Design, Implementation, and Applications of Fully Autonomous Aerial Systems

Dissertation

Presented in Partial Fulfillment of the Requirements for the Degree Doctor of Philosophy in the Graduate School of The Ohio State University

By

Jayson Boubin, B.S 2017, M.S. 2020

Graduate Program in Computer Science and Engineering

The Ohio State University

2022

Dissertation Committee:

Dr. Christopher Stewart, Advisor

Dr. Anish Arora

Dr. Sami Khanal

Abstract

Fully autonomous aerial systems (FAAS) combine edge and cloud hardware with UAVs and considerable software support to create self-governing systems. FAAS complete complicated missions with no human piloting by sensing and responding to their environment in real-time. FAAS require highly complex designs to function properly, including layers of on-board, edge, and cloud hardware and software. FAAS also necessitate complex software used for controlling low-level UAV actions, data collection and management, image processing, machine learning, mission planning, and high-level decision-making which must integrate across the compute hierarchy effectively to meet autonomy goals in real-time.

The complexity of even a relatively simple FAAS makes efficiency difficult to guarantee. Efficiency, however, is paramount to the effectiveness of a FAAS. FAAS perform missions in resource-scarce environments like natural disaster areas, crop fields, and remote infrastructure installations. These areas have limited access to computational resources, network connectivity, and power. Furthermore, UAV battery lives are short, with flight times rarely exceeding 30 minutes. If FAAS are inefficiently designed, UAV may waste precious battery life awaiting further instructions from remote compute resources, delaying or precluding mission completion. For this reason, it is imperative that FAAS designers carefully choose or design edge hardware configurations, machine learning models, autonomy policies, and deployment models.

FAAS have the capability to revolutionize a number of industries, but much research must be done to facilitate their usability and effectiveness. In this dissertation, I outline my efforts toward designing and implementing FAAS that are efficient and effective. This dissertation will focus on the following five topics encompassing design, implementation, and applications of FAAS:

§1. Creation of new general and domain-specific machine learning algorithms and careful utilization of others

§2. Selection of hardware at all levels in the FAAS hierarchy

§3. Power and environmental awareness informing selection and switching of autonomy policies, hardware devices, machine learning techniques and deployment characteristics.

§4. Online learning capabilities resilient to limited cloud access, network interruption, and power scarcity.

§5. Thorough applications which demonstrate the technological value of FAAS, drive adoption, and determine future research challenges.

For Kelly,

without whom this would never have been possible.

Acknowledgments

Modern science can not be produced in a vacuum. The work in this document was performed in collaboration with many excellent scientists and engineers, and is built upon the work of many more. I want to take a moment to acknowledge those who have helped me grow as a scientist, engineer, educator, and thinker. In this document, I have taken the time to reference the works of these scientists which most inspire my own.

First and foremost, I must thank my advisor Dr. Christopher Stewart for his excellent guidance, kindness, and friendship. Without Chris' knowledge, patience, and encouragement, I surely would not be where I am today. I know that his mentorship will be the greatest contributing factor to any future career success I achieve, and I strive to emulate him as a role model in my future endeavors.

I would like to thank my committee members Dr. Sami Khanal and Dr. Anish Arora, whose extensive domain knowledge and mentorship have helped me grow broadly as a scientist. I must also thank Dr. Trevor Bihl, with whom I have collaborated just as deeply as with my committee members. My collaborations with these fine scientists have inspired my research directions in ways that will resound throughout my career.

I would like to thank those who mentored me as an undergraduate. I thank Dr. Dhananjai Rao of Miami University, who fostered my first interest in Research as an

undergraduate. I also thank AFRL and AFIT researchers Dr. Christina Rusnock, Dr. Michael Miller, and Dr. Brett Borghetti who mentored me extensively during my undergraduate career. These researchers all welcomed me into their labs as an untrained freshman undergraduate and taught me how to tackle research challenges. They have instilled in me the importance of undergraduate mentorship: something that has been a focus of mine as a PhD student and will remain a focus as faculty.

I also want to thank my many friends who shared this journey with me in one way or another. Lucas Magee, Ryan Slechta, Pouya Kousha, Erik Clarke, Max Fogle, Sally Dufek, Zichen Zhang, Eduardo Romero, Shiqi Zhang, and many others have all acted as sources of inspiration, motivation, and catharsis. I hope that for them I have provided the same. Most among these, I want to thank my brother Matthew Boubin. Matt has contributed more than he will ever understand to my successes, and I hope that I have contributed to his.

I also want to thank my parents, Felicia and Jay Boubin, who taught me (originally against my will) the value of education and the importance of living a fulfilling life. Thank you for believing in me well before I ever believed in myself. I hope you are as proud of me as I am of you both.

Finally, I dedicate this thesis in its entirety to my wife Kelly. You alone have seen the sweat and toil that has culminated in this document. As you surely know, your emotional and material support provided me an environment in which I could dedicate myself to my work. I hope to now provide you the freedom to pursue your passions unencumbered, as you have done for me these past five years. Thank you and I love you. -Jayson

Vita

2013-2017	B.S. Computer Science and Engineering, Miami University
2017-2020	M.S. Computer Science and Engineering, Ohio State University
2017-2022	PhD Student, Computer Science and Engineering, Ohio State University

Publications

Research Publications

Jayson Boubin, Avishek Banerjee, Jihoon Yun, Haiyang Qi, Yuting Fang, Steve Chang, Kannan Srinivasan, Rajiv Ramnath, Anish Arora "PROWESS: An Open Testbed for Programmable Wireless Edge Systems" *PEARC 2022*

Anthony Baietto, **Jayson Boubin**, Patrick Farr, Trevor J Bihl, Aaron M Jones, Christopher Stewart "Lean Neural Networks for Autonomous Radar Waveform Design" *Sensors 2022*

Max Taylor, **Jayson Boubin**, Haicheng Chen, Christopher Stewart, Feng Qin "A Study on Software Bugs in Unmanned Aircraft Systems" *ICUAS 2021*

Jayson Boubin, Codi Burley, Peida Han, Bowen Li, Barry Porter, Christopher Stewart "Programming and deployment of autonomous swarms using multi-agent reinforcement learning" *arXiv 2021*

Zichen Zhang, **Jayson Boubin**, Christopher Stewart, Sami Khanal "Whole-field reinforcement learning: A fully autonomous aerial scouting method for precision agriculture""Programming and deployment of autonomous swarms using multi-agent *Sensors 2020*

Ming-Der Yang, **Jayson Boubin**, Hui Ping Tsai, Hsin-Huang Tseng, Yu-Chun Hsu, Christopher Stewart "Adaptive Autonomous UAV scouting for rice lodging assessment using edge computing with deep learning EDANet" *Computers and Electronics in Agriculture, December 2020*

Alwyn Burger, Patrick Urban, **Jayson Boubin**, Gregor Schiele "An Architecture for Solving the Eigenvalue Problem on Embedded FPGAs". *ARCS 2020*

Patrick Farr, Aaron Jones, Trevor Bihl, **Jayson Boubin**, Ashley DeMange "Waveform Design Implemented on Neuromorphic Hardware". *IEEE International Radar Conference 2020*

Jayson Boubin, Aaron Jones, Trevor Bihl "NeuroWav: Toward Real-Time Waveform Design for VANETs using Neural Networks". *IEEE VNC 2020*

Jayson Boubin, Naveen T.R Babu, Christopher Stewart, John Chumley, Shiqi Zhang "Managing Edge Resources for Fully Autonomous Aerial Systems". *IEEE/ACM SEC 2019*

Jayson Boubin, John Chumley, Christopher Stewart, Sami Khanal "Autonomic Computing Challenges in Fully Autonomous Precision Agriculture". *IEEE ICAC 2019*

Jayson Boubin, Christina Rusnock, Jason Bindewald "Qualifying Compliance and Reliance Trust Behaviors to Influence Trust in Human-Automation Teams". *HFES 2017*

Christina Rusnock, **Jayson Boubin**, Joseph Giametta, Tyler Goodman, Anthony Hillesheim, Sungbin Kim, David Meyer, Michael Watson "The Role of Simulation in Designing Human-Automation Systems". *HCII 2016*

Fields of Study

Major Field: Computer Science and Engineering

Table of Contents

Page

List of Tables

List of Figures

Chapter 1: Introduction

Unmanned aerial vehicles (UAV) are an emerging technology that has seen considerable adoption in private industry, research, and hobbyist communities in the past decade [129]. UAV are used extensively in aerial photography, crop scouting, search and rescue, infrastructure inspection, surveillance, and military applications [65, 105, 134, 136, 197] so much so that worldwide consumer UAV sales have eclipsed $4 Billion annually and, as shown in Figure 1.1 are projected to rise considerably in the coming years [49, 50].

UAV used in industrial and commercial settings are generally piloted by licensed professionals, posing a number of problems for UAV deployments. Licensed pilots can command considerable compensation [72], which can add up staggeringly depending on the number of unmanned aerial vehicles (UAV) required to complete a given task in the preferred amount of time. Furthermore, human-piloted missions can be complicated by inclement or inhospitable conditions, pilot visibility issues, or waypoint repetition. UAV are flown in disaster areas, remote infrastructure deployments, and crop fields which further necessitate large swarms to meet time and coverage goals, making pilots prohibitively expensive.

Data collected by UAV may also require complicated analysis to make decisions. Crop scouting, surveillance, and infrastructure inspection tasks may require analysis

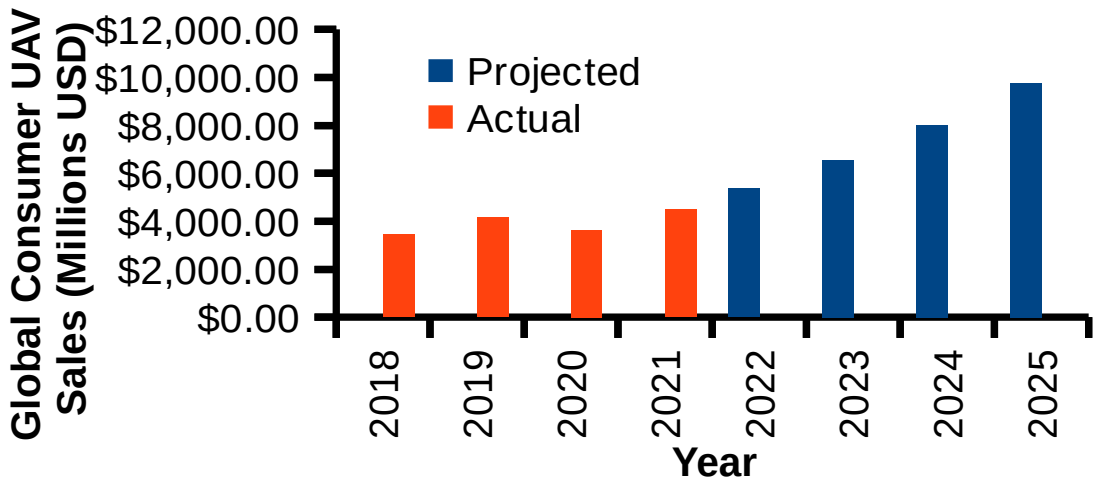

Fig. 1.1: Data from the business research company shows that consumer UAV market is consistently increasing year over year.

by professionals or computers that can not be provided by pilots in flight [95]. The inability to react to sensed data in real-time using complex expert analysis may elongate missions or leave information on the table.

Fully Autonomous Aerial Systems (FAAS) are an emerging technology that replace human pilots with machine learning algorithms, edge hardware, and remote compute resources along the edge and in the cloud. The goal of FAAS design is to create a system that accomplishes high-level goals with little to no human interaction [30]. FAAS provide considerable support beyond conventional UAV groundstation software. Groundstations [137, 190] can set waypoints, fly UAV in an automated manner, and sense data. FAAS, on the other hand, generate waypoint or virtual remote-control missions that respond to sensed data in real-time to solve a predefined problem [30]. On top of conventional groundstation software, FAAS use

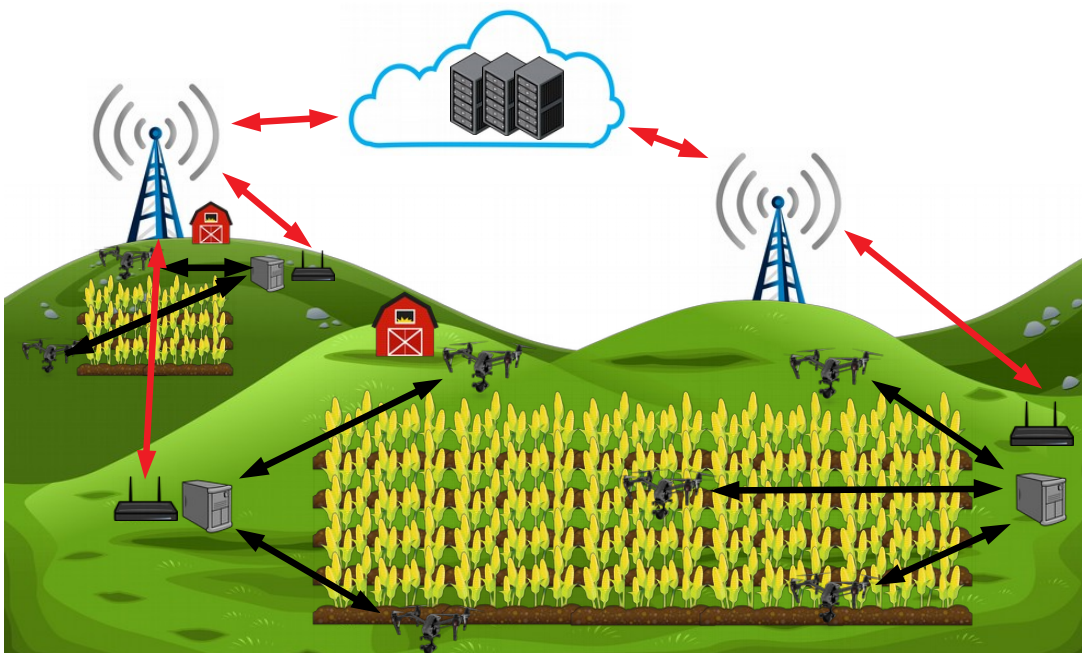

Fig. 1.2: FAAS are incredibly complex. They operate in remote environments, use novel autonomy policies and machine learning algorithms, and must withstand power limitations and leverage creative networking solutions to accomplish their goals.

machine learning techniques like object detection, feature extraction, and reinforcement learning to identify mission goal-states and features of their environment that may lead to goal-states. There are many UAV groundstation platforms, SDKs, and software packages [58, 61, 137, 161, 190], many of which underlie FAAS, but few utilize machine learning, computer vision, autonomy policies, and systems management strategies which FAAS require [30, 167, 197].

There are many challenges to creating FAAS, some of which complement each other. First and foremost, FAAS require considerable machine learning and image processing capabilities to recognize and locate the visual phenomena they are often

seeking to accomplish their missions. FAAS may search for crop diseases in a field, locate humans in rubble, or inspect remote infrastructure for cracks [60,105,136]. FAAS attempt to replicate domain-expert analysis of aerial images in real-time and features of such analysis must be elucidated and implemented as algorithms that can function in real or near-real time on hardware that can be accessed by FAAS. Furthermore, autonomy policies must use image processing and detection algorithms to accomplish a goal. Goals may be as simple as locating a target, or as complex as generating an area map using as few sample points as possible. FAAS must use information at their disposal to accomplish the goal, regardless of whether that information contains features that may be immediately relevant (i.e signatures of a crop disease). FAAS must strive to locate targets as quickly as possible, which requires autonomy policies that can leverage seemingly insignificant data-points to local potential goal states.

Even with correct autonomy policies and machine learning algorithms, FAAS face other natural limitations. FAAS execute in remote environments, potentially with no access to reliable power and network connectivity. Many FAAS must rely on edge systems for the entirety of their machine learning needs, as well as UAV flight control and data storage. If network connectivity is available, bandwidths may be limited meaning online learning, data storage, or model inference in the cloud may be impossible. Creative networking solutions like TVWS [197] have been used in FAAS, but require considerable power to implement. Furthermore, UAV have small batteries [41,73], often providing less than 30 minutes of flight time on a single charge. UAV Must be recharged regularly, which can take multiple hours, so UAV power-efficiency is paramount. FAAS dictate UAV flight paths online, meaning UAV may wait for flight instructions if model inference times are slow, or may explore unnecessary or

unhelpful waypoints if inferences are inaccurate. Hovering UAV waste their precious battery, so it is important to return results to UAV quickly to maximize UAV flight efficiency. This creates a many-way tradeoff between on-site power, network connectivity, edge and cloud hardware, machine learning model accuracy and inference time. The goal of FAAS design and implementation is, in sort, to create software that is low-power, adaptable to heterogeneous architectures, easily expandable to accommodate policies and models from new domains, and general enough to provide useful software artifacts to most domains out of the box.

1.1 Thesis Statement

Fully autonomous aerial systems require considerable software support, precise hardware selection, and novel learning and autonomy policies to complete complex missions with minimal human interaction. Specifically, careful design, planning, and research must inform every facet of FAAS development, including:

§1. Creation of new general and domain-specific machine learning algorithms and careful utilization of others

§2. Selection of hardware at all levels in the FAAS hierarchy

§3. Power and environmental awareness informing selection and switching of autonomy policies, hardware devices, machine learning techniques and deployment characteristics.

§4. Online learning capabilities resilient to limited cloud access, network interruption, and power scarcity.

§5. Thorough applications which demonstrate the technological value of FAAS, drive adoption, and determine future research challenges.

Each chapter of this dissertation, as discussed in section 1.2 contributes to one or more of these four topics.

1.2 Contributions and Outline

Every experimental result presented in this work required some level of FAAS software support to produce. At the outset there was no publicly available FAAS software package, to my knowledge. I created SoftwarePilot [26], an open source FAAS middleware, to facilitate my research and provide easy access to FAAS for other research. SoftwarePilot underlies all of my work, and should be regarded as a major contribution of this dissertation.

This thesis makes a number of other contributions to the state of the art. First, chapter 2 discusses in detail the design decisions at the edge required to implement FAAS efficiently. As covered earlier in this chapter, FAAS are very difficult to design and present many-way trade-offs. The effects of some design decisions (e.g naive hardware or algorithm selection) may not be realized until after implementation and testing. Chapter 2 explores how complicated FAAS can be modeled in software to maximize performance without full implementation(§2). I created autonomy cubes, an n-dimensional hyper-cube data structure for holding entire autonomous system execution environments. Using autonomy cubes and profiled traces of FAAS hardware and algorithm combinations, I demonstrate that it is possible to model FAAS execution accurately. I evaluated my models using 3 common FAAS benchmarks: autonomous photography, search and rescue, and crop scouting, necessitating real

FAAS implementations using softwarepilot (§1, §5). In this work, I also compare system-level optimizations like adaptive model-switching, onboard/embedded computing, cloud offloading, and hardware duty-cycling(§3). I use simulations validated by over 100 real FAAS missions with different hardware and software configurations to perform this analysis.

Chapter 3 outlines the need for and design of Fully Autonomous Precision Agriculture, a key FAAS application(§1, §5). In Fully Autonomous Precision Agriculture (FAPA), UAV autonomously scout crop fields, returning crop field health maps to farmers which can be used to inform fertilizer application, treat pests, disease, and other crop stressors, and predict and improve yield before harvest. This chapter contributes an initial reinforcement learning process (§1) that uses machine learning to guide FAAS through crop fields to generate crop yield maps. I use simulation to generate yield maps of crop regions using ground truth UAV-captured images.

Chapter 4 presents a FAPA Application using new autonomy techniques (§1, §5) and validated modeling using methods presented in Chapter 2. I use UAV to implement an adaptive autonomous approach to FAAS crop scouting for rice lodging. I use semantic segmentation of aerial images in real-time to predict rice lodging using real aerial images from lodged Taiwanese rice paddies. Using EDANet for semantic segmentation, and my adaptive autonomous scouting technique which scouts at multiple heights to minimize UAV batter consumption, my solution achieves very high lodged detection accuracy while executing considerably faster than naive approaches. This approach generates considerable cost savings and demonstrates an approach far beyond the real-world state of the art for lodged rice detection in Taiwan.

Chapter 5 discusses early work in distributed reinforcement learning for FAAS swarms (§1, §3, §4, §5). I describe the design and implementation of a network and power aware scheduling process for online training of FAAS reinforcement learning pathfinding algorithms on consumer edge hardware. I call this technique the Fleet Computer. The Fleet Computer is an end to end platform for designing, programming, and deploying autonomous systems and swarms. The fleet computer relies heavily on multi-agent reinforcement learning theory to train autonomous swarms based on pre-defined goals and representative autonomy cubes. The fleet computer can then interact with robotic control platforms like SoftwarePilot to deploy autonomus swarm applications across a cluster of edge resources. Relying on Kubernetes, the fleet computer can dynamically expand and contract the cluster's resource footprint by duty-cycling resources. Finally, the fleet computer incorporates online federated learning capabilities which improve system performance over time for autonomous systems whose models are built using limited data.

Chapter 6 covers findings from a real-world Fleet Computer deployment (§3,§5). I deployed a fleet computer cluster and swarm of drones in a crop field in Central Ohio to model Soybean Defoliation, an important crop health condition. I flew over 150 real swarm missions to test the effects of environmental conditions on swarm deployments. Using my deployment results, I built an adaptive deployment model which uses weather information to schedule swarm missions in conditions which minimize risk of failure.

Chapter 2: Managing Edge Resources for Fully Autonomous Aerial Systems

Fully autonomous aerial systems (FAAS) fly complex missions guided wholly by software. If users choose software, compute hardware and aircraft well, FAAS can complete missions faster and safer than unmanned aerial systems piloted by humans. On the other hand, poorly managed edge resources slow down missions, waste energy and inflate costs. This paper presents a model-driven approach to manage FAAS. We fly real FAAS missions, profile compute and aircraft resource usage and model expected demands. Naive profiling approaches use traces from previous flights to infer resource usage. However, edge resources can affect where FAAS fly and which data they sense. Usage profiles can diverge greatly across edge management policies. Instead of using traces, we characterize whole flight areas to accurately model resource usage for any flight path. We combine expected resource demands to model mission throughput, i.e., missions completed per fully charged battery. We validated our model by creating FAAS, measuring mission throughput across many system settings. Our FAAS benchmarks, released through our open source FAAS suite SoftwarePilot, execute realistic missions: autonomous photography, search and rescue, and agricultural scouting using well-known software. Our model predicted throughput with 4% error across mission, software and hardware settings. Competing approaches yielded

10–24% error. We used our SoftwarePilot benchmarks to study (1) GPU acceleration, scale up, and scale out, (2) onboard, edge and cloud computing, (3) energy and monetary budgets, and (4) software driven GPU management. We found that model-driven management can boost mission throughput by 10X and reduce costs by 87%.

2.1 Introduction

Unmanned aerial systems (UAS) hover, fly to waypoints and perform defined actions, e.g., landing and takeoff. In addition to rotors and motors, these aircraft carry computer systems, cameras, batteries, etc. They can access high, vast or unsafe places and capture detailed images and sensor readings. Photographers, farmers and first responders pilot UAS via remote control or smart phone [16, 65, 105, 134, 197]. These end users decide where the UAS flies, when it senses data and when missions are complete.

UAS piloting mistakes can have severe consequences. For example, flying UAS in restricted areas risks human lives. Other common mistakes, e.g., flying to unneeded waypoints, degrade mission throughput (i.e., the number of missions completed). Aerial systems that require less human piloting are needed [134,196]. There is growing support for software development kits that control aircraft. Da-Jiang Innovations (DJI) aircraft support software control from iOS, Android and Linux devices [174]. Pixhawk and Aerostack also provide platforms for software control [137, 167, 168].

Fully autonomous aerial systems (FAAS) are an emerging workload wherein UAS execute dynamic missions defined wholly by software. End users do not pilot FAAS nor do they define preset waypoints. Instead, they provide goals, constraints and

software that execute missions. Like edge-driven video analytics [63,81,89,97,201,211, 217], FAAS process images in real time and leverage AI for scene analysis. However, FAAS also control aircraft flight, making flight paths (and which data gets sensed) dynamic.

Recent UAS carry sophisticated processors. For example, the DJI Mavic carries the Myriad 2, a system on chip that includes streaming vector engine processors, hardware accelerators, multiple RISC cores and 2 MB on-chip memory. However, compared to UAS, FAAS increase computational demands significantly [16, 123, 196, 201, 211]. If aircraft surrender battery capacity to onboard processors and carry heavier payloads for data storage, flight times will suffer [201]. Instead of using onboard resources, FAAS workloads may run on land using networked messages to control the aircraft (i.e., edge cloudlets). Another choice connects aircraft to fast cloud data centers. Latency, processing capacity and cost differ among these choices. Professional end-users must understand how these factors affect their bottom line, but it is hard to answer what-if questions comparing architecture and software designs. For example, how many missions would complete per fully charged battery (i.e., **mission throughput**) if I ran FAAS software onboard the aircraft instead of edge servers? Or does data processing speedup provided by a GPU warrant its cost?

Autonomous systems combine many independent software components. Many components support settings that trade compute demand for energy savings. It is hard to predict the effects of these settings on mission throughput because they affect where FAAS fly, how many compute resources they require, and the effects are mission specific. For example, lightweight image classifiers can lower compute demand and save energy, but FAAS also fly to more waypoints which can negate savings. End

users could test each setting and measure mission throughput directly. However, long complex missions and many software settings make exhaustive testing impractical.

This paper presents a modeling approach that predicts mission throughput. Our approach profiles energy demands for aircraft and compute. We model missions as a sequence of waypoints. The waypoint that exhausts battery capacity defines mission throughput. It is hard to profile energy demands across system settings that affect autonomous decisions; we call these autonomy settings. Autonomy settings change flight paths, affecting which data is sensed during mission execution and ultimately energy demands. We propose *autonomy cubes,* data structures that characterize the whole flight area for a mission. Autonomy cubes can approximate sensed data for any flight path, much like data cubes (their intellectual inspiration) [82].

To validate the model, we created SoftwarePilot, an FAAS suite that performs the following complex missions: (1) autonomously capture high quality photographs of human faces, (2) search and map defined areas for first responders, and (3) scout crop fields for representative samples. SoftwarePilot uses path finding and AI approaches found in prior research [123, 167]. Each FAAS supports a wide range of software and hardware settings. Toggling these settings can increase waypoints per mission by 4X and compute demands by 35X.

We collected 122 autonomy cubes, flying in 56 locations and capturing 20970 data readings. We also flew 145 actual FAAS missions and measured ground-truth mission throughput. FAAS used DJI Spark and Mavic Pro aircraft. We compared our modeling approach to Aerostack [167] and Autoware [98, 123]. These approaches use reference traces from prior missions to model energy usage. Our model predicted throughput with 4% error. When trace and mission settings differed on multiple

dimensions, Aerostack and Autoware yielded error up to 5X and 10X larger than our approach.

After validating our modeling approach, we explore model-driven management of edge resources. First, we consider scenarios where end users buy aircraft, software and hardware separately and then combine them to make a FAAS. After purchasing aircraft and software, these end users would like to purchase compute hardware that will provide high throughput. Under a cost budget, these end users may have to maximize throughput per dollar. End users can use our models to answer these questions. We used our modeling approach to compare onboard, edge and cloud architectures. Onboard compute and storage reduced flight time by 50%, significantly degrading mission throughput. Edge computing eventually provided best mission throughput and throughput per dollar. However, we observed that larger aircraft could boost onboard architectures. We also used our modeling approach to compare scale out, scale up and GPU approaches to meet compute demands. We found that the best approach depended on (1) autonomy settings and (2) energy capacity. GPU improve throughput, but deplete batteries quickly. Scaling up cores on chip provided a reliable approach to increase throughput.

We also study the management scenario where end users control the design and implementation of aircraft, processor and software [134]. These end users may sell FAAS to militaries, smart cities or other large operations and can adjust all facets of FAAS to find high throughput settings. We studied software and hardware co-design. We set up an adaptive policy that toggles between deep, compute intensive AI models and less precise but energy efficient AI models. An edge system running CPU and GPU can power off the GPU for less precise models. We compared approaches that

toggle GPU states during missions and between missions. Toggling GPU states during a mission provided higher mission throughput. We also modeled end-to-end cost for an industrial application: agricultural scouting. With model-driven edge management, FAAS missions are 87% cheaper than human piloted UAS missions.

Our contributions are as follows:

- Our modeling approach precisely predicts mission throughput. We show that autonomy settings have large effects on FAAS flight path and energy usage.

- We built three open source FAAS through SoftwarePilot (autonomous photography, search and rescue, and agricultural scouting). We measure their mission throughput directly and validate our modeling approach.

- We demonstrate edge resource management techniques that boost throughput and lower costs.

The remainder of this paper is as follows. Section 2.2 provides an overview of FAAS. Section 5.3 presents an energy model driven by autonomy cubes. Section 2.4 describes the implementation of FAAS. Section 2.5 validates our model and studies FAAS workloads. Section 2.6 uses our modeling approach to guide FAAS system management. Section 2.7 discusses future work and limitations of our approaches. Section 5.7 discusses related work. Section 5.8 presents conclusions.

2.2 Motivation and Background

Autonomous systems perform complex tasks in vaguely defined areas without receiving commands from humans. By this definition, UAS are *not* autonomous. Humans decide (1) where to fly, (2) when to sense data and (3) when a mission is complete. Figure 2.1 depicts UAS workflow. Humans set high-level mission goals,

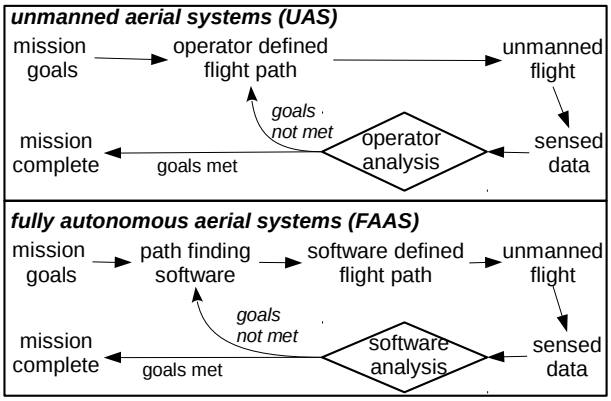

Fig. 2.1: With FAAS, humans set mission goals but do not pilot.

e.g., take a great photo of a human target. Then, they pilot the aircraft to waypoints by (1) using remote control devices, (2) making gestures or (3) providing a list of GPS coordinates. At each waypoint, the UAS senses data from its surroundings, e.g. detailed images or GPS data. After studying data, humans decide if their goals are met. If not, they choose new waypoints and repeat.

Figure 2.1 also depicts workflow for FAAS. Humans set goals, but all remaining work is done by software. The system is capable of completing multiple missions without a human issuing commands. To remove humans from the loop, software must decide when a mission is complete, meaning both human end users and software can understand mission goals. There is a semantic gap between true goals and goals that can be expressed in software. Today, autonomous systems require end users to translate their vague, high-level goals into mathematical equations (called utility functions). If the mission is not complete, FAAS software must also choose the next waypoint.

An Example FAAS Mission: Crop fields are vast. Scouting reports can miss subtle problems, e.g., over crowding or crop disease. Human piloted UAS have transformed scouting. Companies, like Fly The Farm [115], fly over fields and capture detailed images. These images inform farmers, guiding the application of fertilizer and pesticides. However, UAS pilots charge \$1–\$5 per acre. One scouting report can cost nearly 3% of net profits for corn fields [34, 80, 197]. By eschewing human pilots, FAAS can lower costs.

Figure 2.2 depicts flight area and two agricultural scouting missions explored similarly in prior research [23]. Flight area comprises waypoints where the aircraft can fly. Each cell in Figure 2.2 represents a waypoint (here, a GPS location). Following the blue line, our FAAS flight controller directs the aircraft to a waypoint (A1) and captures a detailed image. FAAS software analyzes the image, counting corn crops. If the image contains enough crops to accurately measure the state of the field, the mission is complete. (Note, the image may be fed into subsequent analysis, such as yield modeling [105].) If not, FAAS path finding software chooses a new waypoint (B1). This process repeats and the mission completes at waypoint B2.

FAAS visit only some waypoints on each mission. As shown in Figure 2.2 path finding software (A* search versus nearest neighbors) changes where FAAS fly and how quickly missions complete. The settings represented by the red line choose a longer flight path. This is a key distinction: Video analytics using UAS fly to preset waypoints and adapt video quality dynamically [201]. With FAAS, analytics can affect flight actions, changing resource demands in ways that are hard to predict.

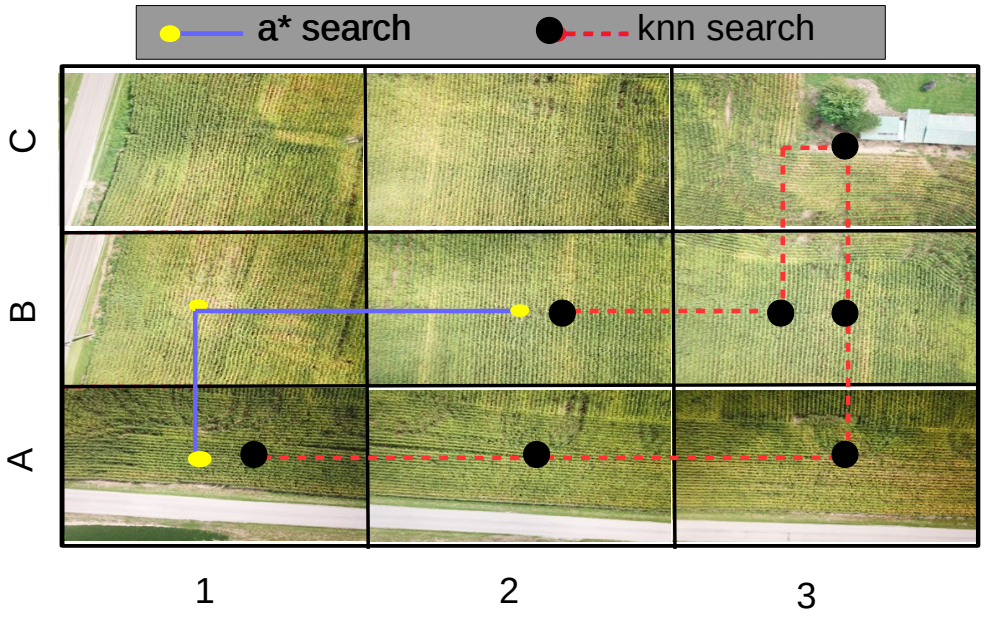

Fig. 2.2: Two agricultural scouting missions. Each image represents an allowed waypoint where the aircraft could have flown (i.e., flight area). Lines represent actual waypoints visited. Both missions begin at A1.

Runtime Execution: Figure 2.1 depicts runtime execution with key software in red. First, software manages *flight controls* for takeoff, landing and maneuvers. *Sensing software* pulls data from aircraft sensors. These data producing and actuation components are latency sensitive and normally use processors placed onboard the aircraft [16]. Like video analytics, FAAS use *AI models* to convert sensed images to multi-dimensional points [81, 89, 150, 201]. Each dimension represents the output of a model. This discussion does not require a specific class of model (e.g., DNN or regression) and models are built offline. During runtime, models are evaluated to classify scenes. This execution can use onboard processors, edge cloudlets or cloud resources [201]. *Reinforcement learning* can be a robust approach to autonomously control devices [109]. In this approach, FAAS are distributed with training data on

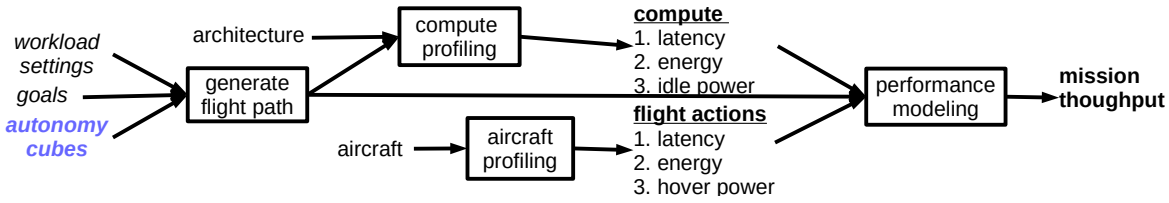

Fig. 2.3: Modeling mission throughput for multiple architectures, aircraft and autonomy settings.

observed settings. Each training data element contains (1) scene features computed using AI models, (2) a flight action and (3) a utility gain. An entry means that taking the flight action after observing the scene features, at one point in time, led to the utility gain. Reinforcement learning maps scenes to actions. End users must define *utility* of sensed data as a function over extracted features. This software too can execute onboard, at edge cloudlets or cloud.

2.3 Performance Modeling

Figure 2.3 outlines our approach to model mission throughput. Model inputs relate to autonomy (goals & workload), compute architecture and aircraft.

Our approach has four stages, shown as boxes in Figure 2.3. First, we collect all data that could be sensed during a mission, i.e., an *autonomy cube*. Autonomy cubes are used to construct precise flight paths along with a pathfinding algorithms discussed later (KNN, A*). Given a flight path, the next stages profile compute and aircraft workloads using empirical data. Finally, bottleneck analysis predicts whether aircraft or compute exhausts batteries first.

2.3.1 Autonomy profiling

FAAS are hard to model, because their flight path depends on *which* data is sensed at runtime. Two FAAS flying only a few feet apart can differ on the utility of their sensed data. Their flight paths could diverge, affecting energy usage per mission and ultimately mission throughput. We mitigate variation caused by spatial displacement by modeling expected mission throughput averaged over many runs. Path finding, AI models, utility functions and other autonomy settings have systemic and non-linear effects on flight path.

Competing Approaches: Before detailing our approach, it helps to explain how recent work models autonomous systems [98, 123, 168]. Autoware [98, 123] uses a long, representative trace from a self-driving car. This trace suffices for research because safety concerns constrain driving maneuvers and execution environments. Aerostack [168] creates multiple traces where users change the environment between traces. This approach captures a wide range of maneuvers, but, if autonomy settings change, flight paths will diverge from previous traces.

