2023 IEEE 34th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC) | 978-1-6654-6483-3/23/$31.00 ©2023 IEEE | DOI: 10.1109/PIMRC56721.2023.10293844

Lifelong Learning for AoI and Energy Tradeoff Optimization in Satellite-Airborne-Terrestrial Edge Computing Networks

Y. Wu*, B. Lorenzo*, B. Liu+
*Dept. Electrical and Computer Engineering, University of Massachusetts Amherst, USA
+Department of Computer Science, University of Illinois at Chicago, USA
*{yinxuanwu, blorenzo}@umass.edu, +liub@uic.edu

***Abstract*—Satellite-airborne-terrestrial edge computing networks (SATECNs) emerge as a global solution for Internet of Things (IoT) applications in 6G. However, their highly dynamic nature with uncertain varying topology and network traffic makes their management and control more challenging. In this paper, we consider a scenario in which IoT devices, UAVs, and satellites with different edge computing capabilities make decisions to balance the freshness of information and energy consumption. Since SATECNs are highly dynamic and data freshness optimization requires timely decisions, we present a new lifelong learning computing resource allocation algorithm (LL-SATEC) that adapts to the environment by exploiting knowledge transfer between devices in different layers in SATECNs and previous experience. Lifelong learning is a promising machine learning algorithm that learns continuously without requiring a new training phase and avoids catastrophic forgetting. Our goal is to find computing resource allocation policies online for IoT devices, UAVs, and satellites that optimize the overall average age-of-information and energy trade-off. Numerical results show that our approach significantly accelerates learning compared to traditional reinforcement learning algorithms, achieves three times lower AoI and energy consumption in just a few iterations, and avoids catastrophic forgetting.**

***Index Terms*—AoI, edge computing, lifelong learning, satellite-airborne-terrestrial edge computing networks, energy efficiency.**

I. Introduction

SATECNs incorporate multi-tier edge computing to serve many IoT applications, such as industrial IoT, intelligent transportation networks, surveillance, control, and environment monitoring [1]–[3]. In these applications, preserving the freshness of information is crucial for accurate decision-making. Smart IoT devices, UAVs, and satellites have edge computing capabilities and can process data collected from the environment. Since these devices are powered by batteries, the data freshness and energy consumption must be jointly optimized. In terrestrial networks, the data freshness and energy consumption trade-off have been researched actively [4]–[6]. Some of these works [5], [6] focus on the partial integration of UAVs to collect and process IoT data given their advantages in terms of fast deployment, low latency, and processing capacity. In [4], UAV trajectory is optimized to reduce the communication latency and energy consumption of IoT devices. However, there has been little work on edge computing facilitated by SATECNS and its trade-off analysis. In [7], satellites and UAVs are considered for heterogeneous traffic offloading. They present a fixed offloading partitioning to offload ultra-reliable low-latency communications traffic to the UAVs and the terrestrial link. In contrast, enhanced mobile broadband traffic is offloaded to the satellite link since it is less delay-sensitive and needs high data rates.

Machine learning (ML) has been adopted in satellite networks for energy management, anti-jamming, and beam-hopping [8]–[10], among others. Lee et al. [10] jointly optimize the source-satellite-UAV association and the location of UAVs using a Deep Reinforcement Learning (DRL) algorithm. However, one major shortcoming in existing ML algorithms is the lack of capabilities in adapting to dynamic environments [11]. In fact, existing works on AoI optimization in SATECNs or UAV-based systems using ML only work under stationary assumptions [12]. Some progress has been made to make learning models respond faster to dynamic environments by transfer learning [13] but still rely on a training phase and thus suffer catastrophic forgetting. The model must be retrained for every new task, which is time-consuming and inefficient. Therefore, dynamic environments need lifelong learning (LL) to adapt the policies to the new environment, accumulate knowledge, and learn continuously [14]. A few works adopted LL in wireless networks [15], [16]. Zhou et al. [15] present a beamforming adaptation scheme based on LL to optimize the downlink beamforming in a dynamic environment. Gong et al. [16] apply LL to learn resource allocation policies in IoT. Our work extends this model to SATECNs and multi-task learning in environments with heterogeneous dynamics and devices.

In this paper, we present a lifelong learning computing resource allocation algorithm (LL-SATEC) to find the optimum policy that minimizes the AoI and energy cost in a dynamic STECNs. We consider a scenario in which IoT devices, UAVs, and satellites with heterogeneous edge computing capabilities

This work is partially supported by the US National Science Foundation under Grant CNS-2008309 and CNS-2225427.

face different environments and exploit multi-task learning to improve performance and accelerate learning. The knowledge exchange is facilitated using a High Altitude Platform (HAP) that collects data from the devices and updates their computing allocation policy. Our results show that our algorithm can adapt faster to dynamic environments and significantly reduces the AoI and the cost.

II. System Model

In section 2, the authors need to clarify the communication model that is adopted in order to perform the task offloading which currently is missing from the paper.

We consider the SATECN scenario presented in Fig. 1, which consists of satellite, airborne, and terrestrial domains. Each domain has different devices i.e., IoT devices, UAVs, and satellites, that collect and process data to monitor a dynamically changing environment. After processing the data, devices can extract meaningful information and take actions (e.g., sending a notification, activating an alarm, etc.) that are time-sensitive. The data generation frequency and the processing requirements may vary over time due to the dynamics of the environment. Therefore, the data freshness decreases over time unless additional CPU resources are allocated. This brings a tradeoff between the freshness of information and energy consumption. To assist devices in making CPU allocation decisions to adapt to the environment, we assume that they interact with a High Altitude Platform (HAP).

