MARbLE: Multi-Agent Reinforcement Learning at the Edge for Digital Agriculture

Conference Paper · December 2022

DOI: 10.1109/SEC54971.2022.00013

6 authors, including:

Jayson Boubin
Binghamton University
25 PUBLICATIONS **188** CITATIONS

SEE PROFILE

Christopher Stewart
The Ohio State University
98 PUBLICATIONS **1,481** CITATIONS

SEE PROFILE

MARbLE: Multi-Agent Reinforcement Learning at the Edge for Digital Agriculture

Jayson Boubin[1,*], Codi Burley[2], Peida Han[2], Bowen Li[2], Barry Porter[3], Christopher Stewart[2]
Department of Computer Science, Binghamton University[1]
Department of Computer Science and Engineering, Ohio State University[2]
School of Computing and Communications, Lancaster University, UK[3]

***Abstract*—Digital agriculture, hailed as the fourth great agricultural revolution, employs software-driven autonomous agents for in-field crop management. Edge computing resources deployed near crop fields support autonomous agents with substantial computational needs for tasks such as AI inference. In large fields, using multiple autonomous agents, called swarms, can speed up crop management tasks if sufficient edge resources are provisioned. However, to use swarms today, farmers and software developers craft their own standalone solutions that are either simple and ineffective or complicated and hard-to-reproduce. We present MARbLE, a platform for developing and managing swarms. MARbLE provides an easy-to-use programming paradigm that helps users build swarm workloads using multi-agent reinforcement learning. Developers supply just two functions *Map()* and *Eval()*. The platform automatically compiles and deploys swarms and continuously updates the reinforcement learning models that govern their actions. Developers can experiment with multiple swarm and edge resource configurations both in simulation and with actual in-field runs. We studied real UAV swarms conducting digital agriculture missions. We observe that swarms demanded edge computing resources in bursts; the ratio of average to peak demand was 2.9X. MARbLE uses energy-saving load balancing policies to duty cycle machines during workload demand troughs, leveraging workload patterns to save edge energy. Using MARbLE, we found that four-agent swarms with load balancing techniques sped up missions by 2.1X and reduced edge energy usage by up to 2X compared to state of the art autonomous swarms.**

I. Introduction

Agriculture is one of humanity's most important endeavors. Throughout three agricultural revolutions, humans have harnessed our greatest technological achievements, from mechanization to generics, to simplify and scale agriculture, decrease costs, increase yield, and meet quality of life standards [2]. However, agricultural practices must advance greatly in the coming decades to sustain us. First, population growth and increased food consumption per capita are projected to necessitate at least a 70% increase in agricultural yields by 2050 [20], [18]. Second, climate change is making farming increasingly difficult by contributing to crop health stressors, e.g., drought, disease, and pest infestations [27]. These effects are expected to decrease crop yield by up to 11% by 2050 [39]. Digital agriculture, the cornerstone of the fourth agricultural revolution, seeks to surmount these challenges [2].

Digital agriculture [26] uses remote sensors (e.g., satellites and UAV), in-field sensors (e.g., embedded soil sensors), and data processing techniques (e.g., machine learning) to inform planting, harvest, and crop treatment in ways that maximize crop yield and minimize the environmental impacts of agriculture. Frequent sensing can detect crop health stress from drought and heat [11], identify diseases [60] and pests [49], and find other harmful phenomena [62]. One common task in digital agriculture is to transform sensed data into *health maps* that provide a geo-spatial characterization of crop health and guide crop treatment. The process of creating health maps is called *crop scouting* [16].

Unmanned Aerial Vehicles (UAV) are a potent technology in digital agriculture. UAV are fast, responsive, and maneuverable sensors that noninvasively sense crop health. In practice today, when UAV are used for crop scouting, they are piloted via remote control to collect images manually. Then, after capturing images of the whole field, a health map is created offline [24], [3]. To be sure, remote control in the context can involve manipulating a joystick or setting GPS waypoints in a smartphone. Both approaches exhaustively cover waypoints for the whole field during runtime. Recent advances in edge computing have made it possible for UAV to map fields in real-time [56], [61], [28], [34]. UAV can transmit data to near-by edge resources to make real-time decisions about flight paths, mapping, and treatment. This trend has allowed UAV to operate as *autonomous agents (AA)*, making high-level decisions with no human interaction. Autonomous UAV leverage edge hardware to model crop health at runtime and avoid visiting redundant and useless waypoints which signficantly speeds up crop scouting.

Groups of AA working toward a common goal are called *swarms* [48]. Compared to AA working alone, swarms can speed up crop-scouting missions. First, missions can be partitioned into tasks that swarm members execute in parallel. Second, swarm members can share observations of their surroundings to help other members take effective actions [58]. In agriculture, fields are vast, crop health varies signficantly within and between fields, and crop stressors change over time. Limited by their batteries, a single UAV can not explore a whole field on one charge and recharging batteries is time consuming. Swarms can help greatly by allowing multiple

UAV to explore a field in parallel.

Swarms can be realized by partitioning waypoints into regions that are assigned to each swarm member, collecting data from the regions in parallel and computing health maps offline [3]. Recent research provides automated partitioning and fault tolerance for such automated swarms [21]. However, pre-programmed behaviors fundamentally waste resources by collecting and processing data of low value relative to the crop-scouting mission. Further, swarm members can not share data to improve results.

Multi-Agent Reinforcement Learning (MARL) is an emerging class of learning algorithms where agents cooperate to maximize a reward [58]. Agents learn their own reinforcement learning policies, but they can also learn from the actions and outcomes of other agents. MARL algorithms applied to AA swarms can speed up missions via partitioning (like the automated approach above) and via efficacy (i.e., taking better actions). Further, recent research on MARL algorithms provides provable guarantees and strong empirical results [32], [8], [57], [10]. However, MARL systems are not simple to develop and manage; they require infrastructure for AA workflows, selection of MARL algorithms and reward functions, and data management policies. Developers must create this infrastructure by hand and incorporate it into a real-world system. The result is that, despite their potential, these algorithms rarely go beyond theoretical studies or highly specialised applications with bespoke components.

This complexity is compounded further by deployment concerns. AA who rely on MARL for decision making often do not have the onboard compute power to extract features and make decisions alone, necessitating offloading [41]. Agricultural areas are rarely provisioned with networks necessary for low-latency cloud communication. For these reasons, edge resources are critical for MARL swarms. Edge resources, however, come with management concerns. Remote edge deployments must use power sparingly to maximize mission goals. Thus, an end-to-end solution for MARL swarms should provide: 1) a programming mechanism for training MARL models and testing swarm configurations to meet user-defined goals, and 2) a power-aware and scalable deployment platform for edge systems.

We present *MARbLE*, a platform for developing and managing MARL-driven swarms. To create a MARL specification in MARbLE, developers provide two functions: *Map()* and *Eval()*. Map() converts sensed observations (e.g., images) to application-specific feature vectors. Eval() evaluates system performance towards autonomy goals. With these two functions and configuration information, MARbLE can be used to quickly deploy systems in the field. MARbLE can also test swarm configurations in simulation, allowing users to explore the performance tradeoffs of configuration settings before committing to costly edge deployments.

To support its novel programming paradigm, MARbLE provides end-to-end control, deployment, and scheduling of an entire swarm-support infrastructure – including UAV and heterogeneous edge computing resources. After sensing data at a waypoint, AA require substantial compute resources to compute crop health at runtime, but these demands dissipate as AA move between waypoints. MARbLE scales resources as needed, turning off edge devices during demand troughs to save power and efficiently bin-packing for energy-efficient load balancing.

We used MARbLE to build a real MARL swarm of UAV to predict crop health. We deployed our swarm on an 85-acre Ohio soybean field for one month, conducting over 150 crop-scouting missions. With MARbLE, we explored 45 custom configurations which ultimately resulted in a 31% agricultural profit increase over an automated swarm implementation. Our MARL swarm outperformed competing approaches, improving mapping times by 2.1X while using 2X less edge energy. With MARbLE, we were able to easily scale up edge compute resources to support multiple AA, allowing AA to learn from each other online while dynamically allocating resources to fit demand and save precious power at the edge.

MARbLE makes three contributions:

1. MARbLE allows users to swap out the building blocks of MARL to develop and deploy a myriad of swarm configurations. Without MARbLE, policies to govern each agent's actions must be handcrafted.
2. MARbLE demonstrates that MARL swarms can improve performance via efficacy; Agents can take more effective actions by sharing data with each other and updating their policies via online learning.
3. MARbLE counteracts increased resource utilization of MARL swarms by leveraging their resource usage patterns to duty cycle edge devices.