We use *autonomy cubes*, a data structure that captures all images that can be sensed during a mission within user defined constraints. Autonomy cubes represent a principled ideal for benchmarking; they can be used to compute flight paths across any FAAS autonomy setting. By depicting captured images at each allowed GPS location, Figure 2.2 presents a simple 2-D autonomy cube.

Defining autonomy cubes: Shown in Equation 2.1, A waypoint x is a multi-dimensional point. Dimensions can abstract (1) GPS or grid positions (e.g., Figure 2.2), (2) aircraft poses (e.g., aircraft attitude, gimbal positions), and localized data (e.g., altimeter and compass readings). Waypoint x is a set of dimensions $d_1 ... d_n$

that uniquely describes the UAVs real world position and state. FAAS fly in a discrete and finite space. where the dimensions are constrained either by limitations of the vehicle, the user, or communication range. For each dimension d_i in which the vehicle can move (e.g yaw, upward motion, northward motion). dimensions are constrained to some maximum magnitude D_i, describing the maximum range of the vehicles motion in said direction as such: $\forall i : 0 < d_i < D_i$.

$$\mathrm{x} = (\mathrm{d}_0, ..., \mathrm{d}_\mathrm{K}) \tag{2.1}$$

$$\mathbf{fa_i} \in \mathbf{FA} : \{\mathrm{x_m}\} \rightarrow \{\mathrm{x_n}\} \tag{2.2}$$

$$\mathbf{FS} : \{\mathrm{x}, \mathrm{fa_i}, \mathrm{y}\} \rightarrow \{1|0\} \quad \text{if} \quad \mathrm{fa_i(x)} = \mathrm{y} \tag{2.3}$$

$$\mathbf{ug} = \frac{\mathrm{u(ai(x))}}{\mathrm{u(ai(y))}} \circ \mathrm{FS}\{\mathrm{x}, \mathrm{fa_i}, \mathrm{y}\} \tag{2.4}$$

As shown in Equation 2.2, we abstract flight actions FA as a set of functions that move the UAV between waypoints. Precisely, a flight action applies a set force. The force moves a hovering aircraft along six degrees of freedom. Actions are calibrated offline. Each action fa_i moves aircraft from one expected waypoint to another. Due to spatial displacement, actual and expected positions may differ slightly. For example, wind can apply unexpected force, moving the aircraft away from its expected position. Note, waypoints reachable by any combination of actions define flight area.

A single step along a flight path has a starting point, action and ending point. The flight step function FS (Equation 2.3) indicates valid steps where the flight action leads to an end point within the confines of the FAAS flight areas dimensions. A waypoint will always have $|FA|$ flight actions, but only some may produce valid flight steps. At each waypoint, FAAS senses its surroundings, transforms sensed data using

AI models (ai) and computes utility ($0 <= u(x) <= 1$) of its current state, which we call a featureset. Each valid flight step, i.e., $FS(...) \rightarrow 1$, has utility gain. Referring to Figure 2.2, the first flight step for the blue line is: $\{[\mathrm{A}, 1], \mathrm{FlyNorth}, [\mathrm{B}, 1]\}$. The AI models for this mission include A* (i.e., $\mathrm{ai}_{\mathrm{A}*}$).

We now define autonomy cube, $\forall x \in \mathbf{X}\ ac\ =\ \cup\,(x, ai(x))$, where $\mathbf{X}$ represents the set of all reachable waypoints. This is to say that an autonomy cube is a data structure that represents the set of all reachable waypoints and their featuresets.

Equation 2.4 shows that ac allows FAAS to compute utility gain for any valid flight step. As shown in Equation 2.5, a flight path is a sequence of N valid flight steps. Autonomous systems aim to maximize total utility gain **TG**, i.e., the product of utility gains acquired at each step in the path.

$$\begin{aligned} \mathrm{fp} &= \{\mathrm{x_n}, \mathrm{fa_{f(n)}}, \mathrm{y_n}\}_{\mathrm{N}} \\ \mathbf{TG} : \{\mathrm{fp}\} \rightarrow \mathrm{r} \in \mathbb{R} : \mathrm{r} &= \prod_{\mathrm{n}} \mathrm{ug}(\mathrm{x_n}, \mathrm{fa_{f(n)}}, \mathrm{y_n}) \end{aligned} \tag{2.5}$$

Using autonomy cubes: Autonomy cubes can be used to simulate a FAAS mission. For this paper, we constrain flight areas to an n-dimensional mesh of waypoints, e.g., a building or crop field. Each dimension corresponds to a FAAS flight action. Each action has an inverse action that is expected to return the aircraft to its original position. Supported actions include x,y, z translation and pitch, yaw, roll and gimbal pitch.

Capturing autonomy cubes: Quadcopters support 6 controllable degrees of freedom, meaning they can use pitch, yaw and roll to fly in any direction along an x,y,z coordinate system [129]. Unlike cars and fixed wing planes, the coordinate system can be explored in any direction relatively quickly, requiring at most rotation and thrust.

However, they only move forward in time. We exploit quadcopter maneuverability to capture sensed data before it changes, i.e., we transform time into discrete blocks based on how frequently the utility of sensed data changes.

Equation 2.6 defines scene persistence P as the minimum discrete time slots t such that a hovering aircraft perceives qualitatively similar utility. As shown in Equation 2.7, to capture an autonomy cube, the aircraft or aircrafts fly to each waypoint in $\mathbf{X}$ before P seconds have elapsed. Due to the unpredictable nature of FAAS pathing, we must safely assure that a cube can be created within persistence constraints before flight. This is done by assuming that each flight action takes worst-case time. If a worst-case flight path can complete an autonomy cube within P seconds, so can all others. for this reason, we use the slowest flight action (fa_s) to model shifting between waypoints.

It is possible to use multiple aircraft to collect an autonomy cube. Equation 2.6 introduces the variable s representing swarm size (the number of UAV capturing the autonomy cube) to account for the decrease in latency of using a swarm of UAV.

$$\mathrm{P} = \mathrm{Latency}(\mathrm{fa_s}(\mathrm{x})) \times \mathrm{t} : \tag{2.6}$$

$$\frac{\mathrm{Max_i}(\mathrm{Latency}(\mathrm{fa_i}(\mathrm{x}))) * |\mathbf{X}|}{\mathrm{s}} < \mathrm{P} \tag{2.7}$$

Examining Equation 2.7, we observe four techniques to scale our approach.

1. Increase scene persistence: Scenes can be tweaked manually so that key features change less frequently. For example, we have tested our FAAS benchmarking suite using mannequins in place of fidgety humans, and farm land.

2. Shrink flight area: We can shrink the total area where aircraft can fly or allow fewer flight actions. Of course, fewer flight actions degrades total utility gain.

3. Speedup flight actions: We could also reduce flight time going between waypoints. Scheduling autonomy cube flight paths by prioritizing waypoints with the shortest flight time relative to the current waypoint would minimize delay per flight step.

4. Use swarms to partition cubes: Finally, multiple quadcopters can be deployed at once, allowing each to capture a fraction ($\frac{1}{S}$) of the flight area. We have used swarms to capture autonomy cubes used with our FAAS. However, for workloads that require tight maneuvering (autonomous photography), partitioning presents several research challenges. First, partitioning to minimize expected delay per action is challenging. Seconding, partitioning should consider the effects of battery capacity. Partitioning on search and rescue (partitioning by rooms) and agricultural sampling (by field region) are much more feasible. Finally, aircraft flying in the same region may interfere with each other.

We used swarms comprising 2 & 3 aircraft to partition flight area along the vertical axis (y-axis) for our autonomous photography benchmark. As expected, we were able to cover up 3X more flight area in the best case. However, we also observed anomalies unique to aerial systems. Placing aircraft immediately under each other (i.e., partitioning y while strictly keeping x & z the same) affected wind patterns, creating suctions. The aircraft crashed into each other. Partitions that worked well slightly offset the aircraft in the x & z dimensions.

2.3.2 Aircraft profiling

In the second stage of our modeling approach, we profile latency and energy functions for flight actions on an input aircraft. In Equation 2.8, the $LatencyA$ function

estimates latency using the average delay of flight actions executed at multiple, sampled points within the flight area denoted by N. Conceptually, we do the same for energy and hover power. This profiling is done offline.

$$\begin{gathered} \text{LatencyA} = \{\text{fa}_\text{i}(\text{x})\} \rightarrow \text{r} \in \mathbb{R} : \\ \text{r} = \frac{\sum_{\text{n}'} \overline{\text{LatencyA}(\text{fa}_\text{i}((\text{x}_{\text{n}'}))}}{\text{N}}, \text{N} << |\mathbf{X}| \end{gathered} \tag{2.8}$$

2.3.3 Compute profiling

Compute latency and energy vary depending on the content of data sensed at a waypoint. Unlike UAV actions, which have a tight latency distribution, compute latency has more variance. Scheduling fluctuations, unpredictable threading overhead, model timing, and network interference all cause compute timings to vary. Compute latency was profiled online, requiring a varied set of execution environments and conditions. experimental latency numbers for all individual components of our benchmark were compiled into normal distributions (represented my μ and σ), truncated to the third standard deviation. Our model uses these distributions to predict compute latency for offline missions similarly to Equation 2.8.

$$\begin{gathered} \text{LatencyC} = \{\text{ai}(\text{x})\} \rightarrow (\mu, \sigma) \in \mathbb{R} : \\ \sigma = \frac{\sum_{\text{n}'} \overline{\text{Latency}(\text{ai}((\text{x}_{\text{n}'}))}}{\text{N}}, \text{N} << |\mathbf{X}| \end{gathered} \tag{2.9}$$

2.3.4 Throughput modeling

Recall, FAAS compute their flight path fp_i at runtime. As shown in Equation 2.10, each flight step $\text{fs}_{[\text{i,n}]} \in \text{fp}_\text{i}$ is informed by data observed during execution.

$$\widehat{\mathbf{ug}_{\text{n,n+1}}} : \{\text{x}_\text{n}, \text{fa}_{\text{f(n)}}, \text{y}_\text{n}\}_\text{N} \rightarrow \{\text{x}_{\text{n+1}}, \text{fa}_{\text{f(n+1)}}, \text{y}_{\text{n+1}}\} \tag{2.10}$$

Specifically, FAAS compute expected utility gain $\widehat{\mathbf{ug}}$, using past and training data comprised of additional autonomy cubes to infer the effects of flight actions. Flight path $\mathrm{fp_i}$ is the result of iterative invocations of expected utility gain given an autonomous cube, i.e., $\mathrm{fp_i} = \widehat{\{\mathbf{ug}} \circ_{\mathrm{N}} \mathrm{ac}\}$. To model throughput, we assume we access to $\widehat{\mathbf{ug}}$ and ac.

We model energy used by the aircraft with two terms. First for every step along a flight path, we sum energy used for the corresponding action. Second, multiply latency for compute by power used when hovering. The inverse applies to compute. When both compute and aircraft have distinct energy sources with known storage capacities (C_{air} and C_{cmp}), mission throughput is computed by looking at the number of missions completed before exhausting one energy sources.

Equations 2.11 shows how we calculate final throughput based on aircraft and compute energy (E_air and E_cmp). E_air can be calculated by summing the energy consumption of all individual flight actions. The energy consumption of a flight action amounts to its latency times the base power consumption of the UAV (hover power) plus the extra energy required to perform that flight action. E_cmp is similarly profiled, using compute idle power as its base. Throughput ($tput$) describes the maximum number (N) of waypoints reached in a mission, which is dependent on E_air and E_cmp. When one component runs out of energy, the system's mission completes, as shown in 2.11.

$$
\begin{aligned}
\mathrm{E_{air}(N)} &: \sum_{\mathrm{n}} (\mathrm{EnA(fa_n)} + \mathrm{LatencyC(ai(x_n))} \times \mathrm{PwrA(fa_{hover})}) \\
\mathrm{E_{cmp}(N)} &: \sum_{\mathrm{n}} (\mathrm{EnC(ai(x_n))} + \mathrm{LatencyA(fa_n)} \times \mathrm{PwrC(idle)}) \\
\mathrm{tput} &= \min \mathrm{N} : \mathrm{C_{air}} - \mathrm{E_{air}(N)} = 0 \ \ \text{or} \ \ \mathrm{C_{cmp}} - \mathrm{E_{cmp}(N)} = 0
\end{aligned}
\tag{2.11}
$$

FAAS mission	autonomous photography	search and rescue	agricultural scouting
utility functions and flight constraints	1) prioritize high utility (util) 2) prefer short missions (tput) 3) good mix		1) high utility req. 2) max waypoints (10, 20 or 30)
flight area	2x2x3x3 hyper cube near subject	15 hyper cubes in a target area	n x m grid over crop field
flight actions	translate x, y and z, gimbal pitch		translate x and y
path finding algorithm	1) choose from K-nearest neighbors 2) use A* search to avoid local optima 3) use energy aware A* (EA*) which prioritizes low power movements		
AI classifier accuracy & complexity	1) Integer precision (int) fast but less accurate, 2) floating point precision (fp), 3) deep most accurate but require GPU (all)		
execution context	edge systems, cloud, or onboard		
arch. support	scale up, scale out, gpu acceleration		

Table 2.1: Layered implementation and system settings.

2.4 Implementing FAAS

Table 2.1 decomposes FAAS and presents a layered, systems view of their components. This section presents each layer and compares system settings for 3 FAAS.

FAAS missions: We implemented (1) autonomous photography, (2) search and rescue, and (3) agricultural scouting. For autonomous photography, the FAAS positions itself and takes high-quality portraits of human faces. It autonomously explores

its flight area. This workload was inspired by computational photography and SkyDio [3, 16].

The search and rescue FAAS extends autonomous photography. It searches multiple areas for humans. During emergencies or disasters, it could help first responders discover victims. The FAAS navigates the aircraft between areas, e.g., rooms in a building, and also explores areas thoroughly.

As discussed, agricultural scouting is commercially viable today. This FAAS takes aerial images of a crop field similar to prior work [23]. For this work, we had access to a corn field, so our missions produce detailed images of corn and planting rows. Scouting as a workload kernel underlies aerial surveillance and military target detection.

Flight area and flight actions: Autonomous photography covers a 2x2x3x3 hyper cube. The aircraft can translate X, Y and Z axes and rotate the camera. We have collected 110 autonomy cubes for this benchmark. Search and rescue explores 15 2x2x3x3 hyper cubes and supports the same actions.

Agricultural scouting covers a 75-acre crop field. The flight area is a 55x43 grid. The aircraft can translate X and Y dimensions only. In total, we have collected 122 autonomy cubes (20970 images) across (1) all 3 FAAS, (2) diverse settings: outdoors, indoors, raining and windy, and (3) with multiple targets: humans and corn. Capturing a cube took roughly 11 minutes for autonomous photography cubes and 4 hours for agricultural scouting.

Utility functions and constraints: Photography and crop analysis use different utility functions. A good portrait contains a centered, bright and crisp face [48]. We created a utility function using the following features: face detection, face location in the image, image brightness and size of the facial bounding box. A good picture

of a crop field avoids blur, contrasts crops and soil and does not include extraneous objects. Our utility function here considers glare, image brightness and corn crops counted. For all utility functions, each feature is weighted and the whole function is normalized.

End users set thresholds. When utility exceeds the threshold, the FAAS mission is complete. Our photography FAAS support 3 thresholds. A high threshold encourages the FAAS to explore its flight area. As a result, missions are longer. A low threshold encourages the FAAS to land quickly. We label this setting as high throughput. Finally, the default settings aims for a medium threshold that provides good mix.

Flight area bounds FAAS flight path spatially. Flight actions that cause the aircraft to leave that area are not executed. If the aircraft battery falls below 10% of its capacity, our FAAS lands immediately. The Max Waypoints setting bounds flight path temporally. After exceeding this threshold, the mission completes.

Path finding: By default, 4000 training data entries are used to decide where to fly. Each training data entry describes a single image from a collected autonomy cube. Training data entries consist of a vector of utility features, as well as pointers to other training data entries that represent sensed data within the autonomy cube that the FAAS could reach with one motion (e.g a training data entry may have pointers to data sensed using the left, right, up, and down flight actions). This reduces dozens or hundreds of autonomy cubes containing tens of gigabytes of image data into portable CSV files on the order of megabytes.

Pathfinding algorithms run on top of our cubes and training set to model FAAS actions. The K-Nearest Neighbors (KNN) algorithm finds 9 entries with utility features nearest to the sensed image. By default, we implement greedy path finding.

The expected utility gain is the mean gain observed by nearest neighbors grouped by flight action. This approach takes the flight action with the largest expected utility gain.

A* Search improves greedy KNN with a linear heuristic to model the whole flight area, choosing a flight action along the best expected path. Energy-aware A* Search weights flight actions according to aircraft profiles. It produces flight paths that prefer low energy actions. A* Search and its energy-aware variant are well studied and have been used in recent research [55, 98, 123].

AI models: Each FAAS characterizes sensed data into a vector with up to 64 dimensions. Each dimension represents the output of an AI classifier. We distinguish classifiers by compute demand and support any subset of these groups. Integer models include OpenCV local binary pattern, cascade models using only integer data types, and RGB image classifiers. These models are lightweight, fast and imprecise. Floating point models include DLIB histogram of gradients. The are more precise than integer models but also slower to compute. Deep models include DLIB's convolutional neural network (CNN) for face recognition and our custom CNN for crop recognition. We execute deep models only when GPUs are available (i.e., not on CPUs).

Execution context and architectural support: SoftwarePilot, our FAAS suite, is composed of micro services. Each micro service provides basic functions, e.g., issuing aircraft commands, data sensing, data storage, running sensed data through an AI model, querying path finding algorithms etc. Micro services exchange messages using Californium UDP CoAP clients and servers [114]. Our suite allows for execution of autonomy cube-based pathfinding and modeling on edge or cloud systems.

2.5 Evaluation

Modeling simplifies testing of a wide array of hardware settings on FAAS throughput. Given ground truth data and profile information, the goal of our modeling approach is to make results from modeled flights and actual FAAS missions virtually indistinguishable. Our FAAS provide ground truth. We can measure mission throughput directly with real aircraft, goals, software settings and compute hardware. This section first compares our model predictions to observed throughput. Then, we compare competing modeling approaches. Finally, we isolate compute and aircraft profiles, and characterize these workloads.

2.5.1 Model validation

We flew each benchmark under the system settings in Table 2.1. Our FAAS uses the DJI Android SDK to control the aircraft via WiFi connected laptop (edge device). Our platform can also run software components across multiple devices or on the cloud. Edge devices run Ubuntu Linux 18.04.

Each test started with fully charged aircraft and edge batteries. We then flew missions until one of the batteries fell below the safe landing threshold. Observed mission throughput is the number of missions completed. We repeated each test 6 times and report mean throughput.

Unless noted otherwise, we used the DJI Spark aircraft [57]. Its body is roughly 6 square inches. It weighs 300 grams. We observed that it can hover for 11–13 minutes without recharging its 16 Wh lithium ion battery. Also unless otherwise noted, we use edge architecture setup, because it is easier to change architecture settings. We tested edge devices with the following compute architectures.

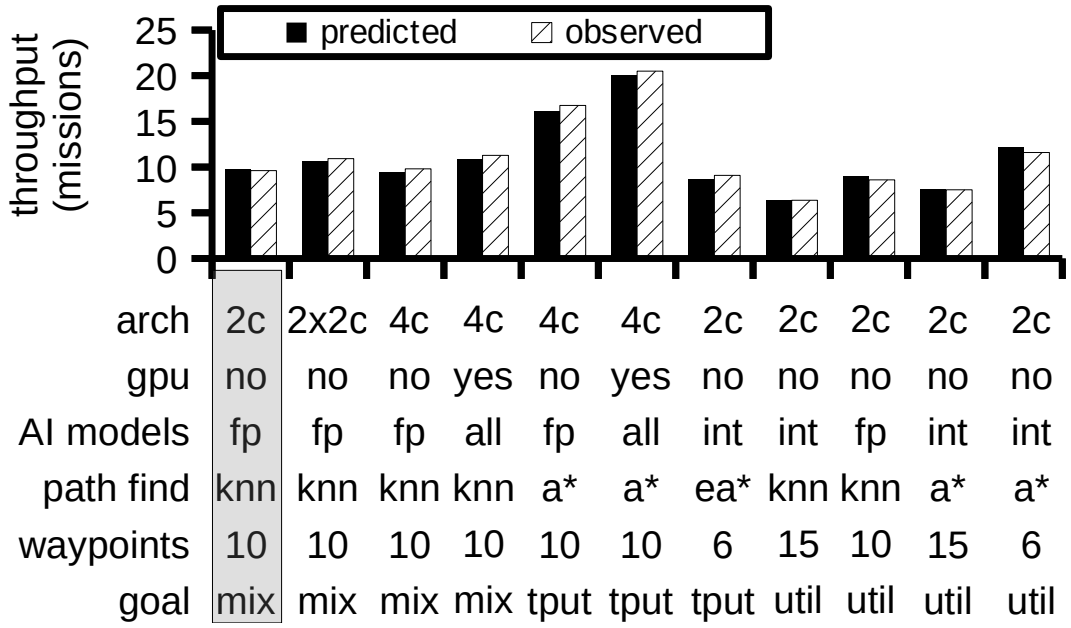

Fig. 2.4: Our model predicts mission throughput precisely. Baseline setting is highlighted

- **2c:** HP G6 laptop; 2-core i5 7200u processor; 3.1 GHz; 3 MB cache; 4 GB DDR4 RAM; 500 GB hard drive.
- **2x2c:** 2 HP G6 laptops using 1 Gbps Ethernet router. One laptop runs flight control, pulls images from the aircraft and computes integer AI models. The other laptop runs path finding algorithms and floating point models.
- **4c:** 4-core i5 7300u processor; 3.5 Ghz Ghz; 3 MB cache; 4 GB DDR4 RAM; 500 GB hard drive.
- **4ci7:** 4-core i7 7500u processor; 3.5 Ghz; 4 MB cache; 8 GB RAM; 256 GB SSD.
- **4c+gpu:** 4ci7 connects to an NVIDIA 1080 Ti.
- **2c+gpu:** GPU is connected to 2c.

Prediction accuracy: Recall, our modeling approach predicts expected mission throughput, i.e., an average over many missions. For autonomous photography and search and rescue, our approach uses autonomy cubes to produce 50 mission flight paths for each autonomy setting. Note, compute hardware settings do not affect flight paths. Autonomy settings include AI models, path finding and utility functions. For each flight path, Section 5.3 describes the workflow to predict mission throughput. Agricultural scouting covers a larger area. We have fewer cubes. Here, we generate 6 flight paths for each autonomy setting. Autonomy cubes were implemented by a micro service that returns an image from a cube waypoint in place of the aircraft camera. FAAS software interacts with the micro service as it would with the aircraft.

Figure 2.4 compares predicted and observed mission throughput for autonomous photography. We shorten the names of mission goal parameters to mix, util and tput for space. We also shorten integer and floating point settings for AI models to int and fp. Our tests cover every autonomy setting supported. Mean absolute percent error (i.e., $\frac{|\text{pred}-\text{obs}|}{\text{obs}}$) was 4%. Error can be attributed to subtle differences in flight conditions, battery age, and hardware timing between profile and test flights. We found that GPU, goals and path finding settings affected throughput by up to 1.8X, 1.75X and 1.71X in isolation. Combined, settings had complex effects. For example, adding a GPU sped up throughput by 1.15X under 4c, KNN and util. However, under 4c, A* and tput speed up was 1.23X–a 7% improvement. Util and KNN missions spent more time hovering. Energy used hovering lessened whole system speedup gained by adding a GPU.

Competing modeling approaches: We also studied modeling approaches inspired by recent research. In *Autoware* [98, 99, 123], researchers use ROSBAG recordings

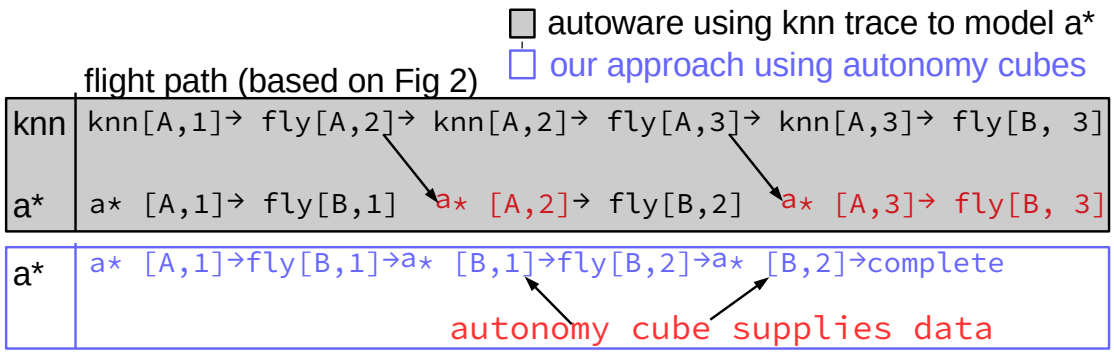

Fig. 2.5: Depicting Autoware versus our approach.

from a real, long-running self-driving car. We mimicked this approach by collecting long traces over multiple missions. For autonomous photography and search and rescue, we combined 100 mission flight paths. Scouting used 12 missions.

Autoware does not consider autonomy settings. As such, this approach does not model flight path well. Figure 2.5 depicts the problem using examples from Figure 2.2. Autoware profiles compute workloads on new hardware. However, Autoware can not acquire data outside of the trace. If autonomy settings change where FAAS would fly, Autoware doesn't have access to the sensed data and profiles using available data. Figure 2.5 highlights the problem: A* missions complete faster than KNN and Mix missions. As a result, Autoware over estimates the total compute workload.

Aerostack flies autonomous aircraft in a wide range of settings by manually inserting obstacles [168]. This approach improves Autoware's methodology, because traces include data from multiple settings. We mimicked this approach by creating 3 long running traces for each setting.

Figures 2.6 (a – d) compare our approach, Autoware and Aerostack. We also compare a simple modeling approach driven by data collected from DJI and Intel

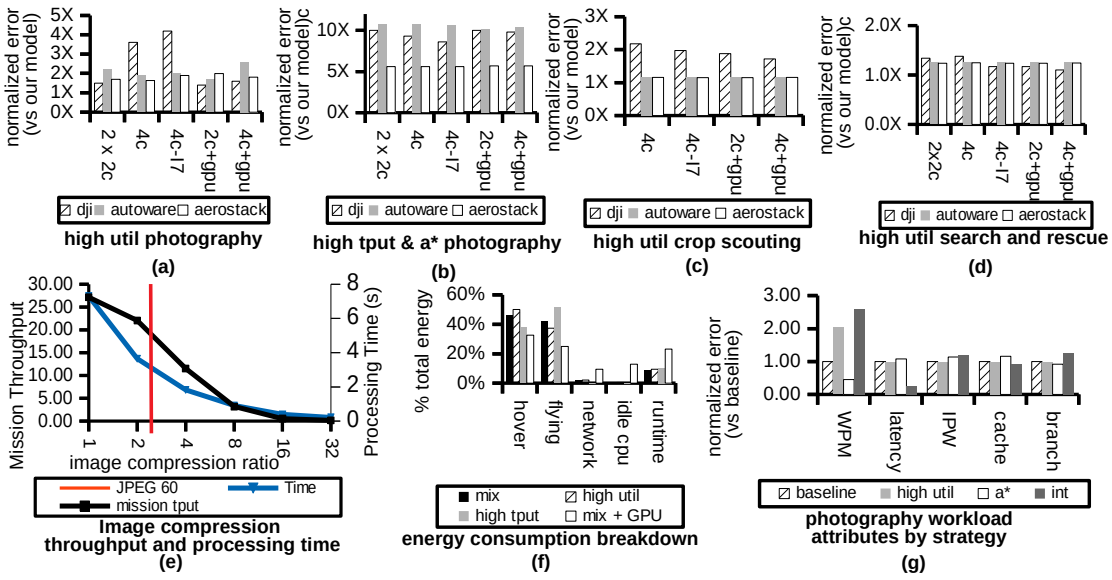

Fig. 2.6: (a – d) Comparing trace and cube driven modeling approaches. (e) poor image quality degrades mission. (f) Autonomy settings shift compute and aircraft energy demands. (g) Autonomy settings affect common architecture counters.

(*DJI*). This approach ignores autonomy and uses flight time and aggregate cycles per second to model throughput as a function of speedup, max waypoints and flight time. Autoware and Aerostack traces used missions conducted under baseline setting. Aerostack traces toggled waypoints (15) and A* for multiple traces.

Across all workloads, settings, and architectures, competing approaches increase relative error from 1.2x–10x. Workloads with high flight overhead and lower detail sensed data experienced less error than low overhead workloads. In autonomous photography, where subtle differences in pathing can cause massive differences in sensed data, sees between 1.7X to over 10X error when using other approaches.

In Figures 2.6 (a,c & d), we used a setting close to the reference trace: we changed mix to util. In these graphs, Autoware and Aerostack avoid inflating error by 2X.

Given our model predicts throughput with 4%, these results are not too bad. However, Figures 2.6(b) makes 2 major changes: we changed mix to tput and knn to A*. As shown earlier, these settings affect throughput greatly. DJI inflates relative error 10X, Autoware by up to 20X and Aerostack by 5%. These results suggest that benchmarking must account for flight path— and, more broadly, software settings related to autonomy.

Changing aircraft: To assure the validity of our modeling approach, we created and validated models for the DJI Mavic Pro as well as the Spark. The DJI Mavic Pro is a 734g personal UAV, roughly 12 inches in length. It has a 43 Wh lithium ion battery and a maximum hover time of 23-25 minutes. Mavic, with more powerful motors and processors, requires more energy to run than Spark. Across the flight components we modeled, Mavic consumes 45-55% more energy than Spark. Validation through 5 fully autonomous missions provided an average error of 3% for our Mavic model.

Image Quality: Figure 2.6(e) describes the effects of image quality on throughput. Recent UAS work suggests using high compression ratios [20, 77] (such as JPEG60) or low resolution images to speed up detection. As shown, processing times decrease with compression ratio. However, image quality degrade object detection. As a result, aircraft explore more waypoints, possibly without producing valuable outcomes.

Figure 2.6(e) shows the decrease in throughput as image quality degrades using DLIB's facial recognition CNN. At the default quality of the DJI Spark camera (12 megapixels), our FAAS can complete 27 missions per charge. At lower image quality (3 megapixels [20,77]), mission throughput has degraded 62%. This result shows that end-to-end metrics are critical in autonomous systems— results driven by processing time alone can miss whole system impacts.

2.5.2 Workload study

Figure 2.6(f) reports the impact of aircraft hover, flying, networked data transfer, idle compute and runtime software on total system energy. The aircraft accounts for 58-90%. The use of GPU increases the impact of compute by 4.6X. Table 2.6(g) delves into the architectural metrics affecting compute latency on facial recognition workloads. These data were collected on the 4c hardware using the Linux Perf tool. We observe that autonomy settings affect waypoints per mission (WPM). Integer models are too imprecise, causing the FAAS to visit many waypoints. However, integer models execute efficiently on general purpose processors, reducing the frequency of cache and branch misses by 25%. This setting provided the lowest latency, speeding up runtime by 4X.

Under A* search, the runtime executes more instructions per waypoint (IPW) before encountering cache misses than baseline setting. However, despite the lower cache miss rate, it also incurs more branch misses and executes more IPW (i.e., instructions spent computing utility gain for a sequence of actions). The net result is a 29% slowdown.

2.6 System Management

Our model can help FAAS end users: (1) manage compute hardware, (2) assess trade-offs between tightly and loosely coupled aircraft, software and hardware, and (3) adapt hardware and software at runtime.

2.6.1 Managing compute resources

Our modeling approach uses autonomy settings to construct realistic flight paths. Flight paths and autonomy cubes yield representative compute workloads. These workloads can be tested without flying the aircraft. Consider an end user that owns a commodity aircraft. This end user may ask, *which hardware resources will provide good throughput?* Reusing flight paths across competing hardware solves this problem.

When upgrading compute resources, there are 3 options. With *scale out*, compute resources are replicated and the workload is balanced across them. Upgrading from our **2c** to **2x2c** setups reflects scale out. *Scale up* replaces resources with faster or more energy efficient resources, e.g., **2c i5** to **4c i7**. Finally, workload targeted *accelerators* can augment existing resources, e.g., **2c+gpu**.

Figure 2.7(a) plots speedup achieved by scaling out, scaling up, and adding GPU using the autonomous photography FAAS. Speedup is $\frac{\text{tput}_{\text{new}}}{\text{tput}_{\text{old}}}$. For this plot, the denominator is from a **2c** processor running on a device that has 2 Wh battery. Under 2 Wh battery, only scale up provides speedup greater than the increase in system cost. If the upgrade includes a 20 Wh battery, scale out and scale up are worthy investments. GPU speedup does not match its 9X cost increase. However, a GPU provides greatest increase in throughput.

2.6.2 Comparing onboard, edge and cloud

DJI software development kits support edge architecture where tablets run AI software and control aircraft remotely [56]. For developers, these devices offer one hop, low latency access to the aircraft and powerful compute. Further, developers can procure resources as needed.

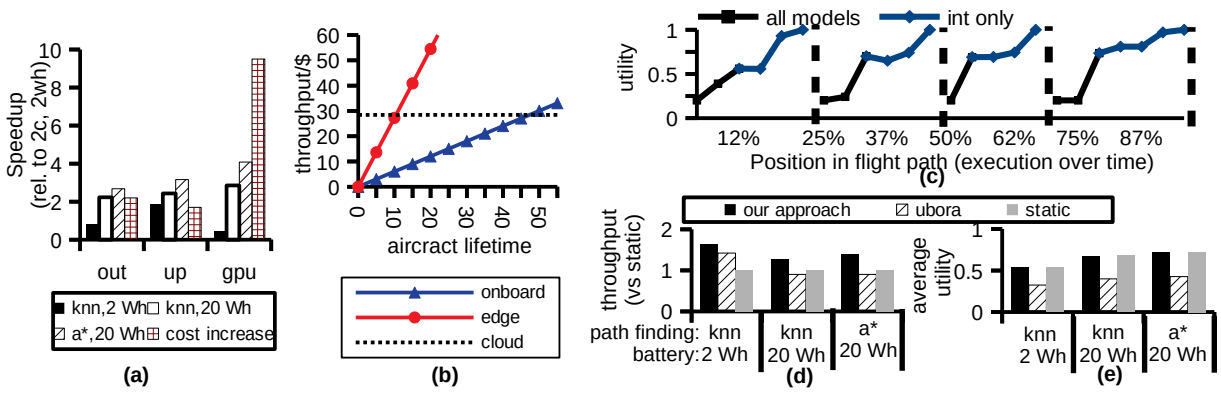

Fig. 2.7: Model-driven management of autonomous photography FAAS: (a) speedup and cost by architecture optimization, (b) efficiency of onboard, edge, and cloud systems, (c) utility of images captured across FAAS missions with adaptive model switching, (d-e) a comparison of throughput and utility across adaptive switching approaches.

Processors located onboard could provide lower latency, but there is a downside: onboard devices take energy from the aircraft, decreasing flight time. Note, flight time decreases for two reasons. First, and most directly, processors use energy for vision processing, path finding, etc. Second, more subtly, their weight increases thrust needed to take off, hover and fly. Small aircraft simply can not move enough air to carry an Nvidia 1080 Ti (1041 g). Even larger unmanned aerial vehicles would notice decreased in flight time.

The cloud is also an option. Elastic cloud services could dynamically provision resources, allowing end users to lease hardware on demand and avoid over provisioning. The downside is that slow network latency reduces responsiveness.

We extended our aircraft profiles to model flight time given added payload. The relationship between flight time and payload weight depends on nominal thrust and

aircraft weight [129]. Specifically, we modeled flight time lost to carry Intel i5 CPU, DDR4 RAM and SSD using manufacturer provided thrust and power loading data.

We compared three aircraft: (1) Spark, a 300g UAV that can carry 500g; (2) Mavic, a 734g UAV that can carry 1300g; And (3) Matrice 100, a 2400g enterprise UAV that carries 3600g. For Spark, onboard CPU and RAM would degrade flight time by 20%. The full compute system would degrade flight time by 50%. For Mavic and Matrice, the full compute system onboard would degrade flight time by 10%.

We updated our aircraft profiles to get onboard throughput. We increased energy needs for each flight action in proportion to flight time degradation caused by onboard payload. Then at each waypoint, we subtracted compute energy from aircraft capacity. For cloud throughput, we deployed **2x2c** set up using an AWS micro instance as the second processor. This led to a 12% throughput degradation due to moving images between the edge and cloud.

Figure 2.7(b) explores the relationship between throughput per dollar and aircraft lifetime (measured in missions). This figure uses the Spark aircraft and assumes that users either purchase hardware or cloud time on an instance that has a static cost per FAAS mission. Throughput per dollar of the cloud system remains static. We used AWS on-demand micro instances for pricing. Onboard and edge systems have overhead cost that cloud systems do not, but minimal maintenance costs, meaning they experience gains in total throughput per dollar as the system is used. Cloud systems also experience much higher latency than edge systems making edge systems more attractive for live FAAS processing. The crossover point is where onboard and edge systems become more cost effective compared to cloud systems. Using our 4c configuration with a high throughput autonomy setting and a DJI Spark, an

edge system would become cost effective after only 10 missions. Moving the system onboard takes 5X as long to cross over.

2.6.3 Adaptive hardware-workload co-design

End users may have many options as to which AI models they choose to deploy on their FAAS. Our benchmarks can switch between multiple models that vary in (1) recognition accuracy and (2) latency. Highly accurate models are needed to detect distant or dark objects. Less complex models suffice for clear, crisp images. However, highly accurate DNN with DLIB (deep models) require a costly, power hungry GPU. We also use the OpenCV LBP cascade classifier (int models) which, when run on a 2 core laptop, has lower latency than DNN, but also lower accuracy. Deep models can find small, unclear faces in large, noisy images, but as images become clearer, it's performance converges with that of int models.

As the performance of Deep and Int models converge, it is prudent to turn off the GPU and use only the faster Int models. This approach conserves edge battery and increases throughput by decreasing feature extraction latency.

