CS-741 NEXT GENERATION CLOUD ARCHITECTURE

A REPORT ON THE PROJECT ENTITLED

Energy-Efficient Computation Offloading in Edge Computing for Multi-UAV Networks

Group Members:

KALP PATEL 242CS028

ABHISHEK PANDEY 242CS004

I SEMESTER M-TECH CSE

DEPT. OF COMPUTER SCIENCE AND ENGINEERING

NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA

SURATHKAL

Contents

Core Concept

The paper explores the integration of **IoT networks**, **edge computing**, **UAVs**, and **computational offloading** to optimize computational performance and energy efficiency. Below are the definitions and their combined advantages:

1. **IoT Networks**:
   The *Internet of Things (IoT)* refers to interconnected devices embedded with sensors, software, and network capabilities that collect and exchange data. Examples include smart home devices, wearable technology, and industrial sensors.
   Challenge: Many IoT devices are resource-constrained, with limited computational power and energy capacity, making it difficult to handle large-scale or computationally intensive tasks in real-time.

2. **Edge Computing**:
   Edge computing shifts computational and storage tasks closer to the data source (i.e., IoT devices) by using edge servers located within or near the network's periphery. This reduces latency, minimizes bandwidth usage, and supports real-time data processing compared to centralized cloud computing.
   Key Advantage: By processing data closer to the devices, edge computing alleviates the computational and energy burdens on IoT devices.

3. **UAVs (Unmanned Aerial Vehicles)**:
   UAVs, or drones, are autonomous or remotely piloted aerial vehicles used for various applications, such as surveillance, logistics, and communications. When equipped with computational and communication capabilities, UAVs can function as mobile edge computing nodes.
   Unique Role: In environments where fixed infrastructure is unavailable or impractical (e.g., disaster zones, remote areas), UAVs provide flexible and scalable support for IoT networks.

4. **Computational Offloading**:
   Computational offloading refers to the process of transferring computationally intensive tasks from resource-constrained IoT devices to more capable servers, such as edge servers on UAVs. This reduces the energy and processing load on IoT devices while maintaining system performance.
   Key Benefit: Offloading tasks to UAVs or edge servers ensures efficient resource utilization, reduces latency, and improves the energy efficiency of the overall system.

Integration and Advantages

The combination of IoT networks, edge computing, UAVs, and computational offloading brings several unique advantages:

- **Enhanced Energy Efficiency:**
  IoT devices offload computationally intensive tasks to UAVs acting as edge servers, reducing their local energy consumption. The proposed system optimizes UAV paths and task allocation, minimizing energy usage for both IoT devices and UAVs.

- **Real-Time Processing with Reduced Latency:**
  By leveraging nearby UAVs as edge servers, the delay caused by long-distance transmission to centralized cloud servers is eliminated. This ensures faster data processing, critical for real-time applications like disaster management or autonomous vehicles.

- **Dynamic and Flexible Coverage:**
  Unlike static edge servers, UAVs can dynamically reposition themselves to serve IoT devices in changing environments. This mobility is particularly beneficial for applications requiring coverage in remote or underserved regions.

- **Extended System Lifespan:**
  Computational offloading allows for balanced energy consumption across the network, enhancing the operational duration of both UAVs and IoT devices. UAVs can efficiently manage computational loads while conserving energy for sustained operation.

- **Improved Scalability:**
  As IoT networks expand, UAV-based edge computing offers a scalable solution by deploying additional drones to cover new areas or meet rising computational demands.

- **Resource Optimization through Computational Offloading:**
  Offloading tasks to UAVs ensures that IoT devices can focus on less demanding operations, while UAVs utilize their enhanced computational capabilities. This results in optimal resource usage and reduced system bottlenecks.

The integration of IoT networks, edge computing, UAVs, and computational offloading creates a synergistic system that addresses the limitations of IoT devices while leveraging the mobility and computational power of UAVs. This results in a network that is energy-efficient, responsive, and adaptive to dynamic environments, making it a robust solution for modern IoT applications.

