IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 21 June 2026, accepted 6 July 2026, date of publication 9 July 2026, date of current version 15 July 2026.

Digital Object Identifier 10.1109/ACCESS.2026.3711682

SURVEY

A Review of Machine Learning in Embedded Space Systems: Challenges, Techniques, and Future Directions

MOHAMMED ESHAQ[1,2], (Member, IEEE), M. SAMI ZITOUNI[2], (Senior Member, IEEE), JAIME ZABALZA[1], (Member, IEEE), AND MALCOLM MACDONALD[1]

[1]Department of Electronic and Electrical Engineering, University of Strathclyde, G1 1XW Glasgow, U.K.
[2]College of Engineering and IT, University of Dubai, Dubai, United Arab Emirates

Corresponding author: Mohammed Eshaq (mohammed.eshaq@strath.ac.uk)

ABSTRACT In the quest for advanced autonomy and real-time data analysis, microsatellites such as CubeSats increasingly rely on embedded machine learning (ML) to meet demanding mission objectives. However, the limited power, memory, and processing resources on these platforms introduce significant challenges to algorithm design, performance, and reliability. This review provides a comprehensive examination of the state of the art in ML for microsatellites, highlighting the constraints inherent in low-power hardware and identifying strategies to overcome them. We begin by exploring model compression, lightweight architectures, and specialized software frameworks—including reusable, flight-software-integrated deployment approaches—that enable efficient onboard inference. Commercial off-the-shelf (COTS) hardware options—ranging from microcontrollers and Field-Programmable Gate Arrays (FPGAs) to System-on-Chip (SoC) devices with integrated Graphics Processing Units (GPUs)—are then discussed, emphasizing their relative merits and trade-offs under Size, Weight, and Power (SWaP) constraints. Drawing on a survey of key missions and research efforts, we outline the most compelling current and future applications, including autonomous navigation, Earth observation, and data filtering. We conclude with a timeline of notable ML-enabled CubeSat missions, synthesizing lessons learned and pinpointing open research issues, such as resilience to radiation, extreme model compression, and standardization. By unifying these insights, the review illuminates practical paths for designing robust, energy-efficient, and mission-ready ML solutions for space environments.

INDEX TERMS Machine learning, embedded systems, CubeSats, microsatellites, onboard processing, space systems, SWaP constraints, COTS hardware, FPGA, edge AI.

I. INTRODUCTION

Artificial Intelligence (AI) and Machine Learning (ML) are central to data-intensive tasks across scientific, commercial, and defense domains. Space-based missions on resource-limited satellites face stringent power, weight, and thermal constraints; miniaturized satellites have nonetheless grown in popularity thanks to reduced launch costs and rapid design-to-orbit lifecycles. This convergence motivates *space edge AI*: moving data analysis, filtering, and decision-making onto the satellite rather than relying on delayed transmission to ground. Microsatellites enable Earth observation, deep-space exploration, and technology demonstration but require specialized algorithms and efficient hardware under low power, intermittent communication, and radiation. This paper reviews state-of-the-art methods for implementing ML onboard CubeSats and similar platforms, including commercial off-the-shelf (COTS) hardware options, model optimization tactics, and implications for future missions.

The associate editor coordinating the review of this manuscript and approving it for publication was Shadi Alawneh.

© 2026 The Authors. This work is licensed under a Creative Commons Attribution 4.0 License.
For more information, see https://creativecommons.org/licenses/by/4.0/

A. MOTIVATION AND SCOPE

Two drivers underpin onboard ML: *bandwidth management* and *mission autonomy*. High-resolution payloads produce vast data volumes while downlinks remain bandwidth-limited; small satellites especially need onboard filtering, compression, or classification for near real-time responsiveness.

Beyond Low Earth Orbit (LEO), round-trip latency and sparse ground contact make autonomy indispensable; onboard fault detection, object tracking, or target recognition enhance safety and enable new science.

The scope of this paper spans ML model optimization techniques and hardware acceleration options. We survey toolchains and deployment frameworks for resource-limited environments and mission demonstrations that validate feasibility in orbit, providing a cohesive perspective on trade-offs and design considerations; recent field maturation–from single-purpose experiments to deployable pipelines with emphasis on reliability under space-induced faults–motivates this updated synthesis.

We use *CubeSats* as the primary reference but generalize to *microsatellites* and *nanosatellites* when discussing hardware, since accelerator-class payloads are often hosted on larger buses (6U–16U). The analysis targets the nanosatellite (∼1–10 kg) and microsatellite (∼10–200 kg) ranges, where Size, Weight, and Power (SWaP), thermal, radiation, and intermittent downlink drive onboard inference. We include CubeSat and related missions executing ML onboard and enabling hardware/toolchains for that regime; we exclude large spacecraft avionics and purely ground-only AI except for context.

