Managing Edge Resources for Fully Autonomous Aerial Systems

Jayson G. Boubin, Naveen T.R. Babu, Christopher Stewart,
John Chumley, and Shiqi Zhang
The Ohio State University Department of Computer Science

ABSTRACT

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%.

SEC 2019, November 7–9, 2019, Arlington, VA, USA

ACM ISBN 978-1-4503-6733-2/19/11…$15.00
https://doi.org/10.1145/3318216.3363306

ACM Reference Format:
Jayson G. Boubin, Naveen T.R. Babu, Christopher Stewart,, John Chumley, and Shiqi Zhang, and The Ohio State University Department of Computer Science. 2019. Managing Edge Resources for Fully Autonomous Aerial Systems. In *The Fourth ACM/IEEE Symposium on Edge Computing (SEC 2019), November 7–9, 2019, Arlington, VA, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3318216.3363306

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 [3, 13, 26, 32, 42]. 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 [32, 41]. 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 [39]. Pixhawk and Aerostack also provide platforms for software control [33, 37, 38].

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 [12, 17, 20, 21, 43–45], 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 [3, 30, 41, 43, 44]. If aircraft surrender battery capacity to onboard processors and carry heavier payloads for data storage, flight times will suffer [43]. 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) [18].

To validate the model, we created SoftwarePilot [6], 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 [30, 37]. 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 [37] and Autoware [22, 30]. 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 [32]. 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.

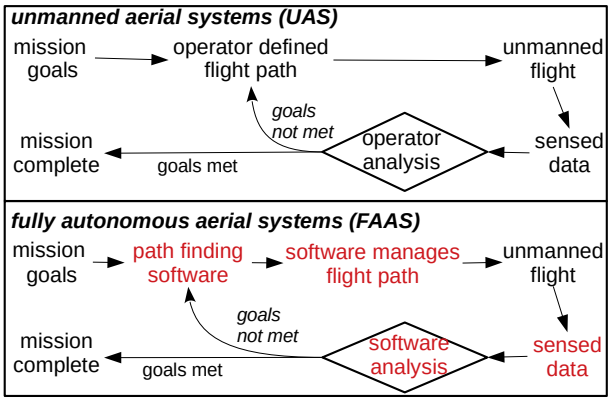

Fig. 1: With FAAS, humans set mission goals but do not pilot.

- 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 provides an overview of FAAS. Section 3 presents an energy model driven by autonomy cubes. Section 4 describes the implementation of FAAS. Section 5 validates our model and studies FAAS workloads. Section 6 uses our modeling approach to guide FAAS system management. Section 7 discusses future work and limitations of our approaches. Section 8 discusses related work. Section 9 presents conclusions.

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 1 depicts UAS workflow. Humans set high-level mission goals, 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 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 [29], 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 [7, 16, 42]. By eschewing human pilots, FAAS can lower costs.

Figure 2 depicts flight area and two agricultural scouting missions explored similarly in prior research [5]. Flight area comprises waypoints where the aircraft can fly. Each cell in Figure 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 [26].) 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 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 [43]. With FAAS, analytics can affect flight actions, changing resource demands in ways that are hard to predict.

Runtime Execution: Figure 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 [3]. Like video analytics, FAAS use *AI models* to convert sensed images to multi-dimensional points [17, 20, 36, 43]. 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 [43]. *Reinforcement learning* can be a robust approach to autonomously control devices [27]. In this approach, FAAS are distributed

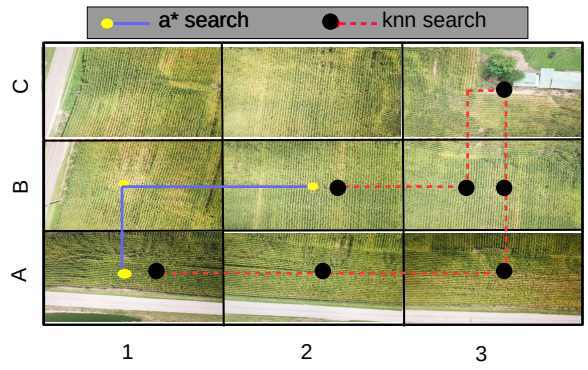

Fig. 2: Two agricultural scouting missions. Each image represents an allowed waypoint where the aircraft could have flown (i.e., flight area). Lines represent actual waypoints visited. Both missions begin at A1.

with training data on 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.

