Known Knowns and Unknowns: Near-realtime Earth Observation Via Query Bifurcation in `Serval`

Bill Tao, Om Chabra, Ishani Janveja, Indranil Gupta, Deepak Vasisht
University of Illinois Urbana-Champaign

Abstract

Earth observation satellites, in low Earth orbits, are increasingly approaching near-continuous imaging of the Earth. Today, these satellites capture an image of every part of Earth every few hours. However, the networking capabilities haven't caught up, and can introduce delays of few hours to days in getting these images to Earth. While this delay is acceptable for delay-tolerant applications like land cover maps, crop type identification, etc., it is unacceptable for latency-sensitive applications like forest fire detection or disaster monitoring. We design `Serval` to enable near-realtime insights from Earth imagery for latency-sensitive applications despite the networking bottlenecks by leveraging the emerging computational capabilities on the satellites and ground stations. The key challenge for our work stems from the limited computational capabilities and power resources available on a satellite. We solve this challenge by leveraging predictability in satellite orbits to bifurcate computation across satellites and ground stations. We evaluate `Serval` using trace-driven simulations and hardware emulations on a dataset comprising ten million images captured using the Planet Dove constellation comprising nearly 200 satellites. `Serval` reduces end-to-end latency for high priority queries from 71.71 hours (incurred by state of the art) to 2 minutes, and 90-th percentile from 149 hours to 47 minutes.

1 Introduction

Low Earth Orbit (LEO) satellites promise to deliver continuous, high-resolution imagery of the Earth through large constellations of low cost cubesats. These constellations, e.g., Planet Dove [50], deploy imaging sensors on cubesats in low orbits nearly 500 Kilometers above the Earth's surface. Due to their low orbits and large constellation size, they can capture an image of every location on Earth multiple times per day. The imagery from these satellites is useful for many applications such as disaster monitoring [11, 18], precision agriculture [9, 40], disease modeling [14], climate monitoring [64], and financial analytics [62].

However, Earth observation constellations today cannot support many latency sensitive applications because they suffer from large latency of few hours to days between an image capture and its availability to the end user [59]. For example, a fire department needs the images of a wildfire within a few minutes so as to limit risks to human lives, forest ecosystems, and property. Such delays largely arise from a satellite's data transfer process (see Fig. 1). Satellite imagery must be transported to ground stations on Earth, and from there to the cloud, where processing, storage, and insight generation can occur. This pipeline can incur a delay of hours and sometimes days. This is due to several factors: (a) orbital dynamics, as satellites have intermittent access to ground stations on Earth—about ten minutes per contact, with 4-5 good quality contacts per day; and (b) the bandwidth from satellite to ground station is limited due to the large distance from satellite to Earth, making it nearly impossible to downlink all the images from a satellite during every contact.

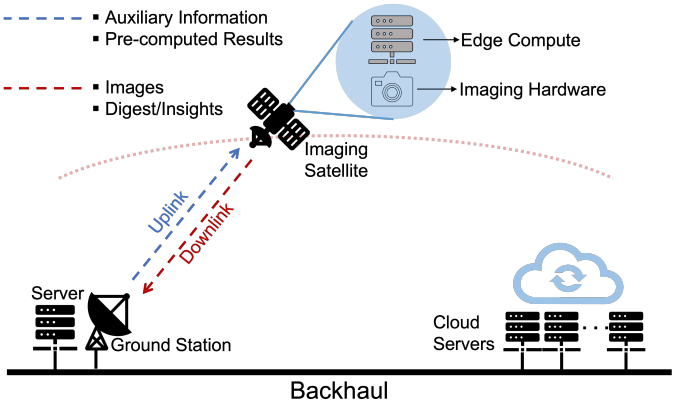


Figure 1: `Serval` **distributes computation across satellites and ground stations to prioritize latency-sensitive imagery and insights.**

This paper's goal is to enable *near-real-time insights* from satellite imagery by reducing the time-to-insight for satellite imagery to O(minutes). We do so by by leveraging emerging compute capabilities on satellites and ground stations. This is feasible today due to the ongoing push to equip satellites with small amounts of compute resources such as a Raspberry Pi or a NVIDIA Jetson, e.g., in 2020, the European Space Agency (ESA) deployed a neural-network based cloud detector on their Φ-Sat-1 mission [28]. Similarly, many recent proposals from industry [7, 46] and academia [60, 61] argue for co-locating ground stations and data centers to reduce terrestrial networking delays and enable compute on ground stations. Our core idea is to leverage the emerging general-purpose computational capabilities available on satellites and ground stations to *prioritize latency-sensitive images* such as those containing forest fires, while deprioritizing other images that are relatively less latency-sensitive.

To achieve this goal, we build Serval [1], a novel edge computing framework designed to derive near-real-time insights from LEO satellites. Serval allows multiple long-running queries to execute simultaneously on incoming satellite imagery. Given a query set, Serval intelligently distributes compute across satellite, ground station, and the datacenter. Similar to comodity products for satellite imagery analysis such as Planet Analytics [1], Serval represents each query as a logical intersection of a sequence of *filters*, e.g. "forest fires in California" is denoted as three sequential filters *{California, forest, fire}*, each of which involves geographical or statistical computation (e.g., a neural network). Different queries may have different latency-sensitivity and compute requirements. Unlike past work [23], Serval does not discard any images because new applications can emerge post-collection (e.g., historical data analysis or disaster management). For example, recently Planet [51], a leading Earth observation company, used their satellite imagery to retroactively track the origin and flight of a balloon that entered the United States airspace [5], which would be impossible if images had been discarded. Instead, Serval focuses on dynamically *reordering* image delivery to reduce end-to-end delays for latency-sensitive content.

The key challenges in Serval stem from the scale of satellite imagery and the limited compute capabilities available inside a LEO satellite. First, each satellite generates nearly a Terabyte of data per day. Second, a satellite needs to perform the query compute on this data using its limited compute capacity. LEO satellites generally have small solar panels that generates limited power. For instance, the model used in [23] has a 7W solar panel, a large fraction of which is utilized for critical satellite function. However, a Jetson TX2 itself consumes 11.3W. Moreover, solar panel power supply is further diminished because it generates no power when the satellite is on the dark side of the Earth. This means that the computer onboard cannot be always on. Together, this means it is infeasible for all images being collected by the satellite to be processed on-board.

Serval's key insight is based on our observation that a query is typically composed of *two kinds* of filters, determined by the rate of change of the data the filter pertains to. The data beneath some filters may change quite quickly—we call these as *dynamic filters*. Examples include (the outline of) fires, (position of) boats, etc. However, the data beneath the second class of filters changes much more slowly—we call these as *glacial filters*. Examples include forest identification, and ocean and land boundaries—these boundaries do not often change within a day (or even weeks).

Serval *bifurcates* a query—it assigns the temporally static (glacial) parts of the query to spatially static entities (ground stations, the cloud) while assigning temporally dynamic parts of the query to spatially dynamic entities (satellites). Consider a query such as "forest fires in California". Serval can decompose this query into identifying "California", identifying "forests", and identifying "fire". In this set, "California" and "forests" are both glacial filters, while "fire" is a dynamic filter. Our key insight is that *glacial filters can be pre-computed on the ground stations using stale imagery (e.g., a day-old image)*. Such glacial filter computation can be done even before an image is captured at the satellite and the results can be conveyed to the satellite. Serval's bifurcated approach has two advantages. First, the pre-computed glacial filter results on the ground means that the LEO satellite only needs to compute the dynamic filters of a query. Second, the same glacial filter inferences can be reused by multiple satellites (single compute, multiple use). The glacial filter offload to ground stations is enabled by the predictability of satellite orbits and as a result, predictability of the geographical location and time of each image.

For some filters that have to run in real-time, such as cloud detection, we can infer high-quality priors by additionally incorporating *auxiliary information* available at the ground station, e.g., weather forecast information. Cloud detection is an important component of RGB image analysis because clouds occlude useful information and must be rejected prior to processing. For instance, if the forecasted cloud cover is either very low or very high, Serval can skip the cloud detection step altogether aboard the satellite.

We evaluate Serval using a combination of trace-driven simulations and hardware emulation on two applications: 'forest fires in California', and 'vessel counting at ports'. These applications represent opposite ends of a spectrum: the former outputs a set of images, while the latter outputs counts. We designed a LEO satellite simulator and evaluated Serval using real image traces collected from PlanetScope, a constellation of over 200 CubeSats launched by Planet Lab. Our paper is, to the best of our knowledge, the largest evaluation performed using data collected by a real operational satellite constellation. Specifically, our traces contain ten million images collected using 151 satellites across 20 days. We evaluate Serval using two different satellite configurations and two different ground station configurations.