B. RELATION TO PRIOR SURVEYS AND CONTRIBUTIONS

Surveys of onboard/edge AI in space remain fragmented across application motivation, algorithmic techniques, hardware/toolchains, and flight evidence. [1] articulate an "AI-first" Earth observation (EO) perspective and a COTS Vision Processing Unit (VPU) (Myriad 2) case study with radiation characterization; [2] focus on hardware platforms and accelerator trade-offs. Table 1 compares these with the present review. Our end-to-end logic proceeds from problem context to constraints, then to design space and solutions, then to landscape evidence, and finally to validation and deployability limits.

C. OUTLINE AND CONTRIBUTIONS OF THIS PAPER

We briefly summarize the paper's layout and key contributions: **Section** II: resource constraints (power, radiation, communication). **Section** III: autonomy, Earth observation, real-time decision-making. **Section** IV: model compression, optimization, hardware (microcontroller units (MCUs) to System-on-Chip (SoC) and Graphics Processing Unit (GPU) platforms), toolchains. **Section** V: synthesis of missions and experiments. **Section** VI: chronological mission overview. **Section** VII: bottlenecks, benchmarking, collaboration. **Section** VIII: insights, open problems, outlook. Our primary contribution is to unify the literature across algorithms, hardware, and mission design and distill practical lessons for space edge AI.

The main contributions are:

- A **structured landscape synthesis** that combines the paper taxonomy (including the Category 2 and Category 4 lines), cross-cutting thematic analysis, and mission-demonstration evidence into one coherent reading of the field.
- A **quantitative requirement-envelope synthesis** (Table 8) and a **signature mission-study design artifact** (Table 9 with Fig. 12) linking mission objectives to constraints, ML strategy, compute substrate, and assurance implications.
- A **reviewer-oriented deployability synthesis** that distills dominant trade-offs (accuracy-latency-memory-energy-reliability) and identifies remaining evidence gaps in validation, benchmarking, and mission assurance.

II. BACKGROUND AND CHALLENGES OF ML ON EMBEDDED PLATFORMS

ML is ubiquitous in vision, natural language processing (NLP), and robotics; historically it ran on cloud infrastructure with abundant Central Processing Units (CPUs) and GPUs [3]. An expanding class of applications demands *onboard* real-time intelligence where power, memory, and bandwidth are severely limited. Similar constraints apply to space systems, particularly microsatellites such as CubeSats, where continuous downlink of high-volume data is infeasible and mission success may hinge on prompt or autonomous responses. Beyond LEO, long delays and limited contact make autonomous, resource-aware inference mission-enabling.

A. KEY RESOURCE LIMITATIONS

Constrained environments–especially *microsatellites* and *nanosatellites*–impose stringent limits versus server-class machines: **size/weight** (form factor limits PCB real estate, heat sinks, memory); **thermal** (little mass for heat, no active cooling, throttling risk); **compute** (embedded CPUs and MCUs offer only hundreds of Million Operations Per Second (MOPS); Digital Signal Processors (DSPs) and GPUs bound by clock and bandwidth); **memory/storage** (scarce on-chip RAM forces activations/weights off-chip; limited flash constrains model footprint and over-the-air (OTA) updates); **power** (solar/battery budgets make duty-cycling, dynamic voltage scaling (DVS), and deep-sleep essential). These constraints couple tightly to ML design: architecture, precision, and runtime scheduling determine whether inference is feasible on embedded flight hardware.

B. DEPLOYMENT-CRITICAL CONSTRAINTS AND OPTIMIZATION LEVERS

Feasibility is governed by sustained ops under a bounded duty cycle, on-chip activation capacity (or minimal off-chip

TABLE 1. Positioning of this review relative to prior surveys.

Survey	Primary emphasis	Typical scope	How this review differs
[1]	System-level "AI-first" EO architecture; COTS VPU case study (Myriad 2) incl. radiation tests	EO-centric architecture and device-level validation; LEO non-mission-critical framing	We broaden across application domains and hardware families and connect algorithmic techniques, toolchains, and mission demonstrations into an end-to-end SWaP-driven co-design perspective
[2]	On-edge AI hardware platforms and accelerator trade-offs	Hardware-centric coverage of accelerators	We integrate hardware discussion with resource-aware ML techniques and a mission timeline, and synthesize the resulting design trade space and adoption barriers
This work	Cross-layer synthesis	Applications + techniques + hardware/toolchains + missions	End-to-end comparative framework (accuracy–latency–energy–memory–reliability) for resource-constrained spacecraft

transfers), and runtime under thermal/power regime [4], [5]. Space-edge workloads are often *memory-bound*; memory planning and operator scheduling are as important as arithmetic [4].