3 PERFORMANCE MODELING

Figure 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 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.

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 [22, 30, 38]. Autoware [22, 30] 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 [38] 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 presents a simple 2-D autonomy cube.

Defining autonomy cubes: Shown in Equation 1, A waypoint x is a multi-dimensional point. Dimensions can abstract (1) GPS or grid positions (e.g., Figure 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: $\forall i : 0 < d_i < D_i$.

$$\mathrm{x} = (\mathrm{d}_0, ..., \mathrm{d}_\mathrm{K}) \tag{1}$$

$$\mathbf{fa_i} \in \mathbf{FA} : \{\mathrm{x_m}\} \rightarrow \{\mathrm{x_n}\} \tag{2}$$

$$\mathbf{FS} : \{\mathrm{x}, \mathrm{fa_i}, \mathrm{y}\} \rightarrow \{1|0\} \text{ if } \mathrm{fa_i(x)} = \mathrm{y} \tag{3}$$

$$\mathbf{ug} = \frac{\mathrm{u(ai(x))}}{\mathrm{u(ai(y))}} \circ \mathrm{FS}\{\mathrm{x}, \mathrm{fa_i}, \mathrm{y}\} \tag{4}$$

As shown in Equation 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 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

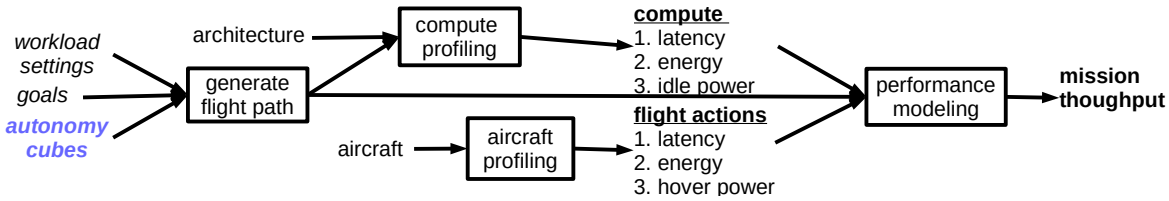

Fig. 3: Modeling mission throughput for multiple architectures, aircraft and autonomy settings.

models (ai) and computes utility ($0 <= u(x) <= 1$) of its current state, which we call a featureset. Each valid flight step, i.e., $FS(\ldots) \rightarrow 1$, has utility gain. Referring to Figure 2, the first flight step for the blue line is: {[A, 1], FlyNorth, [B, 1]}. The AI models for this mission include A* (i.e., $\text{ai}_{\text{A}*}$).

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 4 shows that ac allows FAAS to compute utility gain for any valid flight step. As shown in Equation 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.

$$\begin{aligned} \text{fp} &= \{\text{x}_\text{n}, \text{fa}_{\text{f(n)}}, \text{y}_\text{n}\}_\text{N} \\ \textbf{TG} : \{\text{fp}\} \rightarrow \text{r} \in \mathbb{R} : \text{r} &= \prod_\text{n} \text{ug}(\text{x}_\text{n}, \text{fa}_{\text{f(n)}}, \text{y}_\text{n}) \end{aligned} \tag{5}$$

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 [31]. 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 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 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 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.

$$\text{P} = \text{Latency}(\text{fa}_\text{s}(\text{x})) \times \text{t} : \tag{6}$$

$$\frac{\text{Max}_\text{i}(\text{Latency}(\text{fa}_\text{i}(\text{x}))) * |\mathbf{X}|}{\text{s}} < \text{P} \tag{7}$$

Examining Equation 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.

