NUMA node read
from SysFS had negative value (-1), but there must be at least one NUMA node, so retur
ning NUMA node zero. See more at https://github.com/torvalds/linux/blob/v6.0/Documenta
tion/ABI/testing/sysfs-bus-pci#L344-L355
I0000 00:00:1726749493.057146 3643744 cuda_executor.cc:1015] successful NUMA node read
from SysFS had negative value (-1), but there must be at least one NUMA node, so retur
ning NUMA node zero. See more at https://github.com/torvalds/linux/blob/v6.0/Documenta
tion/ABI/testing/sysfs-bus-pci#L344-L355
I0000 00:00:1726749493.157933 3643744 cuda_executor.cc:1015] successful NUMA node read
from SysFS had negative value (-1), but there must be at least one NUMA node, so retur
ning NUMA node zero. See more at https://github.com/torvalds/linux/blob/v6.0/Documenta
tion/ABI/testing/sysfs-bus-pci#L344-L355
I0000 00:00:1726749493.159254 3643744 cuda_executor.cc:1015] successful NUMA node read
from SysFS had negative value (-1), but there must be at least one NUMA node, so retur
ning NUMA node zero. See more at https://github.com/torvalds/linux/blob/v6.0/Documenta
tion/ABI/testing/sysfs-bus-pci#L344-L355
I0000 00:00:1726749493.160401 3643744 cuda_executor.cc:1015] successful NUMA node read
from SysFS had negative value (-1), but there must be at least one NUMA node, so retur
ning NUMA node zero. See more at https://github.com/torvalds/linux/blob/v6.0/Documenta
tion/ABI/testing/sysfs-bus-pci#L344-L355
2024-09-19 18:08:13.161452: I tensorflow/core/common_runtime/gpu/gpu_device.cc:2021] C
reated device /job:localhost/replica:0/task:0/device:GPU:0 with 1165 MB memory:  -> de
vice: 0, name: NVIDIA GeForce RTX 4070 Laptop GPU, pci bus id: 0000:01:00.0, compute c
apability: 8.9
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 128)	2,816
dense_1 (Dense)	(None, 128)	16,512
dense_2 (Dense)	(None, 3)	387

Total params: 19,715 (77.01 KB)
Trainable params: 19,715 (77.01 KB)
Non-trainable params: 0 (0.00 B)
None

Training Iterations (0.5 M)

Note: print all the episodes with the values of investment and buying. (if not printed then -1 will be done.)