Problem to Address

Main Problem

The paper focuses on optimizing **energy-efficient computation offloading** and **UAV path planning** within multi-UAV-enabled edge computing systems for IoT networks. The main problem arises from the resource constraints of IoT devices and UAVs, including limited energy and computational capacity, which make it challenging to meet real-time processing and energy efficiency requirements.

Core Objective: To maximize *overall energy efficiency* across both IoT devices and UAVs while ensuring effective task offloading and UAV path planning. This includes:

1. Balancing energy consumption between IoT devices and UAVs.
2. Minimizing delays in task execution by optimizing computation offloading decisions.
3. Supporting dynamic and large-scale IoT deployments through scalable and adaptive methods.

Key Challenges

1. **Resource Constraints:**
   - IoT devices have limited computational power and battery life, restricting their ability to process complex tasks locally.
   - UAVs also have constrained energy supplies due to battery limitations, which are further taxed by flight and computational operations.
2. **Dynamic Environments:**
   - IoT devices are often mobile or deployed in dynamic environments, making it difficult to predict their exact locations and computational needs.
   - UAVs must adapt their trajectories and resource allocation in real-time to serve these devices effectively.
3. **High Computational Complexity:**
   - The joint optimization of computation offloading and UAV path planning involves solving a large-scale problem with multiple variables, such as task allocation, energy consumption, and flight dynamics.
   - Traditional approaches struggle to efficiently handle the complexity of multi-UAV networks and dynamic IoT environments.

4. **Energy Efficiency Trade-offs:**
    - Offloading tasks to UAVs can save energy for IoT devices but increases UAV energy consumption for both computation and flight.
    - Ensuring an optimal balance between local computation and offloading is critical to maintaining overall energy efficiency.
5. **Scalability:**
    - With the increasing number of IoT devices, the system must efficiently handle large-scale networks without significant degradation in performance.
    - Multi-UAV coordination adds another layer of complexity as UAVs must collaborate to serve overlapping areas without redundant resource usage.
6. **Real-Time Decision-Making:**
    - The system must make rapid decisions about offloading tasks and planning UAV trajectories to meet latency and real-time processing requirements.
    - This is particularly challenging in scenarios with unpredictable task requests or sudden changes in network conditions.
7. **Coverage and Connectivity:**
    - UAVs must maintain line-of-sight communication with IoT devices while ensuring sufficient coverage for all active devices within the network.
    - This requires strategic positioning of UAVs to optimize coverage while minimizing energy costs.

Addressing these challenges requires a comprehensive approach that combines computation offloading, UAV path optimization, and energy-efficient strategies. The paper introduces the **ECOP (Energy-Efficient Computation Offloading and Path Planning)** algorithm to tackle these issues, ensuring that the system meets energy efficiency and performance goals in dynamic and resource-constrained environments.

Proposed Approach

The paper proposes an integrated approach to optimize **energy-efficient computation offloading** and **UAV path planning** in multi-UAV-enabled edge computing systems. The key aspects of the proposed approach include:

Key Components

1. **Energy Model:** The approach introduces an energy model that considers the energy consumption of both UAVs and IoT devices. This model accounts for:
   - Energy consumed by IoT devices for local task execution.
   - Energy used by UAVs for computation and propulsion.
2. **Problem Formulation:** The problem is formulated as a joint optimization task, combining:
   - **Computation Offloading:** Deciding whether IoT tasks should be processed locally or offloaded to UAVs.
   - **UAV Path Planning:** Optimizing UAV trajectories to maximize energy efficiency while maintaining effective communication and computational support for IoT devices.
3. **Functional Coverage Set:** A novel concept of *functional coverage set* is introduced to determine the optimal set of IoT devices for which a UAV provides computational support. This set is constructed based on energy efficiency metrics and system utility.
4. **Algorithm (ECOP):** The proposed **Energy-Efficient Computation Offloading and Path Planning (ECOP)** algorithm is designed to:
   - Build and update the functional coverage set for each UAV.
   - Optimize UAV trajectories based on energy efficiency and task distribution.
   - Make offloading decisions that maximize system utility and minimize energy consumption.

Algorithm Overview

The ECOP algorithm operates as follows:

1. **Initialization:** Initialize the coverage sets for UAVs and identify task requirements from IoT devices.