The remainder of the paper is organized as follows: Section II provides background information on UAV swarm workloads in agriculture. Section III details MARbLE's high level design and programming model, which allows users to easily build MARL swarms. Section IV discusses MARbLE's runtime, which includes a cluster autoscaling mechanism which saves edge power without sacrificing performance and a priority-based online learning approach. Section V covers our implementation of MARbLE to map crop health using a UAV swarm deployment and the evaluation of 45 potential deployment configurations. Section VI evaluates the performance of MARbLE compared to prior automated and autonomous swarm approaches. Section VII discusses the limitations of MARbLE. Section VIII presents related work and Section IX provides conclusions.

II. Background: Swarm Workloads

We studied three types of AA swarms in digital agriculture, as shown in Figure 1. Automated swarms [50] are the most widely used approach for UAV swarms today. Automated

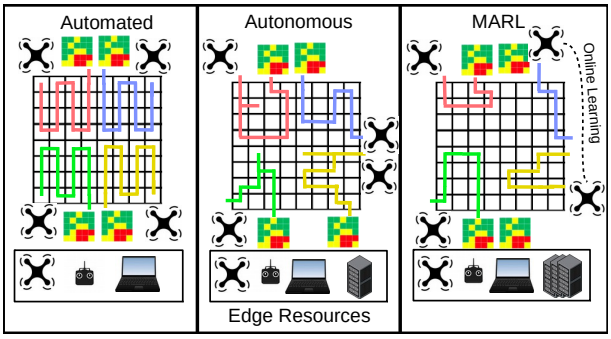

Fig. 1. Types of swarms: Automated, Autonomous, and MARL. Each agent explores its environment from a starting point, senses data, and, when its mission is completed, produces a report. This figure depicts four UAVs scouting a crop field and producing a health map. Depending on the type of swarm, agents visit fewer waypoints, require more edge resources, and communicate more frequently to complete missions faster.

swarms partition an environment into equal sections and sense those sections in their entirety. Environments are represented as sets of GPS waypoints. UAV fly to each waypoint and capture images, video, or other data in sequence. Software packages used for automated systems can be ground control stations [1], onboard control platforms and software development kits [35] or more complicated robotics control platforms [43]. Automated swarms require limited resources on the ground, but produce longer missions, taking days to weeks to scout entire crop fields.

A second way to implement UAV swarms replaces automated GPS routes with AA that creatively sample fields using reinforcement learning (RL) [61]. Researchers can train machine learning algorithms using field data to identify crop diseases, pests, and stressors online. This data can be then used by RL algorithms to determine the best regions to search to properly map phenomena while eschewing regions that are healthy or irrelevant. Finally, unexplored regions can be predicted from nearby values, presenting a sufficiently accurate map in significantly less time. Autonomous agents do, however, require edge resources. UAV often do not have the processing power to analyze observations in real-time and make RL decisions. This makes autonomous deployments more difficult, requiring optimal hardware, software, and model configuration to assure efficient mission execution at the edge [7]. Autonomous swarms act as parallel sets of autonomous collaborating agents. These agents, however, do not communicate with one another. They act alone in their own partitioned environments.

MARL swarms differ from autonomous swarms in that they collaborate directly, using one or more MARL algorithms designed to maximize a global goal as opposed to individual local goals. MARL swarm members share an initial reinforcement learning model that changes over time as observed environments diverge from the original training data. Data sharing and online learning help to increase

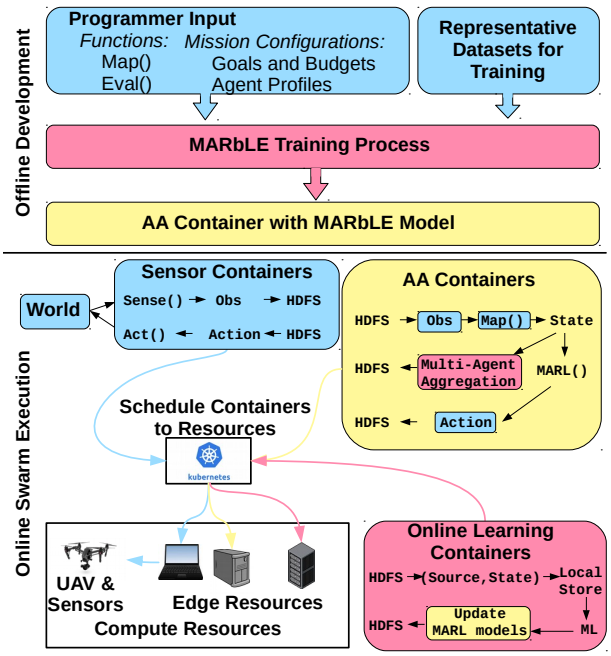

Fig. 2. The architecture of MARbLE. Programmers input mission configurations and Map() and Eval() functions, and MARbLE compiles a MARL algorithm. MARbLE then deploys MARL apps as containers across an edge cluster.

map accuracy and goal performance. These improvements, however, come with a cost. MARL swarms have even greater resource management concerns than autonomous swarms. They add data sharing and retraining workloads beyond those needed for general autonomous flight. MARbLE is designed specifically to simplify the building process for MARL swarms and to manage their deployments and additional resource needs. However, we note that MARL swarms subsume automated and autonomous swarms. MARbLE can be used to build any of these types of swarms.

III. Design

As shown in Figure 2, MARbLE is an end-to-end platform for autonomous swarms, covering the development of AA, their workflow and coordination, and their execution on edge computing devices.

To create a swarm, developers implement two functions, Map() and Eval(), and specify a mission configuration. Map() functions convert quantized inputs from sensing devices (e.g., cameras, GPS, etc.) into a feature vector that represents the state of the swarm. Eval() functions aggregate all outputs from Map() invocations during an epoch and assess the extent to which the mission has been completed. The mission configuration defines key parameters that developers can adjust across swarm applications. Figure 2 depicts three mission configuration settings. First, developers can provide thresholds to determine when missions are complete or

uncompletable, known as goals and budgets. Goals determine if a mission has successfully completed (e.g an agent has exited a maze), budgets determine whether a mission can be completed given current resources (e.g the agent has run out of battery in the maze). MARbLE also uses sensor profiles [7] which specify the amount of each budget that is consumed by a given MARL action (e.g moving to a new state, sensing data). Using budgets and profiles, MARbLE draws a novel link between RL training and resource management at compile time.

MARbLE compiles these inputs to create swarm workflows for sensing surroundings and taking actions. Here, the challenge is to decide which actions to take after running Map() and Eval(). MARbLE automatically builds MARL models by (1) replaying data from prior execution contexts, and (2) learning effective actions that improve Eval() outcomes.

MARbLE models MARL-driven swarms as three asynchronous components: Sensors, AA Workflow, and Online Learning. These components execute in shared-nothing containers connected via distributed storage (e.g., HDFS [46]). Containers are replicated to support swarms. Distributed storage holds model information and aggregates data to update swarm member's models in execution. MARbLE deploys all sensor, agent, and online learning containers, and support software across an edge cluster in-situ. To minimize resource consumption, MARbLE right-sizes via automatic duty-cycling of unused compute and employs a novel, priority-based online learning and scheduling mechanism to manage compute demand.

In this section, we first provide a rigorous primer on MARL algorithms. Then, we introduce the specification of MARbLE applications, i.e., swarms of AA. Finally, we describe each of the key functions and models listed above. In Section IV, we describe the MARbLE runtime and management system.

A. MARL Primer

Broadly, reinforcement learning approaches determine a policy π^* that approximates an optimal solution to a Markov Decision Process (MDP). MDPs comprise States S, Actions A, a transition probability function $P \rightarrow SxA \rightarrow \Delta(S)$, a reward function $R(s_i, a_i, s_{i+1})$ defining the immediate reward an agent receives from performing an action, and a discount factor γ [58]. A policy π is a means of determining which action to take in a given state to transition to another state. The optimal policy π^* is the policy that results in the set of state transitions that maximizes overall reward.

$$MDP : (S, A, P, R, \gamma) \tag{1}$$

MDPs are a useful tool for solving simple state transition problems, but MARL missions are too complex for this framework. Crop scouting is a complex workload where the execution context changes over time and reward is difficult to define. If the system is rewarded based on the estimated yield it predicts in each state, it will likely over or under-predict yield based on how reward is assigned. Furthermore, even if reward is properly assigned, it is likely that states and transition probabilities will change over time as crops grow and conditions change. In many cases, an optimal policy is nearly impossible to determine a priori. For these tasks, reinforcement learning is used to develop $\pi^{*\prime} \approx \pi^*$, an approximately optimal policy for navigating execution contexts while maximizing reward.

There are many methods for approximating π^* through reinforcement learning including value based methods like Q-learning, policy based methods like actor-critic RL, and analogous deep methods like Deep Q-Networks and Deep Deterministic Policy Gradients [54], [19], [38], [31]. Throughout this paper, we will use Q-learning as a basis for MARL, but other techniques fit into our programming model as well.