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 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.

$$\begin{gathered}\text{LatencyA} = \{\text{fa}_\text{i}(\text{x})\} \rightarrow \text{r} \in \mathbb{R} : \\ \text{r} = \frac{\sum_{\text{n}'} \overline{\text{LatencyA}(\text{fa}_\text{i}((\text{x}_{\text{n}'}))}}{\text{N}}, \text{N} << |\mathbf{X}|\end{gathered} \tag{8}$$

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 8.

$$\begin{gathered}\text{LatencyC} = \{\text{ai}(\text{x})\} \rightarrow (\mu, \sigma) \in \mathbb{R} : \\ \sigma = \frac{\sum_{\text{n}'} \overline{\text{Latency}(\text{ai}((\text{x}_{\text{n}'}))}}{\text{N}}, \text{N} << |\mathbf{X}|\end{gathered} \tag{9}$$

3.4 Throughput modeling

Recall, FAAS compute their flight path fp_i at runtime. As shown in Equation 10, each flight step $\text{fs}_{[\text{i,n}]} \in \text{fp}_\text{i}$ is informed by data observed during execution.

$$\widehat{\mathbf{ug}_{\text{n,n+1}}} : \{\text{x}_\text{n}, \text{fa}_{\text{f(n)}}, \text{y}_\text{n}\}_\text{N} \rightarrow \{\text{x}_{\text{n+1}}, \text{fa}_{\text{f(n+1)}}, \text{y}_{\text{n+1}}\} \tag{10}$$

Specifically, FAAS compute expected utility gain $\widehat{\mathbf{ug}}$, using past and training data comprised of additional autonomy

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 1: Layered implementation and system settings.

cubes to infer the effects of flight actions. Flight path fp_i is the result of iterative invocations of expected utility gain given an autonomous cube, i.e., $\text{fp}_\text{i} = \{\widehat{\mathbf{ug}} \circ_\text{N} \text{ac}\}$. 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 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 11.

$$\begin{gathered}\text{E}_\text{air}(\text{N}) : \sum_\text{n}(\text{EnA}(\text{fa}_\text{n}) + \text{LatencyC}(\text{ai}(\text{x}_\text{n})) \times \text{PwrA}(\text{fa}_\text{hover})) \\ \text{E}_\text{cmp}(\text{N}) : \sum_\text{n}(\text{EnC}(\text{ai}(\text{x}_\text{n})) + \text{LatencyA}(\text{fa}_\text{n}) \times \text{PwrC}(\text{idle})) \\ \text{tput} = \min \text{N} : \text{C}_\text{air} - \text{E}_\text{air}(\text{N}) = 0 \text{ or } \text{C}_\text{cmp} - \text{E}_\text{cmp}(\text{N}) = 0\end{gathered} \tag{11}$$

4 IMPLEMENTING FAAS

Table 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 [1, 3].

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 [5]. 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 [8]. 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 [9, 22, 30].

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 [28]. Our suite allows for execution of autonomy cube-based pathfinding and modeling on edge or cloud systems.

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.

5.1 Model validation

We flew each benchmark under the system settings in Table 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 [11]. 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.

- **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.

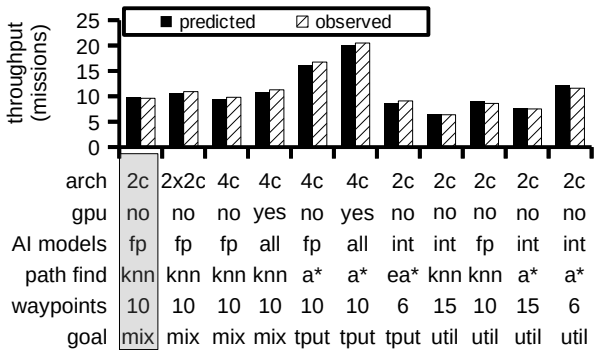

Fig. 4: Our model predicts mission throughput precisely. Baseline setting is highlighted

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 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 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* [22, 23, 30], researchers use ROSBAG recordings from a real, long-running self-driving car. We mimicked this approach by collecting long traces over multiple missions. For autonomous

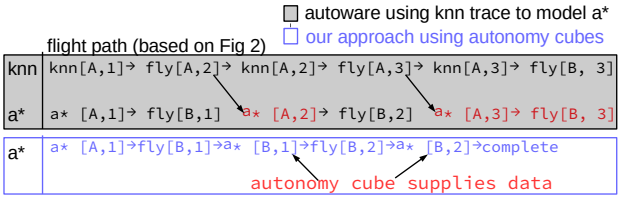

Fig. 5: Depicting Autoware versus our approach.