2. **Utility Coverage Set Construction:** Sequentially add IoT devices to the functional coverage set of each UAV based on their energy efficiency contribution.

3. **Communication and Consensus:** IoT devices share energy efficiency values with nearby UAVs to reach consensus on task allocation.

4. **UAV Path Planning:** UAVs plan their trajectories to maximize energy efficiency and system utility.

5. **Task Offloading Decisions:** Tasks are either offloaded to UAVs or executed locally on IoT devices based on computational resource availability and energy efficiency metrics.

The proposed approach, leveraging the ECOP algorithm, provides an efficient solution for computation offloading and UAV path planning in multi-UAV-enabled edge computing systems. By optimizing energy efficiency and system utility, it addresses key challenges in IoT networks, making it a robust and scalable framework for dynamic environments.

Workflow

The simulation workflow aims to evaluate the proposed ECOP (Energy-Efficient Computation Offloading and Path Planning) algorithm. Below are the major steps and the code implementation:

1. Initialization

- Define the simulation parameters such as the number of IoT devices, UAVs, task properties, UAV flight energy models, and simulation area dimensions.
- Randomly initialize positions for IoT devices and UAVs within the simulation area.
- Generate random computational tasks for IoT devices, specifying task size and computation intensity.

2. UAV and IoT Device Movements

- IoT devices move randomly within the simulation area at each time slot.
- UAVs update their velocities and accelerations, adjusting their positions dynamically to serve IoT devices.
- Movements are constrained within the bounds of the simulation area.

3. Utility Coverage Set Construction

- Identify IoT devices within the coverage radius of each UAV.
- Construct the functional coverage set for each UAV based on the distance to IoT devices.

4. Energy Efficiency Calculations

- Calculate the energy required for task offloading, UAV computation, and UAV flight.
- Compute the energy efficiency as the ratio of processed task size to total energy consumed.
- Select tasks for offloading to UAVs based on maximum energy efficiency.

5. Local and Offloaded Task Execution

- Tasks not offloaded to UAVs are executed locally by IoT devices.
- Track the total energy consumed and the energy efficiency for both local and offloaded tasks.

6. Visualization

The simulation generates several visual outputs:

- Energy efficiency over time slots.
- Distribution of locally executed vs. offloaded tasks, with corresponding energy efficiencies.
- UAV trajectories over time, illustrating their dynamic movement to support IoT devices.