Figure 2.7(c) depicts an experimentally obtained example mission sequence where the GPU is duty cycled. We set a utility threshold of 0.5, turning off the GPU and using Int models only after a 0.5 utility image was found. **All** signifies waypoints where deep models were computed whereas **Int** signifies waypoints where only Int models were computed. For waypoints occurring after the duty cycle threshold, Int models and deep models performed similarly, finding images at comparable utility and choosing the same paths.

Figure 2.7(d-e) explore the differences in throughput and utility of duty cycling GPU using 3 different policies:

- **Our Approach:** Assigns a user defined threshold for duty cycling. Once one image in a flight path exceeds that threshold using the DNN model, the GPU is turned off and the LBP model is used for facial recognition.
- **Ubora:** Mimicks adaptive quality management in recent research [101, 102]. Each mission is treated as a query. GPU and Int models are toggled once at the start of each mission. Average utility taken over the flight path is compared to a duty cycle threshold. If average utility exceeds the threshold, GPU is turned off until average utility falls below the threshold.
- **Static:** Uses deep models for all feature extraction, with no GPU duty cycling.

Figure 2.7(d) shows a 1.3X gain from using our duty cycling approach as compared to the static approach, and a 1.4X gain as compared to Ubora when using either A* configuration. The A* configurations both have large enough edge batteries such that they are bottlenecked by the UAV battery, so gains or losses in throughput are entirely dependent on execution time savings during feature extraction, which are realized by the GPU configuration. The Ubora approach sees a decrease in throughput as compared to both others. Using a cumulative utility threshold allows for the Ubora approach to miss local utility spikes in a high variance workload like UAV data collection. In our test configurations, Ubora duty cycled the GPU either too early or too late. Duty cycling too late (after integer models and deep models converge) causes Ubora to function like our approach, but with more GPU usage. Duty cycling too early potentially switches to integer models before accuracy converges, taking more waypoints on average to meet utility goal. This affect can be seen in Figure 2.7(e),

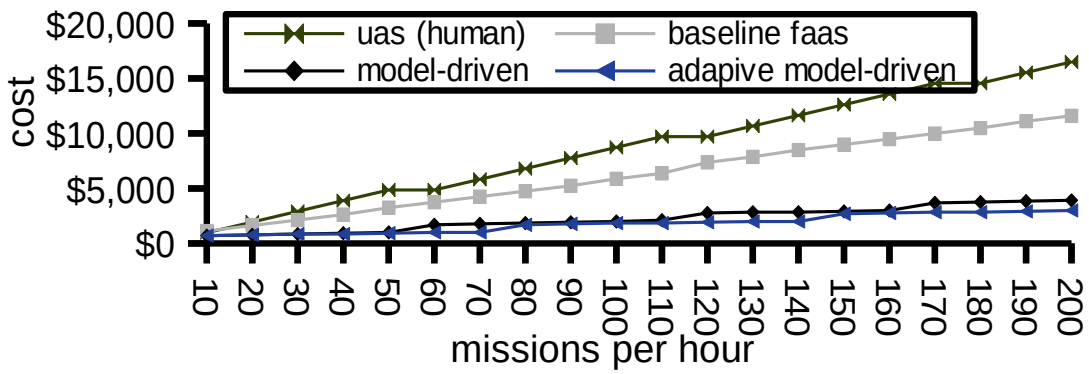

Fig. 2.8: Agricultural scouting cost as end users use parallelism to increase throughput. Our model-driven edge management provides savings at scale.

where Ubora sees considerably lower average utility than both our approach and the static approach. The failings of the Ubora approach in this context, contrasted with the success of our simpler approach, demonstrate that while duty cycling models and hardware in FAAS workloads can be advantageous, one must carefully choose their duty cycling approach.

Average utility across autonomy setting is also important. As our architecture, models, and path-finding algorithms improve, so does average utility. Our KNN configuration sees 0.83X lower image utility as compared to A* using the same models. When transitioning to a deep model on the GPU configuration, we see a 1.06X improvement in average utility which can be attributed to the higher accuracy of the deep model. Using our low battery configuration (where edge battery is a throughput bottleneck), we see that our approach makes a 1.65X improvement over a non-duty cycling approach, and a 1.15X improvement over Ubora.

2.6.4 Speedup for autonomous photography

Our model-driven approaches found highest throughput using 4-core Intel I7 with GPU while using low utility threshold, adaptive duty cycling and A* search. This setting completed 34 missions using Spark without recharging. Compared to 2-core Intel I5 using greedy KNN to find high utility images, the 4-core setting sped up compute latency by 15X. Looking deeper, the following changes were significant:

- **Autonomy settings:** A* search and lower threshold reduced waypoints per mission, providing 4.1X speedup.
- **Using a GPU:** Up to 2.25X over other approaches.
- **Software driven power management:** 1.3X throughput increase over static GPU usage.

Combined, the best settings yield a 10.2X increase in mission throughput compared to the 2-core setting mentioned above.

2.6.5 End-to-end savings for crop scouting

Our scouting FAAS covers roughly 1-acre per mission and completes 15 missions per hour. 1 FAAS would require 10 hours to scout a 150-acre field. However, multiple FAAS can work in parallel to scout the whole field faster. Given a deadline, we can estimate total hardware and software cost for all parallel FAAS. In contrast, UAS require human piloting. Based on first hand experience, we assume human pilots can execute 11 lawnmower missions for $20 per hour [115]. We also model cost for UAS equipment such as batteries, compute resources, and aircraft.

Figure 2.8 shows the cost for parallel UAS and baseline FAAS to achieve x mission throughput per hour. Baseline FAAS (2c, autoware, KNN) outperforms human

piloted UAS. additional equipment and labor costs inflate UAS costs. Our model-driven approaches improve mission throughput significantly, gaining 6X and 4.2X on UAS and baseline. Adaptive GPU power cycling provides further improvements. Model-driven, adaptive FAAS reduce costs by 87% compared to human-piloted UAS.

2.7 Limitations and Future Work

Our Autonomy Cube modeling approach and analysis includes many limitations and future opportunities. First and foremost, autonomy cubes can be difficult to collect. We flew over 100 FAAS missions to collect autonomy cubes for both FAAS modeling and as input to pathfinding algorithms. This is not feasible for all FAAS tasks. Future work should explore the creation of autonomy cubes from extant geo-tagged multidimensional image sets. For instance, large available datasets like Google Street View [8], or autonomous driving datasets like KITTI [76] could be used to construct autonomy cubes for a multitude of relevant FAAS tasks.

Many of our management strategies focused on decreasing compute power consumption of a single FAAS. Multiple FAAS, i.e., swarms, can share edge compute systems while each UAV carries its own battery on board. The aggregate compute demands of a large swarm could transform power usage, making edge system batteries the bottleneck resources. Our approach can adjust battery sizes, but we can not model how swarms will inflate compute demands.

2.8 Related Work

FAAS choose their flight path at runtime similar to self-driving cars. Workload settings that affect their flight path can change energy usage and throughput significantly. We quantified this and proposed autonomy cubes to capture representative traces when settings change. Autoware is a project designed to make autonomous driving more open [98]. Autoware presents open algorithms, libraries, and consumer hardware components for autonomous driving, many of which are applicable to FAAS. It's motion planning design, as referenced in section 5, was improved upon in this paper. Lin et. al extend Autoware to study accelerators [123]. Object detection and tracking for self-driving cars can be sped up 169X using consumer grade hardware, but compute speedup can reduce driving range. This result parallels our observations with mission throughput. Aerostack [168] presents an open source, component based software architecture for aerial robotics, emphasizing full autonomy. Aerostack's design influenced the design and implementation of our own FAAS software.

Other recent studies explore acceleration and edge devices. Sirius [86] studied FPGAs, CPUs, GPUs, and coprocessors on personal assistant benchmarks. In-situ AI [183] studied autonomous IoT. Computational sprinting has targeted interactive, mobile workloads with dynamic architectural optimizations [146, 147].

2.9 Conclusion

Unmanned aircraft are changing industries from agriculture to surveillance and photography. Fully autonomous aerial systems are piloted by software—eschewing costly and mistake prone human piloting. Software and hardware settings affect where these systems fly and when missions complete. Recent benchmarking papers use few settings, e.g., from prior traces, but extrapolate throughput broadly. This paper presents a modeling approach that can model flight paths across autonomy settings. Autonomy cubes provide sensed data for any reachable waypoint, enabling our approach. We have collected autonomy cubes for real FAAS executing diverse missions across a wide range of settings. Our model predicts FAAS throughput within 4%. We used our model to evaluate system management problems and uncovered insights that can improve throughput 10X and FAAS reduce costs 87%. Code for our modeling approach and autonomy cubes are open source, made available through the SoftwarePilot project.

Chapter 3: Autonomic Computing Challenges in Fully Autonomous Precision Agriculture

Precision agriculture examines crop fields, gathers data, analyzes crop health and informs field management. This data driven approach can reduce fertilizer runoff, prevent crop disease and increase yield. Frequent data collection improves outcomes, but also increases operating costs. Fully autonomous aerial systems (FAAS) can capture detailed images of crop fields without human intervention. They can reduce operating costs significantly. However, FAAS software must embed agricultural expertise to decide where to fly, which images to capture and when to land. This paper explores *fully autonomous precision agriculture* where FAAS map crop fields frequently. We have designed hardware and software architecture. We use unmanned aerial systems, edge computing components and software driven by reinforcement learning and ensemble models. In early results, we have collected data from an Ohio cornfield. We use this data to simulate a FAAS modeling crop yield. Our results (1) show that our approach predicts yield well and (2) can quantify computational demand. Computational costs can be prohibitive. We discuss how research on adaptive systems can reduce costs and enable fully autonomous precision agriculture. We also provide our simulation tools and dataset as part of our open source FAAS middleware, SoftewarePilot.

3.1 Introduction

In 2050, the planet will support 9.7B people. People (and the livestock they eat) will demand 1.7X more food [79]. However, rising sea levels and city sprawl will decrease arable land in growing countries. Atmospheric CO_2 will dampen food production as climate change worsens [165, 222]. Producing enough food and providing access to it is a grand challenge.

Food demand will require larger crop yields per arable acre. Today, only 60–80% of planted seeds yield edible crops [170]. Nitrogen deficiency, hydration, crop diseases, pests and fungi degrade yield. *Precision agriculture* uses satellites, aircraft, soil sensors and digital weather stations to sense field conditions, monitor crop health, detect problems early and improve crop yields [194, 215]. For example, Integrated Pest Management uses images collected from aircraft to detect and count insects and vermin in a crop field [62]. Armed with this data, farmers can apply pesticides parsimoniously to reduce costs and sustain the planet. More generally, we broadly define *yield maps* as whole-field characterizations of crop and/or soil health. One goal of precision agriculture is to produce yield maps which farmers may use to manage crops.

Crop health changes over time because nitrogen levels diminish, pests emerge and water needs vary. Yield maps produced frequently can detect changes promptly. Problems detected early afford cost effective solutions. However, frequent mapping is costly and can exceed savings. Figure 3.1 plots costs to map a 250-acre corn field frequently (weekly) for 12 weeks. In comparison, current practice creates a field map once per growing season [105]. Corn fields in Ohio USA profit $148 per acre on average [70]. Figure 3.1 plots costs relative to profit. Airplane pilots charge $5–$10

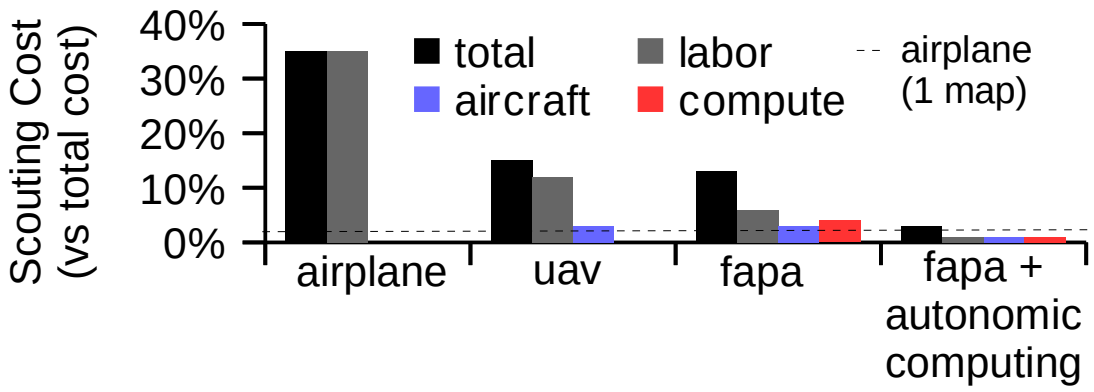

Fig. 3.1: Fully autonomous precision agriculture reduces and shifts costs.

per acre to produce maps [74,155]. At $5 per acre, 12 whole-field maps would cost 40% of profits. Unmanned aerial systems (UAS) use small aircraft and do not carry pilots onboard. Instead, they are piloted remotely. Figure 3.1 assumes UAS pilots charge $20 per hour and cover 12 acres per hour. Frequent mapping using UAS consumes 14% of profits. Despite remote piloting, labor associated with piloting would account for 12% of profits.

Fully autonomous aerial systems (FAAS) eschew human piloting. Instead, FAAS software manages flight, captures images and analyzes data. FAAS software can fly more safely and efficiently than human pilots. Experts agree that fully autonomous systems will replace unmanned aerial systems [135,196]. However, as shown in Figure 3.1, the economic case for precision agriculture is complicated. We model fully autonomous precision agriculture (FAPA), i.e., FAAS that produce yield map reports. FAPA reduces labor costs significantly from $20 per hour to $10 per hour, but the compute resources needed to execute FAAS software increase total costs. Naive FAPA accounts for 13% of profits.

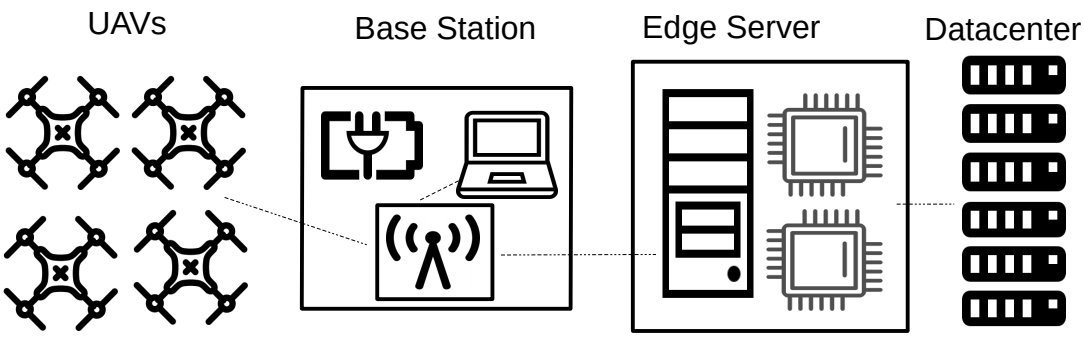

Fig. 3.2: Core agricultural FAAS system components.

Autonomic computing systems adapt their execution at runtime, speedup compute workloads and use fewer resources. FAPA systems can use autonomic computing to reduce labor and compute costs. As shown in Figure 3.1, if autonomic computing techniques can reduce costs by 66%, frequent mapping would be cost competitive with current practices. We contend thatRosa autonomic computing is the missing ingredient in FAPA systems.

This paper reports design and implementation for an early FAPA prototype. Our FAPA system uses consumer-grade UAV, edge and cloud infrastructure. Reinforcement learning and deep feature extraction are key software components. FAPA must compare recently sensed data to anticipated outcomes, because image analysis alone does not fully capture utility for reinforcement learning. Our prototype uses ensemble models. Each model correlates to expected yield. Variance among models captures utility. When our prototype achieves desired utility, it lands and produces a yield map. Our reinforcement learning approach intelligently samples the field and produces a yield map without exhaustively exploring the field in its entirety.

We evaluated our FAPA system on aerial, cornfield images. Each image represents a GPS location. The FAPA aircraft flies between recorded locations. At each location, the aircraft computes its reinforcement learning workload and improves its yield map. Our FAPA system produced low-error yield maps. Error decreased with sample size, but converged when 40% of the field was sampled. In terms of cost, accelerated computing systems inflate hardware costs but also execute faster and reduce hourly labor. Early analysis of the FAPA workload reveals opportunities for autonomic computing. For example, self-aware integration between FAPA systems could reduce sample size. Edge to cloud bursting could reduce hardware costs.

All software tools used to collect and extract our dataset and simulate FAPA can be found as part of our open source FAAS middleware SoftwarePilot, which is hosted on Github [26].

3.2 Design

In this section, we outline a hardware and software architecture for FAPA. The hardware design considers agricultural settings and resource constraints. The software design employs reinforcement learning to manage flight actions, e.g., fly north, south, east, west and land.

Hardware Architecture: Figure 3.2 depicts the main hardware components. Unmanned aerial systems (UAS), base stations, edge gateways, and data centers play unique roles. Below, we describe each.

UAS provide remote sensing. They capture and store images and tag their geographical position (e.g., using GPS). Modern UAS cameras support still images and video across a number of spectra. UAS fly between positions without human pilots,

reducing operating costs. However, their battery life is limited. Modern UAS can fly 20–40 minutes but recharge 60–80 minutes. UAS have onboard processors, and can perform lightweight processing on-board. Complex processing is offloaded to support long battery life and reduce execution delay.

Due to power constraints, maximizing the utility of UAS flight is important. The number of images required to map an acre of cropland has a quadratic relationship with the spatial resolution of the images taken. The spatial resolution of the images taken must be chosen thoughtfully depending on the application. Some systems may only need centimeter precision for tasks like yield monitoring, while more complex pest and disease detection systems may need sub-millimeter precision to properly discern features. The average area covered per UAS mission is dependent on the precision of each image, and is a primary metric that designers should account for when considering UAS storage capacity, charge time, classification needs, total scanning time, total number of UAS, and edge compute resource provisioning.

Base stations enable communication between UAS, power edge computing systems and recharge UAS in the field. The electric grid and/or sustainable off-grid power supplies base stations. As a result, the base station houses wireless networks and edge computers. FAPA require long range wireless networks capable of communicating with multiple UAS over many acres. In swarm setups [164], UAS may proxy communication through base stations. The base station can also provide docking and charging for UAS. Route planners can use in-field charging stations to avoid long return flights to the UAS home base. Configuration of the base station varies across applications and fields. Specifications of radio frequency, charging dock, and

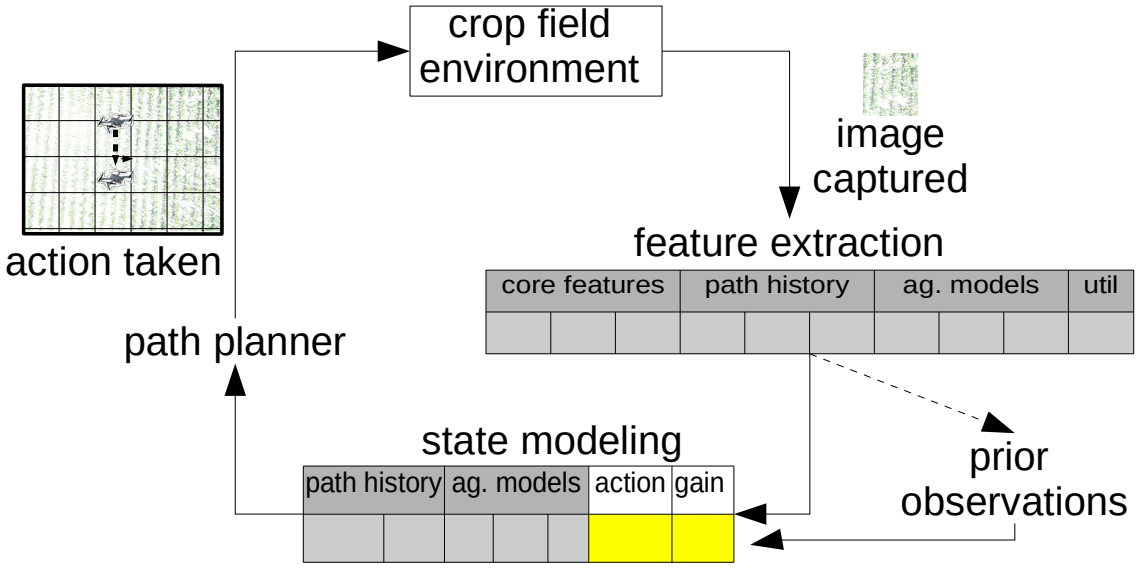

Fig. 3.3: Reinforcement learning underlies our approach.

edge gateway link depend on farm size, distance from the gateway, compute resource power constraints, and the number of UAS supported.

[t]

Edge servers pull images from UAS, extract pertinent features, direct high-level flight paths and produce yield maps. They are the computational engine for FAPA workloads. Workload kernels hosted on edge servers include: image analysis, deep learning, path planning, crop modeling and flight API management. These computationally intensive kernels can require powerful processors and accelerators, e.g., GPUs. Edge servers can have significant energy demands. These demands can stress base stations powered by sustainable, off-grid sources, e.g., power cells or renewable energy. Edge servers may also draw expensive consumer electricity from electric grids.

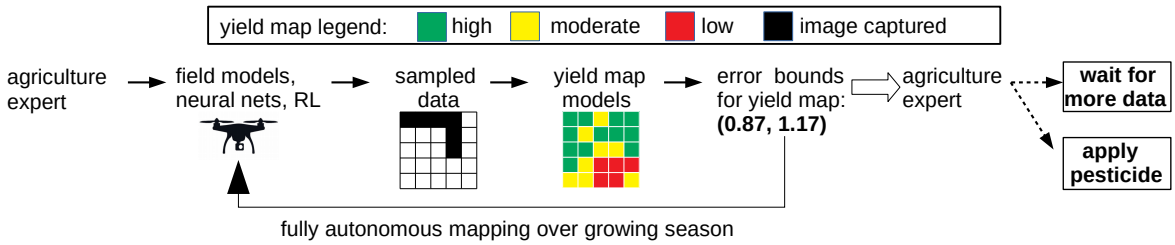

Fig. 3.4: Fully autonomous precision agriculture

Edge servers are constrained by base station power supplies. Large clusters are not permissible, but at times, edge servers may suffer large compute workloads. For example, a swarm of UAS executing FAPA may simultaneously capture images that require heavy processing. Data centers can offload demands when Internet connections are available. Complex feature extraction, i.e., the execution of deep neural networks, SVM and other classifiers, are attractive offloading workloads. Offloading also helps when many models must be run and may be used adaptively, e.g., only when load is high enough to cause service level violations. Cloud offloading requires bi-directional Internet connection capable of transmitting images and classification results rapidly.

Software Architecture: As shown in, Figure 3.3, FAPA software executes feature extraction, state modeling and reinforcement learning. FAPA software is fully autonomous— humans neither fly nor direct UAS. The workflow is (1) collect data, (2) extract features related to FAPA goals, (3) determine if goals have been met, and (4) create new flight paths such that future sampled data will fulfill modeling goals.

We use the term features extraction broadly. Any image or data processing reduction constitutes feature extraction. Examples include geographic location (data

processing), object detection (image processing), pose estimation (image and data processing) and simple brightness and contrast computations. Feature extraction procedures differ greatly in compute needs and their ability to be reused across applications and fields.

Feature extraction produces a *feature vector*, i.e., a vector containing numerical values representing each extracted feature. The feature vector is used to compute *utility*, i.e the usefulness of an image/state toward accomplishing FAPA goals. Utility calculations are customized for each precision agriculture objective. Usefulness clearly depends on purpose. Once a utility value is obtained, the FAPA system uses the utility value along with historical information to model the state of its explored area. This may include updating maps of sensed information, or using high level models to gain meta-information about the collection of feature vectors the FAPA system has already collected. This state model can be used to determine whether the user-provided FAPA system goal has been accomplished and the UAS must land, or whether more data must be collected.

Fully Autonomous Precision Agriculture: This sensing, extracting, modeling, and pathfinding cycle is repeated until the FAPA system runs out of data, a system component runs out of power, or the FAPA system goal is met. Many FAPA systems can be composed using this basic design. In this paper, we implement a FAPA system to model crop yield in corn fields.

Figure 3.4 describes the design of the yield-modeling FAPA system. The goal of this system is to generate a yield map of the crop field. Due to the size of crop fields and the lack of resources available in them, it is wise to sample the field, taking pictures of only a percentage of the field to predict yield. Using our FAPA system

design, we have created a system that can sample a crop field based on the goal of yield modeling and extrapolate sampled ground truth points into a yield map of the entire field with low error.

To do this, the system must have appropriate feature extraction models, state models, and historical training data. Green Excess and Leaf Area Index are strongly correleted with yield. We use them in our feature extraction procedure and to create yield maps. The details of these models are orthoganol to this paper. We refer the reader to [105] for details.

Our FAPA goal targets yield map error. Low error maps bolster confidence for costly management choices. Hence, high utility samples— i.e., useful images— reduce yield map error. However, a problem emerges: how can we compute utility without knowing true yields in advance?

We estimate error by using an ensemble modeling approach. We assume whole-field yield error and aggregate variance coverge comparably with sample size. We use multiple models to obtain better predictive performance, i.e., ensemble modeling. Once the ensemble models converge beyond a user defined threshold, we consider our map complete. If the1.2 Contributions . . . . . error of the ensemble models is above the user defined threshold, we fly to a new location and sample the field again. Using Green Excess and Leaf Area Index, we can calculate utility gain by estimating the change in error for our ensemble models for each possible path the UAS may take. The direction which decreases ensemble error the most is chosen.

Algorithm 1 FAPA Simulation

```
1: yieldmap ← empty
2: i ← starting image
3: ensembleError ← ∞
4: while ensembleError > threshold do
5:     featVec ← extract(i)
6:     yieldmap[featVec[lat],featVec[lon]] = featVec[GE]
7:     errVector ← findNextFlightAction(featVec)
8:     dir = min(errVector)
9:     i = nextImage(i, dir)
10:    imagesMapped ← error(dir)
11: return extrapolate(yieldmap)
```

3.3 Implementation

Using this design, we have implemented a FAPA system that is capable of generating such yield maps in simulation. Our simulation environment relies on data sensed from real UAVs to generate yield maps. Our dataset consists of over 10,000 12 megapixel images captured at 33 feet over a 75 acre Ohio corn field at a spatial resolution of $4mm^2$. Our images were captured at 3 points in the growing season, June, August, and September. The images are all geo-tagged, allowing us to place them in a virtual map. Using this data set, appropriate feature extraction, FAAS control algorithms, and ensemble error models, we fully simulate a FAPA system.

Our simulator takes the series of coarse steps denoted in Algorithm 1 to replicate FAAS action. The simulator described takes three arguments: a set of geo-tagged images representing a crop field, a starting image in the image set, and an ensemble model error threshold. The simulator will use these arguments to output an estimated green excess yield map. Algorithm 1 starts by initializing an empty yield map, a starting image (i), and ensemble error. The yield map is initialized as a 2 dimensional

array of zeros. Each cell in the array represents an image in the image set. The goal of the FAPA simulation is to fill this map with real and extrapolated green excess values that can be used to predict yield. Until ensemble error falls below the user provided threshold, it loops over the sampling procedure.

The sampling procedure begins by extracting features from image i. The goal of feature extraction is to retrieve from an image a vector of floating point values representing specific, user defined features that are useful for pathfinding, yield estimation, and geo-location. Our simulator extracts 15 features, including GPS location, green excess, leaf area index, and corn detection. These features are represented in the *featVec* variable.

The next step in the sampling procedure is to add the information learned from feature extraction to our yield map. To do this, we add the green excess value of i (GE) into the position in *yieldmap* corresponding to the latitude and longitude of i (*Lat, Lon*) obtained from the feature vector.

Once the yield map has been updated, we must find the next image to sample. To conserve energy and sample efficiently, we choose this image based on the images directly adjacent to i. To find the next image, we must find the next flight action (i.e a movement north, south, east, or west) then find the image corresponding to that movement. To find the next flight action, we use a reinforcement learning approach. Our approach uses a series of feature vectors from past FAAS executions to make pathing decisions. For each feature vector, information is also known about the possible flight actions for that position. For our simulation, our feature vector data set includes the green excess of all adjacent images. This information can be used

to calculate utility gain, a quantification of improvement of our model based on the addition of that feature vector's information.

Using reinforcement learning, we can estimate the utility gain for each flight action using similar prior execution data without actually sampling each flight action. To find similar execution data to a feature vector, the simulator runs the K-Nearest Neighbors [92, 213] algorithm using our feature vector data set as the reference set and the current feature vector as the query. This returns the K most similar feature vectors in the data set to the current feature set. These feature sets are used to calculate utility gain.

To determine the best direction to choose for sampling, the simulator uses the green excess of each of the K-Nearest Neighbors to minimize the error in its unfinished yield map. Map error is estimated using ensemble models. Ensemble models are very simple formulae that converge when the map is complete. In the case of this FAPA simulation, ensemble models all converge to mean green excess. We use five models to make up our ensemble: mean, median, 95% confidence interval min and max, and coarse mean (i.e $(min + max)/2$). To determine ensemble error, we find the range between the ensemble model values. For each flight action, we calculate the average ensemble error when adding each of the K-nearest neighbors of the current feature vector to the current yield map. The flight action which decreases ensemble error the most is chosen. In Algorithm 1, the *findNextFlightAction* function takes the feature vector, finds its K-Nearest Neighbors using our feature vector data set, calculates ensemble error, and returns a error vector containing the average ensemble errors of each flight action. The direction with the smallest error is chosen.

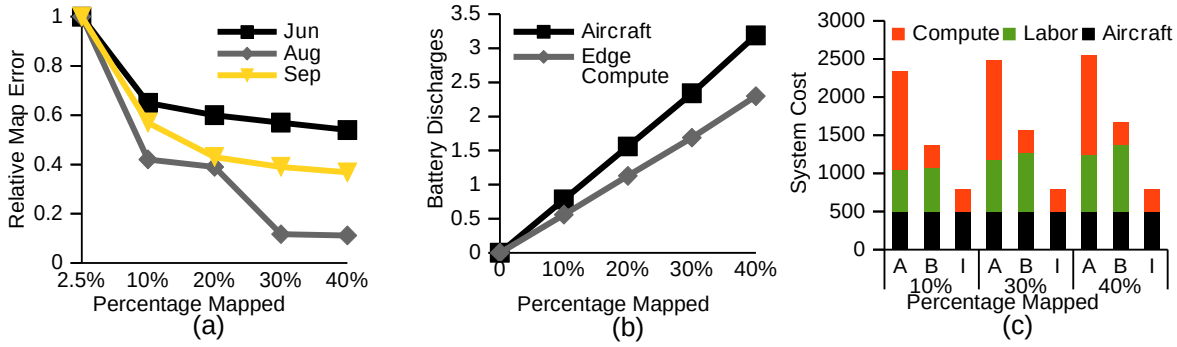

Fig. 3.5: Early results: (a) Yield error decreases as sample size increases, (b) FAPA aircraft and compute have comparable energy demands and (c) low cost FAPA requires balancing compute and labor costs. Sampling percentages are relative to a 75 acre field.

Once the flight action is chosen, the next image is selected based on the current image and flight direction, and is assigned to i. This sampling procedure continues until the user provided amount of images are sampled. Once sampling concludes, empty regions of the yield map must be filled in. This is done using recursive dilation.

The extrapolate function in Algorithm 1 recursively dilates the unfinished yield map until it is full. The dilation procedure begins by creating a copy of the current yield map. The procedure iterates over the original map, looking for empty squares with full indices in their eight-connected neighborhood. Once an index fitting this description is found, the mean of its full eight-connected neighbors is added to the copy map. Once all indices have been checked, the copy map is evaluated. If all indices of the copy map are full, it is returned. If any index in the copy map is empty, it is then extrapolated and its extrapolated version is returned.

3.4 Early Results

The simulator supplied was sensed data from 7,000 cornfield images composed into 1350 sample runs. We profiled energy demands for aircraft and compute offline. For aircraft, we repeatedly tested flight commands in each direction in isolation and use average energy demand. For FAPA software, we measured delay and power on a Lenovo Thinkpad T470 with Intel i7 7500u CPU and NVIDIA 1080 GPU. We did not use cloud offload for these tests. Figure 3.5 shows early results pertaining to yield error, energy demand, and cost.

Figure 3.5 (a) describes the sum of squared error (SSE) between our extrapolated yield maps and ground truth yield maps. We normalized to SSE of a 2.5% sampling size. Figure 3.5 (a) shows extrapolated yield map error decreases as sampling area increases. The selected sampling points reflect knee-points in our ensemble of models. Variance in the ensemble converges with SSE. This property generalizes our reinforcement learning approach, because utility models can transfer across fields and applications, compute local ensemble variance and identify sample sizes that yield good accuracy. For example, consider accuracy reported across monthly data sets. We are able to reuse ensemble models while still finding good sample sizes. The correlation is imperfect. August data converges more quickly September and June. Future work will explore the factors affecting convergence and look for techniques to further align local model calculations with global objectives. In addition, we observed that our pathfinding algorithm (nearest neighbors) performed poorly. At times, the FAPA system took clearly wrong turns. We believe that redundancy elimination in selected features, i.e., PCA, could improve pathfinding. Algorithms like A* search will help also.

Figure 3.5 (b) depicts estimated energy consumption. The aircraft is a DJI spark with a stock battery. Aircraft discharge slightly outpaces edge discharge. FAPA composed with battery powered edge servers must include substantial off-grid batteries to avoid frequent recharge. For our 75 acre field, an aircraft can map about 10% of the field before recharge.

While the edge system can be easily provisioned with greater battery capacity, aircraft must land to be recharged, or have their batteries swapped. Highly provisioning edge system batteries may be an important step towards realizing FAPA. A highly provisioned edge system should be able to control multiple UAS, increasing energy demands linearly. For large fields, links from the edge to the cloud may be required to offload computation to meet objectives for all aircraft.

Figure 3.5 (c) shows the cost of implementing FAPA systems. *System A* is an accelerated FAPA system, equipped with one UAV and an edge device equipped with an accelerator (GPU, FPGA, etc). *System B* is a basic FAPA system, equipped with an edge system with only a simple CPU. *System I* is an ideal FAPA system, including autonomic software components that allow the system to operate labor free.

The cost of each system is broken down between aircraft, compute, and labor. Each system requires the same aircraft and basic compute hardware. Systems A requires accelerators that decrease the runtime of our FAPA algorithms and increase system throughput. Systems A and B also both require labor. Without autonomic software components, workers would be required to set up, execute, and take down the FAPA system. System I eschews this cost by implementing autonomic policies in software. Workers being paid $10 an hour would cost hundreds of dollars over the course of the season performing tasks that can be performed by autonomic software

Topic	Systems Layer	Adaptive Computing Challenges
Self-Aware Integration	Autonomic Management	Seamlessly integrate features extracted by other devices on nearby fields
Quality-Adaptive Models	Autonomic Management	Speedup feature extraction when accuracy does not degrade outcome
Load Balancing	Middleware	In swarms, dynamically re-partition fields to avoid long recharging delays
Adaptive Power Management	OS/Architecture	Duty cycle power hungry devices while keeping execution time low
Edge to Cloud Bursting	OS/Architecture	Use cloud resources to multiplex thin edge clients across multiple fields
Micro-UAV Swarms	Cyber physical	Reuse smaller, cheaper aircraft over multiple mapping missions and fields
Performance Modeling	Pervasive	Model end-to-end execution time of mapping missions

Table 3.1: Autonomic computing opportunities in FAPA.

alone. Labor costs increase as sample rate and field size increase. The addition of autonomic software components to intelligently manage aircraft charging, scheduling, and dispatching, as well as data movement and reporting could considerably cut the cost of FAPA system implementation, especially for large farms.

3.5 Discussion

Table 3.1 outlines autonomic software techniques that could greatly reduce FAPA costs. There are opportunities across the software stack. In this section, we highlight a few opportunities to motivate future work.

Self-aware system integration seeks autonomous component reuse, integrating components without human programmer intervention. FAPA could use self-aware integration to expand their feature vector. Features computed by FAPA on nearby fields could influence where FAPA systems fly and when they land. Integrating more features can reduce sample size required for low yield error which reduces labor costs. The challenge is to produce a scalable infrastructure for feature sharing in resource constrained agriculture settings.

Quality-adaptive models [101,102,117] allow multi modal feature extraction. One mode uses computationally intensive but accurate models. Other modes use less intensive models when inaccurate features do not affect FAPA outcomes. Quality-adaptive models seek to switch between models to conserve battery life on edge and aircraft. The ability to choose lower quality models in situations that allow for them should speed up FAPA compute time, decreasing labor costs or allowing for an increase in sampling percentage and relative battery life.

Self integration and quality-aware models necessitate a new system layer between middleware and application. We propose the Autonomic Management Layer, a new system layer between application and middleware responsible for the management of autonomic software components. To maximize the utility of a FAPA system, an ensemble of autonomic components must be added to decrease labor expenses and increase sampling and mapping accuracy. These components are too high level for the middleware, which handles data movement, aircraft flight, and classification. Conversely, they are too low level for the application layer, which handles user interaction and reporting. The new layer is required to interpret user provided data like field maps, utility functions, and classifiers, into middleware actions that implement high level autonomic policies.

Adaptive power management duty cycles power hungry devices including: unused aircraft, GPUs and accelerators and compute processors. This technique prolongs aircraft and edge battery life and avoids long recharging delays. To be sure, aircraft consume energy hovering idly. Computation delays affect energy usage on edge servers and aircraft. Smart power management prolongs both aircraft and edge batteries (or reduces edge energy costs). For example, by autonomically decreasing the

number of active aircraft in a swarm or the number of accelerators operating [147,199], classification and pathing latencies will change and impact system life. When combined with quality-adaptive models, adaptive power management may be able to save power at zero cost to system performance. In addition, lessons from sustainable computing where multiple energy sources power compute (and now devices) are pertinent [119, 130, 187, 204].

Edge to cloud bursting also requires interaction between the autonomic layer and OS/Architecture layer. The FAPA model described in this paper uses relatively simple models to cover a relatively small area with one aircraft. If the number of aircraft, size of the farm, or model complexity increase within reason, it is plausible that normal edge systems will not be able to process all of the images they receive quickly enough for aircraft. Aircraft will have to hover, wasting power, waiting for instructions from edge systems. The autonomic and OS/Architecture layers can work together to instead move data to the cloud when aircraft hover times are too high.