In [8]:
```
import tensorflow as tf
tf.keras.utils.disable_interactive_logging()
import numpy as np
import random
import tensorflow as tf
from tensorflow.keras.callbacks import Callback

# Suppress TensorFlow logs
tf.get_logger().setLevel('ERROR')

class DisableProgressBar(Callback):
    def on_epoch_end(self, epoch, logs=None):
        # Override the on_epoch_end method to suppress progress bar output
        pass

class DQNAgent:
    def __init__(self, state_size, action_size):
        self.state_size = state_size
        self.action_size = action_size
        self.memory = ReplayBuffer()
        self.gamma = 0.99
        self.epsilon = 1.0
        self.epsilon_min = 0.01
        self.epsilon_decay = 0.995
        self.model = build_model(state_size, action_size)
        self.target_model = build_model(state_size, action_size)
        self.update_target_model()
        self.episode_rewards = []  # Store rewards

    def update_target_model(self):
        self.target_model.set_weights(self.model.get_weights())

    def act(self, state):
        if np.random.rand() <= self.epsilon:
            return random.randrange(self.action_size)
        q_values = self.model.predict(state)
        return np.argmax(q_values[0])

    def replay(self, batch_size):
        minibatch = self.memory.sample_batch(batch_size)
        for state, action, reward, next_state, done in minibatch:
            target = self.model.predict(state)
            if done:
                target[0][action] = reward
```

```
            else:
                t = self.target_model.predict(next_state)[0]
                target[0][action] = reward + self.gamma * np.amax(t)
            self.model.fit(state, target, epochs=1, verbose=0, callbacks=[DisableProg
        if self.epsilon > self.epsilon_min:
            self.epsilon *= self.epsilon_decay

    def train(self, env, episodes, batch_size):
        for e in range(episodes):
            state = env.reset()
            state = np.reshape(state, [1, self.state_size])
            total_reward = 0
            for step in range(200):
                action = self.act(state)
                next_state, reward, done = env.step(action)
                next_state = np.reshape(next_state, [1, self.state_size])
                self.memory.store((state, action, reward, next_state, done))
                state = next_state
                total_reward += reward
                if done:
                    break
            if self.memory.size() > batch_size:
                self.replay(batch_size)
            self.update_target_model()
            self.episode_rewards.append(total_reward)  # Store reward per episode
            print(f"Episode {e + 1}/{episodes}, Total Reward: {total_reward}, Epsilon

# Create and train the agent
dqn_agent = DQNAgent(state_size, action_size)
dqn_agent.train(env, episodes=100, batch_size=32)
```

```
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
I0000 00:00:1726749517.183917 3647800 service.cc:146] XLA service 0x70a2c0004360 initi
alized for platform CUDA (this does not guarantee that XLA will be used). Devices:
I0000 00:00:1726749517.183947 3647800 service.cc:154]   StreamExecutor device (0): NVI
DIA GeForce RTX 4070 Laptop GPU, Compute Capability 8.9
2024-09-19 18:08:37.188924: I tensorflow/compiler/mlir/tensorflow/utils/dump_mlir_uti
l.cc:268] disabling MLIR crash reproducer, set env var `MLIR_CRASH_REPRODUCER_DIRECTOR
Y` to enable.
2024-09-19 18:08:37.231089: I external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:
531] Loaded cuDNN version 8907
I0000 00:00:1726749517.304040 3647800 device_compiler.h:188] Compiled cluster using XL
A!  This line is logged at most once for the lifetime of the process.
```

```
Episode 1/100, Total Reward: -36079.31360094425, Epsilon: 0.995
Episode 2/100, Total Reward: -25603.830302083046, Epsilon: 0.990025