Code Implementation

The following Python code implements the workflow using `SimPy` for simulation and `Matplotlib` for visualization:

```
import simpy
import numpy as np
import matplotlib.pyplot as plt

# Parameters
SIM_TIME = 100  # Total simulation time
NUM_UAVS = 3
NUM_IOT_DEVICES = 500
SIM_AREA = (1000, 1000, 50)
TIME_SLOTS = 20
TASK_GEN_INTERVAL = 5
G = 9.81

# Task properties
TASK_SIZE_RANGE = (0.5, 5.2)  # Mb
COMPUTATION_INTENSITY_RANGE = (500, 1000)  # cycles per bit

# UAV properties
UAV_PROCESSING_POWER = 5e9  # GHz
UAV_FLIGHT_POWER = 20  # Watts
UAV_COVERAGE_RADIUS = 200  # meters
UAV_SPEED = 50  # Maximum UAV speed (m/s)

# UAV Flight Energy Model Constants
W2 = 9.26e5  # UAV flight energy model constant
WA = 2250    # UAV flight energy model constant

# IoT properties
IOT_PROCESSING_POWER = 1e9  # GHz
IOT_ENERGY_PER_CYCLE = 1e-6  # Joules
IOT_SPEED = 10  # Maximum speed of IoT devices in meters per time
    slot

# Initialize IoT devices and UAVs
iot_positions = np.random.uniform(0, SIM_AREA[0], (
   NUM_IOT_DEVICES, 2))
uav_positions = np.random.uniform(0, SIM_AREA[0], (NUM_UAVS, 2))
uav_velocities = np.zeros((NUM_UAVS, 2))  # Initial UAV
   velocities
```

```python
uav_accelerations = np.zeros((NUM_UAVS, 2))  # Initial UAV accelerations

tasks = [
    {
        "size": np.random.uniform(*TASK_SIZE_RANGE),  # MB
        "intensity": np.random.uniform(*COMPUTATION_INTENSITY_RANGE)  # cycles/bit
    }
    for _ in range(NUM_IOT_DEVICES)
]

# Energy Calculation Functions
def compute_offloading_energy(task, distance, transmission_power=1):
    """Compute energy consumed during task offloading."""
    data_size = task["size"] * 8  # Convert MB to bits
    channel_gain = 1 / (distance + 1e-6)  # Simplified path loss model
    return transmission_power * data_size / channel_gain

def compute_uav_computation_energy(task):
    """Compute energy consumed by UAV for task processing."""
    cycles = task["size"] * task["intensity"]
    return cycles / UAV_PROCESSING_POWER

def compute_local_computation_energy(task):
    """Compute energy consumed locally by IoT devices."""
    cycles = task["size"] * task["intensity"]
    return cycles * IOT_ENERGY_PER_CYCLE

def compute_uav_flight_energy(velocity, acceleration):
    """Compute UAV flight energy based on velocity and acceleration."""
    v_magnitude = np.linalg.norm(velocity)
    a_magnitude = np.linalg.norm(acceleration)
    return W2 * (v_magnitude**3) + (WA/v_magnitude) * (1 + a_magnitude**2 / G**2)

# Data for Visualization
energy_efficiencies = []
offloading_patterns = []
task_details = []  # Store task processing location and energy efficiency
uav_trajectories = {i: [] for i in range(NUM_UAVS)}  # Store trajectories for all UAVs

# ECOP Algorithm Simulation
# ECOP Algorithm Simulation (Updated)
def ecop(env, uav_positions, uav_velocities, uav_accelerations, iot_positions, tasks):
```

```
    for t in range(TIME_SLOTS):
        print(f"Time Slot {t+1} at simulation time {env.now}")
        uav_coverage_sets = [[] for _ in range(NUM_UAVS)]
        total_energy = 0
        offloading_decision_map = [False] * NUM_IOT_DEVICES
        task_efficiency_details = []  # Store energy efficiency
            for tasks in this slot

        # Step 1: IoT Device Movement
        for i in range(NUM_IOT_DEVICES):
            iot_positions[i] += np.random.uniform(-IOT_SPEED,
                IOT_SPEED, size=2)
            iot_positions[i] = np.clip(iot_positions[i], 0,
                SIM_AREA[0])

        # Step 2: UAV Movement
        for i in range(NUM_UAVS):
            new_velocity = np.random.uniform(-UAV_SPEED,
                UAV_SPEED, size=2)
            uav_accelerations[i] = (new_velocity - uav_velocities
                [i]) / TASK_GEN_INTERVAL