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 5 depicts the problem using examples from Figure 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 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 [38]. 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 6 (a – d) compare our approach, Autoware and Aerostack. We also compare a simple modeling approach driven by data collected from DJI and Intel (*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 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 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 6(e) describes the effects of image quality on throughput. Recent UAS work suggests using high compression ratios [4, 15] (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 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 [4, 15]), 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.

5.2 Workload study

Figure 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 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.

6 SYSTEM MANAGEMENT

Our model can help FAAS end users: (1) manage compute hardware, (2) assess trade-offs between tightly and loosely

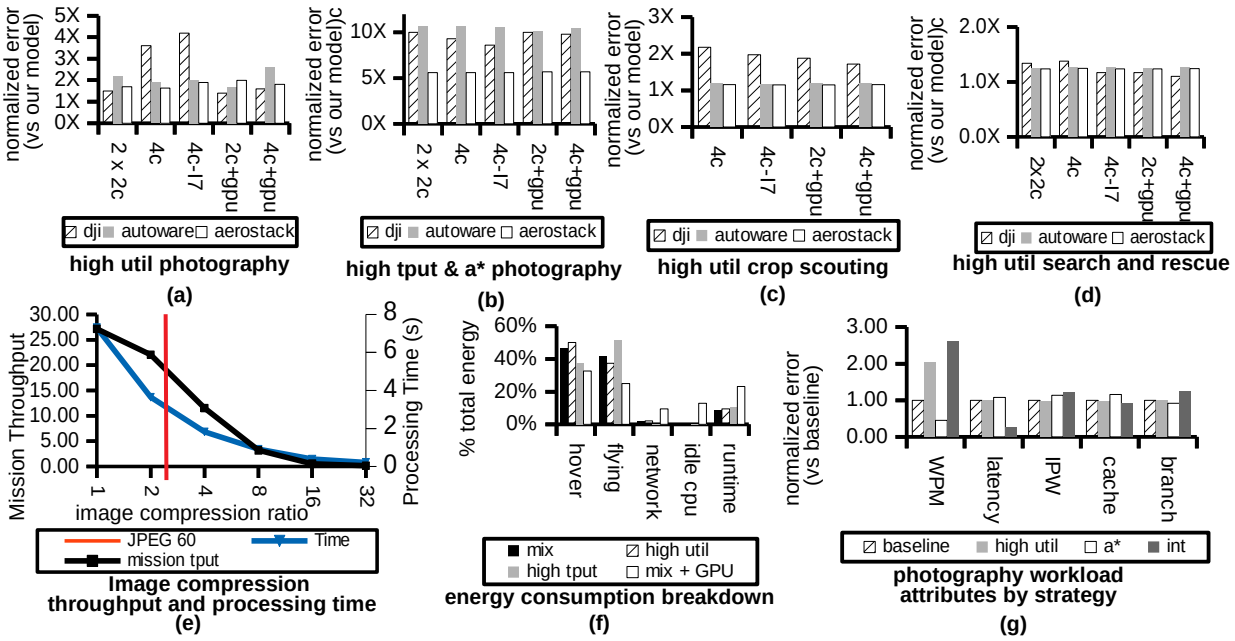

Fig. 6: (a – d) Comparing trace and cube driven modeling approaches. (e) poor image quality degrades mission. (f) Autonomy settings shift compute and aircraft energy demands. (g) Autonomy settings affect common architecture counters.

coupled aircraft, software and hardware, and (3) adapt hardware and software at runtime.

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 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.

6.2 Comparing onboard, edge and cloud

DJI software development kits support edge architecture where tablets run AI software and control aircraft remotely [10]. For developers, these devices offer one hop, low latency access to the aircraft and powerful compute. Further, developers can procure resources as needed.