1) COMPUTATIONAL CONSTRAINTS UNDER BOUNDED DUTY CYCLES

Onboard inference must coexist with guidance, navigation, and control (GNC); payload operations; and comms windows. Consequently, the relevant budget is not peak throughput in Tera Operations Per Second (TOPS) but the *sustained* throughput achievable within a power-positive schedule and thermal limits. This mirrors observations in other safety- and deployment-constrained domains (e.g., clinical decision support), where models that score well offline can still be impractical at scale due to computational cost and resource demands [6], [7].

2) MEMORY FOOTPRINT AND BANDWIDTH OFTEN DETERMINE BOTH "WHAT FITS" AND "WHAT RUNS"

On embedded targets, feasibility is frequently determined by the *peak live (runtime) memory* required during inference. Let M_{weights} denote the memory needed to store the model parameters (weights), and let $M_{\text{act}}(t)$ denote the memory occupied by intermediate activations (feature maps, buffers, and temporaries) at layer/time step t. A first-order estimate of the peak memory is

$$M_{\text{peak}} \approx M_{\text{weights}} + \max_t M_{\text{act}}(t), \tag{1}$$

noting that for convolutional backbones and encoder–decoder networks, $\max_t M_{\text{act}}(t)$ can exceed M_{weights} due to large feature maps. When this working set does not fit in on-chip memory (e.g., SRAM/cache) and spills to off-chip memory (e.g., DRAM), both latency and energy can become dominated by memory transfers rather than arithmetic.

A useful "roofline" intuition is that the end-to-end inference time t is lower-bounded by the worse of a compute limit and a memory-traffic limit:

$$t \gtrsim \max\left(\frac{N_{\text{ops}}}{\Theta_{\text{cmp}}}, \frac{N_{\text{bytes}}}{B_{\text{mem}}}\right), \tag{2}$$

where N_{ops} is the number of arithmetic operations required for one inference, Θ_{cmp} is the sustained compute throughput (operations per second) achievable on the target, N_{bytes} is the total volume of data moved to/from memory (bytes read/written), and B_{mem} is the effective memory bandwidth (bytes per second). This bound highlights why reducing activation size and memory traffic (e.g., via tiling, operator fusion, and quantization-aware memory planning) can yield larger gains than optimizing arithmetic alone [4], [5].

3) COMPUTE, MEMORY, ENERGY, AND LATENCY ARE COUPLED

The energy consumed by a single inference, E (in Joules), is well-approximated by the product of the average electrical power drawn during inference, P_{avg} (in Watts), and the end-to-end inference time (latency), t (in seconds), i.e., $E \approx P_{\text{avg}} \cdot t$ (more generally, $E = \int_0^t P(\tau)\,d\tau$ if power varies over time). This coupling means that reducing runtime *and* reducing power both directly reduce energy per inference. In practice, architectural and implementation choices that reduce intermediate activation sizes and off-chip memory traffic often lower t (by avoiding memory stalls) and can also lower P_{avg}, yielding fewer Joules per inference than simply using a higher-peak-throughput processor [5]. Moreover, for mission workloads the critical requirement is frequently *worst-case* latency (bounded execution time) rather than peak throughput, because inference must be scheduled around payload operations, downlink windows, and power-positive periods [5]. Similar deployment realities arise in applied ML systems that move from offline evaluation to continuous operation (e.g., dynamic prediction pipelines), where latency and compute cost become first-class design constraints rather than secondary implementation details [6].

4) COMMON OPTIMIZATION KNOBS (SPACE-EDGE PRACTICALITIES)

Space-edge ML deployments typically combine model compression (quantization, pruning, weight/activation packing) [4], [8], lightweight backbones and resolution/tiling, feature reduction when appropriate [6], [9], runtime/IO shaping (operator fusion, static allocation, streaming), and cascades/early-exit to cut average energy.

C. THE SPECIAL CASE OF MICROSATELLITES/ SPACE-EDGE AI

Constraints tighten as platforms shrink. *Microsatellites* (10–100 kg, often 6U–16U CubeSats) have modest power and radiator area; *nanosatellites* (1)–10 kg) pack avionics, batteries, and payloads into minimal volumes (e.g. 1U = 10 cm ''cube''). Classification is by launch-ready mass (pico <1 kg, nano 1–10 kg, micro 10–100 kg). The familiar ''U'' form factor is a geometric shorthand. CubeSats often use COTS for cost and lead time; larger craft use space-qualified parts that trail terrestrial technology. In both cases the performance-to-mass-to-cost ratio is constrained [4], making micro- and nanosatellites a demanding testbed for onboard ML.

1) STRICT SWaP AND THERMAL LIMITS

Space-edge SWaP is acute: CubeSats typically have only a few watts; power-gating and scheduling dictate when and how ML tasks run and significantly impact the spacecraft's energy budget and thermal profile.

Traditional CPUs/GPUs draw tens of watts and need heatsinks–often infeasible. GPU adoption remains limited by power budgets, tooling, and few space-tailored architectures [10]. Deploying deep neural networks (DNNs) requires balancing compute and memory against tight power and thermal budgets [4]; oversized models quickly become impractical.