            uav_velocities[i] = new_velocity
            uav_positions[i] += uav_velocities[i] *
                TASK_GEN_INTERVAL
            uav_positions[i] = np.clip(uav_positions[i], 0,
                SIM_AREA[0])
            uav_trajectories[i].append(uav_positions[i].copy())

        # Step 3: Utility Coverage Set Construction
        for i, iot_pos in enumerate(iot_positions):
            for j, uav_pos in enumerate(uav_positions):
                distance = np.linalg.norm(iot_pos - uav_pos)
                if distance <= UAV_COVERAGE_RADIUS:
                    uav_coverage_sets[j].append(i)

        # Step 4: Energy Efficiency Calculation
        for uav_id, coverage_set in enumerate(uav_coverage_sets):
            max_efficiency = 0
            best_task = None
            for device_id in coverage_set:
                task = tasks[device_id]
                distance = np.linalg.norm(iot_positions[device_id
                    ] - uav_positions[uav_id])

                offloading_energy = compute_offloading_energy(
                    task, distance)
                uav_computation_energy =
                    compute_uav_computation_energy(task)
                uav_flight_energy = compute_uav_flight_energy(
                    uav_velocities[uav_id], uav_accelerations[
                    uav_id])
```

```
118
119                 total_uav_energy = offloading_energy +
                        uav_computation_energy + uav_flight_energy
120                 efficiency = task["size"] / total_uav_energy
121
122                 if efficiency > max_efficiency:
123                     max_efficiency = efficiency
124                     best_task = device_id
125
126             if best_task is not None:
127                 offloading_decision_map[best_task] = True
128                 task = tasks[best_task]
129                 distance = np.linalg.norm(iot_positions[best_task
                        ] - uav_positions[uav_id])
130                 total_energy += compute_offloading_energy(task,
                        distance) + compute_uav_computation_energy(
                        task)
131
132         # Step 5: Local Computation for Non-Offloaded Tasks and
                Printing Results
133         local_tasks = 0
134         offloaded_tasks = 0
135         for i, decision in enumerate(offloading_decision_map):
136             task = tasks[i]
137             if decision:
138                 offload_energy = compute_offloading_energy(task,
                        distance) + compute_uav_computation_energy(
                        task)
139                 efficiency = task["size"] / offload_energy
140                 task_efficiency_details.append(("Offloaded",
                        efficiency))
141                 total_energy += offload_energy
142                 offloaded_tasks += 1
143                 print(f"Task {i} is Offloaded to a UAV.")
144             else:
145                 local_energy = compute_local_computation_energy(
                        task)
146                 efficiency = task["size"] / local_energy
147                 task_efficiency_details.append(("Local",
                        efficiency))
148                 total_energy += local_energy
149                 local_tasks += 1
150                 print(f"Task {i} is Executed Locally.")
151
152         # Calculate and Print Total Energy Efficiency
153         total_efficiency = sum(eff[1] for eff in
                task_efficiency_details) / len(task_efficiency_details
                ) if task_efficiency_details else 0
154         energy_efficiencies.append(total_efficiency)
155         total_average_efficiency = sum(energy_efficiencies) / len
                (energy_efficiencies)
```