Contributions: We summarize our contributions below:

- We present the first system that distributes compute across satellites and ground stations to deliver near-real-time insights from Earth observation satellites.
- We propose a new bifurcated query execution approach that offloads glacial (slowly-changing) filter computation to the ground in order to reduce computational load on satellites, and end-to-end latency.
- To the best of our knowledge, we are the first to evaluate our system on a real-world raw and continuous trace collected by the world's largest LEO satellite constellation for Earth observation.

[1] Smart Edge-based Realtime Visual Analytics for LeoSats

- Our evaluation shows that the `Serval` scheduler improves end-to-end median percentile latency on high-priority images from over 70 hours to 2 minutes (90-th percentile from 149 hours to 47 minutes), while also improving detection accuracy and reducing satellite compute load by over 80%.

2 Background

In the 1990's, Iridium, Globestar, and Teledesic [27, 30, 42] planned constellations of tens of satellites to provide direct connectivity to handheld terminals. Similarly, early imaging constellations, such as NOAA's series for weather sensing, comprised of a couple of satellites. These early constellations triggered important research in satellite networking [19, 21, 34, 38, 49, 52].

More recently, within the last 5-10 years, the emergence of large LEO constellations, comprising of hundreds of satellites, has been driven by lower launch and manufacturing costs of small satellites. For instance Planet's Dove constellation for Earth imagery is composed of nearly 200 low-cost cubesats ('shoebox-sized' satellites) with off-the-shelf components. Our work focuses on these modern constellations for Earth observation. These modern constellations differ from traditional satellite constellations in three ways:

- **Constellation size:** Modern satellite constellations (e.g., Planet Inc. [26], Spire Inc. [3], etc.), consist of hundreds of satellites as opposed to few satellites in traditional constellations. This allows modern constellations to get more frequent imagery of any part of Earth with revisit frequency of few hours as opposed to a delay of several days from traditional constellations.
- **Data volumes:** The low orbit of LEO satellites and improved imaging hardware enables high resolution imagery (e.g. $1m^2$ per pixel). They capture images of Earth in different parts of the frequency spectrum, e.g., RGB, Radio Waves, Infrared, etc. The multi-spectral imagery, increased satellite number, and high resolution lead to increased data volumes—from few GBs of data per day to TBs of data per day. For instance, Planet's Dove satellites generates approximately one Terabyte of data per satellite per day.
- **Applications:** Traditional Earth observation satellites could only support delay-tolerant applications like crop yield estimates, land cover use, etc. Modern constellations gather images more frequently and offer the promise of real-time applications like disaster monitoring, traffic analysis, maritime monitoring, etc.
- **Processing pipelines:** Modern data processing pipelines increasingly rely on modern Machine Learning (ML) methods, in contrast to (merely) traditional signal processing based approaches. For example, European Space Agency's Φ-Sat-1 [28] recently demonstrated the ability to perform neural network-based cloud detection on board a satellite.

Network pipeline: Finally, we provide a brief description of satellite network pipeline as context for the rest of the paper. LEO satellites for Earth observation operate in polar orbits and go around the Earth once every 1.5 hours appoximately. During each orbit, they pass over a different part of the Earth due to Earth's rotation. The data from these satellites is usually downloaded using few dedicated ground stations with Gbps link capacities [25]. Due to a satellite's orbital motion, it can contact each ground station four to six times a day, with each contact lasting up to ten minutes. To improve download latency and increase the number of contacts, recent work has proposed distributed ground station designs with multiple general-purpose ground stations [60,61]. In the industry, Amazon and Microsoft have launched ground-station-as-a-service platforms [7, 46], wherein satellite operators can rent time on existing ground stations to download data from satellites.

3 `Serval`'s Design

In this section, we present the problem setup, `Serval`'s approach to distributing compute across Earth and space, and `Serval`'s incorporation of auxiliary information. Finally, we describe `Serval`'s execution engine.

3.1 Problem Setup

Fig. 1 shows the three layers in a satellite networking system: (1) the LEO *satellite constellation* containing hundreds of orbiting satellites, (2) few *ground stations* across the world, each of which can communicate with satellites, and (3) *the cloud* consisting of one or more data centers. The satellites continuously image the Earth's surface and send the captured images down whenever they make the next contact with a ground station. The ground stations eventually upload images to the cloud.

The satellites are energy-constrained, i.e., they have access to limited amounts of power, and only a fraction of the power is available for compute. The primary usage of the power is to maintain temperature, power attitude determination, and control systems, communicate with ground stations, and other critical satellite functionality. We assume each satellite has limited computational capability such as a Jetson TX2 or Jetson AGX Orin. This assumption is validated by recent proposals of incorporating compute in satellites in academia [15, 22, 23, 44, 45], and by recent launches that incorporate computational capabilities on satellites [4, 28].

Ground stations have no power limits and are equipped with more (though not infinite) computational resources than satellites. This assumption is reasonable in both traditional ground station designs which were capital-intensive, as well

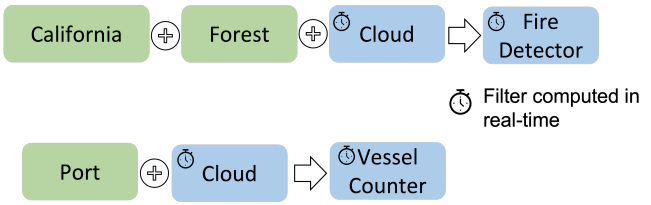


Figure 2: `Serval` **represents queries as a set of sequential filters. The picture above represents two queries: 'images containing forest fire in California', and 'Number of vessels at ports'.**

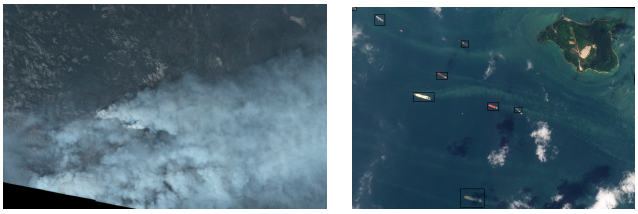


Figure 3: **Example output images for queries 'California Forest Fire' (left) and 'Vessels Counting in Port' (right).**

as in modern ground station designs wherein cloud providers co-locate compute resources at the ground stations.

Graph representation: Application developers who are customers of the satellite constellation operator company submit queries that will run on the Earth observation imagery. For example, Planet Inc. has a product named "Analytics" which allows users to run user-defined analytic applications. The user can choose a specific object detection model to run, and limit the set of images to run the selected model on by defining additional filters based on image metadata, including date, time, geographic location, cloud cover ratio, etc. [1]. However, today such queries are executed offline on the cloud once all the images have been delivered to the cloud — we seek to execute them in real-time.

While some queries may not be sensitive to taking days to generate a response, several queries are latency-sensitive and require responses in minutes. The satellite operator company may charge a premium for such latency-sensitive queries. Examples include: (a) a forest department may design a query like 'images of forest fires in California'; (b) a trading firm may be interested in a query like 'number of ships at major ports across the world'; (c) a disaster response team might be interested in 'images of, or number of, flood-damaged buildings in Florida'.

`Serval` supports two types of queries: (I) *image outputs:* queries that require images to be outputted matching the query, e.g., forest fire images from California may be needed for a detailed inspection of the damage and to create a plan for action, (II) *statistic outputs:* queries that only require the inference to be delivered, e.g., a hedge fund may need the count of cars in different parking lots across Beijing, rather than the actual images, or, financial traders may need to know the count of ships at a port.

We represent each query as a sequence of filters, which is consistent with the current commodity products such as Planet Analytics [1]. For example, 'forest fires in California' can be represented as *California→forest→cloud→fires*. Similarly, 'ships around ports' is represented as *Port→Cloud→Ship count*. Note that each query contains a cloud filter to remove images that are occluded by clouds. The filter representation is depicted visually in Fig. 2.

Each filter is a computational block that takes an image as an input and outputs either a boolean value or a number. For example, the *California* filter performs a geographical check and returns *True* or *False*, i.e., do the geographical coordinates of the image overlap with the geographical boundaries of *California*. Similarly, the filters *Vessel counter* may be a neural network (a stock network, or one supplied by the application developer) that detects and counts the number of vessels in an image. *Cloud*, *Forest*, and *Fire* are other neural network-based filters in examples above.

`Serval`'s goal is to prioritize images that pass all the filters corresponding to at least one latency-sensitive query. We also note that the filter representation of our queries lends itself to cross-query optimization. For example, if two different queries rely on the same underlying filters, `Serval` does not need to perform the computation twice, e.g., if two queries both rely on forests in California, then we do not need to perform forest detection twice. Similarly, *Cloud* filter needs to be computed only once for an image even though the image may be relevant to multiple queries.