2) LIMITED COMMUNICATION WINDOWS

In LEO, microsatellites pass over ground stations only a few times per day with short contact durations. Downlink rates are also limited by the small, low-gain antennas they typically carry. Consequently, the ability to prioritize or filter collected data onboard becomes critical; ML can facilitate advanced triaging of images, classification of phenomena, and pre-extraction of key features before any data is transmitted to Earth.

In deep-space and cislunar missions, the constraint shifts from short ground passes to high latency, sparse contact opportunities, and low link margins. This further increases the value of onboard triage and autonomy: decisions may need to be made without timely ground feedback, and only condensed, high-value products can be returned during limited downlink windows.

3) RELIABILITY AND RADIATION CHALLENGES

Microsatellites that rely on commercial (non-radiation-hardened) electronics are acutely exposed to space radiation, facing both transient *single-event upsets* (SEUs) and long-term *total-ionizing-dose* (TID) degradation. In LEO, repeated passages through the South Atlantic Anomaly intensify particle flux, while higher orbits and deep-space missions must also contend with cosmic rays and solar events. SEUs can flip bits in RAM, configuration memory, or logic, silently corrupting neural-network weights or control registers; cumulative dose gradually weakens transistor junctions, leading to permanent failures.

Mitigation spans circuit/logic level (Error Correction Code (ECC), Triple Modular Redundancy (TMR), conservative margins), system level (scrubbing, checkpointing of model weights), and algorithm level (fault-tolerant training, weight randomization, on-orbit fine-tuning).

a: RADIATION ASSURANCE FOR COTS ML PROCESSORS (SYSTEM-LEVEL HARDENING RATHER THAN TRUE RAD-HARD)

To close the widening performance gap in Fig. 1, recent space-edge ML architectures increasingly adopt *radiation assurance* approaches that make commercial processors acceptable for non-critical onboard analytics, without claiming that the devices are intrinsically radiation hardened. A common pattern is a *mixed-criticality* design, where a rad-tolerant supervisor (or fault-managed OBC (onboard computer)) maintains spacecraft safety and determinism, while a higher-throughput COTS SoC/GPU/VPU executes payload ML as a best-effort service. Reconfigurable computing platforms that explicitly separate reliable nodes from high-performance COTS nodes exemplify this trajectory and enable graceful degradation when faults occur [11].

Practical assurance typically combines a small toolbox of measures:

- **Partitioning and supervision:** isolate payload ML from safety-critical GNC/FDIR (Fault Detection, Isolation, and Recovery) functions, enforce watchdog supervision, and use health-monitoring plus restart/rollback policies for the ML service.
- **Fault mitigation and recovery:** ECC and Error Detection and Correction (EDAC) for memories, selective redundancy for critical state, periodic scrubbing or reloading where applicable, and checkpointing/rollback of model parameters and intermediate products.
- **Test-and-monitor approach:** radiation campaigns and in-orbit monitoring (e.g., memory checkers, consistency tests, repeated runs) to quantify the reliability–performance–energy trade space on the target workload.

Importantly, radiation effects interact with performance and energy in non-trivial ways on low-power heterogeneous SoCs: neutron irradiation studies on Jetson-class devices demonstrate that the reliability of CPU/GPU-offloaded execution depends on workload structure and operating point, so ''more acceleration'' does not automatically imply ''better resilience'' [12]. Complementary evidence from in-orbit deployments also suggests that carefully managed COTS

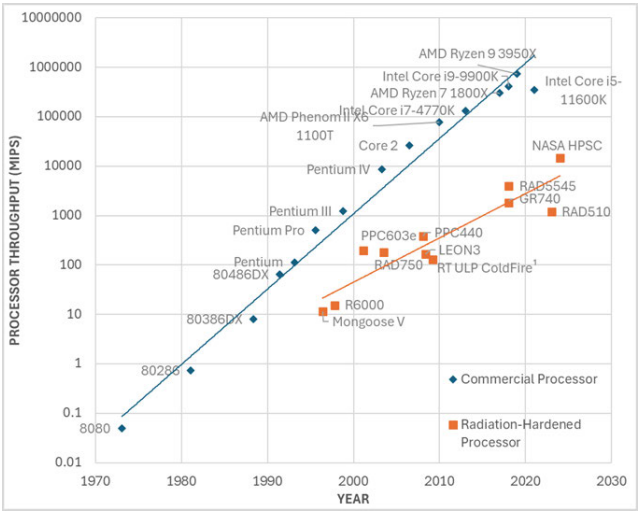

FIGURE 1. Processor throughput versus year for commercial and radiation-hardened CPUs, highlighting the widening performance gap.

accelerators can be viable for payload analytics in LEO: deep-learning models executed on Myriad and Snapdragon processors hosted on Spaceborne Computer-2 showed consistent outputs between ground and International Space Station (ISS) runs, with onboard checkers reporting no detected memory errors during the reported campaign [13]. Collectively, these results support a pragmatic direction: *make COTS usable via architectural containment and assurance*, while reserving mission-critical loops for fault-managed compute until verification standards and long-duration radiation evidence mature [1], [14].