3.6 Conclusion

Precision agriculture is increasing crop yields across the globe in an effort to feed our growing population. Using FAAS in precision agriculture applications can considerably reduce labor costs and increase field mapping frequency. In this paper, we present a design of a FAPA system along with a simulated implementation. We examine trade-offs concerning accuracy, power, and cost of the total system using profile information from real FAAS missions. We demonstrate that our FAPA system can produce low-error yield maps by sampling 40% or less of the total crop field, decreasing the energy and labor costs of a FAPA implementation. We also identify

autonomic computing mechanisms that can help decrease labor and hardware costs of FAPA systems further.

Chapter 4: Adaptive Autonomous UAV Scouting for Rice Lodging Assessment Using Edge Computing with Deep Learning

Rice is a globally important crop that will continue to play an essential role in feeding our world as we grapple with climate change and population growth. Lodging is a primary threat to rice production, decreasing rice yield, and quality. Lodging assessment is a tedious task and requires heavy labor and a long duration due to the vast land areas involved. Newly developed autonomous crop scouting techniques have shown promise in mapping crop fields without any human interaction. By combining autonomous scouting and deep lodged rice detection with edge computing, it is possible to estimate rice lodging faster and at a much lower cost than previous methods. This study presents an adaptive crop scouting mechanism for Autonomous Unmanned Aerial Vehicles (UAV). We simulate UAV crop scouting of rice fields at multiple levels using deep neural networks and real UAV energy profiles, focusing on areas with high lodging. Using the proposed method, we can scout rice fields 36% faster than conventional scouting methods at 99.25% accuracy.

4.1 Introduction

Rice, Oryza sativa L., is an essential staple crop worldwide and has a significant impact on world politics and economics. Under the effects of global climate change and a world population increase by 2 billion persons in the next 30 years [35, 158], maintaining stable rice production is a priority task for many countries to maintain food security. Many studies have shown that rice lodging is the primary factor that weakens rice production. Lodging reduces photosynthesis [177] decrease yield [116], and significantly diminishes rice quality [226]. A large number of studies address rice lodging problems from a cultivation perspective analyzing the mechanisms and the causes of rice lodging [153, 154]. In contrast, other studies concentrate on developing effective methods to assess rice lodging [88, 152, 206]. Traditional rice lodging assessment relies heavily on manual in-situ assessment and random sampling [206]; however, the traditional manual assessment method has significant drawbacks. Typically, a preliminary disaster valuation, a comprehensive investigation, and a review sampling assessment are needed to complete a rice lodging assessment, which may take approximately 1 to 2 months to accomplish in total. Consequently, the rice field cannot be replanted, and the field owner's livelihood is dramatically impacted. Additionally, the delineation of the rice lodging area is performed by officers manually, which is prone to subjectivity and frequently leads to controversies. Lastly, due to the vast land areas involved in natural disasters, the traditional manual assessment faces challenges of high labor cost and efficiency. An effective rice lodging assessment method is urgently needed.

Remote sensing techniques like satellite imagery provide a feasible solution to investigate rice lodging over vast areas of land [126]. Nonetheless, satellite images

are usually limited by their spatial and temporal resolution, as well as their spectral band features [151]. Additionally, cloud contamination can limit the usability of optical satellite images as thick clouds can completely occlude target landscapes. In recent years, utilizing unmanned aerial vehicle (UAV) technology to obtain timely information on crop lodging has created numerous opportunities due to UAVs ability to fly in cloudy conditions [226]. UAVs can be equipped with high-spatial-resolution cameras to collect high spatial aerial red-green-blue (RGB) images for crop lodging assessment. The capability of UAV to obtain high spatial accuracy aerial images is complemented by the benefit of a global positioning system (GPS) and inertial navigation system (INS) technology. Yang et al. (2017) [206] used UAV images combined with a spatial and spectral mixed image classification method, which classified rice lodging at 96.17% accuracy. Based on UAV images, Chu et al. (2017) [45] produced a 3D canopy height model to detect corn lodging severity depending on height percentiles against preset thresholds. Later, Chu et al. (2017) [46] developed a lodging index to automatically reflect the severity of corn lodging and yield after harvesting. Liu et al. (2018) [126] combined thermal infrared images with UAV images to identify lodging rice, which has a false positives rate and a false negative rate of less than 10%.

In the meantime, the development of deep learning techniques has achieved evident results in agriculture applications [94]. Chu and Yu (2020) [47] developed an end-to-end prediction model by fusing two back-propagation neural networks (BPNNs) with an independently recurrent neural network (IndRNN) for rice yield prediction. Wang et al. (2020) [200] proposed a deep learning and depth camera combined solution to improve UAV environmental perception and autonomous obstacle avoidance. Zhao et al. (2019) [225] used a deep learning U-shaped Network (UNet) architecture for

rice lodging, with results showing that the dice coefficients on the RGB and multispectral datasets reach 0.942 and 0.9284, respectively. Yang et al. (2019) [210] compared vegetation index (VI) based with a convolutional neural network (CNN) based approaches for rice grain yield estimation at the ripening stage using UAV images. The results show that this CNN based approach performs much better than the VI-based regression model. Yang et al. (2020) [209] applied deep-learning to UAV images to estimate rice lodging in paddies over a large area. The semantic segmentation networks, including FCN-AlexNet and SegNet, are proven to have lower latency, approximately 10 to 15 times faster, and a lower misinterpretation rate than that of the maximum likelihood method. However, the studies, as mentioned earlier, all perform their analysis from an offline approach. Few applications attempt real-time rice lodging assessment. Mardanisamani et al. (2019) [131] presented a deep convolutional neural network (DCNN) architecture using a transfer learning technique for lodging classification, which achieves comparable results while having a substantially lower number of parameters. The author emphasized that DCNN can be deployed on low-cost hardware, which can be suitable for real-time applications. Recently, the development of edge computing devices and techniques has improved researchers' ability to process large amounts of data in a real-time manner without offloading to the cloud [169]. The usage of graphical processing units (GPUs) for deep learning and the availability of powerful processors at the edge allow practitioners to analyze data quickly without the cloud.

Edge computing has been used to advance many emerging research areas in Computer Science, from the internet of things (IoT) applications [11] to smart homes [5] and smart cities [36]. Edge computing has also been applied to agricultural scouting

applications. Vasisth et al. (2017) [197] used edge computing techniques to create FarmBeats, a novel precision agriculture platform that gathers data from sensors, cameras, and drones to enable precision agriculture techniques. Instead of transmitting all data to the cloud, a local computer or laptop device is used to process drone imagery data, which corresponds to the concept of edge computation [212]. As shown in Figure 4.1, the connection of edge computing nodes to sensors like UAVs provides advantages, including high mobility, simplicity, interactive, and responsive. Additionally, a simplified ad hoc NN (neural network) can be implemented for a specific purpose/target. In contrast, several challenges, such as high communication latency and information complexity, need to be considered in the cloud-based system. In the present study, the connection of edge computing nodes to sensors is taken as the primary focus on developing an effective rice lodging assessment method.

Precision agriculture techniques like satellite imagery, UAV scanning, and even advanced platforms like FarmBeats are all linked in that they are automated. These approaches perform low-level tasks without human decision-making but require high-level human planning to perform well. Advancements at the edge allow for more complex autonomy policies to be implemented, allowing for autonomous, rather than automated systems to perform agricultural scouting and management tasks. Early attempts at fully autonomous precision agriculture using fully autonomous aerial systems (FAAS) have been implemented [23, 30], and early fully autonomous precision agriculture software is available [26]. Boubin et al. (2019) [23] presented an open-source software package for FAAS, which includes autonomous UAV routines for agricultural scouting and demonstrates that fully autonomous routines can benefit

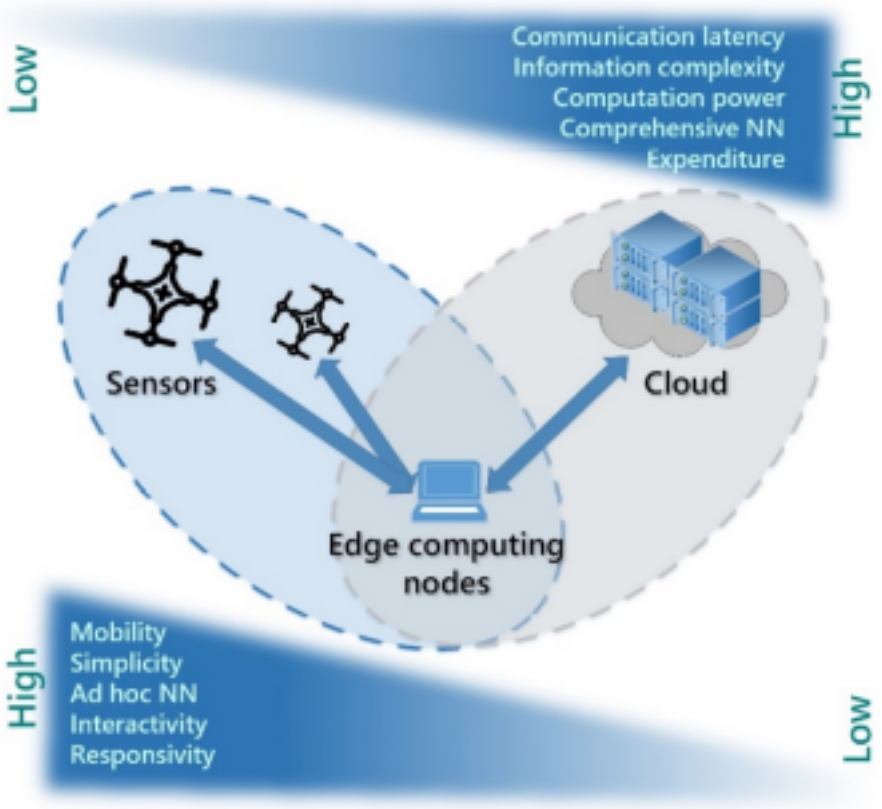

Fig. 4.1: Comparison of Functions of edge computation and cloud-based system

significantly from correct edge compute architectures and autonomy policies. Simulated approaches to FAAS have also been demonstrated. Boubin et al. (2019b) [30] show that given appropriate pathfinding algorithms and autonomy policies, accurate yield maps of crop fields can be generated by viewing only 40% of a field, saving power, time, and money for the farmer. Combining FAAS techniques with deep learning at the edge may greatly impact future precision agriculture techniques. In this study, an effective rice lodging assessment method is proposed by combining deep learning and FAAS techniques with UAV images. Notably, the EDANet model was trained to study rice lodging information extracted by conventional digital RGB images. Three UAV

scouting approaches, namely 200m high altitude scouting, 50m low altitude scouting, and an adaptive fly height, are investigated, and the corresponding energy consumption and rice lodging accuracy are assessed. Specifically, the following contributions of this paper are highlighted as follows. 1. An adaptive autonomous UAV scouting technique utilizing EDANet, a deep learning model, is proposed to assess rice lodging. The adaptive autonomous UAV scouting mechanism is designed based on a threshold-based rice lodging assessment derived from the EDANet model. 2. Visible spectrum information and three vegetation indices are extracted and calculated from UAV images collected from two real rice lodging occasions in Taiwan. Both visible spectrum information and vegetation indices were used for EDANet model training and testing. 3. The proposed adaptive autonomous UAV scouting approach is compared with the other two approaches- 50m low altitude scouting and 200m high altitude scouting in terms of associated rice lodging identification accuracy and scouting time. 4. The comparison of three approaches is performed in a simulation-based research software SoftwarePilot, in which an autonomous cube data structure [30] links UAV images with spatial information and an energy profiling using DJI Phantom 4 Pro (P4P) is applied. 5. Through a series of lodged percentage threshold settings for the simulator, our proposed adaptive autonomous UAV scouting approach demonstrates excellent performance in significantly reducing scouting time and preserving relatively high rice lodging identification accuracy. The remainder of this paper is organized as follows. Section 2 describes our experimental methods, datasets, machine learning process, and adaptive scanning algorithm. Section 3 provides results from experiments. Section 4 discusses the results, their impact, and future work. Section 5 provides conclusions.

4.2 Materials and Methods

Two rice lodging occasions associated with massive rainfall events in June 2017 and May 2019 in the Mozi Shield Park in Wufeng District, Taichung City, Taiwan, were investigated. Visible spectrum information of the field was collected in June 2017 by a fixed-wing UAV and in May 2019 by a rotary-wing UAV for training and testing, respectively. Detailed camera, flight height, area covered, training, and testing dataset information are shown in Table 1. Besides visible spectrum information, three vegetation indexes, Excess Green index (ExG), Excess Red index (ExR), and Excess Green minus Excess Red index (ExGR), were calculated for the training and testing datasets [139,203]. Formulas of these three vegetation indexes are listed below.

$$ExG = 2 \times G_n - R_n - B_n \tag{4.1}$$

$$ExR = 1.4 \times R_n - G_n \tag{4.2}$$

$$ExGR = ExG - ExR = 3 \times G_n - 2.4 \times R_n - B_n \tag{4.3}$$

4.2.1 Semantic segmentation model training

We used an implementation of the Efficient Dense modules with Asymmetric convolution network (EDANet) developed by Lo et al. (2019) [127] to detect rice lodging. The network architecture of EDANet is shown in Table 4.1, which comprises of three major components, including three downsampling blocks, which are adopted from the ENet initial block [18], two EDA blocks consist of 5 and 8 EDA modules respectively and one projection layer. A two-branch design of the downsampling layers saves the

	Training dataset	**Testing dataset**	
Camera	Sony QX-100	DJI Phantom 4 Pro	
Collection Date	2017 / 06 / 08	2019 / 05 / 21	2019 / 05 /23
Resolution(width/ height)	46343 x 25658	5472 x 3648 (image frame)	
Flight Height (m)	200	50	200
Area covered (ha)	430	4.4	120
GSD (cm)	5.3	1.3	5.7
Tile resolution (col / row) pixels	480 x 480	1349 x 899	5472 x 3648
# effective tiles	3485	220	65
# tiles in row / # tiles in col	53 x 96	-	-

Table 4.1: Detail of training and testing datasets

computation of convolutional layers by adding a max-pooling layer when the input channel number is less than the output channel number. Inspired by DenseNet [88], EDANet modifies the concept of dense connectivity to module-level in which the output of each EDA module concatenates its input and the newly learned features. A single EDA module consists of a point-wise convolution layer and two pairs of asymmetric convolution layers. Additionally, a specific dilated asymmetric convolution technique, which inserts zeros between two consecutive kernel values along each dimension, is employed at the second asymmetric convolution pair in an EDA module. The dilated asymmetric convolution technique can aggregate more contextual information by enlarging the effective receptive field without increasing the number of parameters for accuracy improvement purposes. The projection layer, which is a 1 x 1 convolution layer, produces the output C (the number of classes) feature maps. In the end, a bilinear interpolation process is applied to upsample feature maps by a factor of 8 to the size of the input image. In short, the densely connected EDA modules

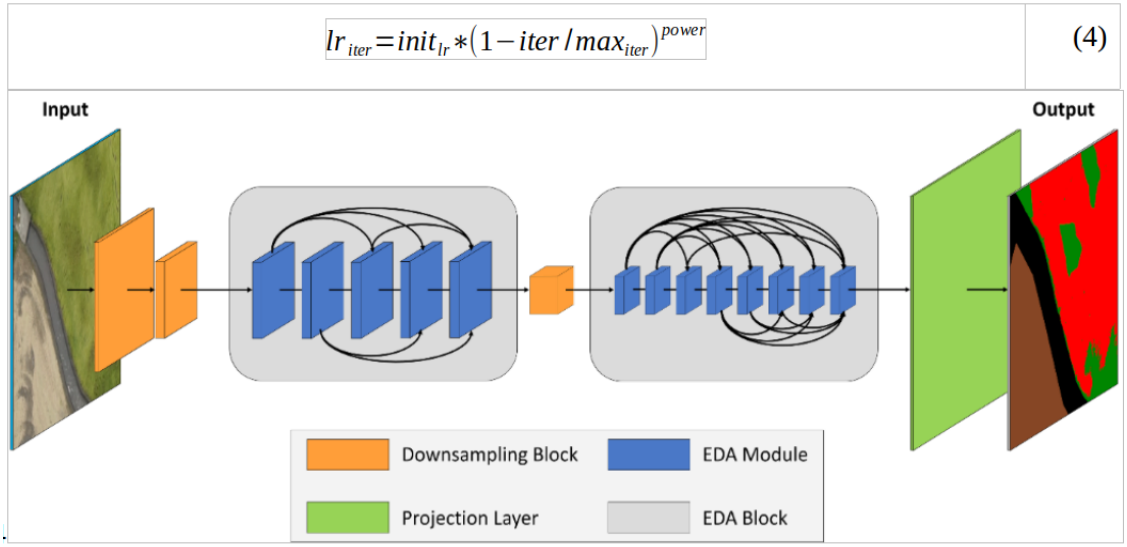

Fig. 4.2: EDANet architecture (Figure adapted from Lo et al. (2019))

with asymmetric convolution structures play the most critical role in reducing the number of parameters and computational costs, which makes EDANet a promising network for real-time semantic segmentation. In the present study, the EDANet was trained using Adam optimizer with weight decay 1e-4, batch size 10, for 50 epochs. As suggested by Lo et al. (2019) [127], the initial learning rate was set to 5e-4, and a polynomial learning rate policy was employed with power 0.9 for the learning rate of each iteration lr_iter in formula (4).

$$lr\_iter = init\_lr \times (1 - iter/max\_iter)^{power} \tag{4.4}$$

The model training environment information is shown in Table 2 below. The EDANet model performance was evaluated using six matrices listed in Table 3. Due to the limited space, only the F1 score and overall accuracy (OA) among the six matrices are reported for the training results. As shown in Table 4, the highest OA reaches

CPU	Intel Xeon Gold 6154 @3.00GHz (4 cores/GPU node)
RAM	90GB/GPU node
Accelerator	NVIDIA Tesla V100 GB SMX2/GPU node
Image	TensorFlow-1.0py3
Libraries	Python 3.6.8, NumPy 1.14.5, scikit-image 0.16.1, Keras 2.3.2, TensorFlow-GPU 1.14, Jupiter notebook, CUDA 10.1

Table 4.2: Model training environment information

94.87% when RGB information is used. For these five classes (rice paddy, rice lodging, road, bare land, and background), the associated F1 scores are quite stable for each class. The bare land class has the highest F1 score of 96.97%, whereas the road class shows the lowest F1 score of 80.26%. Table 5 shows the testing results for the 2017 and 2019 datasets. Based on the testing results, the entire testing process can be performed in around one minute, which is the most compelling evidence demonstrating the real-time capability of EDANet. Figure 4.2 represents testing results from a subset of the 2019 field dataset. As shown in Figure 4.2, EDANet performs well in capturing rice lodging, rice paddy, and other classes. Based on the results of the 2019 dataset, EDANet using RGB+ExG+ExGR information illustrations the highest value in recall (85.22%), accuracy (92.83%), and F1 score (78.51%). Therefore, the EDANet using RGB+ExG+ExGR information is the final model that been applied in this study for further investigation in UAV scouting simulation.

4.2.2 UAV Scouting

To gain an accurate assessment of rice lodging, fields must be scouted in their entirety to determine the percent of total lodged crops throughout. When scouting an area, operators must concern themselves with a number of factors, of which UAV

Evaluation Matrices	Formula
Precision	$precision_c = \frac{TP_c}{TP_c + FP_c}$
Recall	$recall_c = \frac{TP_c}{TP_c + FN_c}$
Accuracy	$accuracy_c = \frac{TP_c + TN_c}{TP_c + TN_c + FP_c + FN_c}$
Overall accuracy	$OA = \sum_{c=1}^{n} \frac{TP_c}{TP_c + TN_c + FP_c + FN_c}$
F_β score	$F_\beta = \frac{(1+\beta^2) \times Precision \times Recall}{\beta^2 \cdot Precision + Recall}$
F_1 score	$F_1 = \frac{2 \times Precision \times Recall}{Precision + Recall}$

Table 4.3: Model performance evaluation matrices

Information	Rice paddy (%)	Rice lodging (%)	Road(%)	Bareland (%)	Background (%)	OA (%)
RGB	**95.28**	**86.17**	**83.02**	**96.97**	**96.94**	**94.87**
RGB+ExG	**95.28**	85.99	81.10	96.86	96.58	94.60
RGB+ExGR	95.24	85.87	81.80	96.48	96.55	94.54
RGB+ExG+ExGR	95.19	86.03	80.26	96.51	96.46	94.45

Table 4.4: EDANet model training results F1 score and overall accuracy (OA) (highest value in bold)

Year	Information	Precision (%)	Recall (%)	Accuracy (%)	F1-score (%)	Time (sec)
2017	RGB	84.03	82.19	94.26	83.10	56
	RGB+ExG	**86.16**	84.19	**94.96**	**85.16**	58
	RGB+ExGR	84.42	85.77	94.84	85.09	61
	RGB+ExG+ExGR	83.34	**86.99**	94.78	85.13	64
2019	RGB	72.71	38.27	88.30	50.15	55
	RGB+ExG	**93.46**	56.45	92.70	70.39	59
	RGB+ExGR	82.24	52.72	90.98	64.25	60
	RGB+ExG+ExGR	72.78	**85.22**	**92.83**	**78.51**	65

Table 4.5: Model testing results for the 2017 and 2019 datasets (highest value in bold)

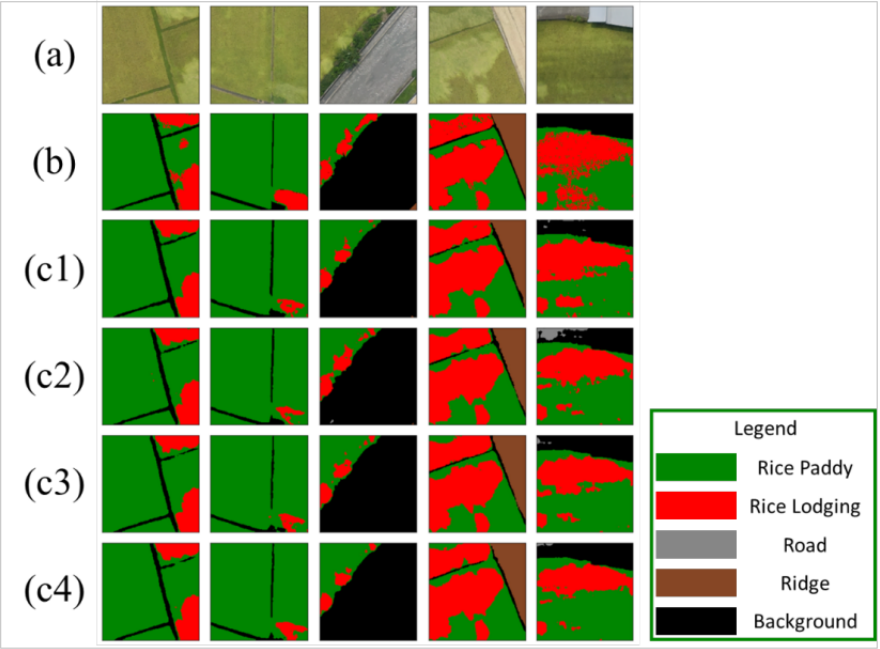

Fig. 4.3: Model testing results demonstration. (a)original image, (b)ground truth, and (c1-c4) represent EDANet with RGB, RGB+ExG, RGB+ExGR, and RGB+ExG+ExGR information, respectively. (Red represents rice lodging, green represents rice paddy, and black represents other classes)

battery life and image ground sample distance (GSD) are principal. UAV batteries are highly constrained, lasting for flight times between 20 and 40 minutes depending on the UAV model. We define one UAV flight as one full discharge of the UAV battery by assuming that the UAV takes off with a full battery and ends the mission with an empty battery. Mapping a field of considerable size may require many flights over hours to days, depending on the number of UAVs and batteries available, their recharge time, and GSD. Flight times can be decreased by increasing UAV altitude, which will, in turn, increase GSD. GSD, and consequently, flight altitude and camera quality, must be considered. Image quality and clarity directly influence the ability of subject matter experts and machine learning algorithms to detect field abnormalities [207,208]. When scouting for lodged rice, it is important to assure that all regions of the field can be accurately classified, and consequently, each image must have a GSD high enough to allow this. Given that raising GSD increases scouting time (independent of camera quality), a tradeoff emerges. Crop scouters must simultaneously balance detection accuracy and total scouting time, which can lead to increased costs, respectively, from unsubsidized lost crops, labor, and equipment.

Three mapping strategies are shown in Figure 4.4 that demonstrate these differences. For a small section of a field (requiring 36 images to fully map at low altitude and four at high altitude), we outline three mapping strategies: high altitude, low altitude, and adaptive scouting. Low altitude and high altitude scouting exhibit the two principal differences previously described. Low altitude scouting requires significantly more energy and time to complete its task, requiring a second UAV mission in this scenario to take each picture. In contrast, high altitude scouting is capable of capturing the entire section of the field in only four images. The tradeoff, however,

	Step						
	1	6	12	18	24	30	36
1) Low Altitude Scouting							
UAV Mission 1 Battery %	95	77	59	41	23	5	7
UAV Mission 2 Battery %	100	100	100	100	100	100	95
Scouting %	4	20	36	52	68	84	100
Current Zone	F0	A0	F1	A2	F3	A4	F5
2) High Altitude Scouting							
UAV Mission 1 Battery %	95	77					
Scouting %	25	100					
Current Zone	E2	Fin					
3) Adaptive Scouting							
UAV Mission 1 Battery %	95	77	59	41			
Scouting %	8	38	69	100			
Current Zone	D2	C1	E3	Fin			

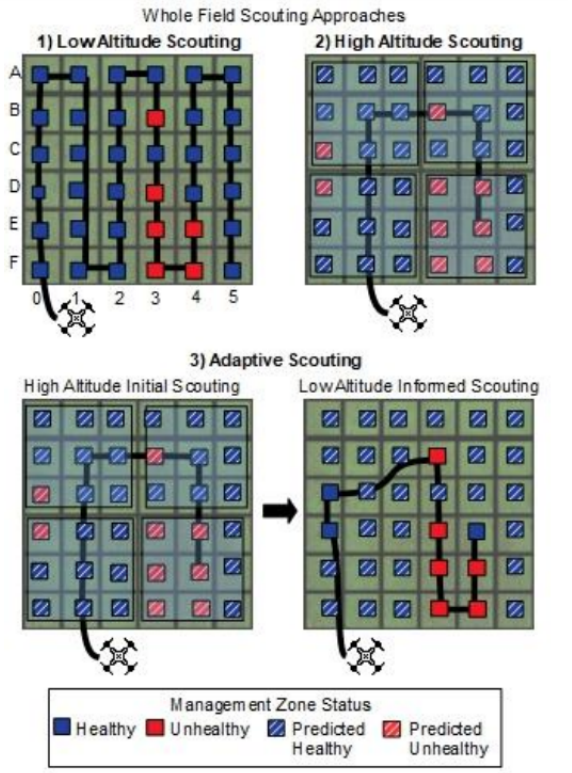

Fig. 4.4: 3 scouting methods over a small area of cropland: 1) Low altitude scouting, 2) high altitude scouting, and 3) adaptive scouting. For each scouting method, scouting is tracked at 4 step intervals, logging battery of each necessary UAV mission, scouting completion percentage, and position.

is that high altitude scouting is less capable of predicting lodging. It can not properly guarantee the correctness of its predictions at its decreased GSD. We can see in Figure 4.4 that high altitude scouting mispredicts lodging in a number of regions. Given the high GSD of low altitude scouting, we can accept its lodging predictions as credible. A strategy that combines the high accuracy of low altitude scouting and the low energy costs of high altitude scouting could be dominant. In this paper, we present an autonomous scouting procedure that uses high altitude scouting to inform selective low altitude scouting.

4.2.3 Autonomous Scouting

Autonomous scouting allows UAVs to make decisions on where to scan for high GSD images based on potentially inaccurate, but informative predictions from low GSD scouts at high altitude. In short, UAVs scout the field at high altitude first. An edge system analyzes these images in situ, then uses machine learning to determine positions to investigate at low altitudes based on classification accuracy and certainty. Figure 4.5 shows how autonomous scouting can help balance the accuracy of low altitude scouting with the energy efficiency of high altitude scouting. The UAV first performs a high altitude scan and predicts lodging in each region of the field at the edge. Once lodged areas are known, the UAV investigates all areas that are classified as lodged at high altitude. In this example scenario, the UAV can gain a 100% accurate picture of lodging in the field subsection by using half the energy of low-altitude scouting.

Autonomous scouting does, however, include a substantive hardware addition to the above requirements for both high and low altitude automated scouting procedures.

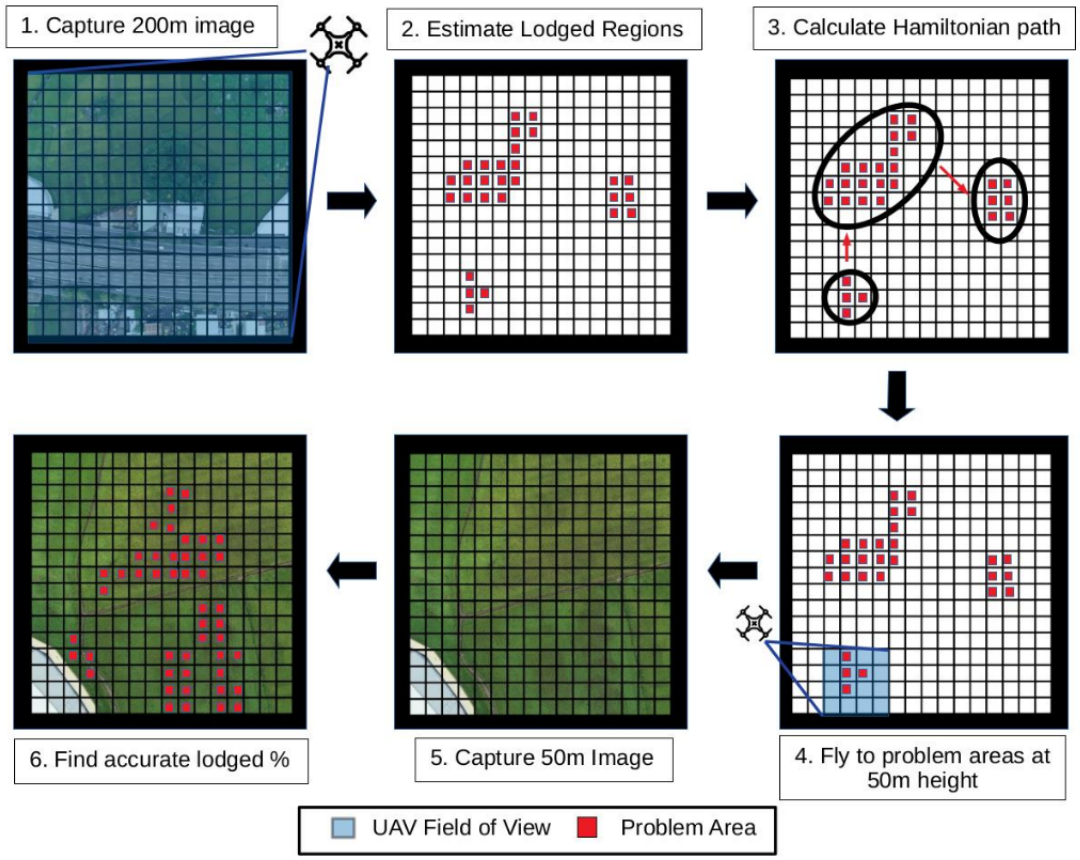

Fig. 4.5: A visual depiction of the autonomous scouting algorithm for 200m and 50m altitudes.

Automated scouting can be done by a pilot or lightweight edge system. Images do not need to be processed locally, so simple systems are sufficient. Autonomous scouting, however, requires an edge system capable of controlling the UAV and executing simplified machine learning algorithms within a reasonable time-frame, which would require a hardware accelerator such as a GPU. Figure 4.5 describes the autonomous scouting algorithm in detail. For this scenario, we define the high altitude scouting height as 200 meters, and the low altitude scouting height as 50 meters. There are six key steps to complete an entire mission.

1. UAV must map at least one high altitude section of the field. This entails the UAV flying to a GPS waypoint at the center of the 200m region being mapped and capturing an image.

2. Once the edge system downloads images of at least one region, machine learning is used to predict rice lodging in regions of high altitude images. Each image is decomposed into subsections that correspond to a possible 50m image (i.e., a 200m image decomposes into 16 50 meter subsections). Each region is then given a certain lodged percentage (i.e., the percentage of pixels in that region represents lodged rice). Regions are then marked as uncertain if their lodged percentage is above a user-provided threshold. Uncertain regions are regions that must be explored further to accurately gauge lodging.

3. Once all regions are predicted, our system finds the most efficient route in which a UAV can fly to visit all uncertain regions. To do this, the edge system calculates the least cost Hamiltonian path between a fully connected graph whose vertices include the UAV's current position and all uncertain regions.

This Hamiltonian path is then remapped to a UAV flight path, such that the UAV flies to each uncertain region once.

4. The UAV descends from its 200m height and flies to the next region in the Hamiltonian path at 50 meters altitude.

5. The UAV then captures a 50m image of each uncertain region, flying along the prescribed Hamiltonian flight path.

6. Once all images are captured, the edge system downloads each image. It calculates its lodged percentage using the high GSD 50m images, thus gaining a more accurate picture of actual rice lodging.

4.2.4 Scouting in Simulation

Crop scouting with autonomous UAV requires considerable infrastructural support, including software construction, testing, and regulation compliance. Furthermore, rice lodging on a large scale happens mainly after severe typhoons or storms. Given the unprecedented trend of global climate change and warming, severe typhoons and storms occur more often and become much stronger. Therefore, developing an efficient crop scouting method is urgently needed. For these reasons, we chose to simulate our scouting algorithms using the lodged rice data sets mentioned earlier in this section. Crop scouting and autonomous UAV simulation mechanisms have been tested and validated in prior work [23, 30]. Using these methods, we were able to construct valid simulations and gather results for our scouting techniques.

Autonomy Profiling

Autonomous systems must have the ability to fully navigate dynamic environments. For UAV, this implies the ability to fly to any point with some two-dimensional or three-dimensional space relevant to the problem at hand. When scouting a crop field for lodged rice, an autonomous UAV must have the ability to fly to and sense data at any point within the three-dimensional region where sensed data would be relevant to the field being scouted. When planning a crop scout, however, certain components of this bounding region can be abstracted to obtain similar results. For instance, the UAV may only fly at one altitude, presumably selected to balance image GSD and flight time. Similarly, UAVs are usually only required to fly to a subset of points within the space, such that the subset of points allows the UAV to fully observe the crop field. Crop scouting and UAV mission plans in this respect are usually represented by a set of GPS waypoints and altitudes read by mission planning software that pilots the UAV automatically. Unlike automatic flight, autonomous systems make high-level decisions, like which waypoint to fly to next, in flight.

Given both the amount of possible sensed data by an autonomous crop scouting UAV and its ability to choose which data is sensed at runtime, profiling autonomous UAV for simulation can be difficult. To simulate lodged rice crop scouting, we used the autonomy cube, a data structure specifically created to aid in autonomous vehicle simulation [23]. Autonomy cubes are data structures that combine sensed data (e.g., images, videos, other sensor readings) with spatial information (e.g., altitude, GPS locations). Each piece of data in an autonomous cube is linked to both an altitude and GPS location. As shown in Figure 4.6, data also have links to surrounding data points based on possible flight actions. A flight action is defined as any movement or

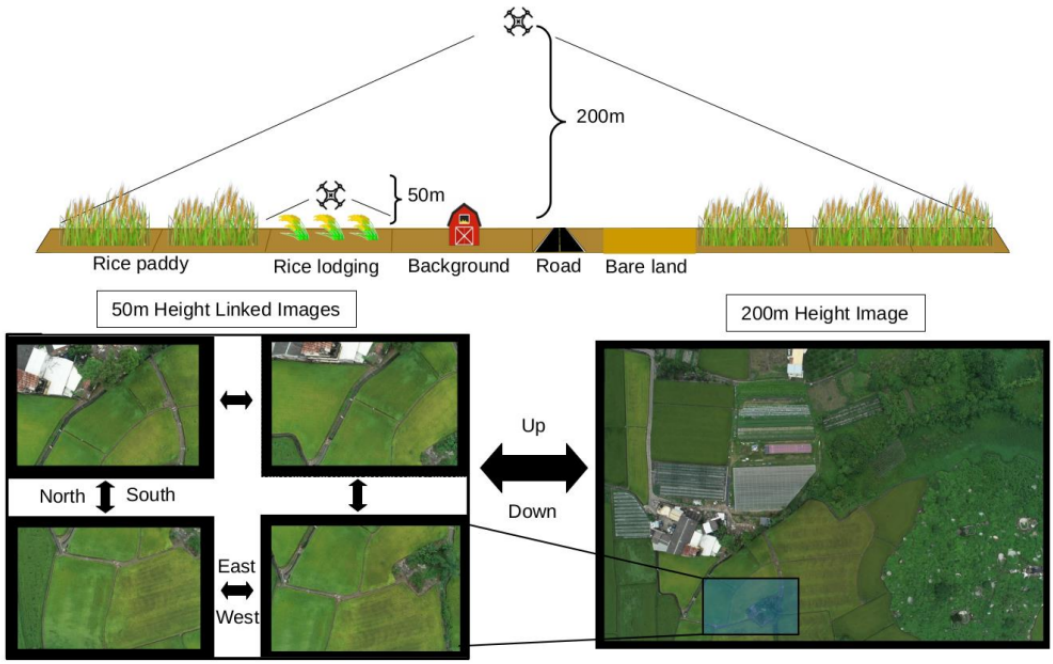

Fig. 4.6: Autonomy cubes capture both sensed data and spatial information. Sensed data points are spatially linked by flight action. In this figure, sensed data is liked by both cardinal direction and altitude.

process that the UAV can undertake to solve its autonomy goal (e.g., takeoff, land, fly up 150 meters, fly west 10 meters, capture an image, hover). Flight actions are combined to create a complete UAV flight. Missions generally consist of UAV takeoff and landing combined with some series of UAV translation actions (e.g., fly left, fly up) and data sensing (e.g., capture images). For our simulation, we have created a set of standard flight actions that can be used for aerial scouting. For every possible flight action of the UAV being modeled that displaces the UAV, a link will exist to data sensed at that position. These links allow simulated UAV to navigate virtual environments easily, accessing sensed data from real UAV missions. To construct autonomous cubes, a considerable amount of profile data is required. We sensed lodged rice data from Mozi Shield Park in Wufeng District in Taichung City, which we built into autonomy cubes using available research software [26].