The HAP builds a knowledge base from all collected information regarding the status of the environment, performs lifelong learning, and transmits the new policy back to every device. Examples of applications include Earth monitoring, surveillance, and transport networks in which data is collected at different heights and combined at the satellites to extract useful information.

We assume that IoT devices, UAVs, and satellites generate data packets independently by observing the environment. Each of these devices is referred to as a data source $n_z \in \mathcal{N}_z$ in a domain z, where $z = 1$ is the terrestrial domain, $z = 2$ is airborne domain, and $z = 3$ is the satellite domain. To capture the decisions at each time, we divide the continuous time into discrete time slots of index t. The data packets are modeled as independent and identically distributed (i.i.d.) across discrete time slots and follow an arrival rate with Poisson distribution $\lambda_{n_z,i}$. The data packets generated are of size $d_{n_z}(t)$ and are stored in each source data buffer of size $B^{col}_{n_z}$. The data packets will be scheduled to be processed using their local CPU on a First Come First Served (FCFS) basis. The processing time for a data packet $\psi_{n_z,i} = d_{n_z}/f_{n_z,i}$ is obtained as a function of the data size and computing rate $f_{n_z,i}$ at each source n_z. We quantify the freshness of the processed data using the age of information (AoI) [16], which is defined as the difference between the current time t and the generation time t_{g,n_z} of the latest processed data packet from device n_z, $\Delta_{n_z}(t+1) = t - t_{g,n_z}$. It can be updated as follows

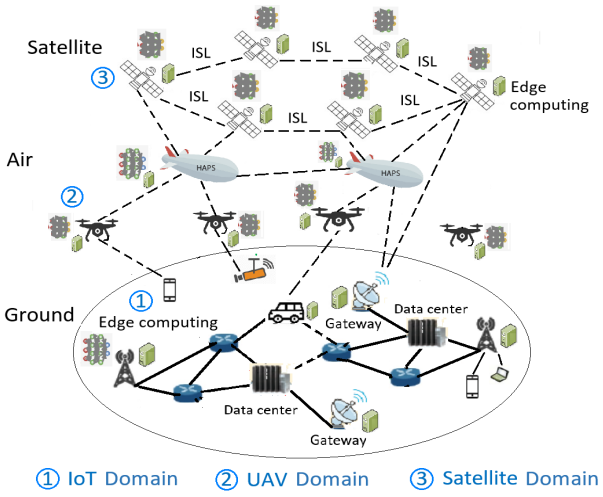


Fig. 1: SATECN scenario.

$$\Delta_{n_z}(t+1) = \begin{cases} \Delta_{n_z}(t) + 1, & \text{if} \quad b_{n_z}(t+1) > 0, \\ \Delta_{n_z}(t+1) - t_{g,n_z}, & \text{if} \quad b_{n_z}(t+1) = 0, \end{cases} \tag{1}$$

where $b_{n_z}(t+1)$ is the number of CPU cycles required to process the remaining data packets at the beginning of time slot $t+1$ obtained as

$$b_{n_z}(t+1) = max\{b_{n_z}(t) + p_{n_z}(t)d_{n_z}(t) - \epsilon_{n_z}(t), 0\}, \tag{2}$$

where $\epsilon_{n_z}(t)$ is the number of CPU cycles allocated at the start of time slot t. The value of $\epsilon_{n_z}(t)$ corresponds to the CPU frequency $f_{n_z,i}(t)$ within time slot t. The energy cost per time slot is $c_{n_z}(t) = \alpha_{n_z}\epsilon^3_{n_z}(t)$, where α_{n_z} denotes the type of chip of node n.

III. Problem Formulation

We characterize the AoI and energy tradeoff of the source nodes in SATECN by a cost function

$$C_{n_z}(t) = \beta\Delta_{n_z}(t) + (1-\beta)c_{n_z}(t), \tag{3}$$

where $\beta \in [0,1]$ is the tradeoff weighting parameter between the first term which is the AoI of source n_z and the second one given by its energy cost.

The overall cost minimization problem is formulated as

$$\begin{aligned} \min_{\Pi} \lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T} \sum_{i=1}^{N} & \beta\Delta_{n_z}(t) + (1-\beta)c_{n_z}(t) \\ \text{s.t.} \quad & (1) - (2), \\ & 0 \le \epsilon_{n_z}(t) \le \epsilon_{\max,n_z}, \quad \forall n_z \in \mathcal{N}_z, \end{aligned} \tag{4}$$

where $\Pi = \{\Pi_1, ..., \Pi_{n_z}, ..., \Pi_{N_z}\}$ is the set of computing allocation policies that indicate the amount of CPU resources allocated per source n_z at each time slot to minimize the cost. Solving (4) using optimization theory is not feasible since the dynamics of the environment are unknown. Therefore, to solve this problem, we adopt lifelong learning and derive a parameterized computing allocation policy.

A. Task Definition

We model the non-stationary environments experienced by source nodes through a series of tasks. We denote by j_{n_z} a task experienced by source n_z in domain z. Each task is a Markov decision process (MDP). We define the tasks per source node n_z as the tuple $\phi_{j_{n_z}} = \{l_{n_z}, \lambda_{j_{n_z}}, d_{j_{n_z}}, \alpha_{j_{n_z}}, \epsilon_{j_{n_z}}\}$, where l_{n_z} defines the domain on which node n_z is located, $\lambda_{j_{n_z}}$ is the data arrival rate (bits/s), $d_{j_{n_z}}$ is the data size, $\alpha_{j_{n_z}}$ is the energy consumption factor per CPU cycle of the device n_z and $\epsilon_{j_{n_z}}$ is the maximum computing capability of the device. We assume that every source n_z experiences i.i.d. environment changes. Source nodes in the same domain z experience the same set of tasks. Hence, in the sequel, we drop the index n for clarity of presentation, and we denote by j_z a task j experienced by any devices in domain z.