Before delving into the details of processor performance, it is useful to recall the basic landscape: most microsatellites, including CubeSats, favor COTS microcontrollers or system-on-modules to maximize computing power per dollar and minimize development time, accepting a higher risk of radiation faults that can be mitigated in software. By contrast, larger and longer-lived spacecraft typically rely on radiation-hardened CPUs–devices engineered for decades of fault-free operation in severe orbital and deep-space environments–even though their raw throughput lags the COTS curve by roughly a decade.

Fig. 1, updated from [15], contrasts the historical performance trajectories of mainstream commercial CPUs (black circles) with radiation-hardened space-qualified processors (red triangles) by plotting their peak integer throughput in Million Instructions per Second (MIPS) on a logarithmic scale against the year of introduction. The commercial line roughly follows the classic ''Moore's Law'' slope, advancing nearly an order of magnitude every five to six years from the Intel 8080 through the Core 2 era. Space-grade chips rise along a similar slope but with a persistent offset of roughly one technology generation–about a decade–so that a modern rad-hard LEON3 or PowerPC 440 offers a throughput comparable to a late-1990s Pentium III. The widening gap underscores the integration challenge facing on-orbit ML: while terrestrial edge devices can rely on multi-gigaflop SoCs, spacecraft designers must either accept far lower compute margins or mitigate risk by hybridizing commercial parts with fault-tolerant architectures and error-mitigation techniques.

Because microsatellites must balance stringent SWaP limits with these protective measures, designers often combine carefully screened COTS parts, board-level shielding, and lightweight redundancy techniques to achieve robust ML inference without exceeding power or mass budgets.

4) SHORT MISSION LIFETIMES, HIGH-VALUE DATA, AND FAST DESIGN LOOPS

Microsatellite platforms typically operate for only ∼ 12-24 months, a fraction of the decade-long lifespans common to large GEO spacecraft. In that compressed window every orbit matters: downlink opportunities are brief, onboard storage is scarce, and a single missed transient (e.g., a volcanic plume, algal bloom, or traffic hotspot) can wipe out a sizable fraction of the mission's total science return. The low cost and rapid build cadence of small spacecraft create a ''fly-learn-refly'' culture where small teams often fail their first CubeSat mission, followed by a successful one that has been refined from the lessons learned from the first failure. Furthermore, models can be refined in flight, uplinked as software patches, and rolled wholesale into the next launch just months later, compounding performance gains from one generation to the next [16]. In short, short lifetimes turn onboard ML from a desirable add-on into a mission-critical enabler that maximizes return on every pass.

On-orbit ML is ultimately a *software product* that must be packaged, uploaded, verified, and recoverable. Practical deployments therefore require format/tool compatibility (conversion and operator support), well-defined rollback paths, and containment of ML services from safety-critical control loops via supervision and mixed-criticality partitioning [1], [11]. Beyond aerospace, reviews of real-world ML systems emphasize analogous ''last-mile'' barriers: integration with existing operational stacks, regulatory/assurance requirements, privacy constraints, and clinician/operator trust driven by interpretability [6], [7]. These realities motivate designs that are not only accurate, but also update-tolerant, resource-aware, and operationally testable under realistic spacecraft power/thermal/radiation conditions [4], [12].

5) DIVERSE MISSION PROFILES AND PAYLOADS

Microsatellites have been deployed for tasks ranging from Earth imaging and climate monitoring to deep-space asteroid rendezvous. Each mission profile has unique performance requirements for sensing, data processing, and communication. Hence, ML solutions for microsatellites must be adaptable to different modalities (e.g., visible, multispectral, hyperspectral) and to varying computational needs (e.g., real-time vs. batch processing).

III. WHY AI ON THE EDGE OF SPACE? (APPLICATIONS)

As noted in Section I, the twin drivers for onboard ML are bandwidth management and mission autonomy. Autonomous ML in orbit significantly enhances the capabilities of nano and microsatellites by enabling real-time or near-real-time decision making, data filtering, and adaptive mission profiles. Beyond the technical demonstration of ML in a constrained environment, these applications broadly fall into two main categories: **autonomy** and **remote sensing**. This section outlines how microsatellites leverage onboard ML within these key areas, showcasing the diverse ways ML can transform both near-Earth and deep-space missions.

A. AUTONOMY

Spacecraft autonomy spans a broad research landscape that touches everything from navigation and multi-vehicle coordination to onboard planning, scheduling, and overall system reliability [17].