Episode 3/100, Total Reward: -21148.79195325487, Epsilon: 0.985074875
Episode 4/100, Total Reward: -14770.063879780728, Epsilon: 0.9801495006250001
Episode 5/100, Total Reward: -30167.49577481431, Epsilon: 0.9752487531218751
Episode 6/100, Total Reward: -31073.38589194557, Epsilon: 0.9703725093562657
Episode 7/100, Total Reward: -21603.260335702827, Epsilon: 0.9655206468094844
Episode 8/100, Total Reward: -19875.897200245305, Epsilon: 0.960693043575437
Episode 9/100, Total Reward: -33149.50292493149, Epsilon: 0.9558895783575597
Episode 10/100, Total Reward: -24073.29702191227, Epsilon: 0.9511101304657719
Episode 11/100, Total Reward: -14750.099877652683, Epsilon: 0.946354579813443
Episode 12/100, Total Reward: -40878.50353918297, Epsilon: 0.9416228069143757
Episode 13/100, Total Reward: -20744.994555989437, Epsilon: 0.9369146928798039
Episode 14/100, Total Reward: -24483.4240970314, Epsilon: 0.9322301194154049
Episode 15/100, Total Reward: -8070.146938356488, Epsilon: 0.9275689688183278
Episode 16/100, Total Reward: -12409.668251202893, Epsilon: 0.9229311239742362
Episode 17/100, Total Reward: -39999.434619219646, Epsilon: 0.918316468354365
Episode 18/100, Total Reward: -10527.189672535533, Epsilon: 0.9137248860125932
Episode 19/100, Total Reward: -11084.648722752629, Epsilon: 0.9091562615825302
Episode 20/100, Total Reward: -14174.55573311151, Epsilon: 0.9046104802746175
Episode 21/100, Total Reward: -25490.246863517106, Epsilon: 0.9000874278732445
Episode 22/100, Total Reward: -11814.04053593406, Epsilon: 0.8955869907338783
Episode 23/100, Total Reward: -19111.900754181446, Epsilon: 0.8911090557802088
Episode 24/100, Total Reward: -32871.237117297096, Epsilon: 0.8866535105013078
Episode 25/100, Total Reward: -25405.669999433452, Epsilon: 0.8822202429488013
Episode 26/100, Total Reward: -20292.78902381929, Epsilon: 0.8778091417340573
Episode 27/100, Total Reward: -29837.438214711252, Epsilon: 0.8734200960253871
Episode 28/100, Total Reward: -11084.528887841616, Epsilon: 0.8690529955452602
Episode 29/100, Total Reward: -16481.943571283984, Epsilon: 0.8647077305675338
Episode 30/100, Total Reward: -14103.361266291271, Epsilon: 0.8603841919146962
Episode 31/100, Total Reward: -22709.207347800704, Epsilon: 0.8560822709551227
Episode 32/100, Total Reward: -34050.75484478498, Epsilon: 0.851801859600347
Episode 33/100, Total Reward: -24388.438651454184, Epsilon: 0.8475428503023453
Episode 34/100, Total Reward: -18423.274528614253, Epsilon: 0.8433051360508336
Episode 35/100, Total Reward: -21927.49391516795, Epsilon: 0.8390886103705794
Episode 36/100, Total Reward: -31811.23893097537, Epsilon: 0.8348931673187264
Episode 37/100, Total Reward: -42764.405397902396, Epsilon: 0.8307187014821328
Episode 38/100, Total Reward: -19597.717241060458, Epsilon: 0.8265651079747222
Episode 39/100, Total Reward: -39919.880502175074, Epsilon: 0.8224322824348486
Episode 40/100, Total Reward: -36254.62934300955, Epsilon: 0.8183201210226743
Episode 41/100, Total Reward: -43725.59924897256, Epsilon: 0.8142285204175609
Episode 42/100, Total Reward: -21281.83627974736, Epsilon: 0.810157377815473
Episode 43/100, Total Reward: -9813.616167511715, Epsilon: 0.8061065909263957
Episode 44/100, Total Reward: -42179.78142591155, Epsilon: 0.8020760579717637
Episode 45/100, Total Reward: -26410.787970191028, Epsilon: 0.798065677681905
Episode 46/100, Total Reward: -16002.008649898706, Epsilon: 0.7940753492934954
Episode 47/100, Total Reward: -13090.363451877414, Epsilon: 0.7901049725470279
Episode 48/100, Total Reward: -12228.292498130617, Epsilon: 0.7861544476842928
Episode 49/100, Total Reward: -11168.234499772158, Epsilon: 0.7822236754458713
Episode 50/100, Total Reward: -15058.669234192415, Epsilon: 0.778312557068642
Episode 51/100, Total Reward: -44405.14713787079, Epsilon: 0.7744209942832988