Note on inter-satellite links: While Inter-satellite Links (ISLs) have generated a lot of interest, none of the Earth observation constellations today are equipped with ISLs. Hence we do not consider ISLs. Starlink demonstrated feasibility of laser-based ISLs [41, 57], and Planet and Telesat announced their plans to explore radio-frequency ISLs across orbits (e.g., from Low Earth Orbit to Middle Earth Orbit or Geostationary Orbits) [36]. Given the uncertainty over feasibility and type of ISLs, we choose to exclude them in our analysis. If ISLs mature in the future, `Serval`'s design can be generalized to accommodate and exploit ISLs.

3.2 Distributing Compute Across Earth and Space

Today's delay between image capture at satellite and image delivery to the cloud/end-user ranges from several hours to many days. To enable near-realtime insights for the end user(s), `Serval`'s primary goal is to reduce this delay to minutes for latency-sensitive queries, while accommodating the constraints imposed by the satellite's limited power and compute capabilities. Towards this goal, we first consider the placement of computation at the different compute units: satellite, ground station, and the cloud servers. There are two existing

approaches to perform inference on Earth imagery:

(i) In-order Delivery and Computation: In the traditional approach, images are delivered to the cloud in the same timestamp order as they were captured by the satellite, and then inference is run on these images in the cloud. This approach suffers from large delays because of networking bottlenecks. Specifically, the LEO satellites exhibit fast motion with respect to ground stations on the Earth [23, 61]. Therefore, any ground station-satellite contact is fleeting – less than ten minutes per contact, few contacts per satellite per day. This intermittent connectivity, in conjunction with the small number of ground stations, leads to large networking delays. Since images cannot be processed before they are received at the data center, the time-to-insight for this approach is very large.

(ii) In-orbit Computation: Recently, there is a growing push to place computation on the satellites. The first wave of this push demonstrated the use of satellite computation to reject cloud-occluded images [28]. Orbital edge computing [23] broadens the scope of satellite computing to reject images that do not meet the application goals, e.g., if the goal is to select images containing buildings, the satellite runs a building detector and rejects images that contain no buildings. This approach places a high computational load on satellites.

Specifically, such in-orbit computation techniques suffer from two problems: (a) as the number of applications increases or becomes more complex (such as the compositional filters discussed in Sec. 3.1), satellites cannot accommodate this compute with their limited resources, and (b) new applications emerge for historical data. Such applications may not be known *a priori*. If a satellite discards data that does not match current applications, new applications that emerge later cannot be served.

`Serval` takes the practical approach of distributing compute between *satellites and ground stations*. Our approach is centered around three properties of satellite imagery:

- **Image locations are predictable:** Satellites follow predictable orbital paths that can be estimated using their orbit descriptors, e.g., using Two Line Element (TLE) orbit descriptors published by observatories as well as many satellite operators. Therefore, even before an image is taken, we can predict what the geographical content of the image is.
- **Content of most images is glacial:** For most images taken at a given location, most of their content is glacial, i.e., stationary—it does not change rapidly within a few days. An image that contains buildings yesterday, will likely contain buildings today. Similarly, forests, desserts, farms, and other land types rarely change over the time span of a few days.
- **Ground station compute capabilities are rising:** Increasingly, ground stations are designed to include computational resources. These computers are more powerful and better resourced in terms of power and networking as compared to computational devices on the satellite.

Based on these observations, `Serval` divides the filters for all the queries into two types: (a) fast-changing dynamic filters that need to be executed on the satellite at run-time, and (b) glacial slow-changing filters that can be executed using stale imagery on the ground stations. `Serval` executes the glacial filters at the ground station or the cloud *before* the image is captured and uses the ground stations to communicate the results to the satellite in advance. This allows the satellite to process a very small fraction of images on the satellite in the bottleneck execution path.

Pre-computing such glacial information can significantly reduce the compute requirements on the satellite. As satellites orbit rapidly around the earth, their footprint looks like thin belts that extend almost vertically from the south pole to the north pole. Therefore, ordinary user requests, such as forests in California or farm land in the U.S., generally only makes up a tiny portion of a satellite's imagery.

`Serval` attaches a *dynamic* or *glacial* attribute to each filter to aid the execution engine. This can either be supplied by the developer, or estimated from data. Consider the query 'forest fires in California'. We find that among the millions of images captured by a constellation, only 0.4% contain any land area in California. Two-thirds of California images contain forests, i.e., only 0.25% of all images captured by the constellation contain 'forests in California'. Since we can estimate the exact set of images that will contain 'forests in California' using historical data, the satellite needs to run a fire detector on only this small set of images. In this case, *California* and *Forest* are glacial filters, and *Fire* is a dynamic filter. Only dynamic filters need to run on satellites using fresh data.

3.3 Incorporating Auxiliary Information Sources

Ground stations, unlike satellites, have continuous access to auxiliary information sources like weather forecasts. We ask if such information can be used by `Serval` to either improve inference quality or to skip computation on the satellite.

We first make the observation that clouds occlude many of the images taken by a satellite, and a cloud-occluded image is not useful for any query. In fact, there has been a lot of work in cloud detection for satellite imagery before [28, 35, 47, 53], both using statistical methods [35, 53] and neural networks [28, 47].

Given this observation, `Serval`'s key idea is to leverage weather forecasts in order to skip on-board processing of images over areas that are *forecast* to have high cloud cover. Concretely, if the probability of cloud cover is high, then `Serval` assigns a 'cloudy' tag to the image and exclude it from any other computation and de-prioritize its transfer to the ground station. Conversely if the probability of cloud cover is very low, then `Serval` assigns a 'cloud-free' tag to the image. In this case, the processing pipeline on the satellite can skip

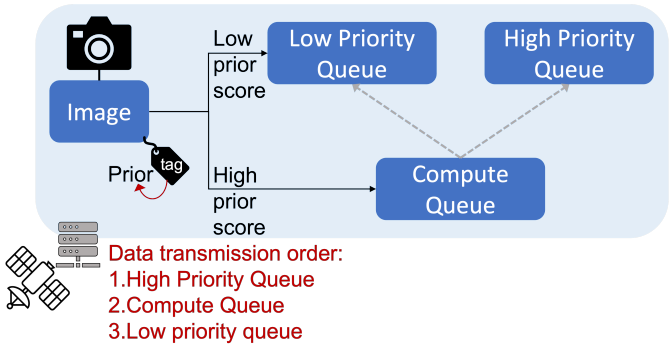


Figure 4: **Serval's satellite execution engine re-orders the images to prioritize latency-sensitive data download.**

the *Cloud* filter and proceed to the next steps of computation. Finally, all other images that are not tagged in either of the above ways are processed through the *Cloud* filter on-board the satellite.

In practice, we use cloud probability thresholds of $\{0.2, 0.8\}$ for the above processing. If the cloud cover probability for an image is less than 0.2, `Serval` assumes the image to be cloud-free. If the probability is greater than 0.8, `Serval` de-prioritizes the image. If the probability is between these two values, `Serval` processes the image through the *Cloud* filter, which is a neural network model.

While the current version of `Serval` does not incorporate weather forecasts in other inferences, there are many other potential optimizations for future work. For applications like forest fire, weather forecasts also contain information regarding forest fire risks, which can be used to select high-likelihood images to run filters on. Also, cloud detectors are known to confuse smoke and cloud which can be improved by combining cloud detection results and weather forecasts. Finally, auxiliary information may contain other types of information such as information extracted from previous satellite images, e.g., if a different satellite observed forest fire in the same region a few hours ago.

3.4 `Serval`'s Execution Engine

We describe how `Serval` captures the insights discussed in Sections 3.1-3.3. We discuss how `Serval`'s execution engine works on a satellite, at a ground station, and in the cloud.

3.4.1 The Satellites

Fig. 4 shows that each satellite maintains three queues for images: (i) a low priority queue, (ii) a high priority queue, and (iii) a compute queue. The low priority queue contains images that do not match any latency-sensitive query. The high priority queue contains images that definitely match latency-sensitive queries. The compute queue contains images that need more computation to be performed on the satellite in order to ascertain their status.

Network delivery order: When the satellite comes into contact with the ground station, it will first downlink images in the high priority queue. If the high priority queue is emptied and ground station contact remains, it will downlink the images in compute queue. This is because even though computation hasn't run on these images on the satellite, these images have a higher likelihood of being latency-sensitive than images in the low-priority queue. Finally, if the compute queue is emptied and ground station contact remains, it will downlink images from the low priority queue. This multi-queue architecture allows `Serval` to benefit from limited compute available on satellites, even when it may not be able to compute on all images.