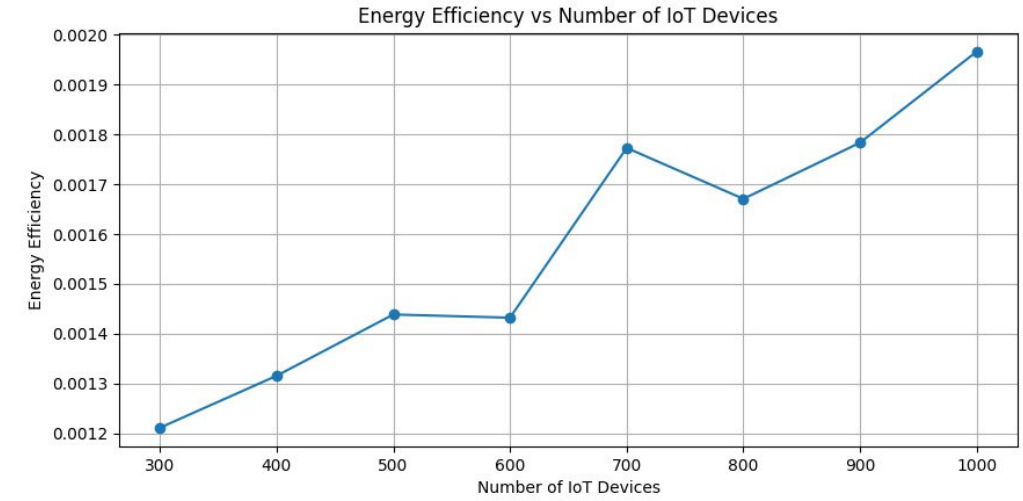
```
156         offloading_patterns.append(offloading_decision_map.copy()
                )
157         task_details.append(task_efficiency_details)
158 
159         print(f"Time Slot {t+1} Summary:")
160         print(f"  Total Energy Consumed: {total_energy:.2f} J")
161         print(f"  Number of Offloaded Tasks: {offloaded_tasks}")
162         print(f"  Number of Locally Executed Tasks: {local_tasks}
                ")
163         print(f"\nTotal Average Energy Efficiency of the System:
                {total_average_efficiency:.4f}")
164 
165         yield env.timeout(TASK_GEN_INTERVAL)
166 
167 
168 
169 # Visualization Functions
170 def visualize_energy_efficiency():
171     plt.figure(figsize=(10, 5))
172     plt.plot(range(1, TIME_SLOTS + 1), energy_efficiencies,
            marker='o', label="Energy Efficiency")
173     plt.title("Energy Efficiency Over Time Slots")
174     plt.xlabel("Time Slot")
175     plt.ylabel("Energy Efficiency (Processed Task Size / Total
            Energy)")
176     plt.legend()
177     plt.grid()
178     plt.show()
179 
180 def visualize_task_processing():
181     local_tasks = []
182     offloaded_tasks = []
183     efficiencies = []
184     for t, details in enumerate(task_details):
185         for decision, efficiency in details:
186             if decision == "Local":
187                 local_tasks.append((t, efficiency))
188             else:
189                 offloaded_tasks.append((t, efficiency))
190     local_times, local_eff = zip(*local_tasks)
191     offload_times, offload_eff = zip(*offloaded_tasks)
192 
193     plt.figure(figsize=(10, 5))
194     plt.scatter(local_times, local_eff, color='blue', label="
            Local Tasks")
195     plt.scatter(offload_times, offload_eff, color='green', label=
            "Offloaded Tasks")
196     plt.title("Task Processing and Energy Efficiency")
197     plt.xlabel("Time Slot")
198     plt.ylabel("Energy Efficiency")
199     plt.legend()
```

```
    plt.grid()
    plt.show()

def visualize_uav_paths():
    plt.figure(figsize=(10, 10))
    for uav_id, trajectory in uav_trajectories.items():
        trajectory = np.array(trajectory)
        plt.plot(trajectory[:, 0], trajectory[:, 1], label=f"UAV {uav_id}")  # Plot path as a line
    plt.scatter(iot_positions[:, 0], iot_positions[:, 1], c='red', label="IoT Devices", alpha=0.5)
    plt.title("UAV Paths Over Time")
    plt.xlabel("X-Coordinate")
    plt.ylabel("Y-Coordinate")
    plt.legend()
    plt.grid()
    plt.show()

# Simulation environment
env = simpy.Environment()
env.process(ecop(env, uav_positions, uav_velocities, uav_accelerations, iot_positions, tasks))
env.run(until=SIM_TIME)

# Call visualization functions
visualize_energy_efficiency()
visualize_task_processing()
visualize_uav_paths()
```