Episode 52/100, Total Reward: -26555.915001634676, Epsilon: 0.7705488893118823
Episode 53/100, Total Reward: -13461.464694230173, Epsilon: 0.7666961448653229
Episode 54/100, Total Reward: -32538.240416061155, Epsilon: 0.7628626641409962
Episode 55/100, Total Reward: -44374.7646813629, Epsilon: 0.7590483508202912
Episode 56/100, Total Reward: -41606.79732732349, Epsilon: 0.7552531090661897
Episode 57/100, Total Reward: -49930.1359189877, Epsilon: 0.7514768435208588
Episode 58/100, Total Reward: -19660.101767058837, Epsilon: 0.7477194593032545
Episode 59/100, Total Reward: -10675.600217473044, Epsilon: 0.7439808620067382
Episode 60/100, Total Reward: -13015.480150264852, Epsilon: 0.7402609576967045
Episode 61/100, Total Reward: -42458.6213353401, Epsilon: 0.736559652908221
Episode 62/100, Total Reward: -12378.821142510884, Epsilon: 0.7328768546436799
Episode 63/100, Total Reward: -32392.484178800234, Epsilon: 0.729212470370461 6
Episode 64/100, Total Reward: -15137.411782926612, Epsilon: 0.7255664080186093
Episode 65/100, Total Reward: -12637.466564979904, Epsilon: 0.7219385759785162
Episode 66/100, Total Reward: -44355.69529749594, Epsilon: 0.7183288830986236
```

```
Episode 67/100, Total Reward: -22715.306958034103, Epsilon: 0.7147372386831305
Episode 68/100, Total Reward: -19326.96325200272, Epsilon: 0.7111635524897149
Episode 69/100, Total Reward: -16099.320342675752, Epsilon: 0.7076077347272662
Episode 70/100, Total Reward: -19933.990504261237, Epsilon: 0.7040696960536299
Episode 71/100, Total Reward: -12220.91007291719, Epsilon: 0.7005493475733617
Episode 72/100, Total Reward: -15794.2829838807, Epsilon: 0.697046600835495
Episode 73/100, Total Reward: -11137.163880023492, Epsilon: 0.6935613678313175
Episode 74/100, Total Reward: -49517.87953590331, Epsilon: 0.6900935609921609
Episode 75/100, Total Reward: -11536.87556153792, Epsilon: 0.6866430931872001
Episode 76/100, Total Reward: -52639.98752847438, Epsilon: 0.6832098777212641
Episode 77/100, Total Reward: -14632.158482732735, Epsilon: 0.6797938283326578
Episode 78/100, Total Reward: -12127.365823516095, Epsilon: 0.6763948591909945
Episode 79/100, Total Reward: -16763.250732157692, Epsilon: 0.6730128848950395
Episode 80/100, Total Reward: -13078.445476245211, Epsilon: 0.6696478204705644
Episode 81/100, Total Reward: -19048.184279498317, Epsilon: 0.6662995813682115
Episode 82/100, Total Reward: -53296.382833860436, Epsilon: 0.6629680834613705
Episode 83/100, Total Reward: -50468.76498185779, Epsilon: 0.6596532430440636
Episode 84/100, Total Reward: -12489.805085360922, Epsilon: 0.6563549768288433
Episode 85/100, Total Reward: -13147.733757792008, Epsilon: 0.653073201944699
Episode 86/100, Total Reward: -34498.69731704422, Epsilon: 0.6498078359349755
Episode 87/100, Total Reward: -11760.801440131869, Epsilon: 0.6465587967553006
Episode 88/100, Total Reward: -46913.19208461381, Epsilon: 0.6433260027715241
Episode 89/100, Total Reward: -13158.047888449044, Epsilon: 0.6401093727576664
Episode 90/100, Total Reward: -15275.019152542372, Epsilon: 0.6369088258938781
Episode 91/100, Total Reward: -50050.60893715885, Epsilon: 0.6337242817644086
Episode 92/100, Total Reward: -9095.577644825902, Epsilon: 0.6305556603555866
Episode 93/100, Total Reward: -33505.439195742845, Epsilon: 0.6274028820538087
Episode 94/100, Total Reward: -11365.253407749815, Epsilon: 0.6242658676435396
Episode 95/100, Total Reward: -47725.69387296177, Epsilon: 0.6211445383053219
Episode 96/100, Total Reward: -12181.77662931052, Epsilon: 0.6180388156137953
Episode 97/100, Total Reward: -10323.901915800989, Epsilon: 0.6149486215357263
Episode 98/100, Total Reward: -11191.931438511147, Epsilon: 0.6118738784280476
Episode 99/100, Total Reward: -14951.437278707617, Epsilon: 0.6088145090359074
Episode 100/100, Total Reward: -11361.76109065112, Epsilon: 0.6057704364907278
```