Q-learning is a value-based method for reinforcement learning which uses a Q-function to determine $\pi^{*\prime}$. $Q(S_i, A_i)$ is the Q-function which predicts the expected reward (Q-value) of an action A_i taken at state S_i. In practice, Q-values are stored in a Q-table $Q[S, A]$ indexed by state-action pairs. When an action is performed, the Q-table is updated using the bellman equation shown in equation 2, which uses dynamic programming to update the Q-value for a state-action pair based on the reward for a given action, plus expected reward of all future actions modified by learning rate α and discount factor γ. Properly informed Q-tables and other RL mechanisms solve MDPs with high reward by learning $\pi^{*\prime}$ through a combination of exploration (taking random actions and learning from results) and exploitation (taking predicted high-quality actions and learning from results).

$$\begin{aligned} Q(s_i, a_i) = (1 - \alpha) * Q(s_i, a_i) + \\ \alpha[R(s_i, a_i, s_{i+1}) * \gamma max(Q(s_{i+1}, a_{i+1}))] \end{aligned} \tag{2}$$

This process translates quite well to multi-agent systems. Transitioning a reinforcement learning algorithm to a multi-agent domain can involve constructing careful global reward functions [9]. Another approach, team-average reward [22], [13], maximizes the reward received by the system given agents with different and potentially discordant reward functions. MARL algorithms of this type are called Markov Games (MGs) [33]. MGs, shown in equation 3, expand the MDP by adding multiple agents, defined by $N \geq 1$. Each agent $i \in N$ has its own action set A_i and reward function R_i. Similar to MDPs, the solution to the MG is policy π^*, the set of state transitions that maximizes reward. Much work has also been done with networked agents [59], [42], [57], agents within a MG that communicate over some time-varying network, may have individual reward functions, and may require data privacy.

$$MG = (N, S, A^i_{i \in N}, P, R^i_{i \in N}, \gamma) \tag{3}$$

Given this specification, MARbLE should accommodate different MARL algorithms using the same base components while assuring that these algorithms fit within the MARbLE framework. To allow the design and deployment of MARL algorithms for real-world AA, MARbLE takes some of these base MARL components as inputs and generates others offline.

B. The MARbLE Spec

$$MARbLE\ Spec : (N, S, A^i_{i \in N}, Map(), Eval(), \mathbb{C}) \quad (4)$$

Equation 4 presents the minimum specification for MARbLE applications, called the MARbLE Spec. Similar to a Markov Game, the MARbLE Spec accepts a number of agents $N \geq 1$, states S, and action sets A_i for each agent. States are non-injective and surjective mappings from action sequences to integers $< a^t_{t=0}, a^t_{t=1} ... a^t_{t=T} > \rightarrow \mathbb{Z}$ where $a^t \in A^i_{i \in N}$. States affect the behavior of action drivers. For example, if a vehicle is at the eastern edge of allowed states, the command *go East* is muted by the action driver.

Unlike Markov Games, the MARbLE Spec eschews state transition probability and reward functions. First, MARbLE developers can reuse action drivers created by others. The action drivers may support states about which the developers are unaware, making state transition models incomplete. Second, constructing mathematical reward functions is challenging. Real-world swarms take on missions that involve complex, domain-specific knowledge. The value of their actions can be subtle and may depend on prior actions, requiring complex non-linear reward functions that overly complicate the development of AS.

Instead, MARbLE simplifies reward engineering through Map() and Eval() functions. The Map() and Eval() functions, coupled with representative observations from prior missions $\mathbb{C}$ suffice to compile initial MARL models which consider goal performance as well as edge resource consumption. The remainder of this section details the training process.

C. Map() and Eval() Functions

Like in MapReduce [12], Map() functions in MARbLE structure input data. AA get their input data from sensor containers. The output is a feature vector, called a state-space vector (SSV), that describes sensed observations in the system's current state. Precisely, let D_j be the sensor data observed in state S_i, $Map(D_j)$ directly translates observed sensor data to a SSV, as shown below.

$$D =< d_1, d_2, ...d_n > \quad (5)$$

$$Map(D) = SSV =< f_1, f_2, ...f_m > \quad (6)$$

By emitting a structured SSV, Map() functions in MARbLE can compose multiple *extractor functions* that process a portion of the sensed data $D' \subset D$ and emit part of the SSV. Extractor functions are shown in Figure 3 for crop scouting.

```
1  func[] extractrs = [ExG(), LAI(), DefoNet()]
2  float[] Map(Obj data) {
3    float[] SSV = [];
4    for(e in extractrs) {
5      SSV.append(e(data));
6    }
7    return SSV;
8  }
9
10 Obj[] Eval(float[][] fs, float[][] budgets,
       Float[][] goals) {
11   goalPerf, finished = GoalPerf(fs, goals)
12   budgetPerf, overrun = BudgetPerf(fs, budg)
13   loss = goalPerf * budgetPerf
14
15   bool done = (finished || overrun)
16   return [loss, done]
```

Fig. 3. Map() and Eval() function pseudocode for crop scouting.

$Map(D_j)$ for crop scouting provides data D_j to extractors including $ExG()$ which determines excess green [23] (a metric for predicting crop yield), $LAI()$ which estimates the leaf area index [44] of crops in the image, and DefoNet, a soybean leaf defoliation detecting neural network [62]. Each of these extractors provides important information about the execution context that can be used to both build final yield maps and predict optimal actions for sampling. Each of these extractors return one or more floating point values which are added to the final SSV.

As Map() is meant to simplify how autonomous agent data is ingested by MARL algorithms, Eval() simplifies the model evaluation and training process that reinforcement learning engineers often perform manually. Reinforcement learning algorithms balance goals (e.g finding targets, navigating through environments) and systems-level budgets (e.g energy expenditure, mission time). Engineers hand-design complex reward functions to accomplish goals within budgets, penalizing poor agent behavior and rewarding positive behavior. Normally, this process at best loosely considers system budgets. MARbLE swarms, however, are intended to execute real-world missions on tight budgets, so edge resource budgets must be considered at every step. To simplify this process, MARbLE automatically engineers reward functions through an iterative, budget-aware training process described in section III D. This process relies on the Eval() function.

$$FS = \{SSV_1, SSV_2, SSV_3 ... SSV_n\} \quad (7)$$

$$B = \{\beta_1, \beta_2, \beta_3 ... \beta_m\} \quad (8)$$

$$\Gamma = \{\gamma_1, \gamma_2, \gamma 3 ... \gamma_o\} \quad (9)$$

$$Eval(FS, \Gamma, B) = \left[\sum_{i=0}^{|\Gamma|} \gamma_i(FS)\right] * \left[\prod_{i=0}^{|B|} \beta_i(FS)\right] \quad (10)$$

Eval() determines whether and to what degree a swarm has accomplished its goals. For some AA, this can be as simple as reaching a certain state. For others, like autonomous crop scouting, goal evaluation is more difficult. Depending on the size and type of field being modeled, the goal may be to make the most accurate yield map possible within some timeframe, cost, or sampling coverage.

Eval() accepts a feature space FS comprised of $n \geq 1$ SSVs and a set of goals Γ and system budgets B, and determines whether the swarm's mission has concluded successfully. Γ is a set of individual goal functions $\gamma_{1..n}$ which, when passed FS, determine whether a goal has been met. Goals are meant to be necessary conditions for mission success. Budgets, on the other hand, denote how well or poorly a mission was completed with respect to system resources. Budgets encompass features of mission completion like mission time, energy expenditure, and efficiency.

As shown in Equation 10, Eval() first determines the sum of all goals, and multiplies it by the product of all budgets. When a goal γ_i is evaluated, that goal function is passed FS, and returns either a positive or negative user-specified number reflecting the value of completion (reward) or failure (penalty) of the goal. Similarly, budgets return a floating point modifier between 0 and 1, based off of FS information, which reflects how well the mission performed with respect to that budget. The product of all budgets is calculated and multiplied by the sum of goal rewards and penalties to return a loss value which scores performance. This performance score is then used in training.

D. Training

MARbLE training automates time-consuming and complicated conventional MARL training tasks. The conventional MARL training process can be decomposed into three steps. First, designers set hyperparameters like rewards, penalties, and budget limits. Next, the model is trained and its performance is evaluated. Finally, designers determine based on the models performance if it is acceptable, or tweak hyperparameters and train again. Hyperparameter tuning and model training are regularly performed in machine learning [55], but rarely consider resource utilization at time of deployment. MARbLE leverages real-world traces and budget considerations to evaluate estimated mission performance in the training process to assure that models meet goals without exceeding budgets. MARbLE then iteratively tweaks reward function hyperparameters as developers would to maximize model performance.