Image placement in queues: When a satellite comes in contact with a ground station, the ground station sends the following information to the satellite: (a) pre-computed values for *glacial* or slow-moving filters for each image that the satellite is expected to capture in its upcoming orbital path leveraging the observation from Sec. 3.2 that satellite orbits are predictable, and (b) weather predictions for the geographical location corresponding to each image the satellite is expected to capture. For each image captured by a satellite, `Serval`'s first goal is to identify whether the image meets the requirements for *any* (at least one) of the on-board queries. `Serval` does so by immediately applying all pre-computed filters. For a given query, if any of its pre-computed filters rejects the image, the image is considered rejected by that query. If an image is rejected by all queries, then the image is placed in the low priority queue. Note that in practice this entire filtering process is fast as it does not involve significant compute but is a quick classifier based on pre-computed inferences received from the ground station.

To be placed in the compute queue, an image must meet two criteria: (a) all the pre-computed glacial filters in at least one query must *select* that image, and (b) there must be at least one dynamic filter in that query that needs computation. For example, for 'forest fires in California', if an image is selected by the *California* and *Forest* filters, it is placed in the compute queue to run the *Cloud* and *Fire* filters.

Any image that is selected by *all* the filters in at least one query is placed by `Serval` into the high priority queue. If a query has some dynamic filters, then an image moves to the high-priority queue from the compute queue. For example, in the example above, if the image in the compute queue is selected by both *Cloud* and *Fire* filters, it will move to the high priority queue. Note that, if all the filters in a query are glacial filter (precomputed on the ground station), then, some images can skip the compute queue and directly move to the high priority queue as well.

Finally, if the output of a query is an inference and not an image (e.g., *Vessel Count*), then `Serval` places the image in the low priority queue and creates an empty image with just

the metadata information (e.g., number of cars in parking lots or number of vessels at a port). Serval places this (much) smaller metadata value in the high priority queue.

In situations where the satellite has sufficient surplus energy to compute on an image, Serval pulls an image from the compute queue and runs the dynamic filters corresponding to the query on the image. At any time, if the satellite determines that an image is not possibly high priority in the multi-stage filtering process, it will immediately put it into the low priority queue and start doing computation on the next image. On the other hand, if the satellite determines that an image is high-priority, it will put the image into the high priority queue without running additional filters on that image. In each queue, images are ordered by their capture timestamp.

3.4.2 The Ground Stations

Recall that ground stations have much higher computation capability than the satellites, while having much fewer power constraints. In Serval, the ground station maintains two queues: a high priority queue and a low priority queue. When a ground station receives an image from the satellite, it first runs the uncompleted filters of the query and the places the image in the high priority if the image is selected by all the filters in at least one query, otherwise it places it in the low priority queue. Whenever the ground station has free computational resources, Serval opportunistically uses the downtime to compute the results of 'glacial' filters for upcoming satellite contacts. A ground station has a steady backhaul connection to the cloud, and uses this backhaul to constantly stream images to the cloud starting from the high-priority queue.

3.4.3 The Cloud

The cloud acts as a frontend to all users. The images and insights will be uploaded to the cloud and made available for users to download. The cloud also acts as the central coordinator of the entire Serval system and computes the future orbits, positions of satellites and other preemptively computed values for the images taken in the future. The cloud will also schedules satellite-ground station contacts and sends the pre-computed values to the appropriate ground station to be relayed to the satellites in the control plane. Fig. 5 demonstrates Serval's placement of different filters for the example queries in Fig. 2.

4 Experimental Setup

We evaluate Serval using a combination of trace-driven simulations and emulations. Our code is open source and available at https://github.com/ConnectedSystemsLab/Serval.

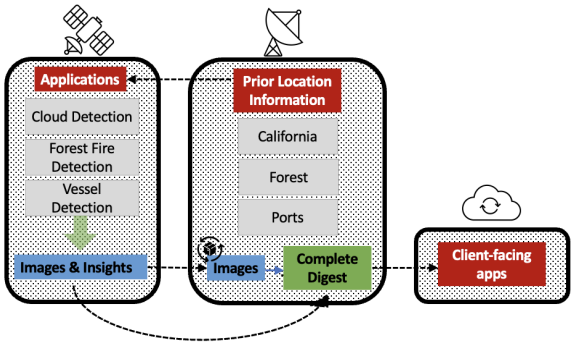


Figure 5: **Serval Example Execution: The placement of different compute units for queries defined in Fig. 2.**

4.1 Applications

We picked two sample user applications to evaluate Serval's performance: (a) identification of all *Forest Fire images in California*, and (b) *Counting Marine Vessels in Ports* in the world's busiest ports, including Shanghai, Singapore, Hong Kong, Hamad, and Jebel Ali. These applications cover vast geographical areas, and yet they are different in output (images vs. count) and in their internal query filters, ML models, and resultant computational needs.

Fig. 2 showed that the California Forest Fire application is a sequence of four filter stages:
1. *Is the Image from California?* A geographic filter (Glacial filter);
2. *Is the Image in a woody zone?* Computed from stale data (Glacial filter);
3. *Is the Image cloud-free?* (Dynamic filter); and
4. *Does the Image contain smoke or haze from a forest fire?* (Dynamic filter).

Using Planet images, we trained three deep-learning models for the identification of the forest, fire, and cloud. The forest and fire identification is based on a ResNet architecture [31] while the cloud identification is based on a MobileNet architecture [33]. The label for woody/non-woody is obtained from the USDA Forest Service; the labels for cloud and non-cloud are obtained from Planet UDM2; the fire labels are manually created. We divided our data into a training set and a testing set.

Our second application, Marine Vessel in Ports Counting, is a sequence of three filter stages:
1. *Is the Image near one of the focus port areas?* A geographic filter (Glacial filter);
2. *Is the Image cloud-free?* (Dynamic filter); and
3. *Counting the number of vessels in the image.* (Dynamic filter).

We trained a ResNet-based [31] vessel classifier using a Kaggle planet imagery dataset[2]. This dataset did not contain bounding boxes or vessel counts as training labels; it just contained vessel classification labels. Therefore, we used

[2]https://www.kaggle.com/datasets/rhammell/ships-in-satellite-imagery

the trained classifier and a weakly supervised object detection [58] (WSOD) to draw object bounding boxes on sample images taken by Planet. We then trained a yolov5 [37] model using our automatically labeled training images. We divided our data into a training set and a testing set.

Obtaining historical data for forests: To simulate extracting forest data from stale images, we initially run the forest model over images collected from the first 10 days. We, then, evaluated `Serval` over the last 10 days of the trace we obtained. To label each image in the later 10 days, we find all images in the first 10 days that intersect with the image, and if any such image is labeled as forest, we determine that the target image is forest.

Weather data: We obtained weather forecast data from openmeteo for all the images [2]. For each image that requires cloud detection, if the cloud coverage is more than 80%, we determine that it is cloudy without running the cloud detection. Similarly, if cloud coverage is less than 20% we determine that it is not cloudy without running the model. If the cloud coverage is between 20% and 80%, we run the cloud detection model.

4.2 Real-world Dataset

We obtained a large image dataset from Planet's Dove satellite constellation [50], comprising nearly 200 satellites. This dataset contains the metadata (size, location, etc.) for all images captured by all 200+ satellites between July 1st and July 20th, 2021. We use this time interval because of the prevalence of forest fires in California during the summer of 2021. This dataset contains metadata for ten million images. Due to the limited quota of images we can download, we analyzed the metadata to find images in California and multiple port areas. Based on our analysis of the metadata, we requested around 40k complete images from Planet's Planetscope API [51]. Each image contains 4 channels (red, green, blue, NIR). The collective size of our downloaded images is 13 TB.

Planet has 3 different types of sensors to collect images: PS2, which covers an area of 24km by 8km; PS2.SD, which covers an area of 24km by 16km; and PSB.SD, which covers an area of 32.5km by 19.6km. The runtime of a neural network largely depends on the image size and not the content of the image. Therefore, for the images that we only downloaded the metadata, we used the average running time of the models as an assumed running time for the corresponding model on those images, when necessary.

4.3 Hardware-benchmarking and Simulator Design

First, we use real hardware to benchmark the performance of different filters, such as the Machine Learning models for various filters. Table 1 shows that we evaluated `Serval` using **two different modes of on-board computers for satellites**: Jetson ORIN 30W and 15W modes. The microcomputer is equipped with a GPU that is able to run neural network based models in the images.

Mode	15W	30W
CPU Cores	4x1.1MHz	8x1.7MHz
GPU Speed	420 MHz	624 MHz
RAM	32 GB	32 GB
Storage	64 GB	64 GB

Table 1: **Jetson AGX Orin Hardware Configurations**

We tested the performance of `Serval` at scale in simulation. Our simulator computes the orbits of satellites using Two-Line-Element [32] orbit descriptors for each Planet satellite. The simulator tracks satellite-ground station contacts and the link quality of each of these contacts by following the specifications of Planet's ground stations in [25]. The simulator also keeps track of the energy generation and consumption on each satellite, as well as the compute time on it. We ran the simulation with a time granularity of $\Delta t = 1min$. For each satellite, we assume that it is equipped with the Jetson AGX Orin and operates in one of its two power modes.