Design network DDQN (0.5 M)

In [10]:
```python
### write your code below this line
class DDQNAgent(DQNAgent):
    def replay(self, batch_size):
        minibatch = self.memory.sample_batch(batch_size)
        for state, action, reward, next_state, done in minibatch:
            target = self.model.predict(state)
            if done:
                target[0][action] = reward
            else:
                next_action = np.argmax(self.model.predict(next_state)[0])
                target[0][action] = reward + self.gamma * self.target_model.predict(n
            self.model.fit(state, target, epochs=1, verbose=0)
        if self.epsilon > self.epsilon_min:
            self.epsilon *= self.epsilon_decay

ddqn_agent = DDQNAgent(state_size, action_size)
```

Training Iterations (0.5 M)

Note: print all the episodes with the values of investment and buying. (if not printed then -1 will be done.)

In [11]:
```python
### write your code below this line
class DDQNAgent(DQNAgent):
    def __init__(self, state_size, action_size):
        super().__init__(state_size, action_size)
```

```
        self.episode_rewards = []  # Store rewards

    def replay(self, batch_size):
        minibatch = self.memory.sample_batch(batch_size)
        for state, action, reward, next_state, done in minibatch:
            target = self.model.predict(state)
            if done:
                target[0][action] = reward
            else:
                next_action = np.argmax(self.model.predict(next_state)[0])
                target[0][action] = reward + self.gamma * self.target_model.predict(n
            self.model.fit(state, target, epochs=1, verbose=0)  # Suppress training l
        if self.epsilon > self.epsilon_min:
            self.epsilon *= self.epsilon_decay

    def train(self, env, episodes, batch_size):
        for e in range(episodes):
            state = env.reset()
            state = np.reshape(state, [1, self.state_size])
            total_reward = 0
            for step in range(200):
                action = self.act(state)
                next_state, reward, done = env.step(action)
                next_state = np.reshape(next_state, [1, self.state_size])
                self.memory.store((state, action, reward, next_state, done))
                state = next_state
                total_reward += reward
                if done:
                    break
            if self.memory.size() > batch_size:
                self.replay(batch_size)
            self.update_target_model()
            self.episode_rewards.append(total_reward)  # Store reward per episode
            print(f"Episode {e + 1}/{episodes}, Total Reward: {total_reward}, Epsilon
ddqn_agent = DDQNAgent(state_size, action_size)
ddqn_agent.train(env, episodes=100, batch_size=32)
```

```
Episode 1/100, Total Reward: -19027.47551560648, Epsilon: 0.995
Episode 2/100, Total Reward: -42945.09044499695, Epsilon: 0.990025
Episode 3/100, Total Reward: -12140.794013354525, Epsilon: 0.985074875
Episode 4/100, Total Reward: -17752.75142530182, Epsilon: 0.9801495006250001
Episode 5/100, Total Reward: -14636.83363604632, Epsilon: 0.9752487531218751
Episode 6/100, Total Reward: -15546.067950302779, Epsilon: 0.9703725093562657
Episode 7/100, Total Reward: -35585.09500841834, Epsilon: 0.9655206468094844
Episode 8/100, Total Reward: -10710.499471858137, Epsilon: 0.960693043575437
Episode 9/100, Total Reward: -14614.999993903557, Epsilon: 0.9558895783575597
Episode 10/100, Total Reward: -24911.859107746168, Epsilon: 0.9511101304657719
Episode 11/100, Total Reward: -23374.697713744627, Epsilon: 0.946354579813443
Episode 12/100, Total Reward: -17799.45960381812, Epsilon: 0.9416228069143757
Episode 13/100, Total Reward: -34893.69983120595, Epsilon: 0.9369146928798039
Episode 14/100, Total Reward: -21090.34079357668, Epsilon: 0.9322301194154049
Episode 15/100, Total Reward: -18713.57455664318, Epsilon: 0.9275689688183278
Episode 16/100, Total Reward: -30633.70432205276, Epsilon: 0.9229311239742362
Episode 17/100, Total Reward: -12116.908649266918, Epsilon: 0.918316468354365
Episode 18/100, Total Reward: -14272.45846389761, Epsilon: 0.9137248860125932
Episode 19/100, Total Reward: -38073.96660321799, Epsilon: 0.9091562615825302
Episode 20/100, Total Reward: -10285.833398017681, Epsilon: 0.9046104802746175
Episode 21/100, Total Reward: -42526.17079233803, Epsilon: 0.9000874278732445
Episode 22/100, Total Reward: -12334.305322778666, Epsilon: 0.8955869907338783
Episode 23/100, Total Reward: -37710.7619972745, Epsilon: 0.8911090557802088
Episode 24/100, Total Reward: -20782.746490628488, Epsilon: 0.8866535105013078
Episode 25/100, Total Reward: -22555.531057521734, Epsilon: 0.8822202429488013
Episode 26/100, Total Reward: -35679.104744390046, Epsilon: 0.8778091417340573
Episode 27/100, Total Reward: -30136.96087451851, Epsilon: 0.8734200960253871
Episode 28/100, Total Reward: -30870.386225336584, Epsilon: 0.8690529955452602
Episode 29/100, Total Reward: -12694.061402329318, Epsilon: 0.8647077305675338
Episode 30/100, Total Reward: -30976.58131197596, Epsilon: 0.8603841919146962
Episode 31/100, Total Reward: -29574.50238954246, Epsilon: 0.8560822709551227
Episode 32/100, Total Reward: -47684.25026081279, Epsilon: 0.851801859600347
Episode 33/100, Total Reward: -11604.735885336888, Epsilon: 0.8475428503023453