1) REAL-TIME DECISION-MAKING & EVENT TRIGGERING

A prime driver for embedding ML in CubeSats is the need to act quickly on incoming sensor data, often within tight windows of opportunity. By integrating lightweight machine-learning models (e.g., convolutional neural networks (CNNs)), satellites can enable rapid onboard decisions.

a: TRIGGER TARGETED OBSERVATIONS

On detecting anomalous events, the satellite can reorient or capture follow-on images. Reported dynamic-targeting workflows explicitly use look-ahead image analysis (e.g., cloud/thermal anomaly or spectral-signature detection) to retask sensing and issue repointing/reconfiguration commands within short LEO timelines [18].

b: COGNITIVE RADIO FOR DYNAMIC SPECTRUM ACCESS

Edge-resident ML is increasingly being applied to the communications stack itself, where real-time spectrum sensing and re-configuration can maximize throughput and minimize interference. A notable demonstration is NASA's Space Communications and Navigation (SCaN) Testbed on the International Space Station, which employs a "cognitive radio" that continuously monitors the RF environment and opportunistically switches to underutilized frequency bands, thereby maintaining a reliable link without manual ground intervention [17].

c: ADAPT SATELLITE OPERATIONS

Machine-learning-based "systems can help improve spacecraft reliability by being trained to react to unexpected situations and guide the spacecraft to safer operational states with autonomous decision-making" [17]. Real-time ML pipelines can monitor telemetry or sensor data for anomalies. Anomaly detection can trigger shutdowns or Field-Programmable Gate Array (FPGA) reconfiguration. Fault Detection, Isolation, and Recovery (FDIR) has evolved from rule-based and statistical ML (SVMs, K-means) toward CNNs and LSTMs for context-aware fault detection [19].

2) DEEP-SPACE MISSIONS (NAVIGATION, COLLISION AVOIDANCE)

For nano and microsatellites venturing beyond LEO, high-latency or intermittent communication makes real-time ground control impractical. "Deep-space nanosats must rely on an even higher degree of autonomy because of the limited ground station availability." [20]. Onboard ML enables satellites to handle navigation, collision avoidance, and other split-second decisions locally.

a: ONBOARD NAVIGATION AND GUIDANCE

Deep-space probes must operate with long communication lags, making onboard autonomy indispensable. The Mars rovers illustrate this principle: Spirit's AutoNav software handles terrain-hazard detection, while Opportunity and Curiosity employ AEGIS to choose high-value science targets without Earth in the loop. Similar capabilities will be even more critical for future missions that spend extended periods out of contact and face unpredictable environments [17].

b: RENDEZVOUS AND DOCKING

For proximity operations (e.g., capturing a drifting sample container or docking with a service module), neural networks track an uncooperative target's pose in real time. This is vital when short contact windows demand rapid maneuvers [21], [22].

c: COLLISION AVOIDANCE AND HAZARD DETECTION

Machine learning can parse optical sensing data to identify conjunction hazards during proximity operations and debris-aware mission scenarios; anomaly- or event-detection outputs can then trigger evasive maneuvers or attitude changes [20], [21].

d: AUTONOMOUS SCHEDULING

Onboard ML planners (e.g. MEXAR2 [17]) optimize task scheduling and downlink timing [17]. In practice, this means deciding which workloads should run immediately onboard versus deferred execution, and prioritizing high-value data products when communication windows are short.

3) LONG-TERM VISION: SWARMS, CONSTELLATIONS, AND IN-ORBIT MANUFACTURING

While single-satellite missions have already demonstrated onboard ML, the near future points to collaborative, multi-unit deployments.

a: SWARM INTELLIGENCE AND CONSTELLATIONS

A network of microsatellites can share locally processed data and coordinate observations. Each unit may run ML algorithms to classify or compress sensor data, transmitting

only summarized insights. This reduces inter-satellite traffic and the burden on ground-stations. DARPA's recent Broad Agency Announcement for the Blackjack program outlines plans for "Pit Boss," a next-generation avionics unit built from commercial off-the-shelf technology to bring powerful edge computing and autonomous mission control to orbit. The project proves that a replenishable constellation of low-cost, self-governing SmallSats in low-Earth orbit can match the capabilities of traditional, monolithic spacecraft stationed in geosynchronous orbit [23].

Constellations can adapt flight paths and sensor modes jointly; partial reconfiguration or in-orbit model updates allow capability evolution.