Power management: Our simulator models the satellite power profile in Table 2. The satellite generates power using a solar panel. The satellite's ADACS (Advanced Data Acquisition and Control System) and other systems always need to be on and consume continuous power. We select these numbers to match previously reported numbers [23]. When the satellite takes photos or transmit power, it will also cost energy. We schedule the computation in a greedy approach: whenever there is spare power available and images requiring computation, the satellite will run the next-queued computation (if any).

Resource limit: We run two applications that target a small fraction of the Earth's surface area. In practice, many more applications may run on satellite imagery and cover a larger geographical area. Therefore, for fairness, we limit network and compute usage for `Serval` to 1% of the total network and compute capacity available on the satellite (roughly proportional to the surface area of the geography we cater to as compared to all the area imaged by Planet's satellites). Note that high-priority images identified by `Serval` are bottlenecked by this network restriction, other images (e.g., from other locations) can still use the full network.

Ground stations: We evaluated `Serval`'s performance **under two different ground station configurations**: (a) a state-of-the-art distributed ground station architecture [61] where we placed 200 ground stations across the globe; and (b) a traditional monolithic ground station configuration from Planet's current ground station system [20,24]. We model approximate

Component	Power
ADACS	1.13W
Computation	Depends on hardware
Camera	6J per photo
Transmission	50 W

Table 2: **Power consumption of the satellites**

Application 1		Application 2	
Filter	Amount	Filter	Amount
Total	**10097097**	**Total**	**10097097**
California	37037	Port	2642
Forest	26153	Cloud-less	1769
Cloud-less	24578	Vessel	1769
Fire	243		

Table 3: **Number of images in each stage of the pipeline.**

locations of Planet's ground stations from publicly accessible information. The ground station system in our simulation consists of 12 different ground stations distributed across the globe, carrying 48 antennas in total, where each antenna can talk to a single satellite at a time. We assume that the ground stations have a steady backhaul Internet connection that is sufficient to transmit everything it receives to the cloud.

We used the link quality model described by Planet in [25] for traditional ground station setup. For distributed ground station network, we scaled down the downlink data rate for each ground station to 25%—this makes the total downlink bandwidth constant across the two scenarios (as done in [61]).

5 Microbenchmarks

5.1 Number of Final Images $\ll$ Number of Collected Images

Table 3 quantifies the proportion of latency-sensitive images across different days. For the California forest fire application, just 0.36% of the images pass the geographic filter (*California*) and just 0.26% of the images pass the *Forest* filter. The number of actual fire images is just 243. Therefore, in `Serval`, the satellite needs to analyze at most 26k images and select 243 latency-sensitive images to download. For the vessel in ports application, we find 1769 images containing vessels at ports. These small final image counts indicate the potential latency benefits from transferring only a small (but just right!) set of images from the satellite down to the cloud.

5.2 Preemptive Compute at Ground Station

We estimate the burden on the ground station to run the required preemptive computation task by profiling it on a large cluster of computation nodes. The results show that on one

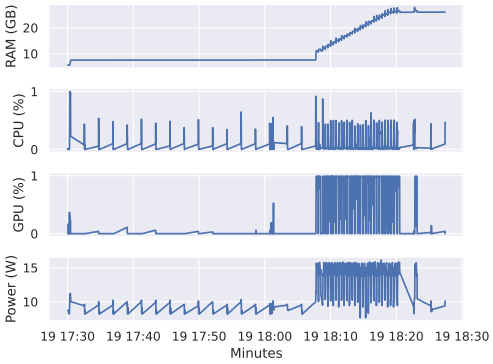


Figure 6: **Resource utilization over time for satellite "103b" passing over California**

data-center-grade GPU such as NVIDIA Tesla K80, it takes an average of 4.39s to run the forest detector on one image taken by the satellite. Running a forest detector on all California images across ten days takes no more than 24 GPU hours. This cost is distributed across multiple ground stations (at least 12 stations in our setup) and due to the station's higher power, this is preferable to running the same compute on the satellites. Further, because such data is glacial, this station runtime (O(hours)) suffices to update the satellite's glacial filters and auxiliary information once a day.

5.3 Hardware Emulation

We tested the system load of a typical satellite under `Serval` via hardware emulation. We emulate the satellite with a Raspberry Pi serving as an onboard control system, connected to a Jetson ORIN serving as the computation system. The Raspberry Pi "captures" images and sends them to the Jetson for running on-board computation. After receiving results from the Raspberry Pi, the Jetson will execute `Serval`'s execution engine (Section 3). The Raspberry Pi will send images over a TCP connection for emulating the satellite-ground station link, to a dedicated server. We emulated the time period from 17:30 on July 19, 2021, to 18:30 on July 19, 2021: when satellite "103b" flies over California and does on-board computation on the interesting images.

Figure 6 shows the variation of system load over time. We observe that before 18:05, the Jetson periodically wakes up to do heartbeat communication with the controller (Raspberry Pi) and consumes low energy. When the satellite starts to pass over California, we see a significant increase in all types of resource usage on the Jetson machine, especially GPU usage, because the applications require running neural networks on the GPU. After the satellite passes California, the CPU, GPU utilization, and power consumption drop to normal levels. We use the numbers obtained from these emulations to benchmark different filters in our simulated at-scale evaluation below.

We trained different deep learning models (as described above) to perform tasks of cloud detection, forest detection,

vessel detection, and forest fire detection. Table 4 presents the average run time of each filter on the Jetson device. Specifically, we report the time taken to classify one image using the filter.

Model	15W	30W
Cloud	1.5	3
Forest	17	8.5
Fire	0.246	0.245
Vessel	3.85	1.846

Table 4: **Average runtime of filters on Jetson machine (seconds)**

6 End-to-End Results

6.1 End-to-end Performance

We evaluate the end-to-end latency for all the latency-sensitve imagery that matches our two applications. We define the latency as the delay between when the corresponding image is generated and when the information is delivered to the cloud and available for the user to download. For the vessel counting application, just the vessel count needs to be delivered to the user. Fig. 7 shows the distribution of the end-to-end latency achieved by `Serval` compared to in-order delivery. We plot this result for two hardware configurations of the AGX ORIN – 15W power consumption and 30W power consumption, to benchmark `Serval`'s benefits across different types of computational platforms. AGX ORIN can operate in these power modes and offers less computational capability in the lower power mode.

Using the 15W power mode, and a monolithic ground station architecture, `Serval` reduces the median latency by 70× from 78.2 hours to 1.1 hours, and 90-th percentile from 145.55 hours to 2.71 hours. For `Serval`, a large part of this latency stems from the fact that even after high-priority images have been identified, they must wait for the first ground station contact. Distributed ground stations (DGS) [7, 46, 61] have recently been designed to provide more opportunities for data download. In the DGS case, `Serval` can download images with a median latency of 0.03 hours, compared to 71.71 hours median latency for in-order delivery. Even the 90th percentile for `Serval` is 0.78 hours, compared to 149.05 hours for in-order delivery. This result shows that even with simple compute capabilities on the satellite, `Serval` can achieve near-realtime delivery of latency-sensitive insights.

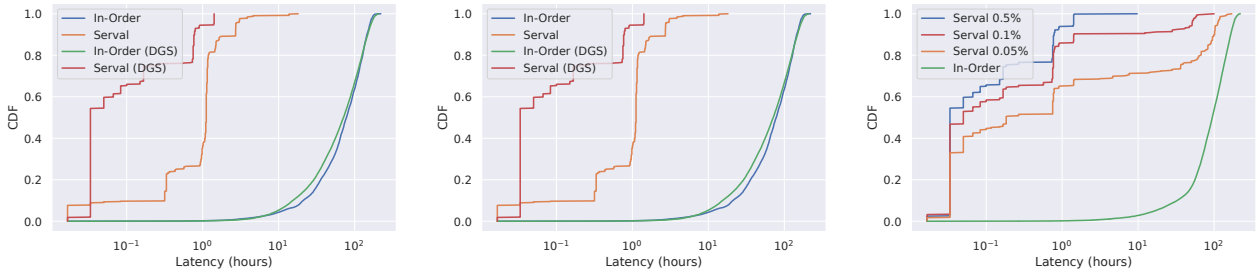
Next, we compare results for the high-power 30W mode on Jetson AGX ORIN deployed on the satellite. In this case, the delays for the baseline do not change because it includes no compute. For `Serval`, the median delay is 1.1 hours (90-th percentile – 2.7 hours) and 0.03 hours (90-th percentile – 0.78 hours) for monolithic ground stations and DGS respectively. Note that the median delay for monolithic stations does not change. This validates our hypothesis that the delay stems from the sparsity of ground station locations. When combined with DGS, `Serval` can achieve a median latency of few minutes. Unless noted otherwise, the evaluation below is performed using the 30W, DGS, 1% resource-constrained setting.