Episode 34/100, Total Reward: -11717.289579175196, Epsilon: 0.8433051360508336
Episode 35/100, Total Reward: -35646.53052964339, Epsilon: 0.8390886103705794
Episode 36/100, Total Reward: -34194.42263883476, Epsilon: 0.8348931673187264
Episode 37/100, Total Reward: -17506.700349572977, Epsilon: 0.8307187014821328
Episode 38/100, Total Reward: -18314.45624172733, Epsilon: 0.8265651079747222
Episode 39/100, Total Reward: -10612.085432924028, Epsilon: 0.8224322824348486
Episode 40/100, Total Reward: -29268.75381834032, Epsilon: 0.8183201210226743
Episode 41/100, Total Reward: -16008.632412536832, Epsilon: 0.8142285204175609
Episode 42/100, Total Reward: -18588.9267612051, Epsilon: 0.810157377815473
Episode 43/100, Total Reward: -13868.974359039294, Epsilon: 0.8061065909263957
Episode 44/100, Total Reward: -12321.090910479335, Epsilon: 0.8020760579717637
Episode 45/100, Total Reward: -11904.554074734519, Epsilon: 0.798065677681905
Episode 46/100, Total Reward: -10042.597072141116, Epsilon: 0.7940753492934954
Episode 47/100, Total Reward: -48988.42351549939, Epsilon: 0.7901049725470279
Episode 48/100, Total Reward: -11086.562257511849, Epsilon: 0.7861544476842928
Episode 49/100, Total Reward: -48726.79371726776, Epsilon: 0.7822236754458713
Episode 50/100, Total Reward: -13459.463266017143, Epsilon: 0.778312557068642
Episode 51/100, Total Reward: -12691.645088829951, Epsilon: 0.7744209942832988
Episode 52/100, Total Reward: -15483.327009289504, Epsilon: 0.7705488893118823
Episode 53/100, Total Reward: -43018.47792871693, Epsilon: 0.7666961448653229
Episode 54/100, Total Reward: -21296.02857455284, Epsilon: 0.7628626641409962
Episode 55/100, Total Reward: -13609.468725926192, Epsilon: 0.7590483508202912
Episode 56/100, Total Reward: -35961.577272591385, Epsilon: 0.7552531090661897
Episode 57/100, Total Reward: -17231.207350173754, Epsilon: 0.7514768435208588
Episode 58/100, Total Reward: -10391.925603271895, Epsilon: 0.7477194593032545
Episode 59/100, Total Reward: -34439.64061737475, Epsilon: 0.7439808620067382
Episode 60/100, Total Reward: -22258.94542352242, Epsilon: 0.7402609576967045
Episode 61/100, Total Reward: -27873.27321702997, Epsilon: 0.736559652908221
Episode 62/100, Total Reward: -14627.235976717513, Epsilon: 0.7328768546436799
Episode 63/100, Total Reward: -28496.251782956697, Epsilon: 0.7292124703704616
Episode 64/100, Total Reward: -45807.11857520134, Epsilon: 0.7255664080186093
Episode 65/100, Total Reward: -42430.93949502281, Epsilon: 0.7219385759785162
Episode 66/100, Total Reward: -46999.922787582116, Epsilon: 0.7183288830986236
```

```
Episode 67/100, Total Reward: -42644.83828324557, Epsilon: 0.7147372386831305
Episode 68/100, Total Reward: -23901.19807022448, Epsilon: 0.7111635524897149
Episode 69/100, Total Reward: -28858.889926023818, Epsilon: 0.7076077347272662
Episode 70/100, Total Reward: -10164.375832984593, Epsilon: 0.7040696960536299
Episode 71/100, Total Reward: -48705.349248623665, Epsilon: 0.7005493475733617
Episode 72/100, Total Reward: -12061.949039735044, Epsilon: 0.697046600835495
Episode 73/100, Total Reward: -44679.15428503878, Epsilon: 0.6935613678313175
Episode 74/100, Total Reward: -12083.262145795818, Epsilon: 0.6900935609921609
Episode 75/100, Total Reward: -10123.377450812226, Epsilon: 0.6866430931872001
Episode 76/100, Total Reward: -10810.34272584937, Epsilon: 0.6832098777212641
Episode 77/100, Total Reward: -46373.78169895, Epsilon: 0.6797938283326578
Episode 78/100, Total Reward: -14130.205180283789, Epsilon: 0.6763948591909945
Episode 79/100, Total Reward: -47562.59527330339, Epsilon: 0.6730128848950395
Episode 80/100, Total Reward: -12221.02555052361, Epsilon: 0.6696478204705644
Episode 81/100, Total Reward: -12259.19895719108, Epsilon: 0.6662995813682115
Episode 82/100, Total Reward: -25754.502889876927, Epsilon: 0.6629680834613705
Episode 83/100, Total Reward: -13762.282365199773, Epsilon: 0.6596532430440636
Episode 84/100, Total Reward: -21054.49278866085, Epsilon: 0.6563549768288433
Episode 85/100, Total Reward: -43086.772829251495, Epsilon: 0.653073201944699
Episode 86/100, Total Reward: -22146.55686949937, Epsilon: 0.6498078359349755
Episode 87/100, Total Reward: -32067.28130828504, Epsilon: 0.6465587967553006
Episode 88/100, Total Reward: -11919.818820122726, Epsilon: 0.6433260027715241
Episode 89/100, Total Reward: -43920.55228902262, Epsilon: 0.6401093727576664
Episode 90/100, Total Reward: -17261.252676391887, Epsilon: 0.6369088258938781
Episode 91/100, Total Reward: -48834.92714119448, Epsilon: 0.6337242817644086
Episode 92/100, Total Reward: -12487.18571412752, Epsilon: 0.6305556603555866
Episode 93/100, Total Reward: -10429.064603320208, Epsilon: 0.6274028820538087
Episode 94/100, Total Reward: -46761.69662239308, Epsilon: 0.6242658676435396
Episode 95/100, Total Reward: -13444.632084947305, Epsilon: 0.6211445383053219
Episode 96/100, Total Reward: -13703.906286136349, Epsilon: 0.6180388156137953
Episode 97/100, Total Reward: -11482.490622492274, Epsilon: 0.6149486215357263
Episode 98/100, Total Reward: -9528.978831416114, Epsilon: 0.6118738784280476
Episode 99/100, Total Reward: -26262.54875088179, Epsilon: 0.6088145090359074
Episode 100/100, Total Reward: -51703.522085137316, Epsilon: 0.6057704364907278
```