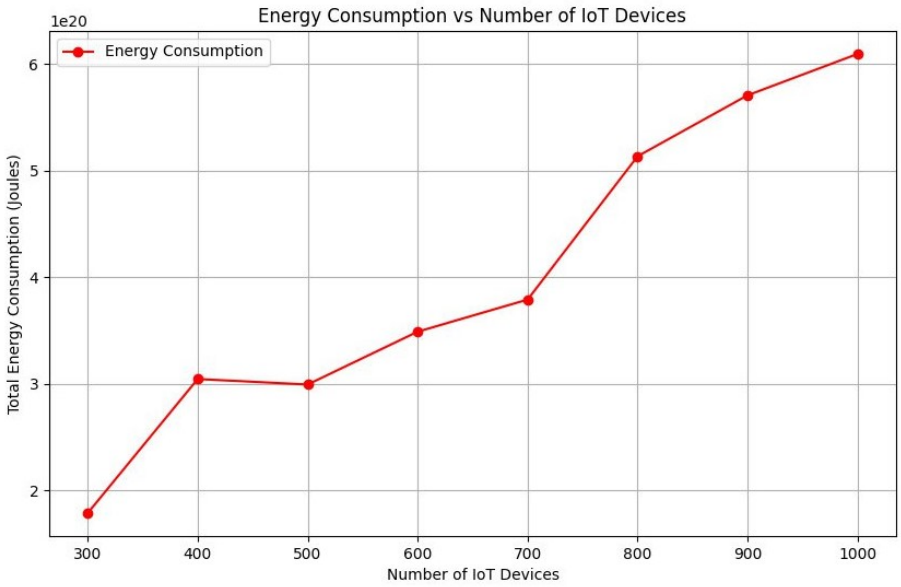
1: Code for ECOP Algorithm

Outputs

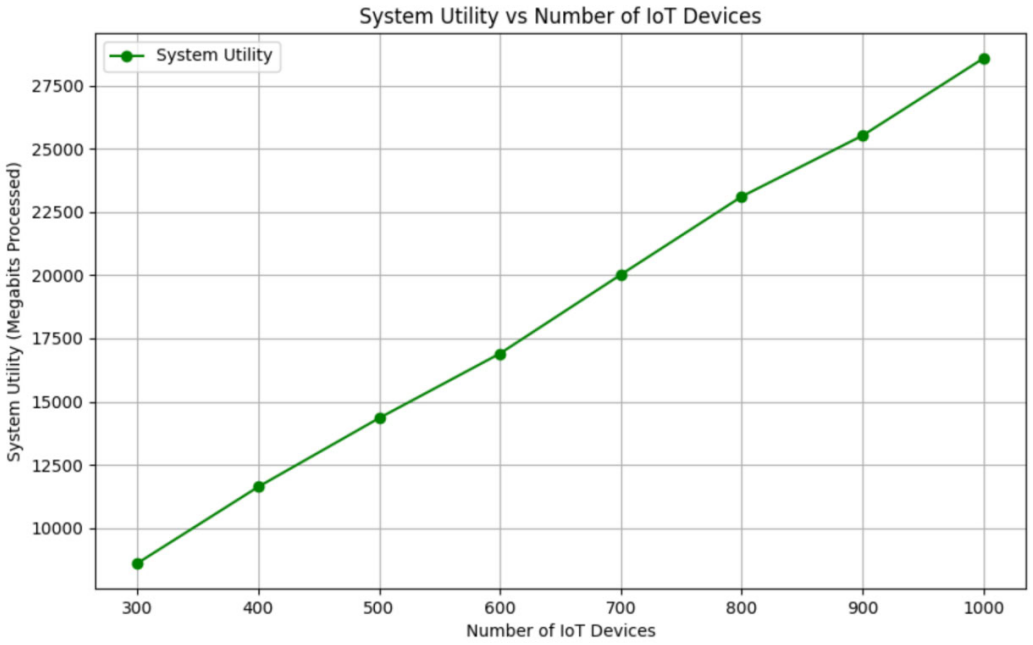
- Energy Consumption Graph



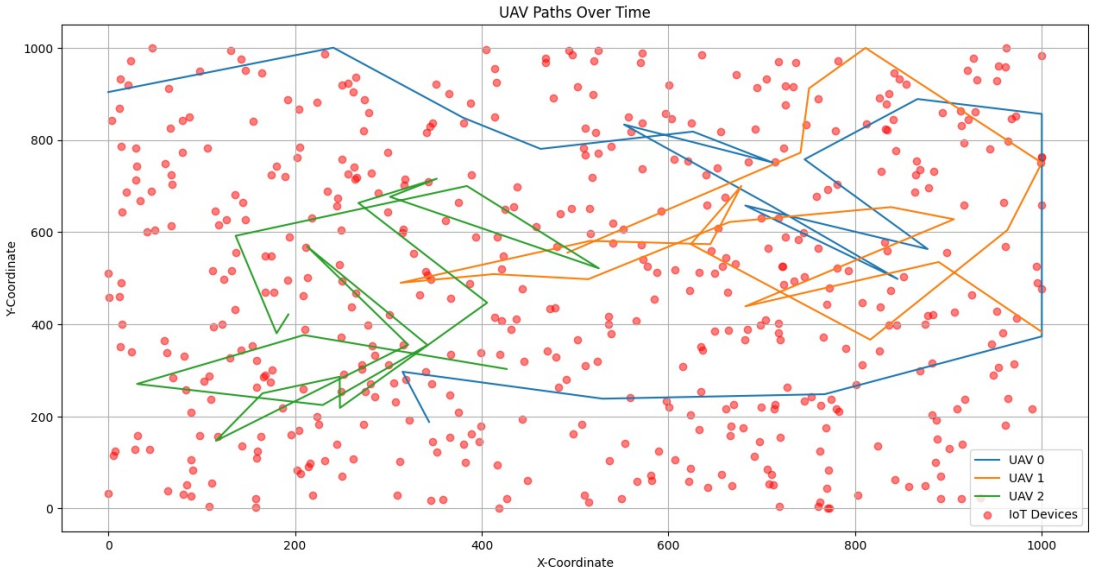
- Energy Efficiency Graph



- System Utility Graph



- UAV Path according to Energy efficient route



Summary of Outputs

The Python simulation provides an in-depth analysis of task offloading, UAV movement, and energy efficiency optimization in a multi-UAV-enabled edge computing system. The key results from the simulation include:

1. UAV and IoT Device Positions

The simulation tracks the dynamic positions of UAVs and IoT devices throughout the simulation. UAV trajectories are optimized to ensure maximum coverage and energy efficiency, adapting to the mobility of IoT devices. The final positions and trajectories are visualized in a 2D plane.

2. Task Offloading Decisions

A binary offloading decision map is generated, showing whether each IoT device's task is processed locally or offloaded to a UAV. This decision is based on energy efficiency calculations, ensuring minimal energy consumption for both IoT devices and UAVs.

3. Energy Efficiency Metrics

The program calculates and displays energy efficiency as the ratio of processed task size to total energy consumption. These metrics provide insights into the system's performance, showing improvements in energy efficiency across simulation time slots.

4. Task Processing Analysis

- **Local vs. Offloaded Tasks:** The simulation categorizes tasks into those processed locally by IoT devices and those offloaded to UAVs, highlighting the resource utilization for each category.
- **Task Efficiency:** Energy efficiency for each task is recorded, distinguishing between locally processed and offloaded tasks.

5. Visualizations

The simulation outputs multiple plots for detailed analysis:

- **Energy Efficiency Over Time Slots:** A line plot showing how energy efficiency evolves during the simulation, reflecting the algorithm's ability to optimize task allocation.
- **Task Processing Efficiency:** A scatter plot of energy efficiency values for local and offloaded tasks, illustrating the impact of task offloading.
- **UAV Trajectories and IoT Positions:** A 2D plot displaying the movement paths of UAVs and the dynamic positions of IoT devices, with coverage circles around each UAV to verify effective service coverage.

6. UAV Flight and Energy Consumption

The simulation calculates the flight energy for each UAV based on its velocity and acceleration. These values, combined with computation and offloading energy, highlight the total energy consumption of UAVs during the simulation.

Overall, these outputs provide a comprehensive evaluation of energy-efficient task offloading and UAV path optimization. The results contribute to better resource management and deployment strategies in multi-UAV-assisted edge computing systems for IoT networks.

Conclusion

The integration of UAV technology with edge computing provides a dynamic and energy-efficient solution for computation offloading in IoT networks. Unlike static infrastructure, UAVs offer mobility and adaptability, enabling them to provide flexible coverage and computational assistance to IoT devices in resource-constrained environments. In this project, we developed a multi-UAV-aided edge computing system aimed at optimizing energy efficiency and task allocation.

The ECOP (Energy-Efficient Computation Offloading and Path Planning) algorithm was central to this effort, addressing the challenges of dynamic IoT environments. By leveraging energy efficiency calculations, the algorithm determined optimal UAV trajectories and task offloading decisions, ensuring balanced resource utilization across UAVs and IoT devices. Through simulation, we demonstrated the algorithm's effectiveness in reducing energy consumption, improving task processing efficiency, and ensuring consistent coverage of IoT devices.

Key results include:

- Substantial improvements in system energy efficiency, with tasks dynamically assigned to UAVs or processed locally based on real-time conditions.
- Optimized UAV positioning, minimizing transmission delays and maximizing IoT device coverage.
- Visual validation of UAV trajectories, energy efficiency trends, and task assignment distributions, confirming the adaptability of the ECOP algorithm.

These findings validate the potential of UAV-aided edge computing systems to enhance the scalability and responsiveness of IoT networks. Moving forward, further advancements in UAV technology, such as predictive mobility models and energy-efficient flight algorithms, can further improve the capabilities of these systems. By integrating renewable energy solutions for UAVs and refining lightweight task scheduling methods, UAV-aided edge computing systems can support the growing demands of IoT applications, paving the way for smarter, sustainable, and more resilient networks.