Energy Profiling

To properly evaluate the performance of a simulated UAV flight, the energy expenditure of each possible flight action for both UAV and compute must be known. To construct valid simulations, we created energy profiles for both UAV and compute. We define an energy profile as the execution time in seconds and energy expenditure in joules for every action available to a given system. In this context, an aircraft's energy profile would include energy and execution time information for each flight action. For a base station, this would include energy and execution time information for each flight control, classification, or data movement task. To obtain this information, we used methods from prior work [23]. We performed each computes and flight action 100 times, fitting the execution time and to normal distributions and recording energy expenditure in watts to construct energy profiles. We gathered energy and

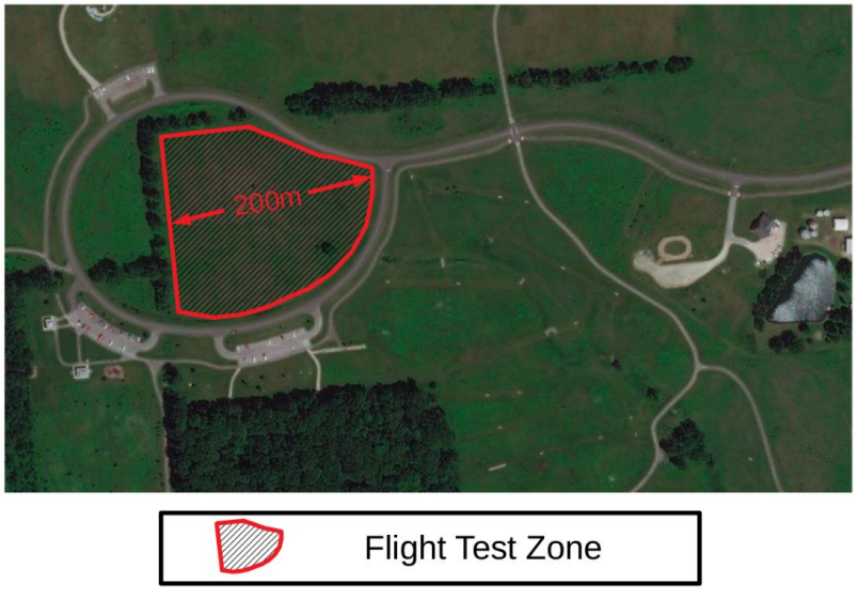

Fig. 4.7: Aerial view of Glacier Ridge Metropark in Dublin, Ohio, where the flight action profiling was performed.

execution time information from UAV and edge systems using SoftwarePilot [26]. Flight actions were profiled at Glacier Ridge Metropark in Dublin, Ohio (Figure 4.7), using a DJI Phantom 4 Pro (P4P).

Flight actions include translation at 2m/s, data capture and transfer, hovering, and takeoff/landing. The simulated UAV translates at 2 m/s in all directions for consistency. At the height of 50m, the UAV must move 10m along one of its horizontal axes to see a different image in our dataset, and similarly, at 200m, the UAV must move 35m. For this reason, we profiled each movement along the horizontal axis (left, right, forward, and backward) at both 10m and 35m. Vertical translations (fly up, down, takeoff/landing) were also profiled. We combined takeoff and landing into one action per prior work [30], and profiled translation between the 50m and 200m heights at 2m/s. Lastly, we profiled interactions with edge systems. The Sense Data

UAV	Flight Action	Energy (J)	Battery Consumption (%)	Time (sec)
P4P	Takeoff/Land	249	0.95	14
	Left 10m	558	0.76	5
	Left 35m	558	2.65	17.5
	Right 10m	558	0.76	5
	Right 35m	558	2.65	17.5
	Forward 10m	661	0.89	5
	Forward 35m	661	3.14	17.5
	Backward 10m	514	0.69	5
	Backward 35m	514	2.44	17.5
	Fly Up 150m	560	2.28	15
	Fly Down 150m	519	2.11	15
	Sense Data	367	0.05	5
	Hover	257	0.07	1

Table 4.6: Results of the profiling with the P4P.

action includes capturing an image with the UAV camera and transferring it to the edge system. The hover action is used when the UAV idles, awaiting commands from the edge system during the adaptive approach.

Simulated Scouting

Using autonomy cubes and energy profiles, we constructed software to simulate the three whole field mapping strategies detailed earlier in this section. Table 4.6 shows how to construct the simulator. Inputs include workload settings (e.g., machine learning models, flightpath type), the autonomy goal (i.e., to estimate the lodged percentage of the crop field), and autonomy cubes. Based on the flightpath type, potentially sensed data and model outputs, flight paths for the UAV are generated. Once a flightpath is generated, it can be used to determine the number of each flight and compute the actions required. We sample the normal distributions of these actions latency, which are used to calculate energy expenditure in joules.

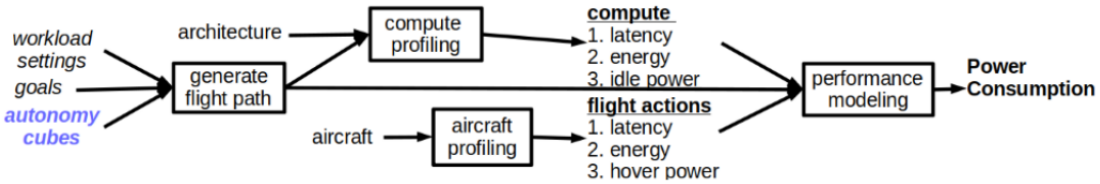

Fig. 4.8: Workload settings, goals, autonomy cubes, and energy profiles are taken as inputs to the simulation.

These inputs are used to generate flight paths and determine the overall power consumption and execution time of UAV missions. Adaptive scouting in the simulation was principally compared to low and high altitude scouting, which both scout the entire field using EDANet to predict lodging at 50m and 200m respectively after the flight. All three methods were simulated with UAV starting at one of 4 corners of the waypoint grid and moving first in either of the two possible directions from the start point in a lawnmower fashion for a total of 8 possible traversals of the grid. Simulations were run on a Lenovo Thinkpad T470 laptop with an Intel 7500U CPU and 24GB of RAM running Ubuntu Linux 18.04. The simulator was written in Python 3.6 and was given pre-segmented images from our EDANet model.

4.3 Results

The adaptive scouting process was evaluated in simulation using autonomy cubes, a DJI P4P UAV profile, and a series of lodged percentage thresholds. Autonomy cubes were constructed from the Wufeng rice crop dataset using the SoftwarePilot autonomy cube builder [26]. Autonomy cubes were created for images captured at both 200m and 50m heights. Out of 215 Wufeng crop images captured at 50m, 73 contained no rice or exclusively rice that could be seen from other waypoints, so

the simulated UAV did not capture data at those waypoints. The 200m autonomy cube contained 12 images that were able to encompass all 215 50m images. An energy profile was created for the P4P by profiling each simulated movement under real-world conditions using SoftwarePilot. Ten lodged percentage thresholds were also assigned for the simulator, denoting what estimated lodged percentage a region must have at 200m in simulation to be investigated at 50m, meaning that a 5% lodged threshold would only scout 50m regions showing 5% or greater lodging at 200m. The lodged percentages were chosen between 2.5% to 25% at intervals of 2.5%. The simulator used EDANet to estimate lodged percentages. Figure 4.9 shows example results from one simulation comparing the three methods where all methods begin on the same path, at the southwest corner of the grid, and move immediately north. The number of each flight action, total discharges, and lodged percentage prediction of each method are reported. Note that the adaptive method's Hamiltonian path is not included as a flight action. Figure 4.9 also shows the route taken by the adaptive approach. The adaptive approach originally follows the same route as the 200m approach, sensing data at each 200m waypoint. Once all data is sensed, uncertain sections are found, and the best possible waypoints are determined to explore them. A Hamiltonian path between those waypoints is determined and is explored at 50m by the UAV. The principal comparison points between the three approaches are accuracy and time in figure 4.10. Figure 4.10a compares a normalized accuracy between 200m and the adaptive scouting as they compare to 50m scouting. 200m scouting has a normalized error of 24.2%, meaning that an overall field scout at 200m differs in predicted lodged percentage by 24.2% from a 50m scout.

Simulation Attribute		50m	200m	Adaptive (T=2.5%)
Flight Actions	Takeoff/Land	1	1	1
	Left 10m	0	NA	NA
	Left 35m	NA	0	0
	Right 10m	9	NA	NA
	Right 35m	NA	1	1
	Forward 10m	103	NA	NA
	Forward 35m	NA	5	5
	Backward 10m	102	NA	NA
	Backward 35m	NA	4	4
	Fly Up 150m	NA	1	1
	Fly Down 150m	NA	1	1
	Sense Data	142	11	29
	Hover	NA	NA	29
UAV Discharges		2.34	0.32	1.60
Predicted Lodged Percentage		26.1	32.4	25.9

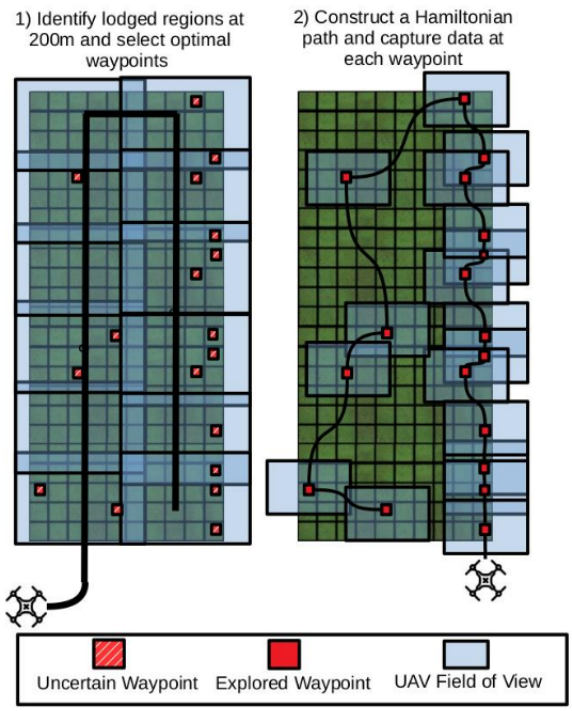

Fig. 4.9: Sample simulation results with the adaptive lodged threshold (T) set to 2.5%. Depicted is the simulated FAAS path for the adaptive approach at both 200m and 50m.

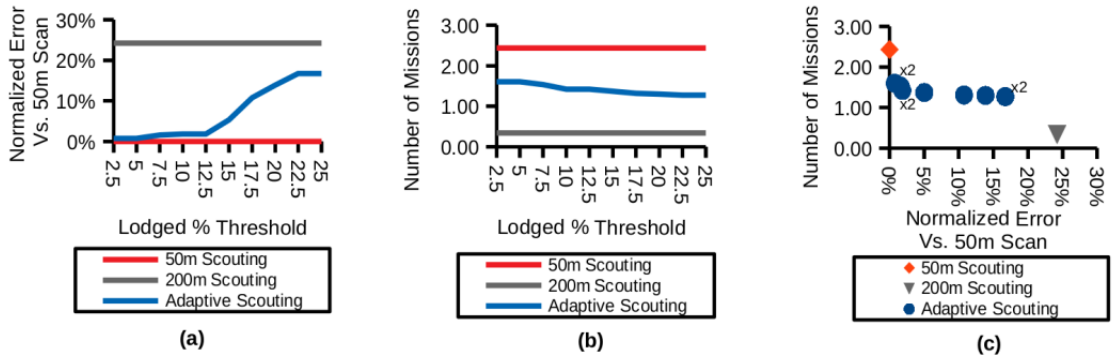

Fig. 4.10: a) Adaptive scouting is up to 99.25% accuracy compared to 50m scouting, while 200m scouting alone is 75.8% accuracy. b) Adaptive scouting takes at most 35% less missions to completely scout the field as compared to 50m scouting. c) Adaptive scouting balances the high speed of 200m scouting with the accuracy of 50m scouting, sacrificing little accuracy for significant speed gains.

Adaptive scouting differs between 0.75% and 16.73% depending on the lodged threshold, steadily increasing as the lodged threshold increases. Until the lodged threshold exceeds 12.5%, the normalized error between 50m scouting and the adaptive scouting does not exceed 2%. Higher lodged percentages, however, increase a normalized error by 5.03% to 16.73% due to their higher frequency to ignore largely lodged regions of the field due to inaccuracies in the 200m scout.

Figure 4.10b shows the differences in UAV missions required to scout the field between 50m, 200m, and the adaptive scouting. In the analysis, a P4P would have access to extra batteries, and when the battery depletes, the UAV battery would be changed, and scouting would resume. 200m scouting is considerably faster than either adaptive or 50m scouting, taking only 0.34 flights (at least one flight) to map the field. 50m scouting, on the other hand, took 2.43 total (at least three flights) flights to map the entire field. The adaptive scouting was able to map the field in

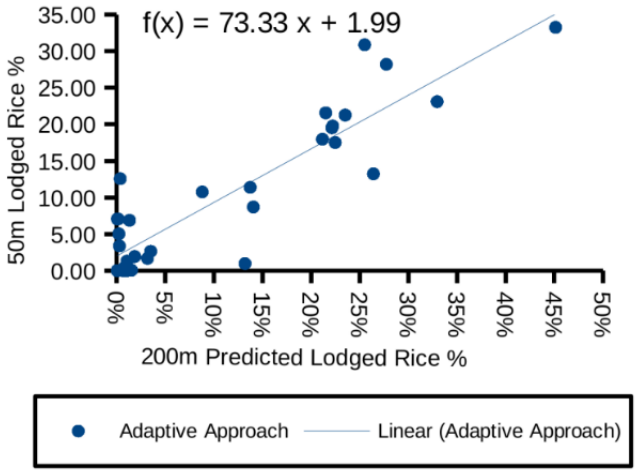

Fig. 4.11: Lodged rice predictions at 200m correlate with observed values at 50m but are inaccurate enough to yield valuable results alone. By informing 50m scouting using 200m results, high accuracy can be achieved while decreasing scout times.

between 1.6 to 1.27 (at least two flights) flights depending on the lodged threshold, saving one recharge period and mapping the field in 65%-52% the total time required by 50m scouting. Figure 4.10c shows the relationship between time and accuracy experienced by each scouting method. Of all approaches, 200m scouting is by far the fastest but suffers from accuracy issues. On the other hand, 50m scouting is highly accurate but requires considerable execution time. The adaptive scouting at low thresholds experiences less than 1% normalized error compared to 50m scouting but completes in 36

The adaptive scouting approach is able to decrease scouting time and preserve accuracy due to the correlation between 200m and 50m lodged predictions. As shown in figure 4.11, predicted lodging at 200m and 50m for highly lodged regions are correlated, with a bias toward higher prediction rates at 200m. By following the

overall correlation, we are able to scout at 200m and successfully predict which regions contain lodging, but not accurately predict the amount. By then scouting only those regions at 50m, the adaptive approach can improve on total mission time without losing significant accuracy. The largest differences between 200m and 50m scouting predictions can be found at low levels of lodging, where 200m scouts predict lodging where there is none or predict no lodging where there is some at 50m. Many of these points are regions which contain smaller amounts of rice crop, in lieu of buildings and roadways. 200m scouting could mispredict these regions as lodged or healthy rice to the detriment of the overall scout, requiring either the scouting of unlodged regions or the skipping of lodged regions. While these outlier points may be mispredicted as highly lodged or healthy, they generally encompass only a small percentage of the overall field. They, therefore, have little effect on the overall performance of the adaptive scouting model, as demonstrated by the results presented in figure 4.9. Handling outlier cases such as these, however, should be addressed by future work. A flight plan must be specifically made before executing the UAV mission, and consists of parameter setting, including the required scale of the photography, the camera focal length, the film format size, (forward) overlap, sidelap, the flying height above the ground, the ground speed of UAV, and the time interval between exposures. The purpose of photography is a major consideration in the flight plan. In this study, the overlap and the sidelap are required only for image mosaic in the 200m scouting. In the 50m flight mission, a higher overlap, larger than 67% of overlap and sidelap so that one feature point and its corresponding points appear on at least three successive photos, is required for image-based modeling. To increase robustness and accuracy, the redundancy should be afforded with a large number of mutually overlapping

photos simultaneously, so-called multi-ray photogrammetry, which requires a very high overlap (80%-90%) and sidelap (up to 60%) [120]. The overlap and the sidelap are 85% and 85%, as well as 80% and 60% for 200m and 50m scouts, respectively. Moreover, the total number of exposures necessary for a mission should be computed prior. Each photo has an incremental area, A, as

$$A = (1-p) \times h/S \times (1-q) \times w/S = (1-p) \times (1-q) \times h \times w \times S^{-2} \qquad (4.5)$$

p is the overlap, q is the sidelap, h is the height of the photo, w is the width of photo, and S is the scale and shows an inverse relationship with the flying height above the ground using the same focal length. As the setting overlap in the previous flight plan, the ratio of the numbers of total photos at 50m scouting over 200m scouting is 35 for a designated area. Furthermore, the overlap can be reduced to 60% for a 200m scouting that can increase the ratio of the total photos up to 71. The cost of 50m, 200m, and adaptive scoutings varying with the scouting area can be illustrated in figure 4.12. The initial costs are 100USD and 500USD for 50m scouting and 200m scouting, respectively, in Taiwan. According to the regulation of Taiwan Civil Aviation Law, an out-of-sight flight that 200m scouting may encounter needs an extra supervisor standing at commanding heights, who is responsible for connecting the nearby airport controlling center in case of emergency, beside a UAV operator. Thus, the extra supervisor results in a different initial cost between 50m and 200m scouting. In general, the cost of 50m and 200m scouting increases with area coverage. Adaptive scouting combines the cost estimation at 200m and 50m and depends on the area of fine and coarse coverages. In this case study, the adaptive scouting combines

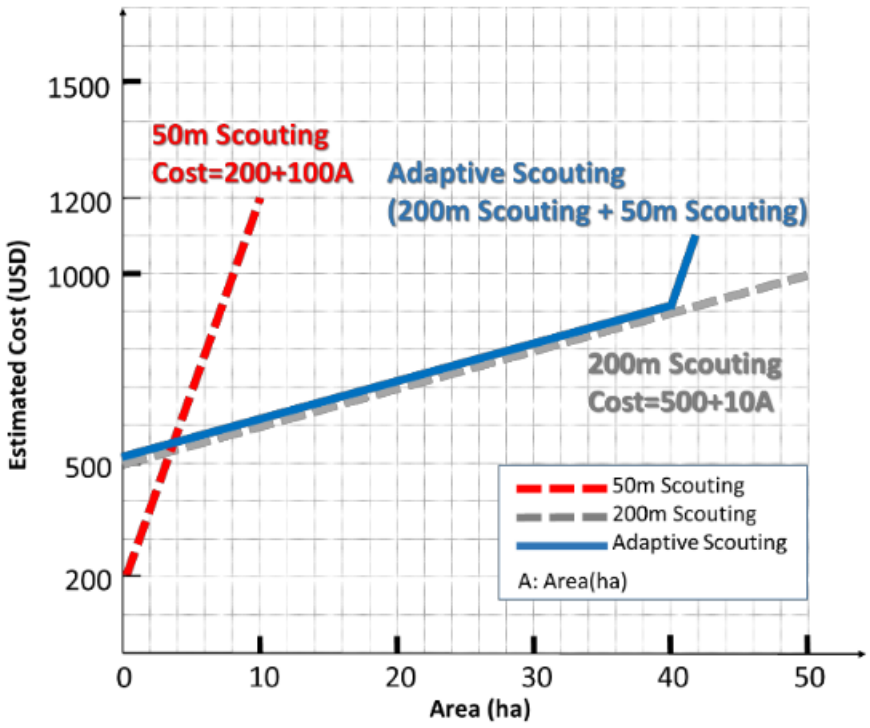

Fig. 4.12: The estimated cost of 50m, 200m, and adaptive scouts varying with the scouting area.

one 200m flight and two 50m flights as one mission to efficiently reduce the total cost and achieve the goal of lodging identification, as shown in figure 4.12.

4.4 Discussion

As demonstrated in the results, we are able to construct a deep neural network for lodged rice detection that outperforms prior work using SegNet and FCN-AlexNet(17). The EDANet approach classifies rice paddy at 95.28% accuracy using RGB, as compared to SegNet at 91.49 and FC-AlexNet at 92.77%. EDANet classifies Lodged rice at 86.17% accuracy compared to SegNet at 70% and FCN-AlexNet

at 77.91%. Even with more information in the form of vegetation indices, the best performance from prior work was garnered using FCN-Alexnet yielding 93% rice paddy accuracy and 80.08% lodged rice accuracy. The adaptive algorithm was able to maintain this newfound accuracy. The adaptive approach scouts at two levels, first estimating highly lodged regions at high altitude, then confirming lodging at low altitude at only those highly lodged areas as efficiently as possible. We were able to maintain 99.25% accuracy using EDANet as compared to a complete scout of the entire field by simply avoiding regions that showed less than 2.5% lodging at high altitude. The adaptive approach allows for a considerable time and costs savings compared to a complete scout of the crop field. Adaptive scouting at optimal accuracy takes 35% less flight time to achieve and saves precious UAV battery life. Flight time is important from a number of perspectives, including urgency, resource savings, and labor costs. Crop scouting for lodged rice is particularly sensitive to these factors. Rice lodging generally only occurs in reasonable quantities during periods of high flooding and serious rainfall, which often coincide with other effects like power loss or limited labor availability and assessment time, which make aerial scouting difficult. Furthermore, lodging must be determined quickly to allow farmers to remove lodged rice or replant crops and promptly receive government subsidies [206] so that aerial scouting resources will be in high demand during these periods. This double effect of high demand and low resources makes the time and energy savings of the proposed approach much more consequential. Scouting approaches also commonly utilize skilled independent pilots or aerial photography companies to survey cropland and locate lodging, which can be expensive. In a period of high volume for aerial scouting, demand for pilots may also increase, raising prices or delaying scouts. The

proposed approach is fully autonomous, requiring only compute resources, UAVs, and labor for setting up and taking down systems. This process could be done easily by farmers, especially if they already use UAVs on the farm and own a reasonably powerful computer. Because the proposed approach does not require human piloting, it costs considerably less over time. There are many available avenues for future work to improve our neural network and approach. First and foremost, increased data collection of lodged rice fields is imperative. Rice lodging, though devastating, is not a regular situation, so gathering lodging datasets is difficult. Increasing the number of samples available will increase the ability of machine learning techniques to accurately detect lodging, like the proposed EDANet approach. Techniques to increase the accuracy of the proposed approach or other approaches with higher lodged rice detection accuracy are also valuable directions for future work. The adaptive approach is not simply useful for rice lodging, but for finding and counting any anomaly that can be detected using aerial image analysis. This could include finding other crop diseases or estimating crop yield but is also applicable to other areas of aerial photography like infrastructure inspection or battlefield surveillance. Applying this technique to other domains may yield superior results compared to simply scouting entire areas at low altitudes, as shown here. The most pressing avenue for future work is to take the proposed approach out of the simulation and test it on an actual rice lodging scenario. Additionally, many newly developed embedded edge computing devices are lightweight and portable, such as Nvidia Jetson TX2, AGX Xavier, or Xavier NX. These devices are suitable for real-time onboard computing power on small drones with restricted space [37], which can be useful to empower our approach. We plan

to address this in future work by using the SoftwarePilot framework for fully autonomous aerial systems [26] and our EDANet for lodged rice detection to implement the proposed approach to scout crop fields in real-time.

4.5 Conclusions

Rice is a globally important crop that will be a necessary component of the earth's food supply for the foreseeable future. Rice lodging is a threat to rice production, hurting yield, and diminishing farmers' income. Assessing rice lodging should be more efficient because current methods rely primarily on random manual sampling. This paper presents an autonomous scouting approach to estimate rice lodging using machine learning and UAVs. The machine learning model using EDANet is capable of identifying rice at 95.28% accuracy and lodging at 86.17%, improving in prior work by 2.51% and 8.26%, respectively. The adaptive scouting approach, which scouts rice at multiple altitudes to target lodged regions, in particular, maintained 99.25% lodged prediction accuracy compared to a complete scan of the field at low altitude while taking 35% less time. The adaptive scouting approach saves considerable money and time and provides a great opportunity to enhance rice lodging assessment over a large area using deep learning techniques on UAV images. The adaptive scouting approach is ready to be implemented in lodging assessments to provide low-cost and in-time digital maps. In the future, edge computing will be integrated into adaptive scouting to identify field anomalies in real-time and complete multi-scale imaging tasks in one flight.

Chapter 5: Programming and Deployment of Autonomous Swarms using Multi-Agent Reinforcement Learning

Autonomous systems (AS) carry out complex missions by continuously observing the state of their surroundings and taking actions toward a goal. Swarms of AS working together can complete missions faster and more effectively than single AS alone. To build swarms today, developers handcraft their own software for storing, aggregating, and learning from observations. We present the Fleet Computer, a platform for developing and managing swarms. The Fleet Computer provides a programming paradigm that simplifies multi-agent reinforcement learning (MARL) – an emerging class of algorithms that coordinate swarms of agents. Using just two programmer-provided functions *Map()* and *Eval()*, the Fleet Computer compiles and deploys swarms and continuously updates the reinforcement learning models that govern actions. To conserve compute resources, the Fleet Computer gives priority scheduling to models that contribute to effective actions, drawing a novel link between online learning and resource management. We developed swarms for unmanned aerial vehicles (UAV) in agriculture and for video analytics on urban traffic. Compared to individual AS, our swarms achieved speedup of 4.4X using 4 UAV and 62X using 130 video cameras. Compared to a competing approach for building swarms that is widely used in practice, our swarms were 3X more effective, using 3.9X less energy.

5.1 Introduction

Autonomous systems (AS) continuously sense their surroundings, model their current state and take actions toward a goal. Edge computing, a paradigm where compute resources are provisioned near sensing devices, has propelled AS in a wide range of industries [23, 87, 144, 228].

Groups of AS working toward a common goal are called *swarms*. Compared to AS working alone, swarms can speed up mission execution. 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. Today, human programmers manually partition missions for swarm operations and each swarm member uses pre-programmed behaviors. For example, in precision agriculture, such *automated swarms* divide a crop field among multiple UAV and each UAV executes pre-programmed scouting routines on its region [15]. Recent research provides automated partitioning and fault tolerance for automated swarms [87]. However, pre-programmed behaviors fundamentally waste resources by collecting and processing data of low value relative to the mission. Further, by limiting autonomy, data sharing between members can not improve efficacy.

Multi-Agent Reinforcement Learning (MARL) is an emerging class of reinforcement learning algorithms where agents cooperate to maximize a reward. Agents learn their own reinforcement-learning policies, but they can also learn from the actions and outcomes of other agents. MARL algorithms applied to AS swarms can speedup 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 [39, 51, 122, 218]. However,

MARL systems are not simple to develop and manage; they require infrastructure for AS workflows, selecting 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.

We present the *Fleet Computer*, a platform for developing and managing MARL-driven swarms. To program AS in the Fleet Computer, 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. In addition, developers also provide hardware resources and mission configuration settings, e.g., allowed actions and goals. With these inputs, the Fleet Computer compiles MARL models that govern autonomous actions. Then, during execution, the Fleet Computer aggregates data from swarm members and retrains the models governing actions. However, retraining stresses limited computational resources at edge sites. The Fleet Computer prioritizes retraining for models most useful for effective actions, making a novel link between online learning and resource management. The Fleet Computer also expands and contracts edge compute resources via duty cycling to save energy and reducing over provisioning.

Just as MapReduce [53] simplified parallel data processing, the Fleet Computer demystifies fully autonomous swarms. In addition to its novel programming paradigm, the Fleet Computer provides end-to-end control, deployment, and scheduling of an entire swarm-support infrastructure – from individual swarm units, such as a UAV, to the supporting heterogeneous compute resources at the edge.

The Fleet Computer comprises 3 main contributions:

1. A programming model and toolchain that allows users to easily build MARL-driven swarms.
2. A novel, online-learning approach to aggregate observations from swarm members and dynamically update models governing actions.
3. A runtime platform that manages, schedules and duty-cycles edge compute resources, simplifying deployment.

We used the Fleet Computer to build both aerial crop scouting and video analytics systems, demonstrating its broad applicability to significantly different problems. We developed a swarm of unmanned aerial vehicles used to predict crop yield from flyover images. We also developed a taxicab tracking workload using a swarm of autonomous cameras. By providing a simple way to take advantage of state-of-the-art MARL algorithms, our crop yield mapping application outperforms state of the art yield mapping, improving mapping times by 4.4X using 3.9X less UAV power. Similarly, our vehicle tracking workload built using the Fleet Computer outperforms prior work, tracking taxis up to 62X faster as a swarm compared to centralized processing while maintaining good performance over time by updating models regularly. Using these applications, we demonstrate that a compute cluster running Fleet Computer software can easily scale up from a single UAV, camera, or other agent to a complete swarm that can learn from its actions while dynamically allocating resources to fit demand and save precious power at the edge.

The remainder of the paper is organized as follows: Section 5.2 describes at a high level the overarching design of the Fleet Computer. Section 5.3 details our programming model, which allows users to easily build autonomous swarms that perform well.

Section 5.4 discusses the Fleet Computer's runtime, which includes a priority-based online learning approach to maintain model performance as environments change, and cluster autoscaling which saves edge power without sacrificing performance. Section 5.5 covers two Fleet Computer implementations: video analytics and autonomous crop scouting. Section 5.6 presents results for the Fleet Computer on both applications. Section 5.7 presents related work and Section 5.8 provides conclusions.

5.2 Background

Self-driving cars [123, 195], UAV [30, 167, 216], and other autonomous systems (AS) [33,66,156,180] are transforming society. Investments in self-driving cars exceed $56B [132]. UAV are transforming delivery, infrastructure monitoring, and surveillance [4,7,30,87,175]. Autonomous cameras with mobile gimbals are powering traffic monitoring and smart cities [90]. In this paper, the term *autonomous* describes systems that sense their surroundings, infer their state, and take actions toward mission goals *without human intervention.*

By design, AS execute in unfamiliar surroundings. Their efficacy depends on how well their actions align to mission goals. Broadly, AS can be characterized by two key design choices: (1) Do they learn from their own observations? And (2) do they learn from the observations of other AS? For many AS used in practice today, the answer to both questions is no. These *pre-programmed AS* repeat the same routine across all missions, often taking unneeded and wasteful actions.

With reinforcement learning, AS can learn from prior observations [30,65,98,123, 167]. However, often in practice, AS observe the world too slowly to capture enough data for learning, especially in non-stationary contexts where the best actions change.

Multi-agent reinforcement learning (MARL) addresses this problem by aggregating data from multiple AS (i.e., swarms). Previously, the challenge for MARL-driven AS has been deciding from which swarm members to learn, but recent research has developed online algorithms to explore data aggregations that have provable guarantees and/or strong empirical results [39, 51, 122, 218]. However, even with recent research, swarms require additional compute resources compared to single-agent AS. To share data, swarms members must be networked, e.g., via hubs [17,197] or wireless [85,198]. Exploring data aggregations also requires additional compute resources.

Autonomous UAV for Crop Yield Modeling: To illustrate these concepts, consider a farmer that owns 4 UAV controlled via apps running on 4 tablets. The tablets and an edge-hub desktop share a wireless Internet connection. The mission is to map the expected crop yield for each $0.01\times$ acre lot in a $1,000$ acre field with accuracy above 80%. For the approach most commonly used in practice, the farmer would use pre-programmed routines to exhaustively capture images from $100,000$ waypoints. This approach is slow and unnecessarily exceeds accuracy goals. A pre-programmed swarm could split the mission into 4 parallel tasks. With reinforcement learning, AS can visit significantly fewer waypoints [23, 167, 223] by continuously modeling the expected accuracy of their map and prioritizing valuable waypoints to visit next. MARL-driven AS improve upon naive reinforcement learning by using shared images to visit fewer waypoints. However, the computational load for reinforcement learning and MARL exceeds the capacity of UAV processors and tablets, requiring resources of an edge hub or the cloud.

5.3 Design

As shown in Figure 5.1, the Fleet Computer is an end-to-end platform for autonomous swarms, covering the development of AS, their workflow and coordination, and their execution on edge computing devices.

To create an AS, developers implement two functions, Map() and Eval(), and specify mission configuration. Map() functions convert quantized input from sensing devices (e.g., cameras, GPS, etc.) into a feature vector that represents the state of the AS. 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 5.1 depicts three examples of mission configuration settings. First, developers can provide thresholds to determine when missions are complete. By default, the Fleet Computer supports thresholds on accuracy and energy usage. Second, developers specify the type of resources needed for sensing, processing, and data aggregation by stipulating quantitative requirements, such as CPU and memory required for Map() execution. Third, users provide qualitative requirements, such as support for specific Action Drivers. Action Drivers support a cyber-physical system's actions, e.g., take off, land, sense, and fly to waypoint.

The Fleet Computer compiles these inputs to create AS workflows for sensing surroundings and taking actions. Here, the challenge is to decide which actions to take after running Map() and Eval(). The Fleet Computer automatically builds state-to-action (SA) models and history-to-action (HA) models by (1) replaying data from prior execution contexts, and (2) learning effective actions that improve Eval()

outcomes. SA models convert a single map output to actions whereas HA models convert multiple observed outputs.

The Fleet Computer models MARL-driven AS as three asynchronous components: Sensors, AS Workflow, and Data Aggregation. These components execute in shared-nothing containers connected via distributed storage (e.g., HDFS [179]). Containers are replicated to support swarms. Precisely, a swarm of size N will comprise N Sensor and AS containers and up to 2^N Data Aggregation containers (representing every possible combination of aggregations). Clearly, Data Aggregation containers impose computational demands that exceed system capacity— a challenge common to all MARL-driven approaches [39, 122, 218]. For small swarms ($N \leq 8$), The Fleet Computer deploys all Data Aggregation containers and employs a novel, priority-based online learning and scheduling to manage compute demand. For larger swarms, the Fleet Computer supports developer-provided heuristics to limit aggregation.

In this section, we first provide a rigorous primer on MARL algorithms. Then, we introduce the specification of Fleet Computer applications, i.e., swarms of AS. Finally, we describe each of the key functions and models listed above.

5.3.1 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 [219]. A policy π is a means of determining

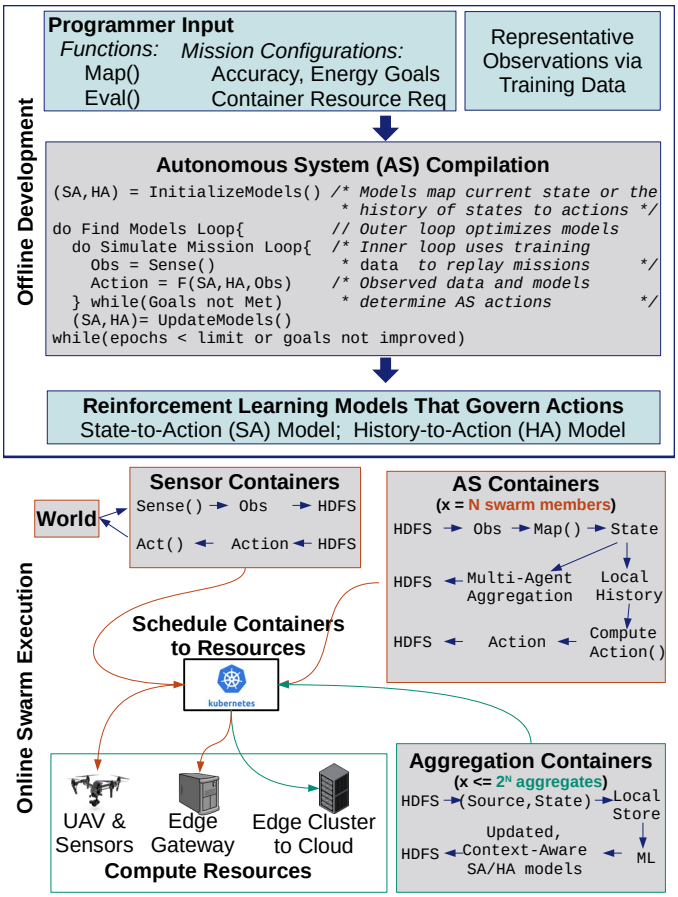

Fig. 5.1: The architecture of the Fleet Computer.

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.

$$MDP : (S, A, P, R, \gamma) \tag{5.1}$$

MDPs are a useful tool for solving simple state transition problems, but fleet missions are too complex for this framework. Consider the crop yield modeling example from Section 2. 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 [83, 121, 142, 202]. 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 value (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 accuracy by learning $\pi^{*\prime}$, allowing them to learn complex behaviors through exploration.

$$\begin{aligned} Q(s_i, a_i) = (1 - \alpha) * Q(s_i, a_i) + \\ \alpha[R(s_i, a_i, s_{i+1}) * \gamma max(Q(s_{i+1}, a_{i+1}))] \end{aligned} \tag{5.2}$$

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 [42]. Another approach, team-average reward [59, 96], 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) [125]. 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 [160, 218, 220], agents within a MG that communicate over some time-varying network, may have individual reward functions, and may require data privacy.

$$MG = (N, S, A^i_{i \in N}, P, R^i_{i \in N}, \gamma) \tag{5.3}$$

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

5.3.2 The FleetSpec

$$FleetSpec : (N, S, A^i_{i \in N}, Map(), Eval(), \mathbb{C}) \tag{5.4}$$

Equation 5.4 presents the minimum specification for Fleet Computer applications, called the FleetSpec. Similar to a Markov Game, the FleetSpec 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, FleetSpec eschews state transition probability and reward functions. First, Fleet Computer 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 AS 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, the Fleet Computer learns state-to-action (SA) models and history-to-action models (HA) automatically through training and reward shaping. These models represent an approximation of $\pi^{*\prime}$. The Map() and Eval() functions, coupled with representative observations from prior missions $\mathbb{C}$, suffice to compile initial MARL models and reward functions. The remainder of this section details the compilation process.