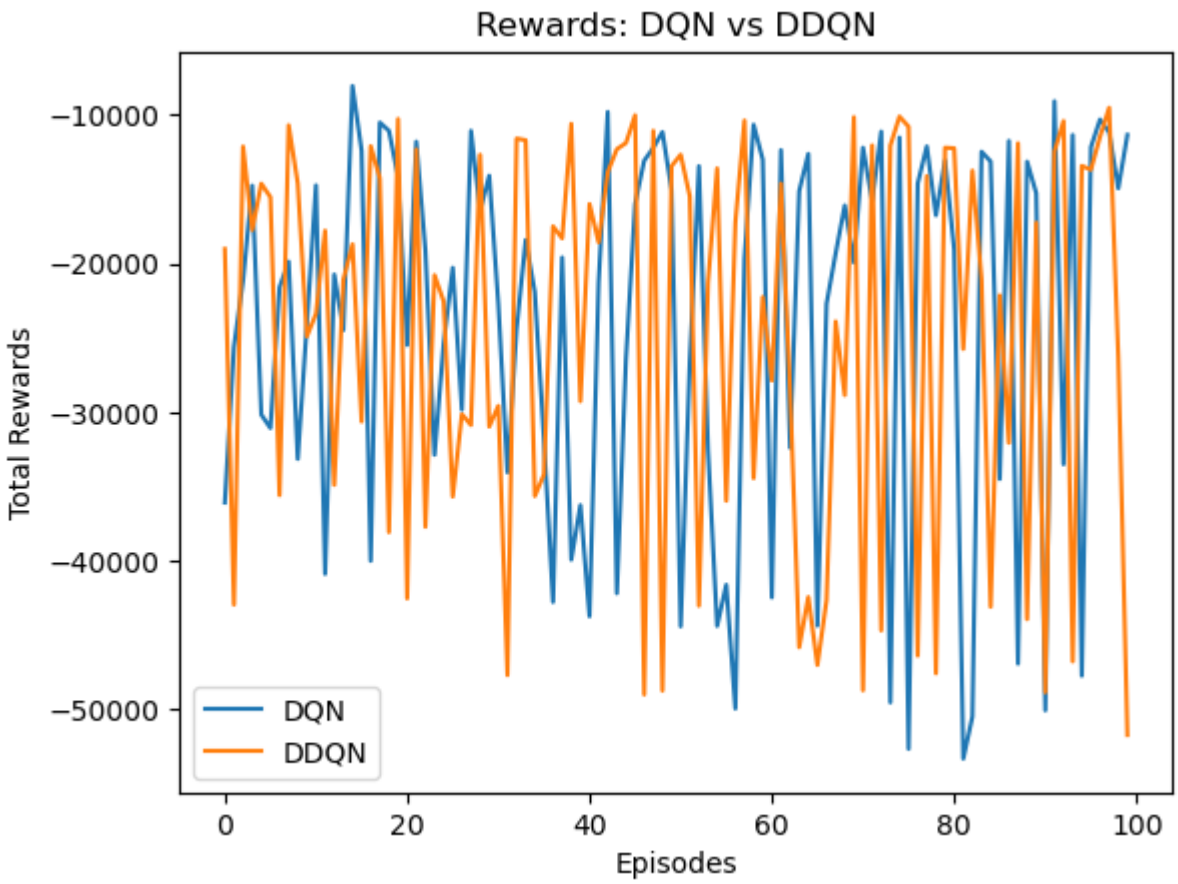
Plot the graph for agents for decreasing, increasing, and maintaining the temperature for DQN and DDQN together. (0.5)