B. MDP Framework

Consider $Z = 3$ domains, each device in domain z experiences a sequence of tasks and takes actions to minimize the cost function or penalty. We model the computing resource allocation problem based on the interaction with the environment as a Markov Decision Process (MDP) defined by $\{\mathcal{X}, \mathcal{Y}, \mathcal{P}, T, \gamma\}$, where $\mathcal{X}$ is the set of states, $\mathcal{Y}$ is the set of actions, $\mathcal{P}$ denotes the penalty, T is the transition probability, and γ is the discount factor. Each component is defined as:

1) State: the state space is the set of AoI values and pending CPU data cycles, $\mathcal{X} = \{x_{j_z}(t)\} = \{\Delta_{j_z}(t), b_{j_z}(t)\}$ to compute the remaining data at the beginning of each time slot t.

2) Action: the action space $\mathcal{Y} = \{y_{j_z}(t)\} = \{\epsilon_{j_z}(t)\}$ is the set of possible computation resource allocations, i.e., CPU cycles, under task j for devices in domain z.

3) Penalty: the penalty is the cost function in (3).

The goal of each agent is to find a policy Π that minimizes the penalty. We define parameterized policies $\Pi'_j = \{\pi_{\theta_j} | \theta_j \in \mathbb{R}^d\}$, where $\pi_{\theta_j}\left(y_{j_z}(t) | x_{j_z}(t)\right) = Pr\{y_{j_z}(t) | x_{j_z}(t), \theta_j\}$. Each agent must learn tasks consecutively and maintains an interaction history as a result of its interaction with the environment in the form of trajectories. A task-specific trajectory $\delta_{j_z} = \{x_{j_z}(t), y_{j_z}(t), P\left(x_{j_z}(t), y_{j_z}(t)\right)\}$. The agent ignore the task tuple (except for the parameters related to the device type), and thus, we assume a HAP will collect the trajectories from different devices and infer the tasks from the trajectories. Besides, the agent has no control over the task order. In fact, tasks may be interleaved from the same or different domains, giving the agent the opportunity to revise previous tasks. In addition, the agent ignores the number of tasks, their distributions, and the number of domains, and has no prior information about mapping knowledge between tasks. Therefore, it has to learn how to transfer knowledge between task domains to optimize the performance.

C. Problem reformulation

Based on the previous assumptions, we transform the initial optimization problem to find a parameterized policy that maximizes the expected average return for all tasks M. For simplicity, we assume that j is the index of the task from any domain [14], [16],

$$\max_{\Pi'} \lim_{M\to\infty} \frac{1}{M} \sum_{j=1}^{M} \mathcal{J}\left(\boldsymbol{\theta}_j\right) \quad \text{s.t. } \mathcal{X} \subset \mathbb{R}^d, \mathcal{Y}, \mathcal{P} \subset \mathbb{R} \tag{5}$$

where $\mathcal{J}\left(\theta_j\right) = \int p_{\theta_j}(\delta)\Re_j(\delta)\mathrm{d}\delta$, $M = \sum_{z=1}^{Z} M_z$, Π' is the set of parameterized policies for all tasks, $p_{\theta_j}(\delta)$ represents the probability distribution for trajectory δ and $\Re_j(\delta)$ is the gain for a given trajectory. In other words, we have:

$$p_{\theta_j}(\delta) = P_0\left(x_{j,\delta}(0)\right) \prod_{t=0}^{T} p\left(x_{j,\delta}(t+1) \mid x_{j,\delta}(t), y_{j,\delta}(t)\right) \pi_{\theta_j}\left(y_{j,\delta}(t) \mid x_{j,\delta}(t)\right)$$

$$\Re_{\mathrm{j}}(\delta) = \frac{1}{T} \sum_{t=0}^{T} \gamma_j^{t-1} R\left(x_j(t), y_j(t)\right)$$

where $p\left(x_{j,\delta}(t+1) \mid x_{j,\delta}(t), y_{j,\delta}(t)\right)$ is the unknown state transition probability that maps a state-action pair at time slot t onto a distribution of states at time slot $t+1$ in trajectory δ. We consider using $\mathcal{M}$ to represent the set of observed tasks by the agent. The set $\mathcal{M}$ is composed of subsets $\mathcal{M}_z$, each of which stores tasks originating from the z-th domain.

Conventional RL algorithms can only learn in stationary environments and thus cannot be applied to solve this problem. Consequently, a lifelong reinforcement learning algorithm for SATECNs is proposed to adapt the computation allocation decisions in a dynamic environment.

IV. Lifelong Learning for Computing Resource Allocation

This section describes our LL algorithm for computing resource allocation. To enable agents to overcome catastrophic forgetting and achieve multi-task learning in dynamically changing STECNs, we developed LL-SATEC based on the parameterized lifelong learning policy gradient (PG) method (Natural Actor-critic [17]). PG methods can find the optimal interaction policy for a single task but are inefficient in non-stationary environments. Therefore, LL combines knowledge transfer with PG frameworks.