Impact of scaling up applications: Recall that, for the evaluation below, we limit the `Serval`'s resource usage for high-priority images to 1%. We would like to see how having more or less applications would impact the performance of `Serval`, which is equivalent to changing the resource usage threshold. We tested resource limit values of 0.05%, 0.1% and 0.5%. Figure 7c shows that `Serval`'s performance remains steady even when the resource limit is reduced to 0.5% from 1%. Its performance starts to decrease when the resource limit is reduced to 0.1%, and the computation starts to emerge as a bottleneck. As discussed in our future work (Sec. 8), such bottlenecks can be overcome by architectural optimizations.

Effect of running computation on satellites: To evaluate the effect of running compute on satellite, we compared the performance of `Serval` against the case when satellites will do no computation tasks on board but only prioritize all images based on glacial filters alone. The results are illustrated in Figure 8a. By running compute on the satellite, `Serval` was able to reduce the median latency by 23×.

One might wonder why filtering based on glacial filters is insufficient since the number of California forest images is small. This is because the high-priority traffic arrives in a bursty manner: when a satellite is over an area of interest, it continuously captures images that need to run the computation. When a satellite is not over an area of interest, it does not require computation. This is true for all applications whose images are not evenly distributed across the globe. Therefore, when a satellite is in contact with a ground station, the proportion of images in transmission queues with high "pre-computed" scores could be much larger than the average during a satellite-ground station contact. On the other hand, we know that for instance, only 1% of images are forest fire in all California forest images. Therefore, running computation on the satellite can help filter a significant chunk of images that pass the glacial filter pre-computed on the ground.

Benefit of using auxiliary information: To validate our decision to use side-channel information for cloud detection, we first checked the accuracy of weather information. The results are shown in Table 5. In this experiment, we only run the cloud detector on images that have a cloud coverage value between 20% and 80%. From the table, we can see that this method saves 83.6% computation on cloud detection while yielding an accuracy of 96.3% and recall of 99.3%. Indeed, we can precisely estimate the cloudiness of a majority of images without sacrificing accuracy, saving a large amount of compute.

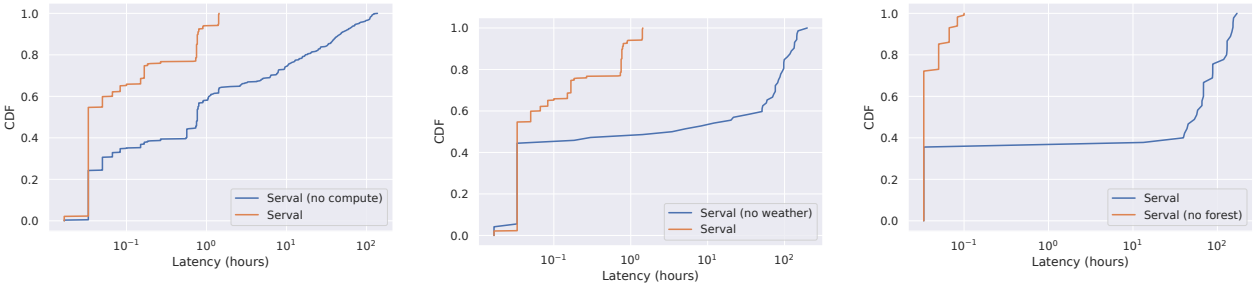


(a) Latency with Jetson ORIN in 15W mode

(b) Latency with Jetson ORIN in 30W mode

(c) `Serval`'s performance with varying resource constraints

Figure 7: **End-to-end latency of `Serval` compared to in-order baseline.** `Serval` can achieve a median end-to-end latency of 0.1 hours when using distributed ground stations. `Serval`'s performance is stable across different hardware constraints.



(a) Effect of running computation on satellites.

(b) Effect of weather information.

(c) Effect of using historical data on forest (California forest fire).

Figure 8: **Effect of different components in `Serval`. (Ground station setup is DGS)**

Prediction	Cloudy (>0.5)	Not Cloudy (<0.5)
High (>0.8)	251	226
Medium (0.2-0.8)	1668	4873
Low (<0.2)	1245	31750

Table 5: **Accuracy of weather information. Each row indicates the weather forecast, and each column indicates the output of the *Cloud* filter.**

We evaluated the weather information's contribution to the end-to-end performance by comparing `Serval` against when the side channel information is disabled. The results are shown in Figure 8b. We can see that by employing weather information, `Serval` improves the median latency by 8.8×.

Benefit of using historical data: Does pre-computation of glacial filters on the ground station have an advantage? To test this hypothesis, we move the *Forest* filter to the satellite and test if this move hurts the latency. For this experiment, we consider a single application: "California forest fire". The comparison is shown in Figure 8c. We can see that the median latency increased by 1670× when we don't use historical data. We observe that there are two reasons for this large performance drop: (a) running forest model on satellites consumes a great amount of computation time (the number of images is larger, and each image requires more computational resources) and (b) since the model is being run in real-time, some of the fire images do not look like forest for the neural network, because of the presence of the smoke. Therefore, these images get misclassified as 'not forest' and placed in the low priority queue. However, `Serval` considers stale data for such analysis when it places the glacial filter execution on ground stations. Such images are not occluded by forest. The second benefit is an unintended consequence of running glacial filters on ground stations.

Comparison to early discard : As an extension to the above experiment, we compared `Serval`'s performance against an early discard scheme inspired by OEC [23]. OEC relies solely on computation on satellites and discards all images deemed low priority. Similar to the above experiment, since OEC does not use historical data or ground station computation, it incorrectly discards some images that contain forest fire. In fact, OEC only successfully identified and downlinked 14.7% of all images with forest fire. Since it discards all "low priority" images on the satellite, the "false negative" images do not even have a chance to reach the user. The "false negative" images appear in the "long tail" distribution in Figure 8c. In contrast, `Serval`'s classifier makes fewer mistakes because

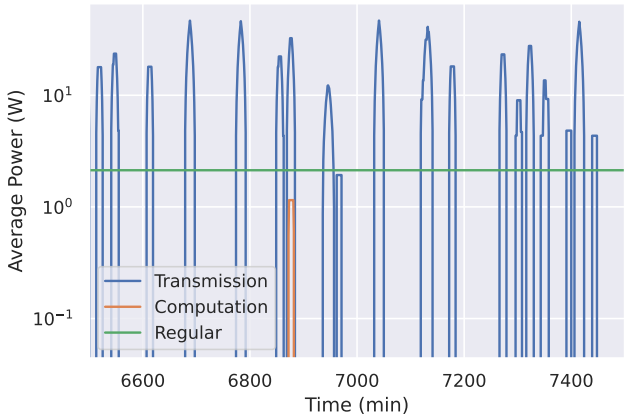


Figure 9: **Average Power over time for different applications**

of its reliance on historical data. Even when Serval makes a mistake, it just adds additional latency to that image rather than discarding that image. For the vessel counting application, both OEC and Serval are able to downlink all images containing vessels primarily because historical data doesn't help with the classification strategy.

6.2 Satellite Power Usage

We monitored the power usage for different functions during the simulation period. The energy cost consists of 3 main parts: regular power (ADACS and other essential functions to keep the satellite alive), transmission power and compute power. Figure 9 illustrates a sample satellite's ("Dove 103b") power consumption profile during a period in which it flys over California. We observed that while the satellite is constantly consuming power for regular functions and from time to time for transmission, the compute function is only activated when the satellite receives some potential high-priority image. We can see that the compute power consumption is much more sparse than either transmission or regular maintenance. In our simulation, we saw 68.1% energy being consumed by transmission, 28.9% consumed by regular maintenance, and 2.9% consumed by computation. Because of our limit on resource utilization for the high-priority applications, we only used 1% of the 2.9% total compute energy, and the rest of the energy was reserved for other computation tasks (e.g., for other tenant applications, satellite maneuvering, etc.).

7 Related Work

Our work builds on results in terrestrial edge computing, video processing systems, and orbital edge computing.

Terrestrial edge computing pipelines: Edge computing has been a widely studied topic in terrestrial networks [8,13,17,29, 54, 55] for diverse applications such as traffic camera analytics [8, 12], augmented reality systems [56], and robotics [63]. These systems tend to push computation as close to the video sensor as possible while being cognizant of the resource constraints of the edge devices. Serval naturally builds on this line of research. Satellites are similar to resource-constrained mobile devices with relatively weak connectivity to the cloud. The LEO setting presents the additional challenge that the sensor devices (satellites) themselves are moving. Serval addresses this by leveraging the predictable orbital paths of satellites, query filters and bifurcation, and using auxiliary information.