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

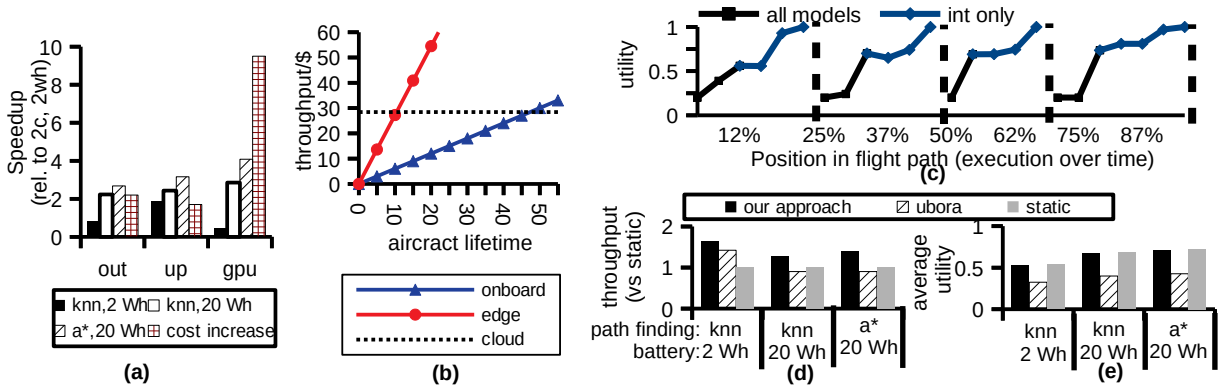

Fig. 7: Model-driven management of autonomous photography FAAS: (a) speedup and cost by architecture optimization, (b) efficiency of onboard, edge, and cloud systems, (c) utility of images captured across FAAS missions with adaptive model switching, (d-e) a comparison of throughput and utility across adaptive switching approaches.

and payload weight depends on nominal thrust and aircraft weight [31]. 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 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.

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 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 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 [24, 25]. 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 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 7(e), 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.

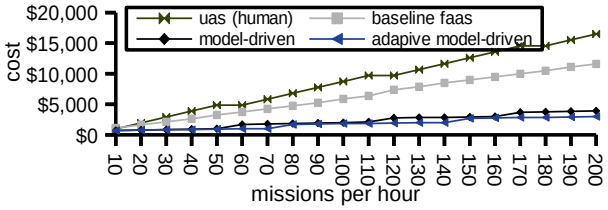

Fig. 8: Agricultural scouting cost as end users use parallelism to increase throughput. Our model-driven edge management provides savings at scale.

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.

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 [29]. We also model cost for UAS equipment such as batteries, compute resources, and aircraft.

Figure 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.

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 geotagged multi-dimensional image sets. For instance, large available datasets like Google Street View [2], or autonomous driving datasets like KITTI [14] 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.

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 [22]. 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 [30]. 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 [38] 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 [19] studied FPGAs, CPUs, GPUs, and coprocessors on personal assistant benchmarks. In-situ AI [40] studied autonomous IoT. Computational sprinting has targeted interactive, mobile workloads with dynamic architectural optimizations [34, 35].

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.

Acknowledgments: This work was funded in part by NSF Grants 1749501 and 1350941 with support from NSF CENTRA collaborations (grant 1550126).

REFERENCES

[1] E. Ackerman. Skydio announces sdk to make worldâĂŹs cleverest drone even cleverer. https://spectrum.ieee.org/.

[2] 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.

[3] 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.

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

[5] 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.

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

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

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

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

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

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

[12] AR Elias, N Golubovic, C Krintz, and et al. Where's the bear?-automating wildlife image processing using iot and edge cloud systems. In *IEEE Internet of Things Design and Implementation*, 2017.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

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

[21] S Kartakis, W Yu, R Akhavan, and et al. Adaptive edge analytics for distributed networked control of water systems. In *IEEE Internet of Things Design and Implementation*, 2016.

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

[23] 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.

[24] 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.

[25] 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.

[26] 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.

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

[28] 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.

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

[30] 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.

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

[32] 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.

[33] 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.

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

[35] 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.

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

[37] 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.

[38] 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.

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

[40] 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.

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

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

[43] 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.

[44] 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.

[45] 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.