A. Lifelong Learning Algorithm

In the policy gradient, the action function for each task j, denoted as $y_j(t) = f(x_j(t), \theta_j)$, is characterized by a task-specific parameter vector $\theta_j \in \mathbb{R}^d$, where $x_j(t) \in \mathbb{R}^d$. We assume that the task parameter vector θ_j is given by a linear combination of shared latent model components from the knowledge base maintained at the HAP, $\boldsymbol{\theta}_j = \boldsymbol{L}\boldsymbol{s}_j$. This facilitates knowledge transfer between tasks. In fact, the HAP stores a library of k latent model components $\boldsymbol{L} \in \mathbb{R}^{d\times k}$, which are shared between tasks. Each task parameter vector θ_j can be represented as a linear combination of the columns of $\boldsymbol{L}$, based on the weight vector $\boldsymbol{s}_j \in \mathbb{R}^k$ that characterizes the task. We encourage the weight vectors $\boldsymbol{s_j}$ to be sparse

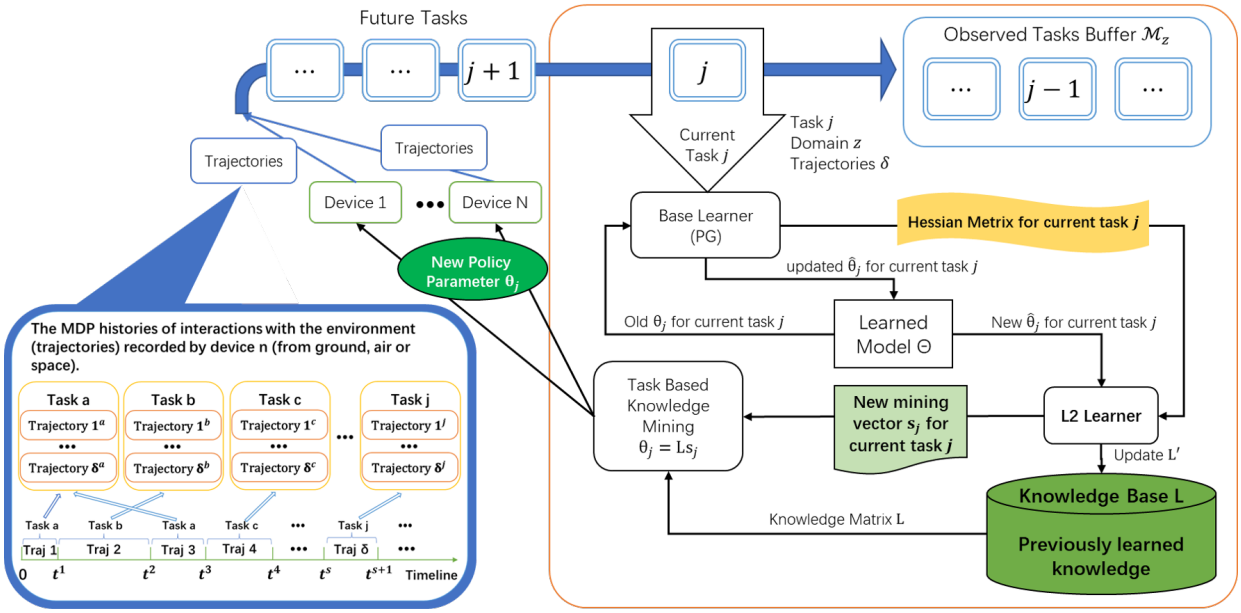


Fig. 2: Implementation Diagram of LL-SATEC Algorithm.

to ensure that each learned model component captures the maximum reusable knowledge chunk.

We use the trajectory data collected for each task as training data to optimize the model. The models are encouraged to share structure while maximizing the average trajectory expectation return for all tasks. The LL loss function is defined as [14], [16]

$$e_{\mathrm{T}}(\boldsymbol{L}) = \frac{1}{M}\sum_{j=1}^{M}\min_{s_j}\left[-\mathcal{J}\left(\boldsymbol{L}\boldsymbol{s}_j\right) + \eta_1\left\|\boldsymbol{s}_j\right\|_1\right] + \eta_2\|\boldsymbol{L}\|_{\mathrm{F}}^2 \tag{6}$$

where η_1 controls the sparsity of $\boldsymbol{s}_j$, the L_1 norm of $\boldsymbol{s_j}$ approximates the true vector sparsity, and $\|\boldsymbol{L}\|_{\mathrm{F}} = \left(\mathrm{tr}\left(\boldsymbol{L}\boldsymbol{L}'\right)\right)^{1/2}$ is the Frobenius norm of the library matrix $\boldsymbol{L}$. To optimize the expected return of M tasks j, we perform averaging by dividing by the number of tasks, denoted as $1/M$.

To solve the previous equation, we need to evaluate all tasks, which is not feasible since tasks need to be learned on the fly. Therefore, we address the explicit dependence on all previous training data (through the inner summation). To improve the efficiency, we approximate the loss using the second-order Taylor expansion of $\frac{1}{M}\sum_{j=1}^{M}\min_{s_j}\left[-\mathcal{J}\left(\boldsymbol{\theta}_j\right)\right]$ around $\boldsymbol{\theta}_j = \hat{\boldsymbol{\theta}}_j$, where $\hat{\boldsymbol{\theta}}_j$ is the local solution of function $\max_{\hat{\boldsymbol{\theta}}_j}\mathcal{J}\left(\boldsymbol{\theta}_j\right)$. In other words, $\hat{\boldsymbol{\theta}}_j$ is an optimal predictor learned only on the trajectories δ_j for task j. By substituting the second-order Taylor expansion, we obtain:

$$\hat{e}_{\mathrm{T}}(\boldsymbol{L}) = \frac{1}{M}\sum_{j=1}^{M}\min_{\boldsymbol{s}_j}\left[\left\|\hat{\boldsymbol{\theta}}_j - \boldsymbol{L}\boldsymbol{s}_j\right\|_{\boldsymbol{H}_j}^2 + \eta_1\left\|\boldsymbol{s}_j\right\|_1\right] + \eta_2\|\boldsymbol{L}\|_{\mathrm{F}}^2 \tag{7}$$

where $\boldsymbol{H}_j$ represents the Hessian matrix [14] and $\hat{\boldsymbol{\theta}}_j$ is the optimal policy parameter that can be obtained using any policy gradient (PG) algorithm that can provide an estimate of the Hessian matrix.