B. REMOTE SENSING

1) BANDWIDTH REDUCTION AND ONBOARD DATA FILTERING

The best-documented use case is image triage and cloud detection for downlink reduction. EO missions produce vast raw imagery while microsatellites have narrow communication windows and low downlink rates; onboard processing and extraction reduce data volume significantly [24], [25].

a: LIMIT DOWNLINK TO HIGH-VALUE DATA

The work in [26] demonstrates how a 1U CubeSat can run a tiny CNN, *CubeSatNet*, to judge image quality onboard. By downlinking only "good" Earth-facing frames, the satellite cuts bandwidth use by roughly two-thirds and boosts the ratio of useful images from 12/30 to nearly 1:1, all within the spacecraft's strict power and memory limits. "The application of deep-learning concepts for onboard processing can enable spacecraft to efficiently process immense volumes of raw sensor data into actionable data to overcome limitations in downlink communication" [23].

b: ONBOARD CLOUD MASKING

Onboard cloud detection is essential given widespread cloud cover. U-Net-style CNNs [27] and CloudScout on PhiSat-1 [8], [19] filter images by cloud cover; FPGAs can further improve efficiency. Onboard processing allows filtering and preliminary decisions without ground intervention and reconfigurability via weight updates [8], [28].

c: DEEP-SPACE REMOTE SENSING

Even when a microsatellite observes Earth–or another planetary body–from higher orbits, raw imagery is degraded by path-dependent absorption and scattering. Performing radiometric and spectral corrections onboard allows the spacecraft to compensate for atmospheric attenuation before any data are sent home, so the downlink contains compact, analysis-ready products rather than bulky raw cubes. For hyperspectral-based attitude reconstruction, representative initial SVM results report $R^2 = 99.49$ and RMSE $= 0.0755$ for quaternion prediction [29]. More broadly, hardware-accelerated onboard pipelines have shown feasibility for demanding pattern-recognition workloads under CubeSat-class constraints [10].

d: DETECTION OF ASSETS

Deploying DNNs directly on the spacecraft makes it possible to spot mission-critical objects–ships, aircraft, vehicles, and even early signs of natural disasters–before the imagery ever leaves the satellite. Because inference occurs in situ, the payload can scan vast low-priority regions, such as open ocean or desert terrain, and still downlink only the frames containing relevant targets. This shifts bandwidth toward information-rich events without sacrificing global coverage; areas historically imaged at lower cadence to save downlink can now be monitored continuously, with alerts or cropped regions sent back whenever the onboard classifier flags a match [4].

2) CHANGE DETECTION AND DATA PRIORITIZATION

More advanced ML pipelines can perform not just classification but also *change detection*–identifying subtle shifts between successive passes over the same region: Gradual changes (land-use, glacial, urban) can be flagged with segmentation/differencing; rapid changes can trigger reprioritization, storage reallocation, or sensor reconfiguration. This focus on "what's new" boosts mission efficiency, aligning data acquisition strategies with real-time environmental dynamics.

3) ML-DRIVEN COMPRESSION AND SUMMARIES

Onboard ML can *transform* raw observations into smaller products: **learned compression** (encoder to latent code, quantized/entropy-coded; decoder on ground) gives rate–distortion trade-offs [30]; **enhancement/fusion** (super-resolution, pan-sharpening, hyperspectral fusion [31], [32]) yields analysis-ready imagery so only enhanced products need downlink.

IV. TECHNIQUES AND METHODOLOGIES FOR FITTING ML IN RESOURCE-CONSTRAINED SYSTEMS

Deploying ML on microsatellites requires balancing accuracy against power, memory, and throughput. This section surveys **model compression**, **optimization** (lightweight architectures), **hardware acceleration**, and **software toolchains** for spaceborne edge AI.

A. MODEL COMPRESSION TECHNIQUES

Compression reduces model size and computational complexity so inference fits CubeSat resource budgets (Fig. 2): reducing precision (quantization) and reducing operations/size (pruning, factorization, compact architectures).

1) QUANTIZATION

Quantization in neural networks is an optimization method that has shown significant success lately. It primarily aims to decrease memory usage and computational duration by substituting floating-point calculations and saving tensor weights

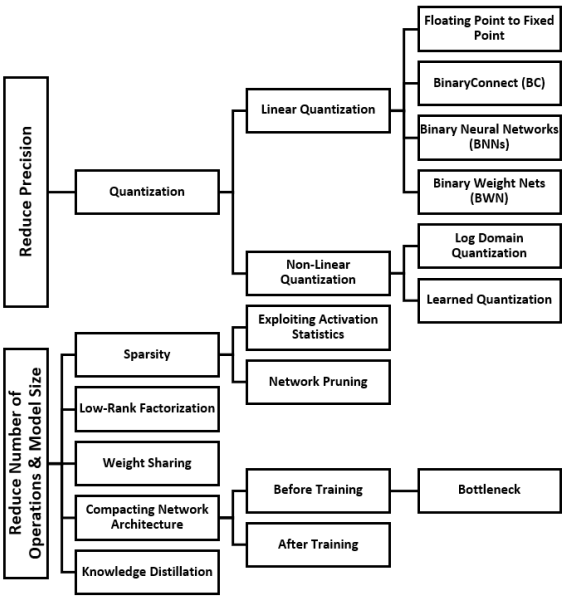

FIGURE 2. Hierarchy of neural-network model-compression and acceleration techniques.