5.3.3 Map() and Eval() Functions

Like in MapReduce, Map() functions in the Fleet Computer structure input data. AS 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.

$$D =< d_1, d_2, ...d_n > \tag{5.5}$$

$$Map(D) = SSV =< f_1, f_2, ...f_m > \tag{5.6}$$

By emitting a structured SSV, Map() functions in the Fleet Computer 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 5.2 for our crop scouting example. $Map(D_j)$ for crop scouting provides data D_j to extractors including $ExG()$ which determines excess green [105] (a metric for predicting crop yield), $LAI()$ which estimates the leaf area index [162] of crops in the image, and a CNN which counts the number of corn stalks in the image among other extractors. 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.

Eval() determines whether and to what degree an AS has accomplished its goal. For some AS, this can be as simple as reaching a certain state. For others, like the autonomous crop scouting example, 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.

$$FS = \{SSV_1, SSV_2, SSV_3...SSV_n\} \tag{5.7}$$

$$P = \{\rho_1, \rho_2, \rho_3...\rho_m\} \tag{5.8}$$

$$Eval(FS, m) = P \tag{5.9}$$

```
func[] extractors = [ExG(), LAI(),
                CornCountCNN(), ...]
float[] Map(Obj data) {
  float[] SSV = [];
  for(e in extractors) {
    SSV.append(e(data));
  }
  return SSV;
}

Obj[] Eval(float[][] FS, float[][] perf) {
  finished = checkGroups(FS, HA)
  map_final = []
  if(finished) {
    map_gt = buildMap(FS)
    map_final = extrap(map_gt)
  }
  P = buildMetrics(perf)
  return [finished, P, map_final]
}
```

Fig. 5.2: Map and Eval function pseudocode for crop scouting.

Eval(), defined above, accepts a feature space FS comprised of $n \geq 1$ SSVs and a set of system-level metrics m, and outputs an evaluation P which includes $x \geq 1$ evaluation metrics $\rho_1..\rho_m$. The number of state-space vectors and evaluation metrics required is task and goal dependant.

For our crop scouting UAV example, many or all sensed SSVs from a mission may be needed to build an accurate yield map. Evaluation metrics for crop scouting may include system execution time, accuracy, energy expenditure, monetary cost and any other metric that determines real-world performance. Eval() pseudocode for crop scouting is shown in Figure 5.2.

Eval accepts two parameters, a featureset of all SSVs collected by the system, and a set of performance metrics for each agent. First, we use the HA model to determine whether the mission has concluded by exploring all state groups, a process detailed in section 3.4. If the mission is finished, a final yield map is generated by converting FS into a ground-truth yield map by mapping ExG from each SSV into a yield prediction and mapping it onto the execution context. Then, unvisited zones are extrapolated using an approach from prior work [223]. Next, Eval() uses performance metrics from every agent to build a set of global metrics that can be used for goal evaluation, P. Finally, Eval() returns a Boolean stopping condition based on HA, evaluation metrics, and potentially a final yield map which will be returned to the user.

5.3.4 State-To-Action and History-To-Action Models

Designing MARL policies and reward functions is a complicated problem with many situational solutions. Each action taken by an agent must be assessed by a policy before it is selected, and assigned reward if it is taken. The Fleet Computer's model training and reward shaping steps simplify this process, allowing users without predefined models or reward functions the option to build them automatically.

Training reinforcement learning policies generally requires releasing an agent into an execution context and allowing it to explore, building a policy using a reinforcement learning model from the reward it garners from its actions. Characterizing entire real-world execution contexts for training models is a well-explored problem. Point-cloud datasets have long been used in robotics and computer vision for simulating navigation and robotic manipulation in real-world environments [52, 149, 181, 188]. Data sets for self-driving vehicles [75, 103, 214] and video analytics [145, 227] provide

Name	**Linkage**	**Data Type**	**Devices**	**Use-Cases**
Point clouds	Spatial	RGBD, ASCII	Robots	Indoor and outdoor Navigation
Video streams	Spatial, Temporal	Video	Self-driving cars, Video analytics	Transportation, Tracking,
Autonomy cubes	Spatial, Temporal	Images, Video	UAV	Precision agriculture, Rescue, Photography

Table 5.1: Execution context data sets and their use cases.

labeled video streams of fully explored environments used to train algorithms in both domains. A similar approach, Autonomy Cubes, provides spatially and temporally linked images in the form of hypercubes representing completely explored execution contexts regularly used to develop autonomous UAV workloads [30, 205, 223].

Any of these execution context representations can be used to train MARL models for the Fleet Computer. In the FleetSpec, these execution contexts are defined as $\mathbb{C}$. Map() and Eval() combined with autonomy goals allow agents to navigate these environments just like the real world, with Map() extracting data from a given state, and Eval() determining whether autonomy goals are sufficiently accomplished. These functions do not, however, constitute a policy. The policy for navigating these environments is defined by State-to-Action and History-to-Action models detailed below.

A State-to-Action model (SA) is a blank or pretrained MARL model $SA \approx \pi^*$. SA accepts a SSV as input and returns an action. SA is not strictly tied to any RL or MARL model format, and is meant to be general. By default, the Fleet Computer supports Q-learning, but other model types, like DQNs or DDPGs can be substituted as an SA baseline. SA can also be provided pre-trained, or the Fleet Computer can instantiate it from a blank model and train it with data from $\mathbb{C}$. The only restriction placed on SA by the Fleet Computer is that it must accept an SSV produced by Map() as an argument.

History-to-Action() models provide spatial support to SA models. An HA model determines if an AS has accomplished its goal within its execution context. As an AS explores its execution context, some states of low relevance can be excluded or extrapolated to conserve time, energy, and compute resources. MARL models generally rely solely on reinforcement learning to determine which states to search, but recent video analytics work [90] demonstrates how spatial and temporal correlations can help prune search spaces effectively. In response to this work, we propose that groups of states (state groups) be allocated in conjunction with HA models to better search across execution contexts.

A state group is a collection of states within the execution context, with each state belonging to one or more state groups. State groups provide spatial bounds on AS. Often, phenomena sought by AS is spatially correlated [90, 223]. By defining spatial bounds, AS can limit their searches, saving time and resources. HA determines the number of states explored in each group. After a state in the state group is visited, the feature vectors of each visited state in that group are provided to HA to determine whether to explore another state in that group, or to visit another group. If the

system decides to visit another state group, all unexplored zones in the group are either disregarded or predicted by Eval(). HA is defined as follows.

$$HA(FS) = \begin{cases} U > T_u \text{ or } V > T_v & 0 \\ else & 1 \end{cases} \tag{5.10}$$

$$U = \sum_{i=0}^{n} R(S_i, A_i, S_{i+1}) \tag{5.11}$$

$$V = \|FS\| \tag{5.12}$$

HA accepts a feature space FS comprised of one or more state-space vectors $SSV_1..SSV_n$ from the same state group. $HA(FS)$ returns a Boolean value representing whether to continue exploring a state group (0) or to move on and explore another (1). This decision is made using U, the total utility of the feature space, and V, the size of the feature space (number of visited states in the group). The utility of a feature space is the aggregate reward received from all state transitions within that space. If U is above T_u, the utility threshold, or V is above T_v the visited states threshold, all remaining states in the group are ignored or predicted by Eval() and another state group is visited. The two thresholds strike a balance between allowing the MARL algorithm time to find high reward state transitions and limiting the total number of states visited to maintain efficiency.

These thresholds, as well as the means for determining reward, are difficult to determine a priori. Rigorous construction of high-quality reward functions is still an open problem, and could be difficult for users attempting to apply autonomy to a new domain or in an unclear application [108, 219, 229]. Work has been done on reward shaping [84, 124, 229], but there is still no best practice on how reward functions should be developed. We take inspiration from recent work that used meta-learning

via gradient descent to construct reward functions [229]. Because our Map() functions have smaller numbers of features, we can define a rigorous means for estimating optimal reward functions and thresholds through a simpler and faster technique, Bayesian optimization [71].

5.3.5 Reward Shaping and Training

Provided one or more execution context data sets, Map() and Eval() functions, and SA, our algorithm initializes hyperparameter values for the reward function and HA thresholds, then simulates autonomous missions over each execution context, and evaluates performance. We use bayesian optimization [71] to find a goal-maximizing combination of Reward, SA, and HA.

$$R(S_i, A_j, S_{i+1}) = \sum_{f=0}^{n} S_{i+1}^{f} * w^{f} \tag{5.13}$$

Reward, shown above, is the sum of each normalized feature of S_{i+1} multiplied by its corresponding weight w_i. Weights are a feature specific hyperparameter between 0 and 1. Using these properties, we can determine that the output of $R(S_i, A_i, S_{i+1})$ will be in the range $[0, \|F_i\|]$. For this reason, T_u may only fall in the same range. Similarly, T_v must be an integer value between $[0, V]$. Through simulation, Bayesian optimization tunes these values until user-specified accuracy and energy requirements are met.

Bayesian optimization is a function approximation technique that minimizes or maximizes objective functions with many parameters through creatively searching their state-space [71]. Users must provide a set of defined autonomy goals G =

```
float[] buildReward(int nFeats, Obj[] C, int numEpochs, float[]
    G) {
  float[] W = initWeights(nFeats)
  int T, T* = GROUP_SIZE
  int V, V* = MAX_INT
  int bestLoss = MAX_INT
  float[] maxWeights = []

  for i in range(numEpochs){
    float loss = 0
    for con in C{
       float[] e = sim(con,W,T,V)
       loss += F(G, e)
    if(loss < bestLoss and goalsMet(G, e)]
       bestLoss = loss
       maxWeights, T*, V* = W, T, V
    W, T, V = Bayes([W, T, V], loss)
   }
  return maxWeights;
}
```

Fig. 5.3: Bayesian reward shaping pseudocode. Reward Shaping seeks to find the set of hyperparameters which minimizes loss and meets goals.

$\{g_1, g_2, ..g_n\}$ corresponding with outputs of the Eval() function. Using G we can construct a multi-variate loss function which serves as the objective function for Bayesian optimization.

Figure 4 shows our Bayesian reward shaping process in pseudocode. Our buildReward function accepts 4 parameters: the number of features from Map() to be weighted, the subsetset of contexts to be simulated across $\mathbb{C}_t$, the number of training epochs, and the list of autonomy goals G. Before training, weights and thresholds are initialized. Training consists of simulating missions using each context $c \in \mathbb{C}_t$. Loss is calculated based on autonomy goals G and the final simulation evaluation

$e \forall con$ using objective loss function $F(e, G)$. The additive loss for all cubes is then compared to the best prior loss. If loss is less than the best previous loss and all goals are met, the hyperparameters are saved. When the last training epoch concludes, the hyperparameters with the least additive loss that met all goals are returned, providing complete HA and reward functions.

Once reward shaping has concluded, SA is retrained with a separate subset of contexts $\mathbb{C}_r$ using a similar process. Missions are simulated across $\mathbb{C}_r$ with the new reward function and evaluated against prior training examples to select the best combination. The number of retraining periods for both SA and reward shaping are specified by user-defined hyperparameters.

5.4 Runtime

Using our goal-based development techniques for AS, users can more easily develop and train MARL algorithms for their individual problems. Output SA models may, however, be insufficiently trained to operate well in specific deployments. To help build high quality systems with limited initial training data, and allow algorithms to adapt to dynamic environments, we introduce a federated online learning system for MARL algorithms within the Fleet Computer. This system allows SA and HA models for individual swarm members to diverge to maximize global reward. Using an online learning component also introduces a scheduling and resource management problem domain for executing online learning tasks at the edge. In this section we first describe our federated learning approach to MARL for AS which uses Bayesian optimization to determine model usefulness for efficient retraining. We then present

our runtime scheduling and resource management system for these online learning tasks, aided by Kubernetes.

5.4.1 Bayesian optimization for system-level hyperparameters

When a swarm is deployed, each member uses the original model SA for pathfinding decisions. As missions complete, swarm members' future performance may benefit from retraining using the the data they sense. It is unclear, however, whether retraining will immediately help or hurt performance, and which data should be used for maximum retraining performance. To solve these problems, we retrain a set of models SA_s using different subsets of collected data and weight their performance for every system.

Our solution to this problem is depicted in Figure 5.4. After a deployment's first mission, each swarm member has sensed and stored new data that can be used by aggregation containers for retraining. The sensed data can be partitioned into an infinite number of subsets to retrain $SA \rightarrow SA_s$. Using intuition from the federated learning community [110], we define the upper bound of SA_s as the set of models retrained using all combinations the powerset of N, $\mathbb{P}(N)$. This gives us latitude in selecting optimal models, but provides resource challenges for large swarms. The Fleet Computer provides developers the option to provide heuristics to prune aggregators from SA_s, and can use usefulness to prune SA_s online. It is unclear a priori how each model in SA_s will affect each swarm member, so training effects must be determined online. Therefore, we track a series of performance-based system level hyper-parameters.

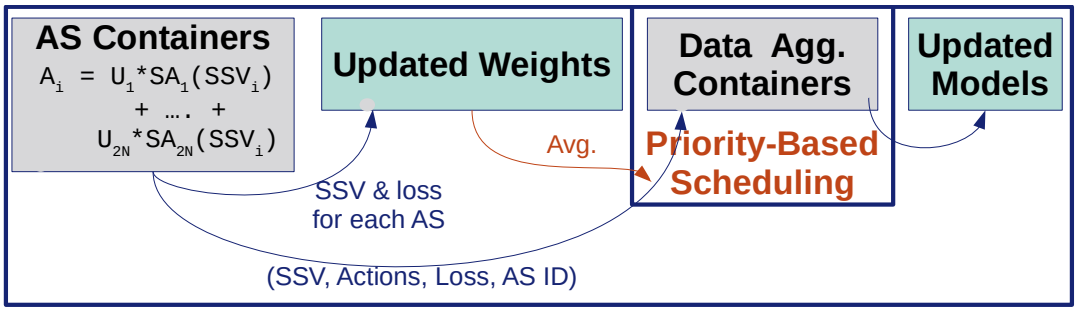

Fig. 5.4: Online learning provides coefficients for priority-based scheduling to update MARL models.

We track model weights $\{U_w^1..U_w^i\}$ for models $\{SA_1..SA_i\} \in SA_s$. Model weights describe the weighted percentage ($0 \leq U_w^x \leq 1$, $\sum_{n=1}^{i} U_w^n = 1$) that decisions from that model are incorporated into a pathfinding decisions. The goal of our approach is to determine the values for these hyperparameters which yield us the best mission performance. Per swarm member, we define these hyperparameters as x.

$$x = \{U_w^1, U_w^2, ..U_w^i\} \tag{5.14}$$

$SA_s(x)$ describes the Fleet Computer mission whose pathfinding decisions for a single agent are determined by SA_s weighted by x. The Fleet Computer's goal is to determine a set of hyperparameters x^* that best accomplishes autonomy goals. More specifically, we want to optimize the following objective function.

$$x^* = x \; s.t. \; argmax(Eval(SA_s(x))|SA_s(x) > G) \tag{5.15}$$

We use $Eval(SA_s(x))$ representing the evaluation of a full mission informed by $SA_s(x)$. Our goal is to find x*, the set of weights that maximize the evaluation, while assuring that our autonomy goals G are met.

Bayesian optimization [71, 143] is a powerful technique often used in hyperparameter tuning to improve machine learning algorithms performance. Bayesian optimization works well for optimization problems where objective functions are derivative free, expensive to evaluate, noisy, lack structures that are easy to optimize, and have small parameter sets [71]. Reinforcement learning approaches often fit all of these criteria, ours included. To optimize our hyperparameters, we use expected improvement to quickly minimize our objective function given specified goals. Expected improvement approximates the global maximum of our objective function by sampling hyperparameters based on the posterior distribution of prior sample points and loss. Expected improvement determines hyperparameter samples with a balance between the highest probability for improvement and the largest magnitude of that improvement. We model this as a Gaussian process.

The above approach frames Bayesian optimization in the scope of a single agent's parameters, but we can use this approach to optimize hyperparameters for multiple agents. Using the same local-optimization approach, we can determine a near-optimal x to maximize global goals. Each swarm member maintains its own set of hyperparameters x_i which is updated asynchronously after each mission. The collective set of hyperparameters provides us a simple metric for determining how 'useful' each model in SA_s is to overall pathfinding decisions. Models with higher average U_w^i provide more insight into pathfinding decisions, and therefore should be updated more regularly and provided more retraining resources.

5.4.2 Scheduling and power management

The Fleet Computer consumes edge resources to support movement, actuation, pathfinding, data storage, and online learning. Within this set of activities it is important to balance the resource needs of critical real-time latency-sensitive processes, like movement control, with compute intensive tasks like retraining routines that offer significant long-term gain. It is also important for edge resources to be responsive to workload changes, powering down edge devices in low load periods to maximize system liveness and mission length, and scaling out in peak loads to support effective decision-making. Here we describe our solutions to both of these challenges: Section 5.4.2 presents our priority-based container scheduling solution for latency-sensitive and compute-intensive tasks, which factors in utility of each compute-intensive training task to overall mission objectives, and Section 5.4.2 describes our cluster autoscaling mechanism to adjust to workload changes.

Priority-based Scheduling

The Fleet Computer deploys all of its core components in containers [138] to maintain hardware independence and support scale-out. Figure 5.1 shows the different container types that encapsulate pre-built inputs like MARL algorithms, retraining routines, and feature extractors along with core Fleet Computer platform elements.

Container scheduling across clusters is well understood [40]. We use Kubernetes [171] for automated bin-backing of containers across clusters, allocating resources for all Fleet Computer components defined as Docker containers. We use priority-based scheduling to ensure that latency-sensitive real-time tasks such as UAV flight control are assured to be executable within their latency window; we then use

additional edge compute to schedule model retraining tasks based on newly-received data from swarm members of the in-progress mission.

For model retraining tasks, we augment the resource allocation algorithm used by Kubernetes to optimise *placement* and *task selection* for Fleet Computer operations. Placement decisions are impacted by data locality, where training data for SA and HA models is typically fragmented across multiple systems within the Fleet Computer. 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 utility of a retraining task: when edge compute resources cannot support all retraining tasks, we choose those with the highest average utility. The Fleet Computer uses Kubernetes' priority scheme for scheduling training procedures using the aggregate usefulness of each model provided by AS containers as shown in Figure 5.4. Model usefulness is simply the floating point value $U_i = [0, 1]$ assigned by Bayesian optimization, as explained in Section 5.4.1. If model usefulness is calculated by multiple swarm members, usefulness is weighted evenly among them. Scheduler priority for model i is then:

$$P_i = \lfloor 10 * U_i \rceil * 100,000,000 \tag{5.16}$$

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 sensor containers.

We also use usefulness 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.

$$CPU_i = \frac{CPU_t}{\sum_{j=0}^{s} P_j} * P_i \tag{5.17}$$

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 are not scheduled. This priority mechanism allows us to provide more resources to retrain models that agents consider useful, and avoid retraining models that provide little to the system.

Cluster Autoscaling

Because Fleet Computers may include many nodes, and edge devices in particular are likely to be provisioned for peak loads rather than average load, it is beneficial to regularly scale the cluster up and down in response to workloads.

The Fleet Computer includes a custom Kubernetes autoscaler which drains compute tasks from superfluous nodes and powers them down, saving on edge power when loads are low, and re-powers decommissioned nodes using Wake-On-LAN [140] when the system detects that more compute will soon be 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 the Hadoop Distributed File System (HDFS) [179] 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.

5.5 Applications

We use two very different application types to help evaluate the Fleet Computer: one using a swarm of crop scouting UAVs, and one using a swarm of taxi tracking cameras.

Our crop scouting swarm builds on prior work [223] where UAVs navigate a corn field and capture images which are used to construct a yield map as outlined in Section 2. Farmers use yield maps to inform crop management strategies like fungicide, pesticide, and herbicide application. Large crop fields may cover hundreds of acres, requiring days of swarm coverage and human labor to gather a complete yield map. Prior work used reinforcement learning to creatively sample fields, covering a fraction of the field autonomously and predicting the rest. This approach creates usable yield

maps in a fraction of the time at lower cost, but requires considerable developer effort. Actions in the crop scouting application are UAV direction movements, in a field which is divided into a grid. In this application we used the same dataset of corn images as prior work, consisting of 684 UAV sensed 4608x3456 images of a corn field in London, Ohio. We input the same Q-learning based model and feature extraction from prior work into the fleet spec as SA and $Map()$, and modified the extrapolation function from prior work which generated crop yield maps to produce an $Eval()$, which provides goal information like edge and UAV energy along with overall map accuracy. We used the Fleet Computer's reward shaping mechanism to build reward and distance functions, defining state-groups as unique 3x3 regions of states.

Our taxi tracking swarm also builds on prior work in video analytics [90]. Spatula is a cross-camera video analytics framework: for a specific target (a person, taxi, etc) in one video stream, it returns all frames across all cameras which contain that target. Spatula avoids searching all cameras by searching only cameras that are highly spatially and temporally correlated. Spatula builds these correlation matrices offline using prior execution data. Online, cameras are only searched if their spatial and temporal correlation scores are higher than user-provided thresholds. This approach performs considerably better than searching all cameras and frames, but runs the risk of staleness as movement patterns change over time, and also requires users to manually determine threshold values. To implement this application we used the Porto Taxi Service Trajectory dataset [145], also used to test Spatula. Similar to the original implementation, we generated spatial and temporal correlations using 130 virtual cameras pinned in an evenly spaced grid within Porto, Portugal. Correlation matrices were provided to the Fleet Spec as the State-to-Action model, and trajectories

from the Porto data set were provided in place of $Map()$. We built a custom $Eval()$ function which reports overall accuracy and the total number of frames checked for targets throughout execution, and we generated an HA function to determine optimal temporal and spatial thresholds. We defined state-groups as 50% overlapped 1-minute sets of video frames from each camera.

Online learning for both systems consists of retraining models and rebuilding reward and distance functions using recently received execution data. For crop scouting, swarm sizes were small, so we dispatched $\mathbb{P}(N)$ aggregators for retraining after every crop-scouting mission (i.e when a swarm generates a final yield map). For Spatula, swarm sizes are large, so we dispatched N aggregators after every simulated day to rebuild correlation matrices and relearned thresholds.

5.6 Evaluation

We implemented swarms for both applications described in Section 5.5 using multiple competing approaches. In this section, we describe our experimental testbed and then evaluate the efficacy and performance of Fleet Computer swarms.

5.6.1 Architecture

The Fleet Computer is designed for edge deployments. Our canonical prototype uses 6 total machines, comprising 5 consumer laptops as gateways and a server as an edge hub. Consumer laptops include 3 HP 250 G6 laptops with i5 CPUs and 8GB RAM and two Lenovo Thinkpad T470s with i7 CPUs and 8GB RAM. The server includes an i9 CPU, 32GB RAM, and an Nvidia RTX 3080 GPU. Each machine runs Ubuntu 20.04 Linux. All systems in the Fleet Computer are connected by Ethernet to a 10 Gbps Netgear router.

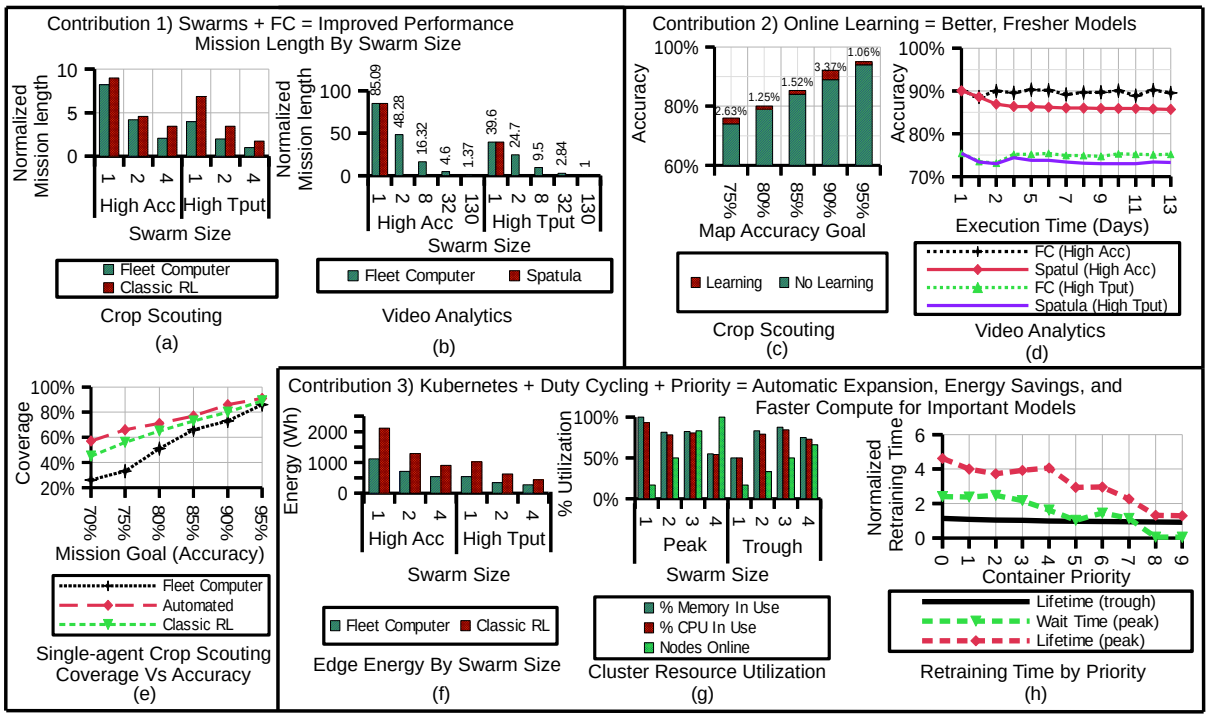

Fig. 5.5: Experimental results: a,b,e) The Fleet Computer's programming model and swarm capabilities improve performance considerably, c-d) Online learning keeps models fresh and helps adapt to new execution contexts, f-h) The Fleet Computer's system management features save energy and improve performance by duty cycling hardware, expanding compute efficiently, and prioritizing high-value model updates.

The Fleet Computer uses one of the Lenovo Thinkpads as a master node to control Kubernetes, Docker, and HDFS. The master node also runs a custom Kubernetes governor, a collection of daemons that create the Kubernetes cluster, start autonomous missions, automatically allocate and prioritize retraining containers, autoscale the cluster, and duty-cycle other machines. When retraining occurs, containers are scheduled by the governor as pods in the Kubernetes cluster. Each pod is scheduled based on its aggregate usefulness as determined by all swarm members as discussed in Section 5.5.

For our crop scouting application, DJI Mavics provide the base UAV characteristics for emulation. Each UAV in a swarm is assigned an HP or Lenovo laptop as a gateway for communication with the Fleet Computer. The UAV's control software runs in a container on the gateway. To gather crop scouting results, we simulated UAV by replaying data captured from previous missions. All UAV control software executed as it would in the field, but data was provided by software. This allowed us to execute all UAV commands without flying, but receive appropriate execution data based on validated energy and latency models [30].

5.6.2 Results

We evaluate the Fleet Computer's comparative performance against state-of-the-art swarm control, the effectiveness of our runtime scheduling approach in saving energy on edge devices, and the effectiveness of our online learning approach at autonomously selecting effective data aggregations for retraining on our resource-constrained edge devices.

Figure 5.5 (a-b) shows the Fleet Computer's performance on our crop scouting and video analytics workloads *without* additional online learning. For crop scouting, we compare against the state of the art in prior work [223] (*Classic RL*) which utilized Q-learning to map crop fields. We test the Fleet Computer against classic RL using 3 swarm sizes (1, 2, and 4 UAVs), and two autonomy goals: one seeking high-accuracy (¿90%) maps; and the other prioritizing throughput, accepting 70% accurate maps in exchange for fast sampling. Both approaches started with the same Q-learning State-to-Action model, but the Fleet Computer generates its own History-to-Action model and reward function from experience.

For the both accuracy conditions, the Fleet Computer decreased mission length by 10-75% compared to Classic RL. Using a swarm of 4 UAVs, compared to a single UAV as used in prior work, we observe a mission length decrease of 3.9-4.4X. This decrease is in part due to having more UAVs, but represents more than 4x the performance gain due to the Fleet Computer's intelligent mission learning and redesign.

Figure 5.5(e) shows why Fleet Computer missions complete more quickly – compared against both a Classic RL and current-industry-practice automated search which scouts waypoints linearly, row by row until the coverage goal is reached [15,87]. The Fleet Computer consistently outperforms both prior approaches, most notably at lower accuracy goals, by sampling an average of just 26% of a crop field to generate a 70% accurate yield map, improving over Classic RL and automated by 1.73X and 2.2X respectively. These improvements are gained because the Fleet Computer learns from experience which areas of a field are likely to be similar to adjacent regions and so do not need detailed data capture.

Figure 5.5(b) shows the Fleet Computer's performance against Spatula. Spatula uses a shared correlation matrix and a single controller to implement cross-camera analysis, and so is only shown in the '1' category on Figure 5.5(b). The Fleet Computer enables us to easily model cross-camera analytics as a distributed swarm of cameras, where each agent is provided compute resources to respond to global queries across one or more cameras. Using this approach, the Fleet Computer can highly parallelize query response and so offers very large potential performance gains without complex programming. We used the Fleet Computer's History-to-Action model to automatically set Spatula's spatial and temporal thresholds, and tested performance by swarm size and for high throughput and high accuracy goals. Our experiments

examine performance gains at increasing numbers of swarm agents applied to the same problem, which is easy to configure through the Fleet Computer; at the highest number of 130 swarm agents we found that the Fleet Computer can parallelize Spatula's workload highly effectively, processing queries 39X and 62X faster than the single Spatula controller.

When we introduce *online learning* to these scenarios we see further improvements for both applications. Figures 5.5(c) and (d) show how crop-scouting can improve over time, and how regularly updating HA models can maintain Spatula's performance as movement patterns evolve.

Figure 5.5(c) shows how online learning improves crop scouting accuracy. In this experiment we run 10 successive real-time swarm missions across simulated crop fields, allowing online learning to continually improve models across a 4 UAV swarm for accuracy goals between 75% and 95%. By the 10th mission we see that every condition moves closer to its target accuracy level by between 1% and 3%; if we measure this as decrease in relative model error from the target accuracy this equates to between 5% and 28% improvement. These savings improve accuracy goals with no increase in resource requirements or mission lengths.

Figure 5.5(e) shows that Spatula's performance with retraining remains consistent with autonomy goals if retrained, but degrades quickly without retraining. We trained Spatula on one day of Porto Taxi Data using all 130 cameras and 448 Taxis, then examined its performance compared to the Fleet Computer on each following day of a two-week period after its initial training. For both high accuracy and high throughput goals, the Fleet Computer remains consistent (within 1%) with accuracy goals. Spatula, in both cases, consistently under-performs accuracy goals after 2 and

4 days for high accuracy and throughput respectively. At the end of the two week period, the Fleet Computer outperforms Spatula by 5% and 3% for high accuracy and throughput respectively by maintaining fresh models.

We next examine the Fleet Computer's resource management approach and its effect on edge site energy usage. Figures 5.5(f-h) show how the Fleet Computer's edge-focused Kubernetes runtime improves overall edge performance on our crop scouting benchmark. Using the prototype Fleet Computer, we ran 10 real-time crop scouting missions for each swarm setting using UAV-collected data with modeled UAV movement, timings, and data-transfer provided by prior work [30]. Energy was calculated using an AC watt meter. We tested 1, 2, and 4 drone swarms with both high accuracy and high throughput goals. Figure 5.5(f) shows the Fleet Computer's performance against Classic RL in terms of energy consumption. The Fleet Computer conserves energy in two different ways: mission lengths are shorter overall, and use of edge sites during a missing is cheaper due to automated power-down of resources in non-peak load periods. Compared to Classic RL, similar sized Fleet Computer missions consume 1.58X-2X less power. A swarm of multiple UAVs is also more energy efficient per-device than fewer UAVs; compared to Classic RL using a single UAV as in prior work, a swarm of four fleet-computer-controlled UAVs uses 3.7X-3.9X less energy depending on swarm size and goals.

Figure 5.5(g) shows how the Fleet Computer manages resources across extremes during a swarm mission. For a single UAV, one node of the Fleet Computer cluster is enough to handle all resource needs. As the swarm grows, more nodes must be provisioned. As nodes increase from 2 to 4, peak and trough allocation change. For instance, a 4-UAV swarm can operate at troughs using just 4 nodes, but requires all

6 to handle peak loads. The Fleet Computer's energy savings shown in Figure 5.5(f) 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 the Fleet Computer's usefulness-aware model retraining; this offers better use of finite edge resources by selecting which training tasks are likely to yield the highest benefit to an ongoing mission. Figure 5.5(h) 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 across it. 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 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 low utility to the system and give them to high utility models.

5.7 Related Work

Much recent work has dealt with tackling the concerns of real-world autonomous systems using strong theoretical foundations. Lin et al [122] explores a federated meta-learning approach to train models with small datasets in an edge setting. Singh

et. al [182] demonstrates a novel reward-training mechanism for reinforcement learning to eliminate the need for reward shaping. Kilinc and Montana [108] constructs a framework for sharing data among agents in execution contexts that are noisy and non-stationary using intrinsic reward and temporal locality. Other recent work [160, 218, 220] on networked agents provide considerable insight into the behaviors of real-world cooperative MARL systems with limited communication capabilities. Porter et. al [159] 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.

Much related work deals specifically with autonomous aerial systems. Boubin et. al [30] demonstrates that naive hardware and algorithm selection for fully autonomous aerial systems can have serious performance consequences. Cui et. al [51] implements MARL for allocating networking resources across a network of UAV base-stations. In agriculture Zhang et. al [223], Yang et. al [205], and FarmBeats [197] provide new techniques for automated and autonomous UAV crop scouting.

5.8 Conclusion

Swarms of autonomous systems powered by resources at the edge can provide insight and actuation that will revolutionize industries like agriculture, construction, transportation, and video analytics. We present the Fleet Computer, an end-to-end platform for building, deploying, and executing swarms. To use the Fleet Computer, developers implement *Map()* and *Eval()* functions and specify mission configurations. The Fleet Computer compiles and deploys swarms using a multi-agent reinforcement

learning framework. At runtime, the Fleet Computer manages data aggregation between swarm members, linking online learning outcomes to the efficient management of edge resources. Our evaluation showed that the Fleet Computer can produce effective and efficient swarms, suggesting this tool chain could make swarms accessible to everyday developers.

Chapter 6: Adaptive Deployment for Fully Autonomous UAV Swarms

Unmanned aerial vehicles play a critical role in many edge computing deployments and applications. UAV are prized for their maneuverability, low cost, and sensing capacity, facilitating many applications that would otherwise be prohibitively expensive or dangerous without them. UAV are cheaper than alternative aerial analysis methods, but still incur costs from expensive human piloting and workloads which necessitate high-resolution coverage of large areas. Recently, autonomous UAV swarms have emerged to increase the speed of deployments, decrease the cost and scope of human piloting, and improve the quality of autonomous decision making through data sharing. Autonomous UAV deployments, however, suffer from external factors. UAV are inherently power-constrained, with low onboard battery lives and limited ability to siphon power from the edge systems that support them. Certain environmental conditions, like inclement weather, wind, extreme heat, and low light also affect UAV power consumption, sensed data quality, and ultimately mission success. In this paper, we present an empirically based model for efficient autonomous swarm deployment. We built and deployed a real autonomous UAV swarm to map leaf defoliation in soybeans. Using this deployment, we determined environmental conditions which led to malfunctions, inefficient edge energy usage, and mispredictions. Using

these findings, we developed a deployment model for UAV swarms which decreases malfunctions and data irregularities by 4.9X and decreases edge energy consumption by 45%, while increasing deployment times by only 4%.

6.1 Introduction

Throughout the past decade, edge computing has matured from an active area of research among academics to an industry. Currently the global edge computing market is valued at over $36 Billion, and is predicted to grow to over $87 Billion by 2026 [133]. The rise of the internet of things (IoT) and the need for privacy and near-sensor processing has spawned a number of interesting consumer edge computing products, including smart homes and buildings [68], medical devices [1], Unmanned Aerial Vehicles (UAV) and more [30]. UAV, in particular, have formed broad academic, industry, and hobby communities. UAV are fast (moving upwards of 40 miles per hour), highly maneuverable, easy to fly, and can be equipped with high resolution cameras. Their flight can also be automated, making UAV an important tool for any task which requires precise, high resolution sensing that is too dangerous, expensive, or time consuming for humans to perform.

This use-case is common to many edge and IoT-related application areas. UAV have found considerable use in areas like precision agriculture [205, 223], search and rescue [6], infrastructure inspection [43], and remote sensing in disaster areas, forest fires, and other areas too dangerous for humans to approach [93]. UAV allow experienced operators to sense broad areas quickly at high altitude, with the freedom to investigate target areas at high spatial resolution. UAV flyover images have been used for coarse-grained crop-health monitoring, forest fire monitoring, and battlefield

target recognition. High spatial-resolution images have allowed subject matter experts and machine learning models to detect specific phenomena like leaf defoliation in soybeans [224], cracks in bridges, or humans in disaster areas.

While UAV are quite useful, they can be difficult to use for large or complex deployments. First, UAV have small battery lives. Most UAV fly for less than 40 minutes on one battery charge. Long missions are often solved by groups of cooperating UAV (called swarms) to both cover areas faster and mitigate short mission times. Second, the intelligence of each UAV agent affects mission time and capability. UAV can be piloted by humans, operating as an extension of their pilot and her expertise, or flow by software. Human piloting is by far the most common piloting method, but comes with drawbacks. UAV pilots are often licensed and can command high hourly rates, can be scarce or difficult to schedule for long deployments, and can not control multiple UAV at once.