We consider computing each of the $\boldsymbol{s}_j$ parameters when the training data for task j is last encountered, and not updating them when training on other tasks. With the above-mentioned simplification, $\boldsymbol{s}_j$ and $\boldsymbol{L}$ can be updated recursively, as follows:

$$\boldsymbol{s}_j \leftarrow \arg\min_{\boldsymbol{s}_j}\ell\left(\boldsymbol{L}, \boldsymbol{s}_j, \hat{\boldsymbol{\theta}}_j, \boldsymbol{H}_j\right) \tag{8}$$

$$\boldsymbol{L} = \arg\min_{\boldsymbol{L}}\frac{1}{M}\sum_{t=0}^{M}\ell\left(\boldsymbol{L}, \boldsymbol{s}_j, \hat{\boldsymbol{\theta}}_j, \boldsymbol{H}_j\right) + \eta_2\|\boldsymbol{L}\|_{\mathrm{F}}^2 \tag{9}$$

$$\ell\left(\boldsymbol{L}, \boldsymbol{s}_j, \hat{\boldsymbol{\theta}}_j, \boldsymbol{H}_j\right) = \left\|\hat{\boldsymbol{\theta}}_j - \boldsymbol{L}\boldsymbol{s}_i\right\|_{\boldsymbol{H}_j}^2 + \eta_1\left\|\boldsymbol{s}_j\right\|_1 \tag{10}$$

The LL-SATEC algorithm is described in Algorithm 1.

B. LL-SATEC Implementation Steps

The diagram of our LL-SATEC algorithm is presented in Fig. 2. The steps are as follows:

1) Initialization: We use random initialization policies for every device n_z per domain z.

2) Trajectory collection: At the beginning of each time slot, if a device in a domain finds that the environment around it has changed (the device encounters a different task), it sends the computed trajectories from the start of the last change in the environment up to now HAP.

3) Tasks collection: When HAP receives new trajectories, it determines which devices in which domains these trajectories come from and whether these trajectories belong to new tasks

Algorithm 1 Algorithm Lifelong Reinforcement Learning for SATECNs

Require: $\boldsymbol{M} \leftarrow 0$, $\boldsymbol{L} \leftarrow \text{zeros}_{d_{max},\ k}$

1: **while** some Device $\boldsymbol{n}$ from domain $\boldsymbol{z}$ have Trajectories $\boldsymbol{\delta}$ for Tasks $\boldsymbol{j}$ **do**
2: $\boldsymbol{\delta}_j^{n_z} \leftarrow$ DeviceSendTrajectoriesToHAP()
3: Identify task-specific tuple $\boldsymbol{\phi}_{j_{n_z}} = \{l_{n_z}, \lambda_{j_{n_z}}, d_{j_{n_z}}, \alpha_{j_{n_z}}, \epsilon_{j_{n_z}}\}$
4: **if** isNewTask($\boldsymbol{\phi}_{j_{n_z}}$) **then**
5: Identify a new task $\boldsymbol{j}$ for Device $\boldsymbol{n_z}$ in domain $\boldsymbol{z}$
6: Add task $\phi_{j_{n_z}}$ to task buffer $\mathcal{M}_z$:
7: $\mathcal{M}_z \leftarrow \mathcal{M}_z \cup \{\phi_{j_{n_z}}\}$
8: $\boldsymbol{M_z} \leftarrow \boldsymbol{M_z} + \boldsymbol{1}$
9: $\boldsymbol{M} \leftarrow \boldsymbol{M} + \boldsymbol{1}$
10: **else**
11: $\boldsymbol{A} \leftarrow \boldsymbol{A} - \left(\boldsymbol{s}_j \boldsymbol{s}_j^\top\right) \otimes \boldsymbol{H}_j$
12: $\boldsymbol{b} \leftarrow \boldsymbol{b} - \text{vec}\left(\boldsymbol{s}_j^\top \otimes \left(\hat{\boldsymbol{\theta}}_j^\top \boldsymbol{H}_j\right)\right)$
13: **end if**
14: $\left(\hat{\boldsymbol{\theta}}_j, \boldsymbol{H}_j\right) \leftarrow$ HAPsingleTaskLearner$\left(\boldsymbol{\delta}_j^{n_z}\right)$
15: $\boldsymbol{L} \leftarrow \text{reinitializeAllZeroColumns}\left(\boldsymbol{L}\right)$
16: $\boldsymbol{A} \leftarrow \boldsymbol{A} + \left(\boldsymbol{s}_j \boldsymbol{s}_j^\top\right) \otimes \boldsymbol{H}_j$
17: $\boldsymbol{b} \leftarrow \boldsymbol{b} + \text{vec}\left(\boldsymbol{s}_j^\top \otimes \left(\hat{\boldsymbol{\theta}}_j^\top \boldsymbol{H}_j\right)\right)$
18: $\boldsymbol{L} \leftarrow \text{mat}\left(\left(\frac{1}{M}\boldsymbol{A} + \eta_2 \boldsymbol{I}_{d\times k, d\times k}\right)^{-1} \frac{1}{M}\boldsymbol{b}\right)$
19: $\boldsymbol{s}_j \leftarrow \arg\min_{\boldsymbol{s}_j} \ell\left(\boldsymbol{L}, \boldsymbol{s}_j, \hat{\boldsymbol{\theta}}_j, \boldsymbol{H}_j\right)$
20: **if** $\boldsymbol{M} < 30$ **then**
21: $\Pi_j^{n_z} \leftarrow$ HAPSendPolicyToDevice$\left(\hat{\boldsymbol{\theta}}_j\right)$
22: **else**
23: $\Pi_j^{n_z} \leftarrow$ HAPSendPolicyToDevice$(\boldsymbol{\theta}_j = \boldsymbol{L}\boldsymbol{s}_j)$
24: **end if**
25: **end while**

or already observed tasks. If there is a new task, it will store the task data in the corresponding task buffer M_z.