Besides Map() and Eval() functions which ingest AA data and score performance, to train a MARL model, MARbLE requires training data. Users provide training data as a set of representative traces ($\mathbb{C}$) of AA environments. Representative traces hold data from real autonomous agent environments and executions. Traces for autonomous agent research are common [17], [7], [6] and regularly used. Traces can involve simulated environments [6], linear execution data [17], or spatially navigable linked data from real AA environments [7]. MARbLE allows users to bring the trace system that best fits their agents and goals, so long as it allows an AA to sense data and take actions.

```
1  Obj[] train(float[] G, float B[], Obj[] C) {
2    float[] W = initWeights(|B| + |G|)
3    Obj MARL = initMARL()
4    int bestLoss = MAX_INT
5    Obj bestMARL = []
6
7    for i from 0 to numEpochs_HP {
8      for j from 0 to numEpochs_T {
9        for trace in C_tr {
10         float[] FS = sim(trace, MARL)
11         loss = Eval(FS, G, B, W)
12         MARL.update(FS, G, B, loss)
13       }
14     }
15     loss = getTestLoss(C_te, MARL)
16     if(loss < bestLoss) {
17         bestLoss = loss
18         bestMARL = MARL
19     }
20     W = BayesOpt(W, loss)
21   }
22   return bestMARL;
23 }
```

Fig. 4. Bayesian reward shaping pseudocode. Reward Shaping seeks to find the set of hyperparameters which minimizes loss and meets goals.

Given Map(), Eval(), and $\mathbb{C}$, the MARbLE training process, as shown in pseudocode in Figure 4, builds a MARL model and tunes its reward function to best fit user goals and budgets. The training process accepts as arguments sets and representative traces ($\mathbb{C}$). MARbLE first initializes weights which modify each goal and budget's reward or penalty applied to the evaluation. The optimization of these weights across MARL training sessions serves to replicate manual hyperparameter tuning that MARL engineers would conventionally perform. Once weights and a blank MARL model are initialized, the training process can begin.

Every trace in $\mathbb{C}$ is used as either a training trace $\mathbb{C}_{tr}$ or test trace $\mathbb{C}_{te}$. For every representative trace $\mathbb{C}_{tr,i} \in \mathbb{C}_{tr}$, MARbLE runs a single iteration of MARL training. Using $\mathbb{C}_{tr,i}$ as its environment, a simulated swarm of agents explores $\mathbb{C}_{tr,i}$ using the MARL algorithm in training, updating that model with the loss value returned by EVAL after goals have been met or budgets exceeded. This training process takes place for each representative trace repeating over a number of epochs decided by the user. Once a MARL model has been trained, it's performance is tested on $\mathbb{C}_{te}$ to determine a final loss value. Once MARL training has concluded, MARbLE uses Bayesian optimization [5], a common hyperparameter tuning technique, to update the weights that goals and budgets have

been given in the Eval() function. This reward engineering step allows MARbLE to determine which goals and budgets to prioritize for determining reward. This reward shaping step takes place for another set number of epochs as determined by the user, at the end of which the best evaluated MARL model and weighted reward function are returned.

IV. Edge-Optimized Runtime

Digital agriculture requires computation on *far-edge* resources that connect to the Internet via endpoints on the last mile for Internet service providers (ISPs). With Kubernetes and Docker, these resources can use cloud-native tools for workload management. However, MARL workloads on far-edge resourcs present new challenges for runtime management. First, unlike cloud workloads that can provision seemingly infinite data center resources on demand, far-edge resources are capped. Further, with low bandwidth to the Internet, it is impractical to offload data and computation to the cloud. Second, in rural areas, the energy burden— that is percentage of monthly income spent on electricity— is three times higher than the national average [29]. Far-edge resources should conserve energy. Finally, crop fields vary spatially and temporally. For example, an aphid infestation will damage crops near the edge of a soybean field first, but without treatment, the infestation will spread to the whole field within days to weeks. Thus, the computational demands of AA at runtime will change as agents move around the field.

Given the challenges above, we advocate for adaptive management of far-edge resources for swarms. Using our resource-aware development techniques, MARbLE provisions the far-edge resources needed for the peak demands of AA. MARbLE uses cloud-native container orchestration and storage tools to auto-scale edge clusters. MARbLE uses duty-cycling to turn off resources when they are not needed, saving energy. We have observed that AA have periodic peaks and troughs in their computational demand that make duty cycling effective. Finally, MARbLE also manages resource usage for retraining and updating MARL models at runtime by using priority scheduling.

A. Runtime Components

MARbLE deploys all of its core components in containers [36] to maintain hardware independence and support scale-out. Figure 2 shows the different container types that encapsulate pre-built inputs like MARL algorithms, retraining routines, and feature extractors along with core MARbLE platform elements. We use Kubernetes [45] to deploy containers in MARbLE, assign containers to available nodes, and migrate containers in case of cluster autoscaling. Kubernetes also ensures that latency-sensitive real-time tasks such as UAV flight control are assured to be executable within their latency windows. Any remaining cluster resources are used sparingly for online learning. The MARbLE runtime also relies on the Hadooop distributed file system (HDFS [46]) for cluster storage. MARbLE stores models and observations in HDFS to provide redundant and distributed access to individual agents. HDFS automatically replicates data across nodes and self-heals when nodes go offline: a feature that the MARbLE autoscaling approach leverages to avoid data loss.

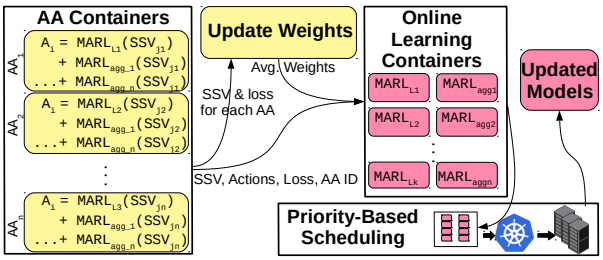

Fig. 5. Online learning evaluates model quality to determine coefficients for priority-based scheduling to update MARL models.

The final piece of the MARbLE runtime is the MARbLE governor. The governor is a Python application which starts missions, deploys all containers using Kubernetes, and autoscales the cluster based on resource utilization thresholds.

B. Cluster Autoscaling

Because MARbLE may include many nodes, and edge devices are provisioned for peak load, it is beneficial to scale the cluster up and down in response to workload needs. MARbLE includes a custom Kubernetes autoscaler which drains compute tasks from superfluous nodes and powers them down to save edge power when loads are low, and re-powers decommissioned nodes using Wake-On-LAN [37] when the system detects that more compute is required.

Powering down nodes in this way must be sensitive to cluster storage implications. Each edge node stores different fragments of data, so we must ensure that no data becomes unavailable. For this purpose we use HDFS for cluster data management, configured to replicate all data twice across the edge node cluster. When data is 'lost' from a decommissioned node it will therefore still be available at one other node in the cluster; after each decommission we simply wait for data to be re-replicated by HDFS before powering down any further nodes, guaranteeing data availability as the active node population changes.

C. Online Learning and Priority-based Scheduling

MARbLE dedicates some of its resources to retraining MARL models online to assure model freshness and to allow agents to diverge when presented with sufficiently different environments. Model retraining is federated, meaning agents maintain local copies of their own models which are updated over time. Updated models are regularly incorporated into a global MARL model, but can also be incorporated into intermediate "aggregator" models. Aggregator models incorporate updates from two or more agent's local models, allowing users or agents themselves to exploit similarities

between agent's individual goals or environments to share an intermediate model. Agents generally use consensus between various models to render decisions.

For model retraining tasks, we augment the resource allocation algorithm used by Kubernetes to optimise *placement* and *task selection* for MARbLE operations. Placement decisions are impacted by data locality, where training data for MARL models is typically fragmented across multiple systems within MARbLE. Our resource allocation algorithm guarantees that containers will be scheduled on an edge node that either (1) has at least some of the data required for model retraining, or (2) is within a user-defined edge hub with expanded compute. This provides Kubernetes with sufficient flexibility in scheduling, but guarantees that data transfer times remain relatively low and allows for the potential of partial or complete data locality at the training site.

Task selection is impacted by the likely quality of a retraining task: when edge compute resources cannot support all retraining tasks, we choose those with the lowest quality first. We rank model quality as determined by relative Eval() outputs. Models who evaluate poorly will receive more priority for retraining as opposed to models that evaluate well. MARbLE uses Kubernetes' priority scheme for scheduling training procedures using the aggregate usefulness of each model provided by AA containers as shown in Figure 5. Model usefulness is simply the floating point value $U_i = [0, 1)$ determined by inverting normalized Eval() results.

$$P_i = \lfloor 10 * U_i \rceil * 100,000,000 \tag{11}$$

P_i then becomes a priority value in range $[0..900,000,000]$ at intervals of $100,000,000$, allowing for 10 possible priority values. This range maps into the entire range of priority levels available in Kubernetes, which is specified by integers between 0 and 1 billion, while providing a coarse granularity that clearly differentiates retraining routines of different utilities and allows us to reserve the priority level 10 for containers with real-time guarantees.