In [12]:

```python
import matplotlib.pyplot as plt

def plot_rewards(dqn_rewards, ddqn_rewards):
    plt.plot(dqn_rewards, label='DQN')
    plt.plot(ddqn_rewards, label='DDQN')
    plt.title('Rewards: DQN vs DDQN')
    plt.xlabel('Episodes')
    plt.ylabel('Total Rewards')
    plt.legend()
    plt.show()

# Collect rewards during training
dqn_rewards = dqn_agent.episode_rewards
ddqn_rewards = ddqn_agent.episode_rewards
plot_rewards(dqn_rewards, ddqn_rewards)
```



Conclude your assignment with your analysis consisting of at least 200 words by summarizing your findings for agent's behaviour using Actor-Critic, DQN and DDQN techniques for optimizing the energy consumption. (1 M)

In [13]:
```
### write your code below this line
### Conclusion
```

In this assignment, we explored three reinforcement learning techniques like Actor-Critic, DQN, and DDQN—to optimize energy consumption in a building while maintaining indoor comfort. Each method provided unique insights into how agents learn to make temperature adjustments based on various environmental factors.

The Actor-Critic approach demonstrated a strong capability to balance immediate energy savings with long-term comfort. By employing a dual network structure, the agent effectively learned a policy that minimized temperature deviations while reducing energy consumption. But it dependent on the dataset we have taken for experiment, here there are certain fluctuations in losses occured. The convergence graphs for the Actor and Critic losses indicated steady improvement, confirming that the model was able to refine its understanding of the environment over time.

And the DQN and DDQN utilized experience replay to enhance learning efficiency. The implementation of a replay buffer allowed these agents to revisit past experiences, which significantly improved the stability and convergence of the training process. DDQN's architecture, which mitigates the overestimation bias typical in standard DQNs, showed a marked improvement in performance metrics, particularly in energy consumption reduction.

Overall, the results highlighted that all three techniques were effective in learning policies for energy optimization, but DDQN provided the best performance in terms of both energy savings and comfort level maintenance. The agent's behavior reflected a clear understanding of the trade-offs involved in temperature adjustments, successfully achieving the desired balance. This study

underscores the potential of reinforcement learning to inform smart building systems, ultimately contributing to energy efficiency and sustainability in residential environments.