4) Knowledge mining for current task j: The task-specific knowledge for task j contained in the Hessian matrix H_j will be computed by the HAP. We use the Natural Actor-critic as the base learner to compute the policy gradient. One run of the policy gradient update will be performed.

5) Knowledge base refinement: When the trajectories of task j come, the knowledge base will be refined according to steps 16-17 in Algorithm 1. If the task j has been observed before, the outdated knowledge will be deducted before knowledge base refinement according to steps 11-12.

6) Update model: The parameterized LL-SATEC model will be updated as in Eq. (8) and (9).

7) Transmission of updated policy: If LL-SATEC model has observed more than 30 tasks, the HAP will transmit the policy obtained by the LL-SATEC model back to the device. Otherwise the HAP will transmit the local solution $\hat{\boldsymbol{\theta}}_j$ for task j, as outlined in steps 20-24 of Algorithm 1. The device will use it as the current policy until a new policy is received from the HAP.

8) When devices encounter new tasks: If the task buffer $\mathcal{M}$ has accumulated enough observed tasks, LL-SATEC can use the average of the existing task-specific coefficient vector s to generate a new policy parameter vector θ_j from the knowledge base $\boldsymbol{L}$ to solve the new task j and improve the initial performance of the agent in the new task.

V. Numerical Results

We have conducted extensive simulations in Matlab to show the performance of our LL-SATEC algorithm. The simulation parameters are summarized in Table I. For the IoT domain, we have followed the settings in [18]. For the airborne domain, we adopted the settings in [19], and we followed [20] for the satellite domain. The duration of each time slot is 1 second. First, we generated 30 tasks per domain by varying the parameters of the tasks (packet size, packet arrival rate, maximum CPU cycles, power consumption factor of CPU) within the ranges given in Table I. These parameter ranges were chosen to ensure the diversity of tasks per domain and across domains and simulate the dynamic nature of the environment, which is heterogeneous per domain. Then, we randomized the order of task arrivals in an iterative manner and evaluated the capability of LL-SATEC to learn online.

We run the algorithm for 50 trajectories with 50-time steps each to perform the update. We compared the performance of our algorithm to the episodic natural-actor critic as the policy gradient base learner (PG) algorithm. We set the weighted parameter β = 0.5 in the reward function. To simulate the high dynamics of the wireless environment, we consider that the packet arrival rate obeys Poisson distribution when the rate λ_{n_z}.

A. Multi-task Learning per Domain

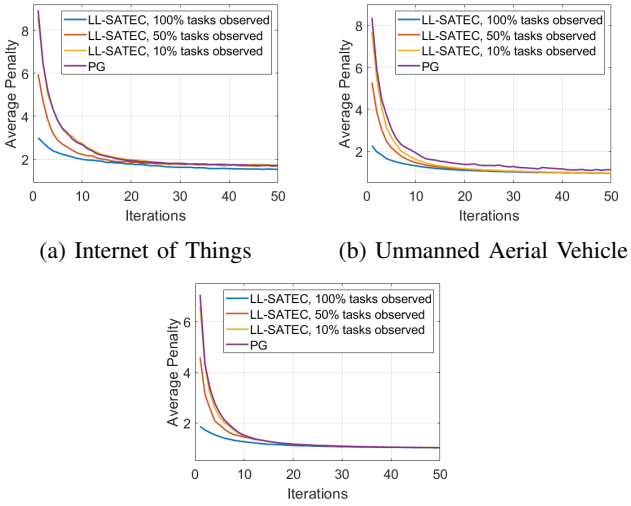
For the IoT domain, we use a CPU frequency between 315MHz and 916MHz. The average number of data packets received by the IoT device at each time slot λ_{n_1} belongs to $[0.1, 5]$, and the mean number of CPU cycles required to process the packets is $[1\times10^6, 1\times10^7]$. For the UAV domain, the CPU frequency is between 1GHz and 5GHz. The average number of data packets received by the UAV device at each time slot λ_{n_2} belongs to $[0.1, 4]$, and the mean number of CPU cycles is $[1\times10^9, 1\times10^{10}]$. For the LEO domain, we use models with a CPU frequency between 5GHz and 10GHz. The average number of data packets received by the LEO device at each time slot λ_{n_3} belongs to $[0.1, 2]$, and the mean number of CPU cycles is $[1\times10^9, 2\times10^{10}]$.

In Figure 3, we compare the performance gap between our proposed LL-SATEC and standard PG in every domains, showing the average performance on all tasks in each domain as a function of the number of learning iterations. We randomly generate 30 tasks for each domain. In each domain, the initialization of the LL-SATEC model was stopped once each LL-SATEC had experienced at least one learning in each of the 30 tasks.