UAV flown by software [137] can replace expensive human pilots by either automating UAV flight for pre-defined missions, or autonomously controlling UAV. Automated UAV fly pre-defined routes where the flight-path and sensing locations are specified before takeoff. Autonomous UAV sense and respond to their environment [167], accomplishing complex missions using machine learning. Software piloting of UAV both decreases deployment cost from human labor and allows for increased intelligence and response to sensed data. Furthermore, edge machines can be provisioned to fly multiple UAV at once as a swarms. Swarms of UAV cooperate to solve high-level goals as a team. Swarms of autonomous UAV can learn from one-another to influence eachothers actions, specialize, and update learning models online [22].

While autonomy can drastically simplify deployment for UAV applications, deployments must still contend with environmental issues. Any outdoor edge or IoT deployment must contend with weatherproofing, but UAV applications are particularly vulnerable. Unlike embedded sensors and edge devices, UAV explore their environments, making them susceptible to failures due to rain, lightning, heat, and other environmental factors. Furthermore, limited UAV battery life and continual recharges can put energy pressure on already constrained edge systems. These issues are compounded again by the remote nature of many UAV deployments. UAV are often most useful in large areas with limited power and network capabilities like crop fields, remote infrastructure, or disaster areas. UAV malfunction, data loss, and weather-related misclassification can lead to incorrect results, extension of deployments, or complete mission failure if incorrectly mitigated.

A successful autonomous UAV deployment must contend with these environmental challenges. Prior work has dispatched UAV and duty-cycled edge hardware based on cloud-cover to conserve renewable power [197]. Other viable flight conditions, such as heat, low light, and moderate wind may affect the energy that UAVs consume in flight, the odds of UAV malfunction, and model classification results. In this paper, we explore a range of environmental effects that UAV deployments experience and provide an empirical model that reserves edge resource for environmental conditions conducive to UAV flight.

We designed a crop scouting UAV swarm that assesses leaf defoliation in soybeans, a globally important crop. We flew over 150 autonomous crop scouting missions in different environmental conditions to measure the effects that environmental conditions

have on UAV malfunctions, data quality, and energy consumption. Using this information, we develop and simulate a deployment model for autonomous UAV swarms which saves UAV batteries for conditions where malfunction is least likely, energy consumption is minimized, and data quality is assured. Our simulation results show that this model decreases malfunctions by 4.9X and decreases UAV battery consumption by 45% over the course of a deployment while only increasing total deployment lengths by 4% on average.

This paper is organized as follows. Section 2 covers background information on UAV deployment concerns and prior deployment models. Section 3 describes the design and implementation of our deployment. Section 4 details results from our deployment. Section 5 presents our empirical deployment model for autonomous UAV swarms.

6.2 Background

Recent UAV work has led to automated and autonomous deployments in a wide range of areas [44, 87, 173, 197]. Particularly in agriculture, UAV have been deployed to scout important crops [44, 205], diagnose disease and pest infections [2], and apply treatments [78]. Crop scouting and treatment application is a continual process, with best practice suggesting repeated scouting every 7-10 days over the course of a growing season [69]. While researchers (and, increasingly, companies [115]) work to build crop scouting models, techniques are often tested in simulation or through manned flights or short-term deployments.

As the ability to scout and treat crops, diagnose infrastructure faults, and map wildfires matures, the need for more long-term deployment automation arises. Today's

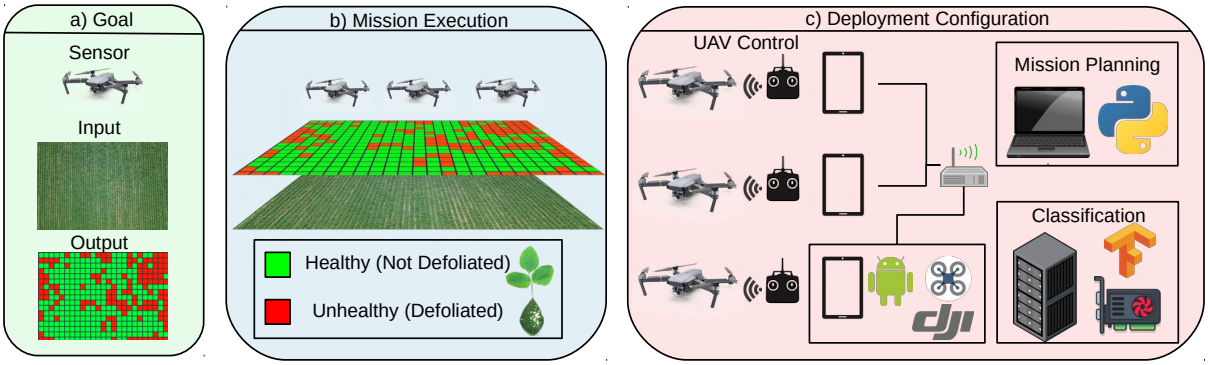

Fig. 6.1: Deployment Overview: UAVs scout a crop field to build defoliation maps over time using edge hardware and machine learning software for classification and map generation.

deployments are generally supervised by research teams regardless of the amount of automation or autonomy the UAV have in flight. This expert supervision removes the need to implement a complex deployment model, replacing it with expert human intuition and planning. FarmBeats [197], however, deployed a long-term UAV scouting solution which required some weather awareness, duty-cycling components of their IoT base-station to save solar-generated edge power. More recent theoretical work has explored the effects that weather can have on package delivery [191, 192] by using optimization to maximize customer satisfaction under different weather conditions. In this paper, we take inspiration from these approaches by deriving a model from long-term deployment experience.

6.3 Design

Computer system deployments in the wild can be complicated and error-prone, with risk factors increasing for UAV deployments where equipment traverses its environment. We deployed a long-term UAV swarm to study the risks that UAV and

equipment face in the wild. In this section, we describe this deployment and it's implementation.

UAV Deployment: Our UAV deployment, depicted in figure 6.1 uses 3 UAV to regularly scout a crop field for leaf defoliation. The goal of aerial crop scouting is to turn sensed field images into useful reports for farmers. UAV fly over crop fields periodically and capture images which are then analyzed and compiled into field reports which farmers can use to diagnose and treat pests, disease, environmental stress, and other crop health issues. Our deployment seeks a specific crop health condition: leaf defoliation. Leaf defoliation denotes loss of leaf area which occurs naturally as plants mature, but can be caused prematurely by pests, resulting in decreased yield [224]. Pre-mature defoliation in soybeans caused by pests is a common problem experienced by farmers around the world.

To scout soybeans using UAV, we implemented a convolutional neural network (CNN) model, called DefoNet [224], to predict the leaf defoliation conditions quickly from aerial images. DefoNet is a binary CNN model that classifies soybean leaves into two classes: defoliated or healthy as shown in figure 6.1. DefoNet accepts as input 108x108 pixel soybean leaf images. The main structure contains eight convolutional layers divided into three sections (Three layers in the fist section, three layers in the section, and two in the last). Following each convolutional section are activation and pooling layers. Before the final fully connected layer, we add a dropout layer to avoid model overfitting. In our test cases, Defonet is able to achieve over 92% accuracy on classifying soybean leaf defoliation.

To efficiently and quickly build crop-scouting maps, we relied on prior work to sample crop fields using multi-agent reinforcement learning. We used WholeField-RL [223], a reinforcement learning based sampling technique for crop health modeling, and the Fleet Computer [22], an edge-conscious autonomous swarm deployment architecture to simplify our deployment. Whole-field RL allows our UAV swarm to build accurate crop maps while sampling a subset of the field, using neural networks to extrapolate ground truth samples across unsampled regions. The fleet computer provides us an efficient dispatch mechanism which automatically schedules UAV flights, conserves edge resources by duty-cycling edge hardware, and retrains reinforcement learning policies online to improve mapping performance.

Implementation: Using this design, we built and deployed a UAV swarm to track soybean defoliation on LaRue farms in Central Ohio. Our swarm was deployed for 3 weeks from August 27th to September 16th 2021, running over 150 missions over that time. Our swarm consisted of 3 DJI Mavic Pro UAV with 6 interchangeable UAV batteries. UAV were controlled by SoftwarePilot [25] running on 3 android Tablets as shown in Figure 6.1c. Each tablet was connected via USB to a Mavic RC Controller via 5GHz WiFi to a fleet computer cluster.

Our fleet computer cluster consisted of two Lenovo T470 Thinkpads and one Dell precision 7920 workstation. Each Lenovo had an Intel i7 CPU and ran Ubuntu 18.04. One Lenovo was used as the Fleet Computer head node, controlling all UAV communication and acting as the Fleet Computer 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 Defonet classification and as the primary node for reinforcement learning retraining.

Each swarm mission was managed by the fleet computer but was manually dispatched by one of two on-site researchers. Missions covered 0.4 hectares of soybeans per UAV, taking between 5 and 20 minutes depending on coverage. To determine the effects of environmental conditions on UAV swarms, we deliberately executed missions in varying degrees of heat, wind, humidity, lighting, and cloud cover. We refrained, however, from executing missions in hazardous conditions with winds higher than 15mph, rain, or storms. We sought specifically to find the effects that seemingly reasonable flight conditions could have on UAV swarm performance.

6.4 Deployment Results

Throughout our deployment, we collected data on UAV and edge battery drain, system malfunctions and causes, machine learning mispredictions, and flight times for certain weather conditions. Our analysis shows that 3 key conditions (wind, temperature, and lighting) have serious effects on mission performance. Shown in Figure 6.2, wind speeds greatly affect flight times. We found that average single-UAV mission times degraded as wind speed increased. Our data shows that UAV in calm conditions (wind less than 5mph) had 19% longer battery lives than UAV flown in conditions where wind was on average faster than 10mph. Wind affects the power required for our UAV to fly between GPS waypoints. Wind can be helpful if it blows in the direction the UAV is traveling [197], but more often UAV must fight the wind to traverse the field and stabilize while capturing images and waiting for instructions, leading to increased battery drain.

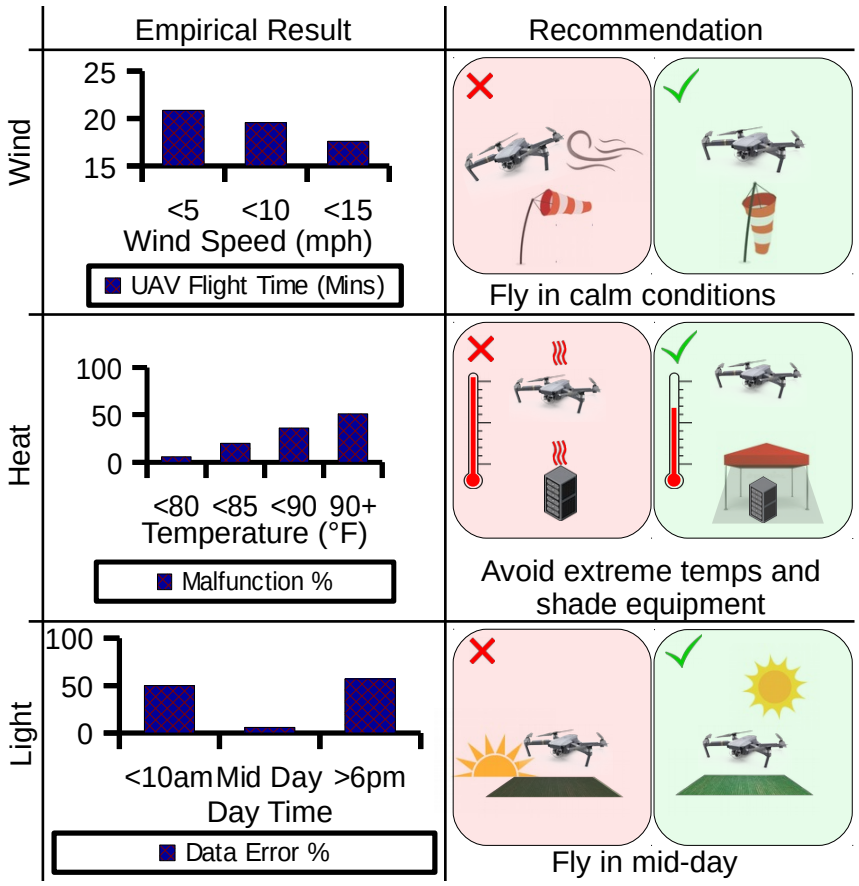

Fig. 6.2: high winds, extreme heat, and low light all contribute to malfunctions. We recommend flying in calm conditions, avoiding extreme temperatures, and flying when the sun is high for best results.

Heat also has negative effects on equipment. Throughout our deployment, we ran missions in various temperatures between 60F and 95F. We found that 5% of missions run in temperatures below 80F experiencing a malfunction due to equipment failure, while 51% of missions run at temperatures over 90F experienced malfunctions. Equipment malfunctions included communication errors between UAV and remotes, network errors, equipment overheating, and heat-based UAV battery malfunctions.

Not all errors were caused directly by overheating, but many were compounded by high temperatures as suggested in Figure 6.2.

Lighting was another major contributor to mission errors. Lighting, in this case sunlight, affects the quality of images that UAV capture. While all missions were flown within United States FAA regulated flight periods (30 minutes before sunrise or after sunset), low light and long shadows from a low solar angle contributed to increased mispredictions from DefoNet. We found that 50% of missions flown between sunrise and 10:00am and 57% of missions flown between 6:00pm and sunset contained mispredictions, while only 6% of missions flown between 10:00am and 6:00pm contained mispredictions. Mispredictions were generally false-negatives (predicting defoliated crop regions as healthy) due to DefoNet's inability to discern holes in leaves obscured by shadow.

Using these identified failure points for UAV missions, we provide recommendations for UAV deployments to avoid failures and unnecessary energy consumption. First, we recommend flying in conditions where sustained winds do not exceed 10 mph. While UAV can fly safely in winds higher than 10mph, we recommend conserving UAV battery for periods where weather is calm to maximize mission lengths, especially for deployments where power is scarce, harvested from compute resources, or generated by renewable sources. Second, we suggest avoiding flights during extreme temperatures, and always providing ample shade for equipment. High temperatures (over 90F) greatly increased incidence of equipment failure from UAV, edge, and networking hardware. For UAV, failures were limited mainly to battery malfunctions from short-term exposure to sun and high temperatures while flying which can be mitigated by conserving UAV batteries for cooler periods of the day. Furthermore,

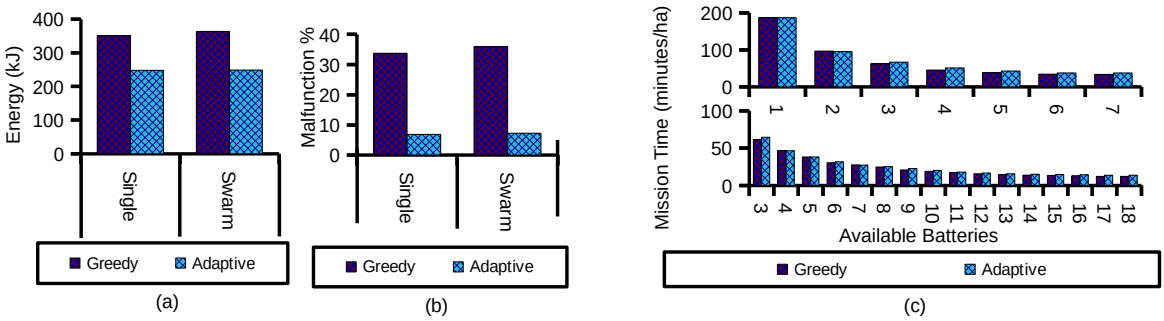

Fig. 6.3: Simulation Results: a) greedy dispatching uses more energy that adaptive dispatching, b) adaptive dispatching encounters less malfunctions, c) adaptive dispatching saves energy without significantly impacting mission time.

edge equipment malfunctions were often due to overheating from direct sun exposure. We suggest shading equipment from direct sunlight and potentially moving throughout the day as shade shifts with solar angle. Lastly, lighting effects on mispredictions can be mitigated by flying UAV when the sun is high, especially when areas are obscured in shadow at dawn or dusk.

6.5 Adaptive Deployment Model

UAV deployments are, at heart, a resource allocation problem. UAV and edge devices require power which can be drawn from electric grids, stored in batteries, or supplied by renewable sources. Conventional UAV rely entirely on batteries which must be recharged between short missions. Flights generally last between 15 and 30 minutes, while battery recharge periods can extend to over 90 minutes, incensing the importance both parallel execution of UAV flights in the form of swarms and over-provisioning of batteries. Swarm deployment workflows are cyclical, with periods of

UAV flight, battery interchange, and downtime where all batteries are depleted or recharging.

Using our deployment results, we designed a simple adaptive deployment model to dispatch UAV flights based on environmental factors. Our goal is to implicitly leverage the flight and charging cycle to charge batteries in unfavorable conditions to maximize flight times in favorable condition. Specifically, our goals are to minimize the following three quantities: 1) total mission time for aerial coverage of a target region, 2) total UAV energy consumption by avoiding re-scouting due to malfunctions, and 3) re-scouting of areas deemed mispredictions.

$$T =< T_t, T_{ds}, T_{de}, T_w >=< 85, 10{:}00, 18{:}00, 10 > \tag{6.1}$$

Model Definition: Our model explicitly schedules UAV flights for periods where conditions are favorable, and keeps UAV grounded while conditions are deemed unfavorable regardless of battery availability. Favorable conditions are determined via user-provided threshold vector T set based on deployment type, risks, and empirical experience. T, shown as an example in Equation 1, holds thresholds for the three unfavorable environmental conditions we determined from our experiments: temperature (T_t), appropriate start and end flight times (T_{ds} and T_{de}), and wind speed (T_w).

$$Dispatch(T) = \begin{cases} True & t \leq T_t, T_{ds} \leq dt \leq T_{de}, w \leq T_w \\ False & \end{cases} \tag{6.2}$$

Equation 2 shows the dispatch equation which determines whether waiting UAV with charged batteries should begin their mission or wait until conditions improve. This simple method charges and conserves batteries through unfavorable conditions,

assuring longer flights and less malfunctions. It may, however, lead to underutilization of UAV resources and greatly increased total mission times if unfavorable conditions persist for too long. To test the effectiveness of our model under a variety of conditions, we tested it in a simulated version of our deployment.

Simulation and Results: Our simulator was created using SoftwarePilot [25], the same UAV control platform used to build our original deployment. We simulated UAV flight over a 50-hectare field similar in size to our deployment field. The field was cut into 10,000 individual management zones for UAV sampling. Our simulated UAV flight characteristics were based on data from our deployment, maintaining similar mission times, battery discharge rates, and sampling rates.

Environmental characteristics and their effects were simulated using prior work and empirical information. Sunrise and Sunset were set at 7:00 am and 8:03 pm respectively, the corresponding sunrise and sunset for September 1st 2021 in Circleville, Ohio when and where our deployment was performed. Temperature was modeled using a sinusoidal curve with each given day being given a random temperature within two standard deviations of Circleville Ohio seasonal weather data obtained from the United States National Weather Service [176]. Temperatures experienced in flight by simulated UAV were between 60F and 93F. Wind was modeled by selecting a random wind value at the beginning of every simulated day and randomly increasing or decreasing it by up to 3mph (between 0 and 15mph) twice per simulated hour. Each simulated configuration was executed 100 times until the field was completely mapped without mispredictions. We simulated configurations with both single-UAV flights and swarms of 3 UAV as performed in our deployment, and with between 1 and 18 interchangeable batteries shared between UAV. For each configuration, our

deployment model was compared against a naive greedy model which dispatches UAV missions whenever sufficient batteries are available.

Figure 6.3 shows results from our simulations. Figure 6.3 (a) shows total energy expenditure for both swarms and single UAV using both greedy and adaptive deployment models. Total energy expenditure is 41.4% less for single UAVs and 45.8% less for swarms when dispatched adaptively. This decrease is due to multiple factors. First, These UAV do not operate in windy conditions, which degrade batteries 19% faster than calm conditions, but this accounts for only part of the decrease. The main decrease comes from repeated scouting of areas that were mispredicted or where data was lost or missions were cut short due to malfunction. Flying back to these potentially remote areas of the field for partial missions cuts other missions short and overall wastes energy compared to flying missions at opportune times.

Figure 6.3 (b) dives deeper into the malfunctions that both deployment models experience. Both single UAV and swarms experience about 4.9X less malfunctions when dispatched using our adaptive model compared to the greedy method. Malfunctions in this case, include any region that has to be re-sampled due to a hardware malfunction or data misprediction. For greedy dispatching 33-35% of all zones must be sampled more than once due to malfunction as opposed to 6% with adaptive dispatching. This decrease in resample is the primary driver for energy savings in Figure 6.3 (a).

Figure 6.3 (c) shows total mapping times for single UAV and swarm mapping using both greedy and adaptive dispatching with varying numbers of available batteries. One concern with adaptive dispatching is that UAV idle while they could be sampling the field which should lead to commensurately increased mapping times. Contrary

to this supposition, figure 6.3 (c) shows that adaptive dispatching does not significantly impact overall deployment times. Across all simulations, deployment times were increased by only 4% when using our adaptive strategy, with the largest increases (11-12%) shown when sufficient batteries are available to eliminate UAV wait-time for battery recharging (7 batteries for single UAV, 18 for swarm). Adaptive dispatching performs best when batteries are scarce and UAV wait-times are long, but maintains effectiveness even when batteries are plenty due to aforementioned decreases in malfunction-based remapping.

6.6 Conclusion

UAV deployments are complicated, requiring environmental considerations beyond those of normal edge and IoT deployments. While extreme weather will clearly impact UAV flight, some viable flight conditions like excessive heat, moderate winds, and low lighting can cause malfunctions and waste UAV energy. In this paper, we use data from over 150 missions of a long-term autonomous crop scouting UAV swarm to inform UAV deployment scheduling. In simulation, our empirical model decreases machine learning mispredictions by 4.9X, decreases overall swarm energy consumption by 45%, and increases total deployment times by only 4%.

Chapter 7: Conclusion

In this chapter, I restate the problems previously posed and their solutions presented throughout this dissertation. I also discuss future work opportunities in FAAS design, implementation, and applications. In the introduction, I posed five key problems for FAAS research to overcome:

§1. Creation of new general and domain-specific machine learning algorithms and careful utilization of others

§2. Selection of hardware at all levels in the FAAS hierarchy

§3. Power and environmental awareness informing selection and switching of autonomy policies, hardware devices, machine learning techniques and deployment characteristics.

§4. Online learning capabilities resilient to limited cloud access, network interruption, and power scarcity.

§5. Thorough applications which demonstrate the technological value of FAAS, drive adoption, and determine future research challenges.

Chapters 2-5 all contributed to the creation and careful usage of machine learning algorithms for FAAS development (§1). Chapter 3 introduces a novel reinforcement

learning and feature extraction technique for FAAS pathfinding in agricultural environments. Chapter 2 improves on this algorithm, expanding it to incorporate any domain and FAAS where sensed data can be analyzed in real-time. Chapter 2 also improves the mechanism further by using A* search on the reinforcement learning data set to improve results. Chapter 2 further explores the efficacy of different object detector models in autonomous photography and search and rescue, and how model-aware edge hardware selection and duty-cycling can affect their performance. Chapter 4 introduces a novel semantic segmentation approach to rice lodging detection using EDANet that is capable of executing using FAAS provisioned with edge GPUs. Chapter 4 also introduces a novel FAAS pathfinding mechanism that uses a high-cost low-altitude search of problem areas detected using a low-cost high altitude exhaustive search. This approach takes 36% less time to accomplish compared to a low-altitude search while being 99.25% as accuracy. Chapter 5 generalizes prior pathfinding approaches using MARL theory. By defining autonomy in terms of a Markov Game, a set of autonomy goals, and initial input data, the Fleet Computer can automatically design reinforcement learning models for swarms of autonomous agents. I also present an algorithm for online learning that can allow FAAS members to inform each other's pathfinding. I used hyper-parameter optimization to assure that pathfinding is weighted towards useful information, and that useful models are retrained quicker than less useful models.

All chapters discuss hardware selection, but chapters 2 and 5 discuss it most prominently. Chapter 2 outlines specific hardware selection mechanisms for FAAS, primarily at the edge. Many edge hardware configurations are compared with different software configurations, models, and autonomy policies to determine best practices

in selecting compatible edge hardware and machine learning algorithms. Chapter 2 also discusses systems-level optimizations like hardware duty-cycling and adaptive model switching, and compares edge configurations to potential onboard and cloud offloading solutions. Chapter 5 discusses the implications that multiple layers of edge hardware resources can have on autonomous system model retraining.

Chapters 2, 4, 5, and 6 all discuss power and environmental awareness for FAAS. Chapter 2 poses FAAS power problems in terms of mission throughput, the amount of FAAS missions that can be completed in one UAV battery discharge. This is an important metric for determining the effects that hardware architectures, machine learning inference times, and network latency and throughput have on overall FAAS performance. Chapter 2 poses findings for all configurations in terms of mission throughput. All models in Chapter 2 also incorporate network throughput as part of their mission throughput calculations. Chapter 4 also focuses on the need to maximize throughput for FAAS. The adaptive autonomous exhaustive search algorithm presented in chapter 4 entirely focuses in maximizing throughput to limit edge and UAV power consumption. Much of the proposed work in chapter 5 is focused on maximizing throughput over time across an entire swarm and accounting for underprovisioning at the edge to conserve power. The purpose of the Bayesian optimization scheduling algorithm is to assure that the models most likely to maximize throughput are updated ahead of less impactful models given that potentially few models will be updated before the start of the next FAAS mission, where new models can be used. Chapter 6 takes into account environmental characteristics in an effort to not only conserver prescious UAV power and minimize mission times, but also to minimize mission and equipment failure.

Chapter 5 primarily deals with online learning and its resilience at the edge. My Fleet Computer design assures that, even if only local edge compute node is accessible, that the most effective models will be retrained first. My design can utilize cloud and distant edge resources, but is tolerant to network connectivity issues and limited or unavailable cloud resources, and attempts to maximize throughput given the resources at its disposal to shorten missions and save edge and UAV power in the field.

Finally, all chapters deal with applications. FAAS applications drive my research. Throughout this thesis I have implemented FAAS to solve problems in search and rescue, autonomous photography, and a number of precision agriculture tasks. These systems were used to build models (chapter 2-3), test techniques (3-6), and act as real-world deployment case-studies (chapters 5-6). I have also built and distributed an open source FAAS software package SoftwarePilot, which was used throughout this thesis to implement FAAS.

7.1 Future Work

Many future work opportunities exist for FAAS research. Much work is currently being done on onboard and embedded computing for FAAS [20, 38, 77]. My work assumes a hard link between UAV sensed data and processing at the edge. Depending on the application domain, some or all machine learning may efficiently occur on the UAV. New hardware accelerators for machine learning such as FPGAs [193], neuromorphic hardware [123, 172], and embedded systems equipped with GPUs [141] may inform future FAAS applications.

Considerable work can also be done on how emerging network technologies, like 5G [157] affect FAAS deployments. With sufficient bandwidth, direct data transfer

from UAV to cloud may be possible. Given that FAAS operate in remote environments, increased cellular data coverage and the growing prevalence of 5G may improve FAAS efficiency and expand their potential application domains. Domain specific improvements to FAAS also require future work. Improvements in machine learning and architecture selection for search and rescue, crop scouting, infrastructure inspection, autonomous photography, and other domains FAAS may service could lead to increased industry adoption.

One key challenge in artificial intelligence that broadly affects FAAS is the building of accurate general models using limited data. FAAS operate on tasks which often do not have large datasets. Almost all of the work presented in this thesis relied on hand collected and curated datasets produced by myself or my colleagues. Data collection is difficult, time consuming, and presents a significant barrier to adoption. The construction of accurate, general, and fast models with few-shot learning could prove to be a breakthrough in the wide-spread adoption and and industrialization of FAAS.

Bibliography

[1] Alaa Awad Abdellatif, Amr Mohamed, Carla Fabiana Chiasserini, Mounira Tlili, and Aiman Erbad. Edge computing for smart health: Context-aware approaches, opportunities, and challenges. *IEEE Network*, 33(3):196–203, 2019.

[2] Jaafar Abdulridha, Yiannis Ampatzidis, Sri Charan Kakarla, and Pamela Roberts. Detection of target spot and bacterial spot diseases in tomato using uav-based and benchtop-based hyperspectral imaging techniques. *Precision Agriculture*, 21(5):955–978, 2020.

[3] E. Ackerman. Skydio announces sdk to make world's cleverest drone even cleverer. https://spectrum.ieee.org/.

[4] Abdulla Al-Kaff, Francisco Miguel Moreno, Luis Javier San José, Fernando García, David Martín, Arturo de la Escalera, Alberto Nieva, and José Luis Meana Garcéa. Vbii-uav: Vision-based infrastructure inspection-uav. In *World Conference on Information Systems and Technologies*, pages 221–231. Springer, 2017.

[5] Muhammad Raisul Alam, Mamun Bin Ibne Reaz, and Mohd Alauddin Mohd Ali. A review of smart homes—past, present, and future. *IEEE transactions on systems, man, and cybernetics, part C (applications and reviews)*, 42(6):1190–1203, 2012.

[6] Ebtehal Turki Alotaibi, Shahad Saleh Alqefari, and Anis Koubaa. Lsar: Multi-uav collaboration for search and rescue missions. *IEEE Access*, 7:55817–55832, 2019.

[7] Amazon. Prime air. `https://www.amazon.com/Amazon-Prime-Air/b?ie=UTF8&node=8037720011`, 2020.

[8] Dragomir Anguelov, Carole Dulong, Daniel Filip, Christian Frueh, Stéphane Lafon, Richard Lyon, Abhijit Ogale, Luc Vincent, and Josh Weaver. Google street view: Capturing the world at street level. *Computer*, 43(6):32–38, 2010.

[9] Anish Arora, Prabal Dutta, Sandip Bapat, Vinod Kulathumani, Hongwei Zhang, Vinayak Naik, Vineet Mittal, Hui Cao, Murat Demirbas, Mohamed Gouda, et al. A line in the sand: A wireless sensor network for target detection, classification, and tracking. *Computer Networks*, 46(5):605–634, 2004.

[10] Anish Arora, Rajiv Ramnath, Emre Ertin, Prasun Sinha, Sandip Bapat, Vinayak Naik, Vinod Kulathumani, Hongwei Zhang, Hui Cao, Mukundan Sridharan, et al. Exscal: Elements of an extreme scale wireless sensor network. In *11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'05)*, pages 102–108. IEEE, 2005.

[11] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Computer networks*, 54(15):2787–2805, 2010.

[12] Anthony Baietto, Jayson Boubin, Patrick Farr, and Trevor J Bihl. Lean neural networks for real-time embedded spectral notching waveform design. In *IEEE International Symposium on Industrial Electronics*, 2022.

[13] Anthony Baietto, Jayson Boubin, Patrick Farr, Trevor J Bihl, Aaron M Jones, and Christopher Stewart. Lean neural networks for autonomous radar waveform design. *Sensors*, 22(4):1317, 2022.

[14] Samik Banerjee, Lucas Magee, Dingkang Wang, Xu Li, Bing-Xing Huo, Jaikishan Jayakumar, Katherine Matho, Meng-Kuan Lin, Keerthi Ram, Mohanasankar Sivaprakasam, et al. Semantic segmentation of microscopic neuroanatomical data by combining topological priors with encoder–decoder deep networks. *Nature machine intelligence*, 2(10):585–594, 2020.

[15] Antonio Barrientos, Julian Colorado, Jaime del Cerro, Alexander Martinez, Claudio Rossi, David Sanz, and Joao 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.

[16] Brendan Barry, Cormac Brick, Fergal Connor, David Donohoe, David Moloney, Richard Richmond, Martin O'Riordan, and Vasile Toma. Always-on vision processing unit for mobile applications. *IEEE Micro*, 35(2), 2015.

[17] Ejder Bastug, Mehdi Bennis, Muriel Médard, and Mérouane Debbah. Toward interconnected virtual reality: Opportunities, challenges, and enablers. *IEEE Communications Magazine*, 55(6):110–117, 2017.

[18] S Basu and RP Woodard. Testing an ansatz for the leading secular loop corrections from quantum gravity during inflation. *Classical and Quantum Gravity*, 33(20):205007, 2016.

[19] Trevor J Bihl, Kenneth W Bauer, and Michael A Temple. Feature selection for rf fingerprinting with multiple discriminant analysis and using zigbee device emissions. *IEEE Transactions on Information Forensics and Security*, 11(8):1862–1874, 2016.

[20] Behzad Boroujerdian, Hasan Genc, Srivatsan Krishnan, Wenzhi Cui, Aleksandra Faust, and Vijay Reddi. Mavbench: Micro aerial vehicle benchmarking. In *MICRO*, 2018.

[21] Jayson Boubin, Avishek Banerjee, Jihoon Yun, Haiyang Qi, Yuting Fang, Steve Chang, Kannan Srinivasan, Rajiv Ramnath, and Anish Arora. Prowess: An open testbed for programmable wireless edge systems. 2022.

[22] Jayson Boubin, Codi Burley, Peida Han, Bowen Li, Barry Porter, and Christopher Stewart. Programming and deployment of autonomous swarms using multi-agent reinforcement learning. *arXiv preprint arXiv:2105.10605*, 2021.

[23] Jayson Boubin, John Chumley, Christopher Stewart, and Sami Khanal. Autonomic computing challenges in fully autonomous precision agriculture. In *2019 IEEE International Conference on Autonomic Computing (ICAC)*. IEEE, 2019.

[24] Jayson Boubin, Aaron M Jones, and Trevor Bihl. Neurowav: Toward real-time waveform design for vanets using neural networks. In *2019 IEEE Vehicular Networking Conference (VNC)*, pages 1–4. IEEE, 2019.

[25] Jayson Boubin and Christopher Stewart. Softwarepilot: Fully autonomous aerial systems made easier. In *2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion (ACSOS-C)*, pages 250–251. IEEE, 2020.

[26] Jayson Boubin, Christopher Stewart, Shiqi Zhang, Naveen T.R. Babu, and Zichen Zhang. Softwarepilot. `http://github.com/boubinjg/softwarepilot`, 2019.

[27] Jayson Boubin, Shiqi Zhang, Venkata Mandadapu, and Christopher Stewart. Characterizing computational workloads in uav applications. In *2018 IEEE/ACM Third International Conference on Internet-of-Things Design and Implementation (IoTDI)*, pages 275–276. IEEE, 2018.

[28] Jayson Boubin, Zichen Zhang, John Chumley, and Christopher Stewart. Data-parallel versus task-parallel swarms for small unmanned aerial systems. In *2022 IEEE/ACM Seventh International Conference on Internet-of-Things Design and Implementation (IoTDI)*, pages 28–29. IEEE, 2022.

[29] Jayson Boubin, Zichen Zhang, Shiqi Zhang, and Christopher Stewart. Softwarepilot: An open source middleware for fully autonomous aerial systems. 2019.

[30] Jayson G Boubin, Naveen TR Babu, Christopher Stewart, John Chumley, and Shiqi 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.

[31] Jayson G Boubin, Christina F Rusnock, and Jason M Bindewald. Quantifying compliance and reliance trust behaviors to influence trust in human-automation teams. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, volume 61, pages 750–754. SAGE Publications Sage CA: Los Angeles, CA, 2017.

[32] Matthew Boubin and Sudhir Shrestha. Microcontroller implementation of support vector machine for detecting blood glucose levels using breath volatile organic compounds. *Sensors*, 19(10):2283, 2019.

[33] Amanda Bouman, Muhammad Fadhil Ginting, Nikhilesh Alatur, Matteo Palieri, David D Fan, Thomas Touma, Torkom Pailevanian, Sung-Kyun Kim, Kyohei Otsu, Joel Burdick, et al. Autonomous spot: Long-range autonomous exploration of extreme environments with legged locomotion. *arXiv preprint arXiv:2010.09259*, 2020.

[34] T. Brechman. 3 examples showing why crop scouting pays, even in an off year. `www.indianaprairiefarmer.com`, 2016.

[35] Molly E Brown and Chris C Funk. Food security under climate change. 2008.

[36] Anup W Burange and Harshal D Misalkar. Review of internet of things in development of smart cities with data management & privacy. In *2015 International Conference on Advances in Computer Engineering and Applications*, pages 189–195. IEEE, 2015.

[37] Alwyn Burger, Christopher Cichiwskyj, and Gregor Schiele. Elastic nodes for the internet of things: A middleware-based approach. In *ICAC*, 2017.

[38] Alwyn Burger, Patrick Urban, Jayson Boubin, and Gregor Schiele. An architecture for solving the eigenvalue problem on embedded fpgas. In *International Conference on Architecture of Computing Systems*, pages 32–43. Springer, 2020.

[39] Lucian Busoniu, Robert Babuska, and Bart 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.

[40] Emiliano Casalicchio. Container orchestration: A survey. In *Systems Modeling: Methodologies and Tools*, pages 221–235. Springer, 2019.

[41] Kyle Cesare, Ryan Skeele, Soo-Hyun Yoo, Yawei Zhang, and Geoffrey Hollinger. Multi-uav exploration with limited communication and battery. In *2015 IEEE international conference on robotics and automation (ICRA)*, pages 2230–2235. IEEE, 2015.

[42] Yu-han Chang, Tracey Ho, and Leslie Kaelbling. All learning is local: Multi-agent learning in global reward games. *Advances in Neural Information Processing Systems*, 16:807–814, 2003.

[43] Siyuan Chen, Debra F Laefer, Eleni Mangina, SM Iman Zolanvari, and Jonathan Byrne. Uav bridge inspection through evaluated 3d reconstructions. *Journal of Bridge Engineering*, 24(4):05019001, 2019.

[44] Steven W Chen, Shreyas S Shivakumar, Sandeep Dcunha, Jnaneshwar Das, Edidiong Okon, Chao Qu, Camillo J Taylor, and Vijay Kumar. Counting apples and oranges with deep learning: A data-driven approach. *IEEE Robotics and Automation Letters*, 2(2):781–788, 2017.