Video processing systems: Much recent work [6,8,13,39,48, 65] has focused on improving the execution of video analytics pipelines on edge devices. These systems consider different aspects of optimizing video analytics such as efficient model retraining [13], model merging for efficient execution on edge GPUs [48], etc. This line of work makes varying assumptions about the availability of compute resources such as powerful GPUs and continuous connectivity with the cloud—these luxuries are not available on satellites. Nevertheless, ideas from video processing systems are complementary to Serval, and our work opens an avenue to explore such future directions. For instance, model merging, an idea from video processing systems, can be useful if the models (inside the filters of different Serval queries) share a lot of common layers.

Satellite edge computing: Traditional satellites packaged radiation-hardened specialized hardware [10]. Due to the lower orbits of LEOSats which suffer much less radiation exposure, and for economies of scale in manufacturing, there has been a move towards general-purpose hardware for LEOSats [28]. This has blossomed research in satellite edge computing systems [15, 16, 22, 23, 43–45]. Some papers propose new edge-enabled functionalities for communication megaconstellations [15, 16], e.g., deploying content delivery networks in space to improve network performance. This work is independent of Serval due to its focus on a different class of satellites. [45] and [44] evaluate the performance of compression techniques and Machine learning models on satellite-compatible hardware such as NVIDIA Jetson Nano and/or NVIDIA Jetson AGX. We believe such optimizations demonstrate the feasibility of deploying Machine Learning workloads on satellites, and such architectural optimizations (including from other domains like approximate computing) can be applied orthogonally to Serval to improve performance.

Our work is closest to Orbital Edge Computing (OEC [23]) and Kodan [22]. Both OEC and Kodan aim to reduce the amount of satellite imagery transferred to Earth by enabling early rejection of imagery that is not considered useful. While OEC reimagines different satellites in a constellation as a computational nanosatellite pipelines that seamlessly distribute

computational tasks, Kodan focuses more on the computation at each satellite. In contrast to both these works, Serval does not discard any images on the satellite, and focuses on reducing the delay of latency-sensitive images via prioritization and reordering. This has the advantage that *post facto* queries on historical can be performed, e.g., in a recent high-profile incident, Planet used historical satellite imagery to trace the historical motion of a Chinese balloon entering into the US airspace from its origin to destination [5]. Kodan's contribution is orthorgonal to ours, since Kodan aims to train the optimal neural network model for specific user applications, while we focus on how to optimally schedule compute on satellites and ground stations given a fixed neural network. Hence Kodan can be used alongside Serval.

8 Concluding Discussion

We build Serval, a distributed computation framework for near-realtime insights from Earth imagery satellites. Serval can deliver latency sensitive imagery such as forest fire imagery in minutes as opposed to hours or days of delay for traditional in-order delivery systems. We conclude by listing some possible extensions of Serval in future work:

- **Multi-modal imagery:** Serval currently focuses on RGB imagery captured by satellites. Increasingly, satellites capture other forms of imagery such as radar, hyperspectral, and multi-spectral. We believe Serval can naturally extend to support these emerging image types.
- **Merging filters across queries:** As the number of applications scales, there are more opportunities to reduce redundant compute within and across queries. Multiple queries may share filters to reduce compute on satellites. Moreover, multiple neural network models may share weights for a subset of the layers and present opportunities for model merging techniques like [48] to optimize compute on the satellite.
- **Architectural optimizations:** We did not consider architectural optimizations such as model pruning or precision drop to pack more compute on the limited satellite resources. Such techniques can further improve Serval's performance.

Acknowledgments

We are grateful to anonymous reviewers and our shepherd Lixia Zhang for feedback on this work. This work was partially sponsored by NSF Award CNS-2237474, by NSF Award CNS-1908888, and by generous contributions from Cisco and Microsoft. We are grateful to Kiruthika Devaraj from Planet Inc. for discussions and feedback on early ideas. Access to Planetscope imagery was provided by the UIUC library's contract with Planet Inc.

References

[1] Analytics Overview. https://developers.planet.com/docs/analytics/.

[2] Open-meteo: Open-source weather api. https://open-meteo.com/.

[3] Spire Global Inc. https://spire.com/.

[4] Living on the edge: Satellites adopt powerful computers. https://spacenews.com/living-on-the-edge-satellites-adopt-powerful-computers/, 2022.

[5] One more way ai can help us harness one of the most underutilized datasets in the world. https://www.planet.com/pulse/one-more-way-ai-can-help-us-harness-one-of-the-most-underutilized-datasets-in-the-world, 2023.

[6] Neil Agarwal and Ravi Netravali. Boggart: Towards General-Purpose acceleration of retrospective video analytics. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 933–951, Boston, MA, April 2023. USENIX Association.

[7] Amazon Inc. AWS Ground Station . https://aws.amazon.com/ground-station/.

[8] Ganesh Ananthanarayanan, Victor Bahl, Landon Cox, Alex Crown, Shadi Nogbahi, and Yuanchao Shu. Video analytics-killer app for edge computing. In *Proceedings of the 17th annual international conference on mobile systems, applications, and services*, pages 695–696, 2019.

[9] Bruno Aragon, Rasmus Houborg, Kevin Tu, Joshua B. Fisher, and Matthew McCabe. CubeSats Enable High Spatiotemporal Retrievals of Crop-Water Use for Precision Agriculture. *Remote Sensing*, 2018.

[10] William Bamford, Luke Winternitz, and Curtis Hay. Gps world, innovation: Autonomous navigation at high earth orbits. 2005.

[11] Panagiotis Barmpoutis, Periklis Papaioannou, Kosmas Dimitropoulos, and Nikos Grammalidis. A review on early forest fire detection systems using optical remote sensing. *IEEE Sensors*, 2020.

[12] Johan Barthélemy, Nicolas Verstaevel, Hugh Forehead, and Pascal Perez. Edge-computing video analytics for real-time traffic monitoring in a smart city. *Sensors*, 19(9):2048, 2019.

[13] Romil Bhardwaj, Zhengxu Xia, Ganesh Ananthanarayanan, Junchen Jiang, Yuanchao Shu, Nikolaos Karianakis, Kevin Hsieh, Paramvir Bahl, and Ion Stoica. Ekya: Continuous learning of video analytics models on edge compute servers. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 119–135, Renton, WA, April 2022. USENIX Association.

[14] A. Bhattachan, N. Skaff, S. Vimal, J. Remais, and D. P. Lettenmaier. Using geospatial datasets to characterize mosquito larval habitats in the Los Angeles Basin. In *AGU Fall Meeting Abstracts*, 2019.

[15] Debopam Bhattacherjee, Simon Kassing, Melissa Licciardello, and Ankit Singla. In-orbit computing: An outlandish thought experiment? In *Proceedings of the 19th ACM Workshop on Hot Topics in Networks*, HotNets '20, 2020.

[16] Vaibhav Bhosale, Ketan Bhardwaj, and Ada Gavrilovska. Toward loosely coupled orchestration for the LEO satellite edge. In *3rd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 20)*. USENIX Association, June 2020.

[17] Christopher Canel, Thomas Kim, Giulio Zhou, Conglong Li, Hyeontaek Lim, David G Andersen, Michael Kaminsky, and Subramanya Dulloor. Scaling video analytics on constrained edge nodes. *Proceedings of Machine Learning and Systems*, 1:406–417, 2019.

[18] Kejie Chen, Jean-Philippe Avouac, Saif Aati, Chris Milliner, Fu Zheng, and Chuang Shi. Cascading and pulse-like ruptures during the 2019 ridgecrest earthquakes in the eastern california shear zone. *Nature Communications*, 2020.

[19] M. Chu, D. Drynan, and L. R. Benning. Integrating satellite links into a land-based packet network. *ACM SIGCOMM Comput. Commun. Rev.*, 1985.

[20] Kyle Colton, Joseph Breu, Bryan Klofas, Sydney Marler, Chad Norgan, and Matthew Waldram. Merging Diverse Architectures for Multi-Mission Support. In *Small Satellite Conference*, 2020.

[21] Olivier L. de Weck, Richard de Neufville, and Mathieu Chaize. Staged deployment of communications satellite constellations in low earth orbit. *Journal of Aerospace Computing, Information, and Communication*, 2004.

[22] Bradley Denby, Krishna Chintalapudi, Ranveer Chandra, Brandon Lucia, and Shadi Noghabi. Kodan: Addressing the computational bottleneck in space. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, ASPLOS 2023, New York, NY, USA, 2023. Association for Computing Machinery.

[23] Bradley Denby and Brandon Lucia. Orbital edge computing: Nanosatellite constellations as a new class of computer system. In *ACM ASPLOS*, 2020.