We also use model quality to assign compute resources. Kubernetes allows users to provide minimum and maximum resource usage constraints when pods are instantiated. We use the same priority numbers $[0, 9]$ to assign relative CPU and RAM maximums to pods. All available RAM and CPU units are portioned among pods based on their priority levels.

$$CPU_i = \frac{CPU_t}{n + \sum_{j=0}^{n} P_j} * (P_i + 1) \tag{12}$$

Shown above, all CPU cores available across the system CPU_t for retraining are split evenly among pods based on priority. This same process is used to allocate memory. Pods with a priority of 0, representing the best performing class of models, are scheduled with very few resources, while poorly performing pods are scheduled with the most resources. This priority mechanism allows us to provide more resources to retrain models that perform poorly, and avoid retraining models that already perform well.

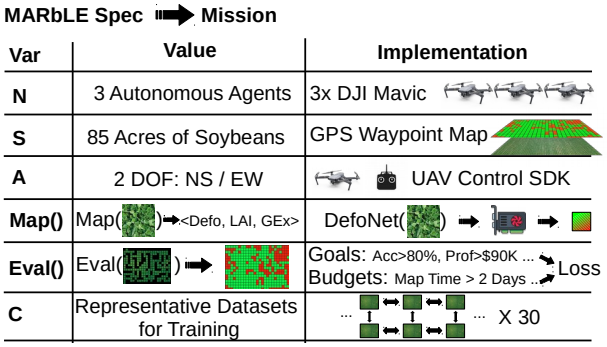

Fig. 6. Our crop scouting application takes MARbLE spec inputs and maps them to a real autonomous swarm implementation.

V. Application

To evaluate MARbLE, we built a real agricultural MARL swarm. Shown in Figure 6, we map MARbLE variables directly to implementation details. Our swarm focuses on modeling a specific crop health condition: leaf defoliation. Leaf defoliation is the process by which crops lose leaf area through predation, disease, or natural aging. Defoliation is a normal part of many plant's lifecycles, but premature defoliation is associated with decreased yield [30]. In soybeans, a globally important crop with over $40 Billion of production annually in the US alone [51], leaf defoliation signifies diseases and pests which can be treated to improve yields, or can decrease yield if ignored. Our swarm uses MARbLE in combination with DefoNet [62], a soybean defoliation neural network, to model crop health. To build our swarm, we first evaluated 45 different hardware, training, and mission length configurations using MARbLE to determine which configuration was most profitable. We then implemented that configuration using the MARbLE runtime and 3 DJI Mavic UAVs in an 85 acre soybean field. In this section, we describe our configuration analysis process and the implementation of our swarm.

A. Building Optimal Models with MARbLE

Before deployment, we built a series of MARL models with MARbLE and tested them on various edge configurations. Autonomous agents are delicate and configuration parameters have cascading performance effects that are difficult to predict a priori [7]. We wanted to understand the performance our system would garner based on hardware configurations, mission flight goals, and model training times. We used MARbLE to build MARL models for 45 separate configurations, and tested each to determine their effects on overall agricultural profits.

Figure 7 shows the performance of each configuration with respect to normalized agricultural profit, with 0% being

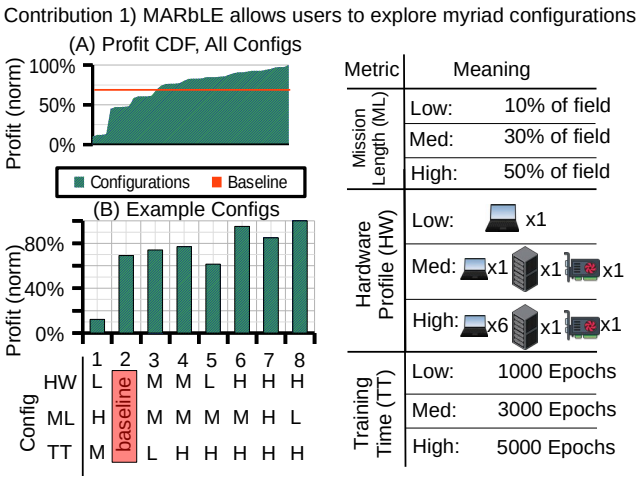

Fig. 7. MARbLE allows developers to test goal and budget combinations to maximize performance.

break-even, and 100% being the best performing configuration. We tested all permutations of 3 edge hardware profiles, 3 mission lengths, and 5 training lengths (1000-5000 epochs of training). Mission lengths describe how much of the field we map with UAV compared to how much we extrapolate from field-sensed data. Hardware profiles describe the hardware accessible to the MARbLE cluster. Training epochs describes the length to which MARbLE trains MARL models before each reward engineering step. All models were trained using 20 reward engineering steps and evaluated using custom UAV and hardware profiles.

We modeled profit by estimating the total profit available for our target 85 acre soybean field and subtracting the costs of equipment, treatment, labor, and crop loss from each configuration. We used agricultural data to determine treatment costs [52] and profits [51] for soy beans. We calculated labor at $30 per hour for a single swarm operator. We simulated final model execution across a validation set of soybean images reserved from the original training set and applied treatment to each region that the UAV either sensed or predicted as defoliated. In our profit model, we assumed that any treated or healthy region would produce an average yield, and that any untreated and unhealthy region would have it's yield uniformly decreased. Our baseline system (shown as configuration 2 in Figure 7 (B)) is an automated 3-UAV swarm which senses every region of the field with offline analysis and generates a treatment plan. All other configurations are MARL 3-UAV swarms which use online analysis to inform mapping.

Figure 7 (A) shows a cumulative density function (CDF) describing the performance of our models as configurations changed from worst to best performance. We found that some configurations performed very poorly despite running as MARL swarms. These configurations suffered from parameter mismatches like long mission times and minimal hardware which decreased startup costs but drastically increased the deployment length and in turn labor costs. Figure 7 (B) shows example configurations of varying levels of performance. Configuration 1, for example, performs poorly due to a mismatch between hardware and mission length. Configuration 1 maps 50% of the crop field using MARL with limited hardware, making each operation take considerably longer than automated scouting and essentially erasing any performance gains from MARL by adding labor costs.

Configurations 3 and 4 show how increased training time provides benefit to MARL algorithms which directly translate to profits. Both configurations use the same hardware and mission lengths, but configuration 3's model was trained for only 1000 epochs while configuration 4 was trained for 5000, adding a benefit of 3% increased profit. This profit improvement holds for all 45 configurations, meaning an additional 1000 epochs of training always either improves profit or rarely (n=6) leaves profit the same. Configurations 5 and 6 demonstrate the importance of hardware selection. In this scenario, our low hardware profile takes considerable time ($> 10s$) to return classification and MARL results for each image which inflates labor costs. Conversely, our highly provisioned profile returns classifications faster ($< 3s$) and can retrain more often, leading to better results over time and faster results constantly, resulting in a 33% profit improvement despite increased initial costs. Finally, configurations 7 and 8 demonstrate the importance of well-calibrated mission goals. Configuration 7 sets a goal to map 50% of the crop field to return results, while configuration 8 maps a more sparing 10%. The benefit configuration 7 provides from increased mapping does not outweigh the cost in time and labor to map additional field regions. Configuration 8 was the most profitable of all configurations, beating the baseline automated scouting approach by 31%.

B. Implementation

We deployed a MARbLE cluster at a private 85 acre soybean field. Our deployment ran from August 26th to September 16th 2021, the period at which those soybeans were most susceptible to yield loss due to premature defoliation. Over this time, we ran 150 UAV swarm missions. Figure 6 shows our MARbLE spec implementation and configuration. First, we initialized all MARbLE inputs: Map(), Eval(), and $\mathbb{C}$, with added budgets and goals. Our Map() and Eval() functions for crop scouting are shown in pseuocode Figure 3. Our Map() function uses DefoNet along with two simpler crop health metrics (Green Excess and Leaf Area Index) to determine health and inform map building. Eval() uses an extrapolation procedure from prior work [61] to build crop health maps from Map() feature sets collected by our UAV agents. $\mathbb{C}$ was a set of 30 Autonomy Cubes collected from 5 separate soybean fields located over 100 miles from our test field. Each autonomy cube contains 1344 spatially linked aerial soybean images composed into a 32x42 matrix.

We used an approach from prior work [7] to build and validate profiles for UAV and edge systems. Profiles included energy consumption and time distributions for flight between waypoints, data capture, and data transfer times.

Our swarm's runtime used a custom autonomous UAV software package to control UAV. We connected each DJI Mavic via remote control to it's own android tablet running our UAV control platform. Our MARbLE cluster, based on configuration 8 in figure 7, consisted of two Lenovo T470 Thinkpads, 4 HP G6 laptops, and one Dell precision 7920 workstation. Each Lenovo had an Intel i7 CPU and ran Ubuntu 18.04. One Lenovo was used as the MARbLE head node, controlling all UAV communication and acting as the MARbLE Kubernetes master. This machine was provisioned with 24 GB of RAM. The second Lenovo was used for UAV control and retraining offloading, and was provisioned with 8GB of RAM. The Dell workstation had one Intel Xeon 6258R CPU, 64 GB of RAM, and an NVIDIA RTX 2080Ti GPU. This machine was used for classification and as the primary node for reinforcement learning retraining. Each HP G6 laptop was used for additional compute for retraining and control pods.