We evaluate the performance of LL-SATEC on all tasks using the knowledge base $\boldsymbol{L}$ learned after observing various

TABLE I: Main parameter settings for simulations

Parameter	Description	Value / Function
ϵ^{bit}	The number of cycles each bit of data requires.	1×10^5 CPU cycles per bit
λ_{n_z}	The average number of packets arriving device n_z in each slot.	$[0.1, 5]$
d_{n_z}	The average size of each new packet arriving device n_z.	$[1 \times 10^1, 2 \times 10^5]$ bits
α_{n_1}	The energy cost factor of chip of the IoT device. (due to the chip type of IoT)	$[1 \times 10^{-21}, 2 \times 10^{-21}]$ J/cycles^3
ϵ_{n_1}	The maximum computing capability of the IoT device n_1 in each slot.	$[3 \times 10^6, 8 \times 10^6]$ cycles per slot
$E_{n_1}^{exe}$	The energy cost of the IoT device n_1.	$\alpha_{n_1}\epsilon_{n_1}^3$
α_{n_2}	The energy cost factor of chip of the UAV. (due to the chip type of the UAV)	$[1 \times 10^{-27}, 2 \times 10^{-27}]$ J/cycles^3
ϵ_{n_2}	The number of CPU cycles allocated by the UAV n_2 in each slot.	$[1 \times 10^9, 5 \times 10^9]$ cycles per slot
$E_{n_1}^{exe,n_2}$	The energy cost of UAV to serve the IoT device n_1.	$\alpha_{n_2}\epsilon_{n_2}^3$
α_{n_3}	The energy cost factor of chip of the satellite. (due to the chip type of the satellite)	$[1 \times 10^{-28}, 2 \times 10^{-28}]$ J/cycles^3
ϵ_{n_3}	The number of CPU cycles allocated by the satellite n_3 in each slot.	$[5 \times 10^9, 1 \times 10^{10}]$ cycles per slot
$E_{n_1}^{exe,n_3}$	The energy cost of satellite to serve the IoT device n_1.	$\alpha_{n_3}\epsilon_{n_3}^3$



(a) Internet of Things (b) Unmanned Aerial Vehicle

(c) Low Earth Orbit Satellites

Fig. 3: The performance of LL-SATEC versus standard PG per domain in SATEC.

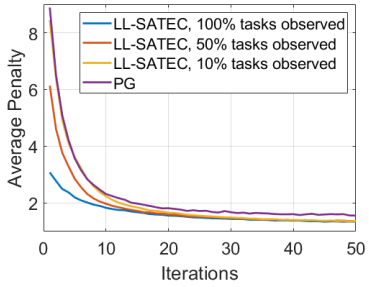


Fig. 4: The performance of LL-SATEC versus standard PG across domains in SATEC.

subsets of tasks, i.e., three tasks (10%) to observing all 30 tasks (100%). These experiments evaluate the quality of the learned knowledge base $\boldsymbol{L}$ on both known and unknown tasks, showing that performance improves as LL-SATEC learns more tasks. When a task has not been observed, the nearest task coefficient is used, and its initialization $\boldsymbol{s}_j$ use a zero vector.

The initial performance and convergence speed of LL-SATEC outperforms standard PG in all task domains. In the domains of IoT and UAVs, the final performance of LL-SATEC is even better than that of standard PG. In fact, we have checked that our algorithm achieves the global optimum, while the standard PG achieves a local optimum in these cases. Our algorithm also accelerates the learning of tasks, observed by faster convergence, by transferring knowledge from other tasks.

B. Cross-Domain Learning

We study the performance of cross-domain learning by letting our algorithm face tasks from different domains simultaneously. Since the computational resources of the devices at each domain are different, and they experience different dynamics, the tasks in this setting are more heterogeneous.

We have generated 30 different tasks randomly from each domain, with each domain generating 10 tasks initially. The initialization of the model ceases once LL-SATEC has experienced at least one learning instance in each of these 30 tasks from the three domains.

As shown in Fig. 4, we can observe that the initial performance, convergence speed, and final performance of LL-SATEC when facing random tasks from the three domains is still superior to the standard PG, and achieves comparable performance as when learning independently per domain.

C. Online Learning Across Domains

We compare the performance gap between our proposed LL-SATEC and the standard PG in online learning of multiple tasks from any of the three domains. We randomly generate 20 tasks for each domain, resulting in a total of 60 tasks. Then we randomly divide these 60 tasks into two halves, with 30 tasks used to initialize the LL-SATEC model. The initialization of the LL-SATEC model ceases once it has experienced at least one learning instance in each of these 30 tasks as before. Subsequently, we let both LL-SATEC and the standard PG encounter the remaining 30 new tasks simultaneously and independently during each iteration. Each task will last about

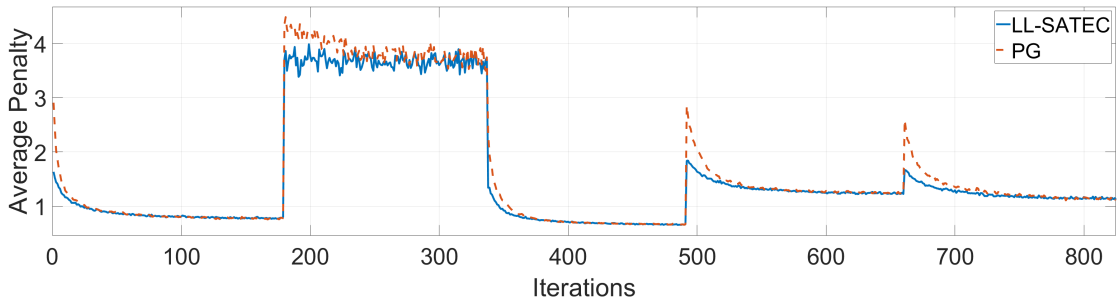


Fig. 5: The performance of LL-SATEC versus standard PG, randomly encountering different new tasks from all three domains.

150 iterations and will not be reencountered by the intelligent agent until all 30 tasks have been encountered once.