[24] Kiruthika Devaraj, Ryan Kingsbury, Matt Ligon, Joseph Breu, Vivek Vittaldev, Bryan Klofas, Patrick Yeon, and Kyle Colton. Dove High Speed Downlink System. In *Small Satellite Conference*, 2017.

[25] Kiruthika Devaraj, Matt Ligon, Eric Blossom, Joseph Breu, Bryan Klofas, Kyle Colton, and Ryan Kingsbury. Planet High Speed Radio: Crossing Gbps from a 3U Cubesat. In *Small Satellite Conference*, 2019.

[26] Anna Escher. Inside Planet Labs' new satellite manufacturing site. TechCrunch. https://techcrunch.com/2018/09/14/inside-planet-labs-new-satellite-manufacturing-site/, 2018.

[27] C.E. Fossa, R.A. Raines, G.H. Gunsch, and M.A. Temple. An overview of the iridium (r) low earth orbit (leo) satellite system. In *IEEE National Aerospace and Electronics Conference*, 1998.

[28] Gianluca Giuffrida, Luca Fanucci, Gabriele Meoni, Matej Batič, Léonie Buckley, Aubrey Dunne, Chris van Dijk, Marco Esposito, John Hefele, Nathan Vercruyssen, Gianluca Furano, Massimiliano Pastena, and Josef Aschbacher. The -sat-1 mission: The first on-board deep neural network demonstrator for satellite earth observation. *IEEE Transactions on Geoscience and Remote Sensing*, 2022.

[29] Peizhen Guo, Bo Hu, and Wenjun Hu. Mistify: Automating DNN model porting for On-Device inference at the edge. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, pages 705–719. USENIX Association, April 2021.

[30] JOHN HANSON, MARIA EVANS, and RONALD TURNER. *Designing good partial coverage satellite constellations*. Astrodynamics Conference. 1990.

[31] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[32] Felix R. Hoots and Ronald L. Roehrich. Models for Propagation of NORAD Element Sets. Technical report, Aerospace Defense Command, United States Airforce, 1980.

[33] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications, 2017.

[34] Y.C. Hubbel. A comparison of the iridium and amps systems. *IEEE Network*, 1997.

[35] Gary J. Jedlovec, Stephanie L. Haines, and Frank J. LaFontaine. Spatial and temporal varying thresholds for cloud detection in goes imagery. *IEEE Transactions on Geoscience and Remote Sensing*, 2008.

[36] Rachel Jewett. Planet to support nasa relay networks for telesat, ses. Satellite Today. https://www.satellitetoday.com/government-military/2022/08/23/planet-to-support-nasa-relay-networks-for-telesat-ses/.

[37] Glenn Jocher, Alex Stoken, Jirka Borovec, Ayush Chaurasia, Liu Changyu, Adam Hogan, Jan Hajek, Laurentiu Diaconu, Yonghye Kwon, Yann Defretin, et al. ultralytics/yolov5: v5. 0-yolov5-p6 1280 models, aws, supervise. ly and youtube integrations. *Zenodo*, 2021.

[38] H. Keller, H. Salzwedel, G. Schorcht, and V. Zerbe. Comparison of the probability of visibility of the most important currently projected mobile satellite systems. In *IEEE Vehicular Technology Conference. Technology in Motion*, 1997.

[39] Mehrdad Khani, Ganesh Ananthanarayanan, Kevin Hsieh, Junchen Jiang, Ravi Netravali, Yuanchao Shu, Mohammad Alizadeh, and Victor Bahl. RECL: Responsive Resource-Efficient continuous learning for video analytics. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 917–932, Boston, MA, April 2023. USENIX Association.

[40] J. Kong, Y. Ryu, R. Houborg, and M. Kang. Monitoring canopy photosynthesis in high spatial and temporal resolution using CubeSat imagery. In *AGU Fall Meeting Abstracts*, 2019.

[41] Kuiper Systems LLC. Application of kuiper systems llc for authority to launch and operate a non-geostationary satellite orbit system in ka-band frequencies. FCC, https://licensing.fcc.gov/myibfs/download.do?attachment_key=1773885.

[42] R.J. Leopold. The iridium communications systems. In *Singapore ICCS/ISITA '92*, 1992.

[43] Israel Leyva-Mayorga, Marc M. Gost, Marco Moretti, Ana Pérez-Neira, Miguel Ángel Vázquez, Petar Popovski, and Beatriz Soret. Satellite edge computing for real-time and very-high resolution earth observation. arXiv 2212.12912, 2022.

[44] Martina Lofqvist and José Cano. Optimizing data processing in space for object detection in satellite imagery. *CoRR*, abs/2107.03774, 2021.

[45] Martina Lofqvist and José Cano. Accelerating deep learning applications in space. arXiv 2007.11089, 2020.

[46] Microsoft. Azure Orbital. https://azure.microsoft.com/en-us/services/orbital/.

[47] Sorour Mohajerani and Parvaneh Saeedi. Cloud-net: An end-to-end cloud detection algorithm for landsat 8 imagery. In *IGARSS 2019 - 2019 IEEE International Geoscience and Remote Sensing Symposium*, 2019.

[48] Arthi Padmanabhan, Neil Agarwal, Anand Iyer, Ganesh Ananthanarayanan, Yuanchao Shu, Nikolaos Karianakis, Guoqing Harry Xu, and Ravi Netravali. Gemel: Model merging for Memory-Efficient, Real-Time video analytics at the edge. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 973–994, Boston, MA, April 2023. USENIX Association.

[49] C. Partridge and T.J. Shepard. Tcp/ip performance over satellite links. *IEEE Network*, 1997.

[50] Planet Inc. Dove Satellite Constellation. https://www.planet.com/our-constellations/.

[51] Planet Inc. Planetscope. https://developers.planet.com/docs/data/planetscope/.

[52] Stephen R. Pratt, Richard A. Raines, Carl E. Fossa, and Michael A. Temple. An operational and performance overview of the iridium low earth orbit satellite system. *IEEE Communications Surveys*, 1999.

[53] William B Rossow and Leonid C Garder. Cloud detection using satellite measurements of infrared and visible radiances for isccp. *Journal of climate*, 1993.

[54] Mahadev Satyanarayanan, Paramvir Bahl, Ramon Caceres, and Nigel Davies. The case for vm-based cloudlets in mobile computing. *IEEE Pervasive Computing*, 8(4):14–23, 2009.

[55] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. Edge computing: Vision and challenges. *IEEE internet of things journal*, 3(5):637–646, 2016.

[56] Yushan Siriwardhana, Pawani Porambage, Madhusanka Liyanage, and Mika Ylianttila. A survey on mobile augmented reality with 5g mobile edge computing: architectures, applications, and technical aspects. *IEEE Communications Surveys & Tutorials*, 23(2):1160–1192, 2021.

[57] SpaceX. Spacex non-geostationary satellite system. FCC, https://fcc.report/IBFS/SAT-LOA-20161115-00118/1158350.pdf.

[58] Peng Tang, Xinggang Wang, Angtian Wang, Yongluan Yan, Wenyu Liu, Junzhou Huang, and Alan Yuille. Weakly supervised region proposal network and object detection. In *Proceedings of the European conference on computer vision (ECCV)*, pages 352–368, 2018.

[59] Bill Tao, Maleeha Masood, Indranil Gupta, and Deepak Vasisht. Transmitting, fast and slow: Scheduling satellite traffic through space and time. In *Proceedings of the 29th Annual International Conference on Mobile Computing and Networking*, ACM MobiCom '23, New York, NY, USA, 2023. Association for Computing Machinery.

[60] Deepak Vasisht and Ranveer Chandra. A distributed and hybrid ground station network for low earth orbit satellites. In *ACM HotNets*, 2020.

[61] Deepak Vasisht, Jayanth Shenoy, and Ranveer Chandra. L2d2: Low latency distributed downlink for low earth orbit satellites. In *ACM SIGCOMM*, 2021.

[62] Maria Fernandez Vidal and Peter Bull. Using satellite data in financial inclusion. 2019.

[63] Yiding Wang, Weiyan Wang, Duowen Liu, Xin Jin, Junchen Jiang, and Kai Chen. Enabling edge-cloud video analytics for robotics applications. *IEEE Transactions on Cloud Computing*, pages 1–1, 2022.

[64] M. Wiseman and A. Bradley. Impact of the length of the sea ice-free summer season on Alaskan Arctic coastal erosion rates. In *AGU Fall Meeting Abstracts*, 2019.

[65] Zhujun Xiao, Zhengxu Xia, Haitao Zheng, Ben Y. Zhao, and Junchen Jiang. Towards performance clarity of edge video analytics. In *2021 IEEE/ACM Symposium on Edge Computing (SEC)*, pages 148–164, 2021.