VI. Evaluation

We implemented our swarm as described in Section V across a real farm and in software simulation using various configurations and swarm sizes. In this section, we describe MARbLE performance results from our deployment using single UAV missions and 3 UAV swarms, and use deployment-validated simulation to provide results for 2 and 4 UAV swarms.

We evaluate MARbLE's comparative performance against state-of-the-art autonomous swarm control and automated control methods as described in section 2. Figure 8 (a-b) show MARbLE's performance on our crop scouting workload as compared to commensurate autonomous and automated approaches. Figure 8 (a) shows results for 8 mission configurations. The first four configurations show results for our maximum accuracy setting which is shown as configuration 7 in figure 7. This setting covers a wide range of our field to maximize prediction accuracy. The final four configurations represent our maximum profit configuration, shown as configuration 8 in Figure 7. This configuration maximizes field profit by minimizing labor costs while providing sufficiently accurate maps with short missions. For each of these configurations, we show how MARbLE performs against autonomous and automated scouting in various swarm sizes. Results for swarm sizes 1 and 3 were obtained from our real world MARbLE deployment, while results for swarm sizes 2 and 4 were produced using mission-validated simulations.

Figure 8 (a) shows how long swarms of various autonomy settings take to accomplish the same task. Shorter times are preferred due to decreased labor costs and additional time for more missions. Figure 8 shows that MARbLE swarms across all configurations are shorter than automated swarms by (1.2-2.1X) and autonomous swarms by (1.1-1.73X). When compared to automated swarms, MARbLE converges quickly by carefully selecting waypoints to sample. Automated swarms visit waypoints faster because their flight-paths are predetermined and do not have to be calculated online. This, however, limits their flexibility. MARbLE swarms outperform automated swarms on all settings despite additional decision-making because the benefits of model-based sampling outweigh the speedup from pre-determined paths. Furthermore, automated swarms must search up to 30% more waypoints to meet the same accuracy goals MARbLE swarms, further increasing automated mission lengths. When compared to autonomous swarms, MARbLE's better optimized models, online learning capabilities, and engineered reward function outperform stale autonomous models. We found that even short periods (10 missions) of online learning contribute between 1.1% and 3.4% improvements to final map accuracy, allowing MARbLE swarms to converge to goals faster. Interestingly, autonomous swarms are outperformed by automated swarms for two configurations: high accuracy swarms with 3 and 4 agents. Autonomous swarms decrease performance relative to swarm size due to increased contention for cluster resources which increases inference times for individual UAV. This prohibits scale, allowing automated flight to outperform autonomous flight at larger swarm sizes. MARbLE, on the other hand, slightly improves performance as swarm size increases (1%) as the benefits of data-sharing outweigh the costs of scale.

Figure 8 (b) shows one way that MARbLE is able to complete goals faster than autonomous and automated scouting while requiring additional compute power. Figure 8 (b) shows the relative number of samples each approach required to create a maps of various quality. Low quality maps (70%+ accuracy) are quick to produce but provide only some treatment benefit. Medium and high quality maps (80%+ and 90%+ accuracy) provide additional benefit for targeted treatment, but take longer to capture. Figure 7 shows that lower accuracy fast maps are optimal for maximizing profit in this agriculture scenario, but other scenarios may require higher quality swarm outputs to maximize performance. Figure 8 shows that MARbLE requires considerably less samples than autonomous (up to 1.7X) and automated (up to 2.2X) to meet accuracy goals. Even at the highest accuracy settings, MARbLE requires less samples than both approaches to accomplish its mission due to MARbLE's coordination among swarm members and goal-based development.

MARbLE swarms require considerably more resources at peaks than autonomous or automated swarms. Figure 8 (c) and (d) show the CPU and memory needs of 3-UAV autonomous and MARbLE swarms over a 2 hour period. Our UAV swarms are deployed on 8 mapping missions, where UAV fly to waypoints in the crop field, sense them to build a crop scouting map, and return for a battery exchange.

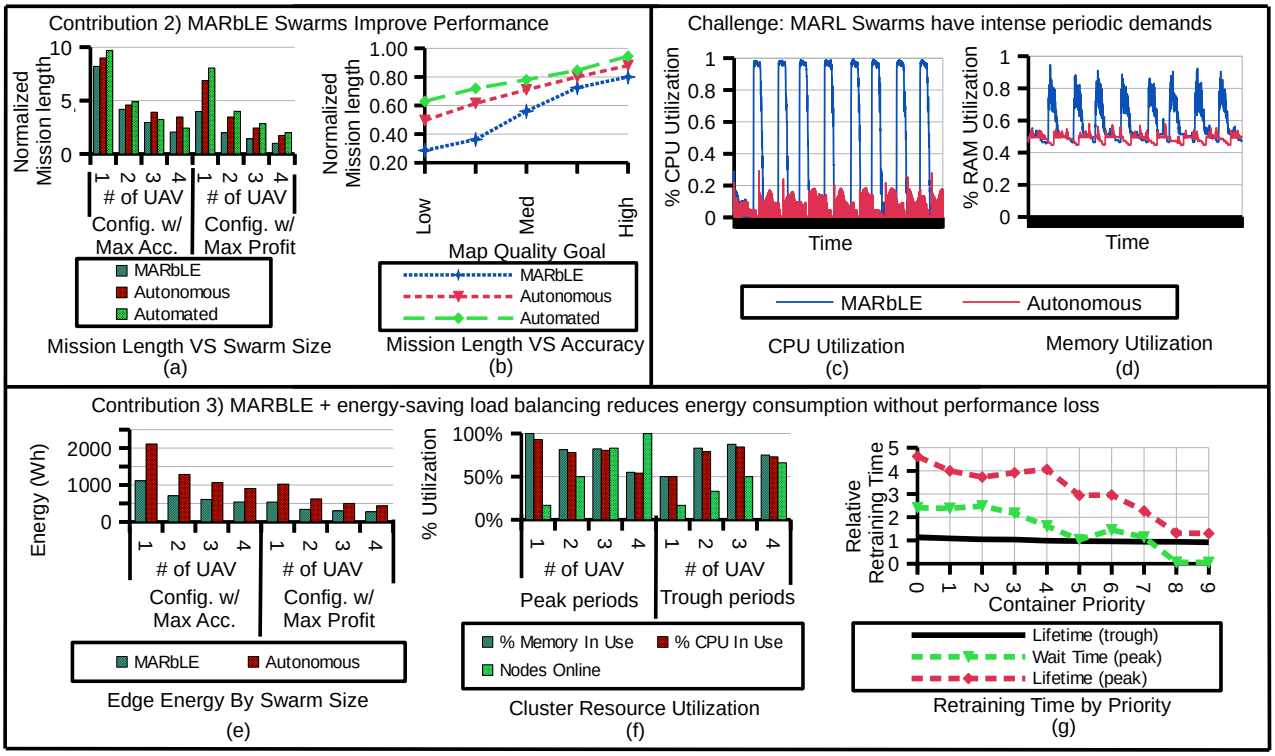

Fig. 8. Experimental results: a,b) MARbLE's programming model and swarm capabilities improve performance considerably by decreasing mission length while increasing accuracy. c-d) MARL swarms, however, require more resources than autonomous swarms. e-g) MARbLE's system management features save energy without performance loss by responding to the periodic demands of MARL and prioritizing high-value model updates.

Both autonomous and MARbLE swarms experience periodic demands in-mission. At some points, multiple UAV request resources for inference, pathfinding, or flight control and at others no UAV are making requests. For autonomous swarms, these periods can be seen clearly peaking when all swarm members are deployed, and tapering off as members actions diverge and individual UAV finish their missions. Eventually, all UAV conclude their mapping or run out of battery, and must return for a manual battery exchange where we see troughs in workload utilization. MARbLE swarms use the periods directly after mission execution to learn from prior inference. MARL models are updated to improve performance, which consumes considerable resources. For this reason, MARbLE swarms require 3.4X more CPU and 1.6X more memory at peak than autonomous swarms, and notably 9.7X more CPU on average due to the effects of online learning. MARL swarms also experience steep differences in average to peak demand, requiring 2.9X compute resources at peaks compared to averages. These consistent but periodic demands require additional resource provisioning. To meet these demands while minimizing overall deployment energy consumption, we implemented our energy-saving load balancing system described in Section 4.

Figures 8(e-g) show how MARbLE's edge-focused load-balancing approach improves overall edge energy consumption on our crop scouting workload. Using our prototype MARbLE cluster, we ran real-time crop scouting missions for swarm sizes 1 and 3, and modeled swarm sizes 2 and 4 in deployment-validated simulation. Energy was calculated using an AC watt meter connected to the cluster. We tested all swarms using our most profitable configuration, shown as configuration 8 in Figure 7.