In Fig. 5, we compare LL-SATEC and the standard PG when encountering random new tasks every 180 iterations. When LL-SATEC encounters a task it has not observed before, the nearest task coefficient is used, and its initialization $\boldsymbol{s_j}$ is the average value of $\boldsymbol{s_{j'}}$ of observed tasks. As shown in Fig. 5, LL-SATEC accelerates the learning of new tasks by leveraging experience from previous tasks, exhibiting faster initial performance and convergence than the standard PG on all new tasks and avoiding catastrophic forgetting. Our findings show that the average penalty of LL on all tasks is lower, and the convergence speed is faster than the standard PG.

VI. Conclusion

In this paper, we study computing resource allocation policies to optimize the overall average age-of-information and energy efficiency trade-off in SATECNs. To solve this problem in a dynamic environment, we present a lifelong learning algorithm (LL-SATEC) for computation resource allocation in IoT devices, UAVs, and satellites. Our algorithm is able to adapt to heterogeneous network dynamics and learn new tasks online by exploiting multi-task learning per domain and across domains. Numerical results show that LL-SATEC reduces the AoI and energy cost by 3 times compared with conventional reinforcement learning algorithms and avoids catastrophic forgetting.

References

[1] B. Wang, Y. Sun, Z. Sun, L. D. Nguyen, and T. Q. Duong, "Uav-assisted emergency communications in social iot: A dynamic hypergraph coloring approach," *IEEE Internet of Things Journal*, vol. 7, no. 8, pp. 7663–7677, 2020.

[2] H. H. Esmat, B. Lorenzo, and W. Shi, "Towards resilient network slicing for satellite-terrestrial edge computing iot," *IEEE Internet of Things Journal*, pp. 1–1, 2023.

[3] A. Kak and I. F. Akyildiz, "Designing large-scale constellations for the internet of space things with cubesats," *IEEE Internet of Things Journal*, vol. 8, no. 3, pp. 1749–1768, 2021.

[4] H. Lee, S. Eom, J. Park, and I. Lee, "Uav-aided secure communications with cooperative jamming," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 10, pp. 9385–9392, 2018.

[5] L. Liu, A. Wang, G. Sun, and J. Li, "Multi-objective optimization for improving throughput and energy efficiency in uav-enabled iot," *IEEE Internet of Things Journal*, pp. 1–1, 2022.

[6] S. Eom, H. Lee, J. Park, and I. Lee, "Uav-aided wireless communication designs with propulsion energy limitations," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 1, pp. 651–662, 2020.

[7] L. Zhang, W. Abderrahim, and B. Shihada, "Heterogeneous traffic offloading in space-air-ground integrated networks," *IEEE Access*, vol. 9, pp. 165 462–165 475, 2021.

[8] S. G. Glisic and B. Lorenzo, *Artificial Intelligence and Quantum Computing for Advanced Wireless Networks*. John Wiley & Sons, 2022.

[9] N. Kussul, M. Lavreniuk, S. Skakun, and A. Shelestov, "Deep learning classification of land cover and crop types using remote sensing data," *IEEE Geoscience and Remote Sensing Letters*, vol. 14, no. 5, pp. 778–782, 2017.

[10] J.-H. Lee, J. Park, M. Bennis, and Y.-C. Ko, "Integrating leo satellite and uav relaying via reinforcement learning for non-terrestrial networks," in *GLOBECOM 2020 - 2020 IEEE Global Communications Conference*, 2020, pp. 1–6.

[11] L. Lei, Y. Yuan, T. X. Vu, S. Chatzinotas, M. Minardi, and J. F. M. Montoya, "Dynamic-adaptive ai solutions for network slicing management in satellite-integrated b5g systems," *IEEE Network*, vol. 35, no. 6, pp. 91–97, 2021.

[12] Z. Zhu, S. Wan, P. Fan, and K. B. Letaief, "Federated multiagent actor–critic learning for age sensitive mobile-edge computing," *IEEE Internet of Things Journal*, vol. 9, no. 2, pp. 1053–1067, 2022.

[13] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[14] H. B. Ammar, E. Eaton, P. Ruvolo, and M. Taylor, "Online multi-task learning for policy gradient methods," in *International conference on machine learning*. PMLR, 2014, pp. 1206–1214.

[15] H. Zhou, W. Xia, H. Zhao, J. Zhang, Y. Ni, and H. Zhu, "Continual learning-based fast beamforming adaptation in downlink miso systems," *IEEE Wireless Communications Letters*, vol. 12, no. 1, pp. 36–39, 2023.

[16] Z. Gong, Q. Cui, C. Chaccour, B. Zhou, M. Chen, and W. Saad, "Lifelong learning for minimizing age of information in internet of things networks," in *ICC 2021 - IEEE International Conference on Communications*, 2021, pp. 1–6.

[17] J. Peters and S. Schaal, "Natural actor-critic," *Neurocomputing*, vol. 71, no. 7-9, pp. 1180–1190, 2008.

[18] Z. Gong, Q. Cui, C. Chaccour, B. Zhou, M. Chen, and W. Saad, "Lifelong learning for minimizing age of information in internet of things networks," in *ICC 2021-IEEE International Conference on Communications*. IEEE, 2021, pp. 1–6.

[19] T. Zhang, Y. Xu, J. Loo, D. Yang, and L. Xiao, "Joint computation and communication design for uav-assisted mobile edge computing in iot," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 8, pp. 5505–5516, 2019.

[20] C. Ding, J.-B. Wang, H. Zhang, M. Lin, and G. Y. Li, "Joint optimization of transmission and computation resources for satellite and high altitude platform assisted edge computing," *IEEE Transactions on Wireless Communications*, vol. 21, no. 2, pp. 1362–1377, 2021.