[45] Tianxing Chu, Michael J Starek, Michael J Brewer, Tiisetso Masiane, and Seth C Murray. Uas imaging for automated crop lodging detection: a case study over an experimental maize field. In *Autonomous air and ground sensing systems for agricultural optimization and phenotyping II*, volume 10218, page 102180E. International Society for Optics and Photonics, 2017.

[46] Tianxing Chu, Michael J Starek, Michael J Brewer, Seth C Murray, and Luke S Pruter. Assessing lodging severity over an experimental maize (zea mays l.) field using uas images. *Remote Sensing*, 9(9):923, 2017.

[47] Zheng Chu and Jiong Yu. An end-to-end model for rice yield prediction using deep learning fusion. *Computers and Electronics in Agriculture*, 174:105471, 2020.

[48] CNN. 7 tips for taking better selfies. `https://www.cnn.com/2013/12/11/tech/mobile/selfie-photo-tips/`, 2013.

[49] The Business Research Company. Commercial drones market global report 2020-30: Covid19 growth and change, 2020.

[50] The Business Research Company. Global commercial drones market 2021, 2021.

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

[52] Mark De Deuge, Alastair Quadros, Calvin Hung, and Bertrand Douillard. Unsupervised feature learning for classification of outdoor 3d scans. In *Australasian Conference on Robitics and Automation*, volume 2, page 1, 2013.

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

[54] Tamal K Dey, Marian Mrozek, and Ryan Slechta. Persistence of conley–morse graphs in combinatorial dynamical systems. *SIAM Journal on Applied Dynamical Systems*, 21(2):817–839, 2022.

[55] Carmelo Di Franco and Giorgio C Buttazzo. Energy-aware coverage path planning of uavs. In *ICARSC*, pages 111–117, 2015.

[56] DJI. Prerequisites-dji mobile sdk documentation. `https://developer.dji.com/`, 2018.

[57] DJI. Spark specs. `https://www.dji.com/spark/info`, 2018.

[58] DJI. Dji developer sdks. https://developer.dji.com/, 2020.

[59] Thinh Doan, Siva Maguluri, and Justin 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.

[60] Patrick Doherty and Piotr Rudol. A uav search and rescue scenario with human body detection and geolocalization. In *Australasian Joint Conference on Artificial Intelligence*, pages 1–13. Springer, 2007.

[61] dronekit.io. Developer tools for drones. http://dronekit.io/, 2015.

[62] Lester E Ehler. Integrated pest management (ipm): Definition, historical development and implementation, and the other ipm. 62:787–9, 09 2006.

[63] Andy Rosales Elias, Nevena Golubovic, Chandra Krintz, and Rich Wolski. Where's the bear?-automating wildlife image processing using iot and edge cloud systems. In *2017 IEEE/ACM Second International Conference on Internet-of-Things Design and Implementation (IoTDI)*, 2017.

[64] Emre Ertin, Anish Arora, Rajiv Ramnath, Vinayak Naik, Sandip Bapat, Vinod Kulathumani, Mukundan Sridharan, Hongwei Zhang, Hui Cao, and Mikhail Nesterenko. Kansei: A testbed for sensing at scale. In *Proceedings of the 5th international conference on Information processing in sensor networks*, pages 399–406, 2006.

[65] Matthias Faessler, Flavio Fontana, Christian Forster, Elias Mueggler, Matia Pizzoli, and Davide Scaramuzza. Autonomous, vision-based flight and live dense 3d mapping with a quadrotor micro aerial vehicle. *Journal of Field Robotics*, 33(4), 2016.

[66] Farbod Fahimi. Autonomous robots. *Modeling, Path Planning and Control*, 2009.

[67] Patrick Farr, Aaron M Jones, Trevor Bihl, Jayson Boubin, and Ashley DeMange. Waveform design implemented on neuromorphic hardware. In *2020 IEEE International Radar Conference (RADAR)*, pages 934–939. IEEE, 2020.

[68] Francisco-Javier Ferrández-Pastor, Higinio Mora, Antonio Jimeno-Morenilla, and Bruno Volckaert. Deployment of iot edge and fog computing technologies to develop smart building services. *Sustainability*, 10(11):3832, 2018.

[69] Fred Fishel, Wayne Charles Bailey, Michael L Boyd, William Gary Johnson, Maureen H O'Day, Laura Sweets, and William John Wiebold. Introduction to crop scouting. *Extension publications (MU)*, 2009.

[70] Linda Foreman. Characteristics and Production Costs of U.S. Corn Farms, Including Organic, 2010, 2014.

[71] Peter I Frazier. A tutorial on bayesian optimization. *arXiv preprint arXiv:1807.02811*, 2018.

[72] Sally French. Want to make six figures? try being a drone pilot. https://www.marketwatch.com/, 2020.

[73] Boris Galkin, Jacek Kibilda, and Luiz A DaSilva. Uavs as mobile infrastructure: Addressing battery lifetime. *IEEE Communications Magazine*, 57(6):132–137, 2019.

[74] S. Garvey. Considering trying out a uav? `http://www.grainnews.ca`, 2015.

[75] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 32(11):1231–1237, 2013.

[76] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3354–3361. IEEE, 2012.

[77] H. Genc, Y. Zu, T. Chin, M. Halpern, and V. J. Reddi. Flying iot: Toward low-power vision in the sky. *IEEE Micro*, 37(6):40–51, November 2017.

[78] D Giles and R Billing. Deployment and performance of a uav for crop spraying. *Chemical engineering transactions*, 44:307–312, 2015.

[79] H. Charles J Godfray, John R. Beddington, Ian R. Crute, Lawrence Haddad, David Lawrence, James F. Muir, Jules Pretty, Sherman Robinson, Sandy M. Thomas, and Camilla Toulmin. Food security: The challenge of feeding 9 billion people. *Science*, 2010.

[80] D. Gomez-Candon, A. De Castro, and F. Lopez-Grandos. Assessing the accuracy of mosaics from unmanned aerial vehicle (uav) imagery for precision agriculture purposes in wheat. In *Remote Sensing*, 2014.

[81] G Grassi, K Jamieson, P Bahl, and G Pau. Parkmaster: An in-vehicle, edge-based video analytics service for detecting open parking spaces in urban environments. In *ACM Symposium on Edge Computing*, 2017.

[82] Jim Gray, Surajit Chaudhuri, Adam Bosworth, Andrew Layman, Don Reichart, Murali Venkatrao, Frank Pellow, and Hamid Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *Data mining and knowledge discovery*, 1997.

[83] Ivo Grondman, Lucian Busoniu, Gabriel AD Lopes, and Robert 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.

[84] Marek Grześ. Reward shaping in episodic reinforcement learning. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, pages 565–573, 2017.

[85] Najmul Hassan, Kok-Lim Alvin Yau, and Celimuge Wu. Edge computing in 5g: A review. *IEEE Access*, 7:127276–127289, 2019.

[86] Johann Hauswald, Michael A Laurenzano, Yunqi Zhang, Cheng Li, Austin Rovinski, Arjun Khurana, Ronald G Dreslinski, Trevor Mudge, Vinicius Petrucci, Lingjia Tang, et al. Sirius: An open end-to-end voice and vision personal assistant and its implications for future warehouse scale computers. In *ASPLOS*, 2015.

[87] Justin Hu, Ariana Bruno, Brian Ritchken, Brendon Jackson, Mateo Espinosa, Aditya Shah, and Christina Delimitrou. Hivemind: A scalable and serverless coordination control platform for uav swarms. *arXiv preprint arXiv:2002.01419*, 2020.

[88] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[89] CC Hung, G Ananthanarayanan, and et al. Videoedge: Processing camera streams using hierarchical clusters. In *ACM Symposium on Edge Computing*, 2018.

[90] Samvit Jain, Xun Zhang, Yuhao Zhou, Ganesh Ananthanarayanan, Junchen Jiang, Yuanchao Shu, Paramvir Bahl, and Joseph Gonzalez. Spatula: Efficient cross-camera video analytics on large camera networks. In *2020 IEEE/ACM Symposium on Edge Computing (SEC)*, pages 110–124. IEEE, 2020.

[91] Peter John-Baptiste, Graeme E Smith, Aaron M Jones, and Trevor Bihl. Rapid waveform design through machine learning. In *2019 IEEE 8th International Workshop on Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP)*, pages 659–663. IEEE, 2019.

[92] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with gpus. *CoRR*, abs/1702.08734, 2017.

[93] Kyle D Julian and Mykel J Kochenderfer. Distributed wildfire surveillance with autonomous aircraft using deep reinforcement learning. *Journal of Guidance, Control, and Dynamics*, 42(8):1768–1778, 2019.

[94] Andreas Kamilaris and Francesc X Prenafeta-Boldú. Deep learning in agriculture: A survey. *Computers and electronics in agriculture*, 147:70–90, 2018.

[95] Dongho Kang and Young-Jin Cha. Autonomous uavs for structural health monitoring using deep learning and an ultrasonic beacon system with geo-tagging. *Computer-Aided Civil and Infrastructure Engineering*, 33(10):885–902, 2018.

[96] Soummya Kar, José MF Moura, and H Vincent 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.

[97] Sokratis Kartakis, Weiren Yu, Reza Akhavan, and Julie A McCann. Adaptive edge analytics for distributed networked control of water systems. In *2016 IEEE First International Conference on Internet-of-Things Design and Implementation (IoTDI)*, pages 72–82. IEEE, 2016.

[98] Shinpei Kato, Eijiro Takeuchi, Yoshio Ishiguro, Yoshiki Ninomiya, Kazuya Takeda, and Tsuyoshi Hamada. An open approach to autonomous vehicles. *IEEE Micro*, 35(6), 2015.

[99] Shinpei Kato, Shota Tokunaga, Yuya Maruyama, Seiya Maeda, Manato Hirabayashi, Yuki Kitsukawa, Abraham Monrroy, Tomohito Ando, Yusuke Fujii, and Takuya Azumi. Autoware on board: enabling autonomous vehicles with embedded systems. In *International Conference on Cyber-Physical Systems*, 2018.

[100] Jaimie Kelley, Christopher Stewart, Nathaniel Morris, Devesh Tiwari, Yuxiong He, and Sameh Elnikety. Measuring and managing answer quality for online data-intensive services. In *2015 IEEE International Conference on Autonomic Computing*, pages 167–176. IEEE, 2015.

[101] Jaimie Kelley, Christopher Stewart, Nathaniel Morris, Devesh Tiwari, Yuxiong He, and Sameh Elnikety. Measuring and managing answer quality for online data-intensive services. In *ICAC*, 2015.

[102] Jaimie Kelley, Christopher Stewart, Nathaniel Morris, Devesh Tiwari, Yuxiong He, and Sameh Elnikety. Obtaining and managing answer quality for online data-intensive services. In *ACM Transactions on Modeling and Performance Evaluation of Computing Systems*, 2017.

[103] R. Kesten, M. Usman, J. Houston, T. Pandya, K. Nadhamuni, A. Ferreira, M. Yuan, B. Low, A. Jain, P. Ondruska, S. Omari, S. Shah, A. Kulkarni, A. Kazakova, C. Tao, L. Platinsky, W. Jiang, and V. Shet. Lyft level 5 perception dataset 2020. `https://level5.lyft.com/dataset/`, 2019.

[104] Sami Khanal, John Fulton, Nathan Douridas, Andrew Klopfenstein, and Scott Shearer. Integrating aerial images for in-season nitrogen management in a corn field. *Computers and Electronics in Agriculture*, 148:121–131, 2018.

[105] Sami Khanal, John Fulton, Nathan Douridas, Andrew Klopfenstein, and Scott Shearer. Integrating aerial images for in-season nitrogen management in a corn field. *computers and electronics in agriculture*, 148, 2018.

[106] Sami Khanal, John Fulton, Andrew Klopfenstein, Nathan Douridas, and Scott Shearer. Integration of high resolution remotely sensed data and machine learning techniques for spatial prediction of soil properties and corn yield. *Computers and electronics in agriculture*, 153:213–225, 2018.

[107] Sami Khanal, Kushal Kc, John P Fulton, Scott Shearer, and Erdal Ozkan. Remote sensing in agriculture—accomplishments, limitations, and opportunities. *Remote Sensing*, 12(22):3783, 2020.

[108] Ozsel Kilinc and Giovanni Montana. Multi-agent deep reinforcement learning with extremely noisy observations. *arXiv preprint arXiv:1812.00922*, 2018.

[109] Dong Ki Kim and Tsuhan Chen. Deep neural network for real-time autonomous indoor navigation. *arXiv preprint arXiv:1511.04668*, 2015.

[110] Jakub Konečnỳ, H Brendan McMahan, Daniel Ramage, and Peter Richtárik. Federated optimization: Distributed machine learning for on-device intelligence. *arXiv preprint arXiv:1610.02527*, 2016.

[111] Pouya Kousha, Bharath Ramesh, Kaushik Kandadi Suresh, Ching-Hsiang Chu, Arpan Jain, Nick Sarkauskas, Hari Subramoni, and Dhabaleswar K Panda. Designing a profiling and visualization tool for scalable and in-depth analysis of high-performance gpu clusters. In *2019 IEEE 26th International Conference on High Performance Computing, Data, and Analytics (HiPC)*, pages 93–102. IEEE, 2019.

[112] Pouya Kousha, Kamal Raj Sankarapandian Dayala Ganesh Ram, Mansa Kedia, Hari Subramoni, Arpan Jain, Aamir Shafi, Dhabaleswar Panda, Trey Dockendorf, Heechang Na, and Karen Tomko. Inam: Cross-stack profiling and analysis of communication in mpi-based applications. In *Practice and Experience in Advanced Research Computing*, pages 1–11. 2021.

[113] Pouya Kousha, Kamal Raj SD, Hari Subramoni, Dhabaleswar K Panda, Heechang Na, Trey Dockendorf, and Karen Tomko. Accelerated real-time network monitoring and profiling at scale using osu inam. In *Practice and Experience in Advanced Research Computing*, pages 215–223. 2020.

[114] Matthias Kovatsch, Martin Lanter, and Zach Shelby. Californium: Scalable cloud services for the internet of things with coap. In *International Conference on the Internet of Things*, 2014.

[115] Meg Kummerow. Fly the farm. `http://www.flythefarm.com.au/`, 2018.

[116] You-zhong LANG, Xiao-dong YANG, Mei-e WANG, and Qing-sen ZHU. Effects of lodging at different filling stages on rice yield and grain quality. *Rice Science*, 19(4):315–319, 2012.

[117] L. larson, W. Tarneberg, C. Klein, and E. Elmroth. Quality-elasticity: Improved resource utilization, throughput and response times via adjusting output quality to current operating conditions. In *IEEE ICAC*, 2019.

[118] Bowen Li, Nat Shineman, Jayson Boubin, and Christopher Stewart. Comparison of object detectors for fully autonomous aerial systems performance. In *Companion of the ACM/SPEC International Conference on Performance Engineering*, pages 165–166, 2021.

[119] C Li, R Wang, T Li, D Qian, and J Yuan. Managing green datacenters powered by hybrid renewable energy systems. In *ICAC*, 2014.

[120] Thomas Lillesand, Ralph W Kiefer, and Jonathan Chipman. *Remote sensing and image interpretation*. John Wiley & Sons, 2015.

[121] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[122] Sen Lin, Guang Yang, and Junshan 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.

[123] Shih-Chieh Lin, Yunqi Zhang, Chang-Hong Hsu, Matt Skach, Md E Haque, Lingjia Tang, and Jason Mars. The architectural implications of autonomous driving: Constraints and acceleration. In *ASPLOS*, 2018.

[124] Xi Victoria Lin, Richard Socher, and Caiming Xiong. Multi-hop knowledge graph reasoning with reward shaping. *arXiv preprint arXiv:1808.10568*, 2018.

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

[126] Tao Liu, Rui Li, Xiaochun Zhong, Min Jiang, Xiuliang Jin, Ping Zhou, Shengping Liu, Chengming Sun, and Wenshan Guo. Estimates of rice lodging using indices derived from uav visible and thermal infrared images. *Agricultural and Forest Meteorology*, 252:144–154, 2018.

[127] Shao-Yuan Lo, Hsueh-Ming Hang, Sheng-Wei Chan, and Jing-Jhih Lin. Efficient dense modules of asymmetric convolution for real-time semantic segmentation. In *Proceedings of the ACM Multimedia Asia*, pages 1–6. 2019.

[128] Lucas Magee and Yusu Wang. Graph skeletonization of high-dimensional point cloud data via topological method. *arXiv preprint arXiv:2109.07606*, 2021.

[129] Robert Mahony, Vijay Kumar, and Peter Corke. Multirotor aerial vehicles. *IEEE Robotics and Automation magazine*, 20(32), 2012.

[130] Hosein Mohammadi Makrani, Hossein Sayadi, Devang Motwani, Han Wang, Setareh Rafatirad, and Houman Homayoun. Energy-aware and machine learning-based resource provisioning of in-memory analytics on cloud. In *Proceedings of the ACM Symposium on Cloud Computing*, pages 517–517, 2018.

[131] Sara Mardanisamani, Farhad Maleki, Sara Hosseinzadeh Kassani, Sajith Rajapaksa, Hema Duddu, Menglu Wang, Steve Shirtliffe, Seungbum Ryu, Anique Josuttes, Ti Zhang, et al. Crop lodging prediction from uav-acquired images of wheat and canola using a dcnn augmented with handcrafted texture features. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 0–0, 2019.

[132] MarketDataForecast. Self-driving cars market. `https://www.marketdataforecast.com/market-reports/self-driving-cars-market`, 2020.

[133] Markets and Markets. Edge computing market with covid-19 impact analysis. `https://www.marketsandmarkets.com/Market-Reports/edge-computing-market-133384090.html`, 2021.

[134] Lockheed Martin. The future of autonomy isn't human-less. it's human more. `https://www.lockheedmartin.com/en-us/capabilities/autonomous-unmanned-systems.html`, 2018.

[135] Lockheed Martin. The future of autonomy isn't human-less. it's human more. `https://www.lockheedmartin.com/en-us/capabilities/autonomous-unmanned-systems.html`, 2018.

[136] Koppány Máthé and Lucian Buşoniu. Vision and control for uavs: A survey of general methods and of inexpensive platforms for infrastructure inspection. *Sensors*, 15(7):14887–14916, 2015.

[137] Lorenz Meier, Petri Tanskanen, Lionel Heng, Gim Hee Lee, Friedrich Fraundorfer, and Marc Pollefeys. Pixhawk: A micro aerial vehicle design for autonomous flight using onboard computer vision. *Autonomous Robots*, 33(1-2), 2012.

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

[139] George E Meyer and João Camargo Neto. Verification of color vegetation indices for automated crop imaging applications. *Computers and electronics in agriculture*, 63(2):282–293, 2008.

[140] Nilesh Mishra, Kameswari Chebrolu, Bhaskaran Raman, and Abhinav Pathak. Wake-on-wlan. In *Proceedings of the 15th international conference on World Wide Web*, pages 761–769, 2006.

[141] Sparsh Mittal. A survey on optimized implementation of deep learning models on the nvidia jetson platform. *Journal of Systems Architecture*, 97:428–442, 2019.

[142] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[143] JB Mockus and LJ Mockus. Bayesian approach to global optimization and application to multiobjective and constrained problems. *Journal of Optimization Theory and Applications*, 70(1):157–172, 1991.

[144] Nitinder Mohan, Lorenzo Corneo, Aleksandr Zavodovski, Suzan Bayhan, Walter Wong, and Jussi Kangasharju. Pruning edge research with latency shears. In *Proceedings of the 19th ACM Workshop on Hot Topics in Networks*, pages 182–189, 2020.

[145] Luis Moreira-Matias, Joao Gama, Michel Ferreira, Joao Mendes-Moreira, and Luis Damas. Predicting taxi–passenger demand using streaming data. *IEEE Transactions on Intelligent Transportation Systems*, 14(3):1393–1402, 2013.

[146] N Morris, C Stewart, L Chen, R Birke, and et al. Model-driven computational sprinting. In *ACM Eurosys*, 2018.

[147] Nathaniel Morris, Siva Meenakshi Renganathan, Christopher Stewart, Robert Birke, and Lydia Chen. Sprint ability: How well does your software exploit bursts in processing capacity? In *ICAC*, 2016.

[148] Nathaniel Morris, Christopher Stewart, Lydia Chen, Robert Birke, and Jaimie Kelley. Model-driven computational sprinting. In *Proceedings of the Thirteenth EuroSys Conference*, pages 1–13, 2018.

[149] Daniel Munoz, J Andrew Bagnell, Nicolas Vandapel, and Martial Hebert. Contextual classification with functional max-margin markov networks. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 975–982. IEEE, 2009.

[150] Krishna Giri Narra, Zhifeng Lin, Ganesh Ananthanarayanan, Salman Avestimehr, and Murali Annavaram. Collage inference: Tolerating stragglers in distributed neural network inference using coding, 2019.

[151] Andrew Nelson, Tri Setiyono, Arnel B Rala, Emma D Quicho, Jeny V Raviz, Prosperidad J Abonete, Aileen A Maunahan, Cornelia A Garcia, Hannah Zarah M Bhatti, Lorena S Villano, et al. Towards an operational sar-based rice monitoring system in asia: Examples from 13 demonstration sites across asia in the riice project. *Remote Sensing*, 6(11):10773–10812, 2014.

[152] R Todd Ogden, Carl E Miller, Kunio Takezawa, and Seishi Ninomiya. Functional regression in crop lodging assessment with digital images. *Journal of Agricultural, Biological, and Environmental Statistics*, 7(3):389, 2002.

[153] Ayako Okuno, Ko Hirano, Kenji Asano, Wakana Takase, Reiko Masuda, Yoichi Morinaka, Miyako Ueguchi-Tanaka, Hidemi Kitano, and Makoto Matsuoka. New approach to increasing rice lodging resistance and biomass yield through the use of high gibberellin producing varieties. *PLoS One*, 9(2):e86870, 2014.

[154] Taiichiro Ookawa, Tokunori Hobo, Masahiro Yano, Kazumasa Murata, Tsuyu Ando, Hiroko Miura, Kenji Asano, Yusuke Ochiai, Mayuko Ikeda, Ryoichi Nishitani, et al. New approach for rice improvement using a pleiotropic qtl gene for lodging resistance and yield. *Nature communications*, 1(1):1–11, 2010.

[155] Chris Paterson. A perspective from above. `http://www.agadvance.com/issues/jan-2013/a-perspective-from-above.aspx`, 2013.

[156] Mark Pfeiffer, Michael Schaeuble, Juan Nieto, Roland Siegwart, and Cesar Cadena. From perception to decision: A data-driven approach to end-to-end motion planning for autonomous ground robots. In *2017 ieee international conference on robotics and automation (icra)*, pages 1527–1533. IEEE, 2017.

[157] Quoc-Viet Pham, Fang Fang, Vu Nguyen Ha, Md Jalil Piran, Mai Le, Long Bao Le, Won-Joo Hwang, and Zhiguo Ding. A survey of multi-access edge computing in 5g and beyond: Fundamentals, technology integration, and state-of-the-art. *IEEE Access*, 8:116974–117017, 2020.

[158] Gilles Pison. The population of the world (2019). *Population & Societies*, (569):1–8, 2019.

[159] Barry Porter, Matthew Grieves, Roberto Rodrigues Filho, and David 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.

[160] Guannan Qu, Adam Wierman, and Na Li. Scalable reinforcement learning of localized policies for multi-agent networked systems. In *Learning for Dynamics and Control*, pages 256–266, 2020.

[161] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, 2009.

[162] Rahul Raj, Jeffrey P Walker, Rohit Pingale, Rohit Nandan, Balaji Naik, and Adinarayana Jagarlapudi. Leaf area index estimation using top-of-canopy airborne rgb images. *International Journal of Applied Earth Observation and Geoinformation*, 96:102282, 2021.

[163] Eduardo Romero, Christopher Stewart, Angela Li, Kyle Hale, and Nathaniel Morris. Bolt: Fast inference for random forests. In *Proceedings of the 23rd ACM/IFIP International Middleware Conference*, pages 94–106, 2022.

[164] Martin Rosalie, Matthias R. Brust, Gregoire Danoy, Serge Chaumette, and Pascal Bouvry. Coverage optimization with connectivity preservation for uav swarms applying chaotic dynamics. In *ICAC*, 2017.

[165] Cynthia Rosenzweig and Martin L. Parry. Potential impact of climate change on world food supply. *Nature*, 367, 1994.

[166] Christina F Rusnock, Jayson G Boubin, Joseph J Giametta, Tyler J Goodman, Anthony J Hillesheim, Sungbin Kim, David R Meyer, and Michael E Watson. The role of simulation in designing human-automation systems. In *International Conference on Augmented Cognition*, pages 361–370. Springer, 2016.

[167] Jose Luis Sanchez-Lopez, Ramón A Suárez Fernández, Hriday Bavle, Carlos Sampedro, Martin Molina, Jesus Pestana, and Pascual Campoy. Aerostack: An architecture and open-source software framework for aerial robotics. In *International Conference on Unmanned Aircraft Systems*, 2016.

[168] Jose Luis Sanchez-Lopez, Martin Molina, Hriday Bavle, Carlos Sampedro, Ramón A Suárez Fernández, and Pascual Campoy. A multi-layered component-based approach for the development of aerial robotic systems: the aerostack framework. *Journal of Intelligent & Robotic Systems*, 88, 2017.

[169] Mahadev Satyanarayanan. The emergence of edge computing. *Computer*, 50(1):30–39, 2017.

[170] Serge Savary, Andrea Ficke, Jean-Noel Aubertot, and Clayton Hollier. Crop losses due to disease and their implications for global food production losses and food security. *Food Security*, 4.2, 2012.

[171] Gigi Sayfan. *Mastering kubernetes*. Packt Publishing Ltd, 2017.

[172] Johannes Schemmel, Daniel Briiderle, Andreas Griibl, Matthias Hock, Karlheinz Meier, and Sebastian Millner. A wafer-scale neuromorphic hardware system for large-scale neural modeling. In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, pages 1947–1950. IEEE, 2010.

[173] Jürgen Scherer, Saeed Yahyanejad, Samira Hayat, Evsen Yanmaz, Torsten Andre, Asif Khan, Vladimir Vukadinovic, Christian Bettstetter, Hermann Hellwagner, and Bernhard Rinner. An autonomous multi-uav system for search and rescue. In *Proceedings of the First Workshop on Micro Aerial Vehicle Networks, Systems, and Applications for Civilian Use*, pages 33–38, 2015.

[174] Popular Science. Dji wants you to develop software for their drones. `https://www.popsci.com/`, 2015.

[175] Eduard Semsch, Michal Jakob, Dušan Pavlicek, and Michal Pechoucek. Autonomous uav surveillance in complex urban environments. In *2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology*, volume 2, pages 82–85. IEEE, 2009.

[176] National Weather Service. Noaa's online weather data (nowdata). `https://www.weather.gov/wrh/Climate?wfo=iln`.

[177] TL Setter, EV Laureles, and AM Mazaredo. Lodging reduces yield of rice by self-shading and reductions in canopy photosynthesis. *Field Crops Research*, 49(2-3):95–106, 1997.

[178] Sudhir Shrestha, Casey Harold, Matthew Boubin, and Logan Lawrence. Smart wristband with integrated chemical sensors for detecting glucose levels using breath volatile organic compounds. In *Smart Biomedical and Physiological Sensor Technology XVI*, volume 11020, pages 158–165. SPIE, 2019.

[179] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The hadoop distributed file system. In *2010 IEEE 26th symposium on mass storage systems and technologies (MSST)*, pages 1–10. Ieee, 2010.

[180] Roland Siegwart, Illah Reza Nourbakhsh, and Davide Scaramuzza. *Introduction to autonomous mobile robots*. MIT press, 2011.

[181] Nathan Silberman, Derek Hoiem, Pushmeet Kohli, and Rob Fergus. Indoor segmentation and support inference from rgbd images. In *European conference on computer vision*, pages 746–760. Springer, 2012.

[182] Avi Singh, Larry Yang, Kristian Hartikainen, Chelsea Finn, and Sergey Levine. End-to-end robotic reinforcement learning without reward engineering. *arXiv preprint arXiv:1904.07854*, 2019.

[183] M. Song, K. Zhong, J. Zhang, Y. Hu, D. Liu, W. Zhang, J. Wang, and T. Li. In-situ ai: Towards autonomous and incremental deep learning for iot systems. In *High Performance Computer Architecture*, 2018.

[184] Christopher Stewart, Terence Kelly, and Alex Zhang. Exploiting nonstationarity for performance prediction. In *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, pages 31–44, 2007.

[185] Christopher Stewart and Kai Shen. Performance modeling and system management for multi-component online services. In *Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation-Volume 2*, pages 71–84, 2005.

[186] Christopher Stewart and Kai Shen. Some joules are more precious than others: Managing renewable energy in the datacenter. In *Proceedings of the workshop on power aware computing and systems*, pages 15–19. IEEE, 2009.

[187] Christopher Stewart and Kai Shen. Some joules are more precious than others: Managing renewable energy in the datacenter. In *ACM Workshop on Power Aware Computing and Systems*, 2009.

[188] Zhiqiang Sui, Zheming Zhou, Zhen Zeng, and Odest Chadwicke Jenkins. Sum: Sequential scene understanding and manipulation. *arXiv preprint arXiv:1703.07491*, 2017.

[189] Max Taylor, Jayson Boubin, Haicheng Chen, Christopher Stewart, and Feng Qin. A study on software bugs in unmanned aircraft systems. In *2021 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 1439–1448. IEEE, 2021.

[190] ArduPilot Dev Team. Ardupilot. *URL: www. ardupilot. org, accessed*, 2:12, 2016.

[191] Amila Thibbotuwawa, Grzegorz Bocewicz, Grzegorz Radzki, Peter Nielsen, and Zbigniew Banaszak. Uav mission planning resistant to weather uncertainty. *Sensors*, 20(2):515, 2020.

[192] Amila Thibbotuwawa, Grzegorz Bocewicz, Banaszak Zbigniew, and Peter Nielsen. A solution approach for uav fleet mission planning in changing weather conditions. *Applied Sciences*, 9(19):3972, 2019.

[193] Stephen M Trimberger. *Field-programmable gate array technology*. Springer Science & Business Media, 2012.

[194] Aqeel ur Rehman, Abu Zafar Abbasi, Norman Islam, and Zubair Ahmed Shaikh. A review of wireless sensors and networks' application in agriculture. *Computer Standards and Interfaces*, 36.2, 2014.

[195] Chris Urmson, Joshua Anhalt, Drew Bagnell, Christopher Baker, Robert Bittner, MN Clark, John Dolan, Dave Duggins, Tugrul Galatali, Chris Geyer, et al. Autonomous driving in urban environments: Boss and the urban challenge. *Journal of Field Robotics*, 25(8):425–466, 2008.

[196] Kimon P Valavanis and George J Vachtsevanos. Future of unmanned aviation. In *Handbook of unmanned aerial vehicles*. Springer, 2015.

[197] Deepak Vasisht, Zerina Kapetanovic, Jongho Won, R̃anveer Chandra, Anish Kapoor, Sudipta Sinha, Madhusudhan Sudarshan, and Sean Strãtman. Farmbeats: An iot platform for data-driven agriculture. In *NSDI*, 2017.

[198] Shuo Wan, Jiaxun Lu, Pingyi Fan, and Khaled B Letaief. Toward big data processing in iot: Path planning and resource management of uav base stations in mobile-edge computing system. *IEEE Internet of Things Journal*, 7(7):5995–6009, 2019.

[199] Cheng Wang, Bhuvan Urgaonkar, Aayush Gupta, Lydia Y. Chen, Robert Birke, and George Kesidis. Effective capacity modulation as an explicit control knob for public cloud profitability. In *ICAC*, 2016.

[200] Dashuai Wang, Wei Li, Xiaoguang Liu, Nan Li, and Chunlong Zhang. Uav environmental perception and autonomous obstacle avoidance: A deep learning and depth camera combined solution. *Computers and Electronics in Agriculture*, 175:105523, 2020.

[201] J Wang, Z Feng, Z Chen, S George, and et al. Bandwidth-efficient live video analytics for drones via edge computing. In *ACM Symposium on Edge Computing*, 2018.

[202] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[203] David M Woebbecke, George E Meyer, Kenneth Von Bargen, and David A Mortensen. Color indices for weed identification under various soil, residue, and lighting conditions. *Transactions of the ASAE*, 38(1):259–269, 1995.

[204] Zichen Xu, Nan Deng, Christopher Stewart, and Xiaorui Wang. Cadre: Carbon-aware data replication for geo-diverse services. In *ICAC*, 2015.

[205] Ming-Der Yang, Jayson G Boubin, Hui Ping Tsai, Hsin-Hung Tseng, Yu-Chun Hsu, and Christopher 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.

[206] Ming-Der Yang, Kai-Siang Huang, Yi-Hsuan Kuo, Hui Ping Tsai, and Liang-Mao Lin. Spatial and spectral hybrid image classification for rice lodging assessment through uav imagery. *Remote Sensing*, 9(6):583, 2017.

[207] Ming-Der Yang, Kai-Siang Huang, Jin Wan, Hui Ping Tsai, and Liang-Mao Lin. Timely and quantitative damage assessment of oyster racks using uav images. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 11(8):2862–2868, 2018.

[208] Ming-Der Yang, Tung-Ching Su, Nang-Fei Pan, and Yeh-Fen Yang. Systematic image quality assessment for sewer inspection. *Expert Systems with Applications*, 38(3):1766–1776, 2011.

[209] Ming-Der Yang, Hsin-Hung Tseng, Yu-Chun Hsu, and Hui Ping Tsai. Semantic segmentation using deep learning with vegetation indices for rice lodging identification in multi-date uav visible images. *Remote Sensing*, 12(4):633, 2020.

[210] Qi Yang, Liangsheng Shi, Jinye Han, Yuanyuan Zha, and Penghui Zhu. Deep convolutional neural networks for rice grain yield estimation at the ripening stage using uav-based remotely sensed images. *Field Crops Research*, 235:142–153, 2019.

[211] S Yi, Z Hao, Q Zhang, Q Zhang, W Shi, and et al. Lavea: Latency-aware video analytics on edge computing platform. In *ACM Symposium on Edge Computing*, 2017.

[212] Ashkan Yousefpour, Caleb Fung, Tam Nguyen, Krishna Kadiyala, Fatemeh Jalali, Amirreza Niakanlahiji, Jian Kong, and Jason P Jue. All one needs to know about fog computing and related edge computing paradigms: A complete survey. *Journal of Systems Architecture*, 98:289–330, 2019.

[213] Chenhan D. Yu, Jianyu Huang, Woody Austin, Bo Xiao, and George Biros. Performance optimization for the k-nearest neighbors kernel on x86 architectures. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '15, pages 7:1–7:12, New York, NY, USA, 2015. ACM.

[214] Fisher Yu, Haofeng Chen, Xin Wang, Wenqi Xian, Yingying Chen, Fangchen Liu, Vashisht Madhavan, and Trevor Darrell. Bdd100k: A diverse driving dataset for heterogeneous multitask learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2636–2645, 2020.

[215] Chunhua Zhang and John M. Korvacs. The application of small unmanned aerial systems to precision agriculture. *Precision Agriculture*, 13.6, 2012.

[216] Chunhua Zhang and John M Kovacs. The application of small unmanned aerial systems for precision agriculture: a review. *Precision agriculture*, 13(6):693–712, 2012.

[217] Haoyu Zhang, Ganesh Ananthanarayanan, Peter Bodik, Matthai Philipose, Paramvir Bahl, and Michael J Freedman. Live video analytics at scale with approximation and delay-tolerance. In *NSDI*, 2017.

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

[219] Kaiqing Zhang, Zhuoran Yang, and Tamer Başar. Multi-agent reinforcement learning: A selective overview of theories and algorithms. *arXiv preprint arXiv:1911.10635*, 2019.

[220] Kaiqing Zhang, Zhuoran Yang, Han Liu, Tong Zhang, and Tamer Basar. Fully decentralized multi-agent reinforcement learning with networked agents. In *International Conference on Machine Learning*, pages 5872–5881. PMLR, 2018.

[221] Shiqi Zhang and Christopher Stewart. Computational thinking curriculum for unmanned aerial systems. In *2019 IEEE National Aerospace and Electronics Conference (NAECON)*, pages 122–125. IEEE, 2019.

[222] Xiao Zhang and Ximing Cai. Climate change impacts on global agricultural land availability. *Environmental Research Letters*, 6, 2011.

[223] Zichen Zhang, Jayson Boubin, Christopher Stewart, and Sami Khanal. Whole-field reinforcement learning: A fully autonomous aerial scouting method for precision agriculture. *Sensors*, 20(22):6585, 2020.

[224] Zichen Zhang, Sami Khanal, Amy Raudenbush, Kelley Tilmon, and Christopher Stewart. Assessing the efficacy of machine learning techniques to characterize soybean defoliation from unmanned aerial vehicles. *Computers and Electronics in Agriculture*, 193:106682, 2022.

[225] Xin Zhao, Yitong Yuan, Mengdie Song, Yang Ding, Fenfang Lin, Dong Liang, and Dongyan Zhang. Use of unmanned aerial vehicle imagery and deep learning unet to extract rice lodging. *Sensors*, 19(18):3859, 2019.

[226] Hengbiao Zheng, Xiang Zhou, Jiaoyang He, Xia Yao, Tao Cheng, Yan Zhu, Weixing Cao, and Yongchao Tian. Early season detection of rice plants using rgb, nir-gb and multispectral images from unmanned aerial vehicle (uav). *Computers and Electronics in Agriculture*, 169:105223, 2020.

[227] Yu Zheng, Quannan Li, Yukun Chen, Xing Xie, and Wei-Ying Ma. Understanding mobility based on gps data. In *Proceedings of the 10th international conference on Ubiquitous computing*, pages 312–321, 2008.

[228] Yongkun Zhou, Bin Rao, and Wei Wang. Uav swarm intelligence: Recent advances and future trends. *IEEE Access*, 8:183856–183878, 2020.

[229] Haosheng Zou, Tongzheng Ren, Dong Yan, Hang Su, and Jun Zhu. Reward shaping via meta-learning. *arXiv preprint arXiv:1901.09330*, 2019.