Figure 8(f) shows MARbLE's performance against a MARbLE cluster with no load-balancing. When compared to autonomous execution, MARbLE conserves energy in two different ways: mission lengths are shorter overall, and edge cluster resources are more efficiently used due to our load-based duty-cycling approach. Compared to autonomous flight, similar sized MARbLE missions consume 1.58X-2X less power. A swarm of multiple UAVs is also more energy efficient per-device than fewer UAVs; compared to autonomous flight using a single UAV as in prior work, a swarm of four MARbLE controlled UAVs uses 3.7X-3.9X less energy depending on swarm size and goals.

Figure 8(f) shows exactly how MARbLE manages resources across extremes during a swarm mission. For a single UAV, a one node MARbLE cluster is enough to handle all resource needs. As the swarm grows, more nodes must be provisioned. As swarm size increases from 2 to 4, peak and trough allocations change. For instance, a 4-UAV swarm can

operate at troughs using just 4 nodes, but requires all 6 to handle peak loads. MARbLE's energy savings shown in Figure 8(e) are a direct result of its ability to spread resources evenly across the cluster and shut down unnecessary nodes until they are needed.

Finally, we examine MARbLE's usefulness-aware model retraining; this offers better use of finite edge resources by prioritizing retraining for poorly performing models. Figure 8(g) shows how our usefulness metric affects model retraining times compared to average retraining times. When the system is not under load, Kubernetes is easily able to distribute containers. If the system is correctly provisioned for the edge, however, it may experience peak loads where containers must contend for resources. We evaluated retraining times for 4-UAV swarms on our crop-scouting benchmark. We found that wait-times for high-priority containers (8-9 on the x-axis) were insignificant even at peak loads, but could be up to 2.4X normal retraining time for very low (0-2) priority containers. Similarly, high-priority containers experienced only modest (1.3X) lifetime increases even when the system was highly pressured. This was at the expense of lower priority containers, which experienced lifetimes up to 4.6X longer than usual. This behavior allows the system to take resources from models with high utility to the system and give them to low utility models.

VII. Limitations and Future Work

Contribution 1 shows that MARbLE simplifies building MARL systems by replacing handcrafted reward functions with Map() and Eval() functions that are easier to program. However, MARbLE also requires representative traces from the target environment ($\mathbb{C}$). Generating realistic traces, especially in novel and dynamic environments can be challenging and costly. Future work should explore tradeoffs in cost, realism and data availability. Also, we have not evaluated training time for MARbLE swarms. In our tests, training time ranged from 29 and 250 minutes on a Lenovo laptop with one GPU. Future work exploring more complex swarm workloads will demand efficient training procedures.

Contribution 2 showed that MARbLE can use Bayesian optimization to improve swarm performance. In general, Bayesian optimization has admirable attributes like fast convergence, but other popular techniques, such as gradient descent, could be applied as well. We have not explored the tradeoffs in convergence time and learning efficacy. Contribution 3 showed that container management with priority scheduling can reduce energy usage and improve utilization. Advanced edge-aware approaches to manage inference and training [14], [15] should be explored in this context. In our evaluation, we used Q-learning for our crop-scouting scenario. Conceptually, MARbLE is compatible with other techniques, e.g., deep Q-networks. Future work should test efficacy across such modeling frameworks. Lastly, we tested MARbLE on digital agriculture workloads only. While important and illustrative of many edge computing challenges, MARbLE should work well for many other MARL problems which require autonomous agents deployed in edge contexts.

VIII. Related Work

Much recent work has tackled the concerns of real-world autonomous agents using strong theoretical foundations. Lin et al [32] explores a federated meta-learning approach to train models with small datasets in an edge setting. Singh et. al [47] demonstrates a novel reward-training mechanism for reinforcement learning to eliminate the need for reward shaping. Kilinc and Montana [25] constructs a framework for sharing data among agents non-stationary environments using intrinsic reward and temporal locality. Other recent work [59], [57], [42] on networked agents provide considerable insight into the behaviors of real-world cooperative MARL systems with limited communication capabilities. Porter et. al [40] presents a novel development platform for creating software that autonomously assembles itself and discovers optimal execution policies online without the need for expert model building and reward shaping. Edge-SLAM [4] provides mechanisms offloading for SLAM, a critical component of autonomy, using edge computing in a way that is suitable for AA like our autonomous UAV.

Much related work deals specifically with autonomous aerial systems. Boubin et. al [7] demonstrates that naive hardware and algorithm selection for fully autonomous aerial systems can have serious performance consequences. Cui et. al [10] implements MARL for allocating networking resources across a network of UAV base-stations. In agriculture Zhang et. al [61], Yang et. al [56], and FarmBeats [53] provide new techniques for automated and autonomous UAV crop scouting.

IX. Conclusion

Swarms of autonomous agents powered by resources at the edge can provide insight and actuation that can solve globally important challenges in digital agriculture. We present MARbLE, an end-to-end platform for building, deploying, and executing these swarms. To use MARbLE, developers implement *Map()* and *Eval()* functions and specify mission configurations. MARbLE compiles and deploys swarms using a multi-agent reinforcement learning framework and allows users to explore myriad configurations to build the best models for their application's goals. At runtime, MARbLE manages load-balancing for edge resources by duty-cycling compute resources when demands are low, and linking online learning outcomes to retraining resource allocation. Our evaluation shows that MARbLE can produce effective and efficient swarms beyond state of the art techniques, suggesting that this tool chain could make swarms more accessible to researchers and developers.

Acknowledgments: This work was funded by NSF Grants OAC-2112606 and DGE-1343012, and the Ohio Soybean Council.

REFERENCES

[1] Ardupilot autopilot suite. Accessed on Dec 2021.

[2] H. Barrett and D. C. Rose. Perceptions of the fourth agricultural revolution: What's in, what's out, and what consequences are anticipated? *Sociologia Ruralis*, 62(2):162–189, 2022.

[3] A. Barrientos, J. Colorado, J. d. Cerro, A. Martinez, C. Rossi, D. Sanz, and J. Valente. Aerial remote sensing in agriculture: A practical approach to area coverage and path planning for fleets of mini aerial robots. *Journal of Field Robotics*, 28(5):667–689, 2011.

[4] A. J. Ben Ali, Z. S. Hashemifar, and K. Dantu. Edge-slam: edge-assisted visual simultaneous localization and mapping. In *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services*, pages 325–337, 2020.

[5] J. Bergstra, D. Yamins, and D. D. Cox. Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms. In *Proceedings of the 12th Python in science conference*, volume 13, page 20. Citeseer, 2013.

[6] B. Boroujerdian, H. Genc, S. Krishnan, W. Cui, A. Faust, and V. Reddi. Mavbench: Micro aerial vehicle benchmarking. In *MICRO*, 2018.

[7] J. G. Boubin, N. T. Babu, C. Stewart, J. Chumley, and S. Zhang. Managing edge resources for fully autonomous aerial systems. In *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing*, pages 74–87. ACM, 2019.

[8] L. Busoniu, R. Babuska, and B. De Schutter. A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38(2):156–172, 2008.

[9] Y.-h. Chang, T. Ho, and L. Kaelbling. All learning is local: Multi-agent learning in global reward games. *Advances in Neural Information Processing Systems*, 16:807–814, 2003.

[10] J. Cui, Y. Liu, and A. Nallanathan. Multi-agent reinforcement learning-based resource allocation for uav networks. *IEEE Transactions on Wireless Communications*, 19(2):729–743, 2019.

[11] S. Das, S. Chapman, J. Christopher, M. R. Choudhury, N. W. Menzies, A. Apan, and Y. P. Dang. Uav-thermal imaging: A technological breakthrough for monitoring and quantifying crop abiotic stress to help sustain productivity on sodic soils–a case review on wheat. *Remote Sensing Applications: Society and Environment*, 23:100583, 2021.

[12] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[13] T. Doan, S. Maguluri, and J. Romberg. Finite-time analysis of distributed td (0) with linear function approximation on multi-agent reinforcement learning. In *International Conference on Machine Learning*, pages 1626–1635. PMLR, 2019.

[14] B. Fang, X. Zeng, F. Zhang, H. Xu, and M. Zhang. Flexdnn: Input-adaptive on-device deep learning for efficient mobile vision. In *Proceedings of the 5th ACM/IEEE Symposium on Edge Computing (SEC)*, 2020.

[15] B. Fang, X. Zeng, and M. Zhang. Nestdnn: Resource-aware multi-tenant on-device deep learning for continuous mobile vision. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, pages 115–127, 2018.

[16] F. Fishel, W. C. Bailey, M. L. Boyd, W. G. Johnson, M. H. O'Day, L. Sweets, and W. J. Wiebold. Introduction to crop scouting. *Extension publications (MU)*, 2009.

[17] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 32(11):1231–1237, 2013.

[18] H. C. J. Godfray, J. R. Beddington, I. R. Crute, L. Haddad, D. Lawrence, J. F. Muir, J. Pretty, S. Robinson, S. M. Thomas, and C. Toulmin. Food security: The challenge of feeding 9 billion people. *Science*, 2010.

[19] I. Grondman, L. Busoniu, G. A. Lopes, and R. Babuska. A survey of actor-critic reinforcement learning: Standard and natural policy gradients. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(6):1291–1307, 2012.

[20] L. T. Hickey, A. N Hafeez, H. Robinson, S. A. Jackson, S. Leal-Bertioli, M. Tester, C. Gao, I. D. Godwin, B. J. Hayes, and B. B. Wulff. Breeding crops to feed 10 billion. *Nature biotechnology*, 37(7):744–754, 2019.

[21] J. Hu, A. Bruno, B. Ritchken, B. Jackson, M. Espinosa, A. Shah, and C. Delimitrou. Hivemind: A scalable and serverless coordination control platform for uav swarms. *arXiv preprint arXiv:2002.01419*, 2020.

[22] S. Kar, J. M. Moura, and H. V. Poor. Qd-learning: A collaborative distributed strategy for multi-agent reinforcement learning through consensus + innovations. *IEEE Transactions on Signal Processing*, 61(7):1848–1862, 2013.

[23] S. Khanal, J. Fulton, N. Douridas, A. Klopfenstein, and S. Shearer. Integrating aerial images for in-season nitrogen management in a corn field. *computers and electronics in agriculture*, 148, 2018.

[24] S. Khanal, J. Fulton, and S. Shearer. An overview of current and potential applications of thermal remote sensing in precision agriculture. *Computers and Electronics in Agriculture*, 139:22–32, 2017.

[25] O. Kilinc and G. Montana. Multi-agent deep reinforcement learning with extremely noisy observations. *arXiv preprint arXiv:1812.00922*, 2018.

[26] L. Klerkx, E. Jakku, and P. Labarthe. A review of social science on digital agriculture, smart farming and agriculture 4.0: New contributions and a future research agenda. *NJAS-Wageningen Journal of Life Sciences*, 90:100315, 2019.

[27] M. S. Kukal and S. Irmak. Climate-driven crop yield and yield variability and climate change impacts on the us great plains agricultural production. *Scientific reports*, 8(1):1–18, 2018.

[28] Y. Lan, K. Huang, C. Yang, L. Lei, J. Ye, J. Zhang, W. Zeng, Y. Zhang, and J. Deng. Real-time identification of rice weeds by uav low-altitude remote sensing based on improved semantic segmentation model. *Remote Sensing*, 13(21):4370, 2021.

[29] G. Levy. Report: Energy costs are a higher burden on the rural poor some low-income rural residents pay nearly a fifth of their monthly income on their energy bills. usnews.com, 2018.

[30] X. Li, P. An, S. Inanaga, A. E. Eneji, and K. Tanabe. Salinity and defoliation effects on soybean growth. *Journal of plant nutrition*, 29(8):1499–1508, 2006.

[31] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[32] S. Lin, G. Yang, and J. Zhang. A collaborative learning framework via federated meta-learning. In *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*, pages 289–299. IEEE, 2020.

[33] M. L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Machine learning proceedings 1994*, pages 157–163. Elsevier, 1994.

[34] J. Liu, J. Xiang, Y. Jin, R. Liu, J. Yan, and L. Wang. Boost precision agriculture with unmanned aerial vehicle remote sensing and edge intelligence: A survey. *Remote Sensing*, 13(21):4387, 2021.

[35] L. Meier, P. Tanskanen, L. Heng, G. H. Lee, F. Fraundorfer, and M. Pollefeys. Pixhawk: A micro aerial vehicle design for autonomous flight using onboard computer vision. *Autonomous Robots*, 33(1-2), 2012.

[36] D. Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux journal*, 2014(239):2, 2014.

[37] N. Mishra, K. Chebrolu, B. Raman, and A. Pathak. Wake-on-wlan. In *Proceedings of the 15th international conference on World Wide Web*, pages 761–769, 2006.

[38] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[39] G. C. Nelson, H. Valin, R. D. Sands, P. Havlík, H. Ahammad, D. Deryng, J. Elliott, S. Fujimori, T. Hasegawa, E. Heyhoe, et al. Climate change effects on agriculture: Economic responses to biophysical shocks. *Proceedings of the National Academy of Sciences*, 111(9):3274–3279, 2014.

[40] B. Porter, M. Grieves, R. Rodrigues Filho, and D. Leslie. Rex: A development platform and online learning approach for runtime emergent software systems. In *Symposium on Operating Systems Design and Implementation*, pages 333–348. USENIX, November 2016.

[41] C. Qu, P. Calyam, J. Yu, A. Vandanapu, O. Opeoluwa, K. Gao, S. Wang, R. Chastain, and K. Palaniappan. Dronecoconet: Learning-based edge computation offloading and control networking for drone video analytics. *Future Generation Computer Systems*, 125:247–262, 2021.

[42] G. Qu, A. Wierman, and N. Li. Scalable reinforcement learning of localized policies for multi-agent networked systems. In *Learning for Dynamics and Control*, pages 256–266, 2020.

[43] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, 2009.

[44] R. Raj, J. P. Walker, R. Pingale, R. Nandan, B. Naik, and A. Jagarlapudi. Leaf area index estimation using top-of-canopy airborne rgb images. *International Journal of Applied Earth Observation and Geoinformation*, 96:102282, 2021.

[45] G. Sayfan. *Mastering kubernetes*. Packt Publishing Ltd, 2017.

[46] K. Shvachko, H. Kuang, S. Radia, and R. Chansler. The hadoop distributed file system. In *2010 IEEE 26th symposium on mass storage systems and technologies (MSST)*, pages 1–10. Ieee, 2010.

[47] A. Singh, L. Yang, K. Hartikainen, C. Finn, and S. Levine. End-to-end robotic reinforcement learning without reward engineering. *arXiv preprint arXiv:1904.07854*, 2019.

[48] A. Tahir, J. Böling, M.-H. Haghbayan, H. T. Toivonen, and J. Plosila. Swarms of unmanned aerial vehicles—a survey. *Journal of Industrial Information Integration*, 16:100106, 2019.

[49] E. C. Tetila, B. B. Machado, G. Astolfi, N. A. de Souza Belete, W. P. Amorim, A. R. Roel, and H. Pistori. Detection and classification of soybean pests using deep learning with uav images. *Computers and Electronics in Agriculture*, 179:105836, 2020.

[50] D. C. Tsouros, S. Bibi, and P. G. Sarigiannidis. A review on uav-based applications for precision agriculture. *Information*, 10(11):349, 2019.

[51] USDA. Usda national agricultural statistics service, 2017 census of agriculture. https://www.nass.usda.gov/AgCensus/, 2017.

[52] USDA. Usda economic research service, commodity costs and returns. https://www.ers.usda.gov/data-products/commodity-costs-and-returns/, 2022.

[53] D. Vasisht, Z. Kapetanovic, J. Won, . R. Chandra, A. Kapoor, S. Sinha, M. Sudarshan, and S. Strãtman. Farmbeats: An iot platform for data-driven agriculture. In *NSDI*, 2017.

[54] C. J. Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[55] L. Yang and A. Shami. On hyperparameter optimization of machine learning algorithms: Theory and practice. *Neurocomputing*, 415:295–316, 2020.

[56] M.-D. Yang, J. G. Boubin, H. P. Tsai, H.-H. Tseng, Y.-C. Hsu, and C. C. Stewart. Adaptive autonomous uav scouting for rice lodging assessment using edge computing with deep learning edanet. *Computers and Electronics in Agriculture*, 179:105817, 2020.

[57] K. Zhang, Z. Yang, and T. Basar. Networked multi-agent reinforcement learning in continuous spaces. In *2018 IEEE Conference on Decision and Control (CDC)*, pages 2771–2776. IEEE, 2018.

[58] K. Zhang, Z. Yang, and T. Başar. Multi-agent reinforcement learning: A selective overview of theories and algorithms. *arXiv preprint arXiv:1911.10635*, 2019.

[59] K. Zhang, Z. Yang, H. Liu, T. Zhang, and T. Basar. Fully decentralized multi-agent reinforcement learning with networked agents. In *International Conference on Machine Learning*, pages 5872–5881. PMLR, 2018.

[60] X. Zhang, L. Han, Y. Dong, Y. Shi, W. Huang, L. Han, P. González-Moreno, H. Ma, H. Ye, and T. Sobeih. A deep learning-based approach for automated yellow rust disease detection from high-resolution hyperspectral uav images. *Remote Sensing*, 11(13):1554, 2019.

[61] Z. Zhang, J. Boubin, C. Stewart, and S. Khanal. Whole-field reinforcement learning: A fully autonomous aerial scouting method for precision agriculture. *Sensors*, 20(22):6585, 2020.

[62] Z. Zhang, S. Khanal, A. Raudenbush, K. Tilmon, and C. Stewart. Assessing the efficacy of machine learning techniques to characterize soybean defoliation from unmanned aerial vehicles. *Computers and Electronics in Agriculture*, 193:106682, 2022.