with reduced bit precision [33]. Converting floating-point parameters and activations to fixed-point (e.g., 8-bit or even 4-bit) formats significantly cuts down on memory usage and accelerates inference on hardware that supports efficient integer arithmetic. Quantized networks also consume less energy by reducing the number of bits transferred between memory and processing units. Techniques such as *post-training quantization* and *quantization-aware training* help retain as much predictive performance as possible. We first divide quantization into *linear* and *non-linear* and study both methods.

a: LINEAR QUANTIZATION

The most trivial way of reducing precision is usually to convert values and operations from floating point to fixed point. A 32-bit floating point number has a larger *memory* footprint than an 8-bit fixed point [34]. Using 8-bit fixed point arithmetic significantly reduces both *energy* consumption and area requirements compared to 32-bit operations.

Energy and area scale favorably with reduced bit width (e.g. 8-bit vs 32-bit yields large savings; [34]).

By pushing network precision to a single bit–often called ''binary nets''–multiplications in multiply-accumulate (MAC) operations collapse to simple additions, subtractions, or even XNORs. Early work such as BinaryConnect (BC) introduced binary weights $(-1, +1)$, reducing multiplications to add/subtract, while Binarized Neural Networks (BNN) extended this to both weights and activations, implementing MACs via XNOR. However, BC and BNN incur substantial accuracy drops (19% and 29.8%, respectively). To narrow this gap, Binary Weight Networks (BWN) and XNOR-Nets apply refined quantization and scaling strategies, cutting the loss to about 0.8% and 11%. Further improvements arise when activations are quantized to 2 bits while keeping weights binary–as in Quantized Neural Networks (QNN), DoReFa-Net, and HWGQ-Net–bringing accuracy loss down to roughly 5.2% [34]. Recent advancements in binary neural networks (BNNs) have significantly closed the accuracy gap with real-valued networks. A notable example is ReActNet [35], which introduces learnable activation functions–RSign and RPReLU–that explicitly reshape and shift activation distributions. Combined with a distributional loss that aligns binary and real-valued output distributions, ReActNet achieves 69.4% top-1 accuracy on ImageNet using only 87 million operations. This performance not only surpasses all prior binary methods, including Real-to-Binary Net and MeliusNet29, but narrows the accuracy gap to within 3% of its full-precision counterpart–all while maintaining the binary network's efficiency and hardware compatibility.

Table 2 distills the main flavors of extreme-low-precision CNNs into three columns–weight precision, activation precision, and the net benefit/trade-off they deliver. The table supports choosing an operating point along the efficiency–accuracy continuum for space-qualified FPGAs, VPUs, or MCUs.

TABLE 2. Representative binary and ultra-low-precision CNN variants, showing how increasingly refined quantization strategies trade memory/computation for accuracy.

Method	Weight precision	Activation precision	Key advantages / trade-offs
Binary Connect (BC)	1-bit $(-1, +1)$	Full-precision	Minimal change to the compute graph–every multiply by ±1 becomes an add/subtract–and 32× model-memory reduction for weights with only a small accuracy drop on simple datasets.
Binary Neural Networks (BNN)	1-bit $(-1, +1)$	1-bit $(-1, +1)$	Maximises compression and speed: convolutions reduce to XNOR + popcount; delivers 32× memory and 58× arithmetic savings, but suffers the largest accuracy hit.
Binary Weight Networks (BWN)	1-bit $(-1, +1)$	Full-precision	Uses scaling factors per filter to recover much of the lost accuracy (<1% on CIFAR-10 vs FP32) while keeping weight memory 32× smaller and removing real multiplications at inference.
XNOR-Net	1-bit $(-1, +1)$	1-bit $(-1, +1)$	Adds data-driven scaling to both weights and activations, cutting the accuracy gap with full-precision nets to 11% while still reaping the full binary speed/memory benefits.
ReActNet	1-bit $(-1, +1)$	1-bit $(-1, +1)$	Introduces learnable activation functions (RSign and RPReLU) and a distributional loss to optimize activation distributions; achieves 69.4% top-1 accuracy on ImageNet with only 87M OPs–narrowing the gap to FP32 networks to just 3%.
QNN / DoReFa-Net / HWGQ	1-bit $(-1, +1)$	2-bit	Quantising activations to 2 bits (and sometimes weights to 2-4 bits) strikes a sweet spot: only a 5% accuracy penalty on ImageNet while remaining amenable to bit-serial or packed-bit hardware.

In general, reduced precision has not only been a focus of research, but it has also been implemented in recent commercial platforms for DNN processing. For example, Google's Tensor Processing Unit (TPU), introduced in May 2016, and Nvidia's PASCAL GPU, unveiled in

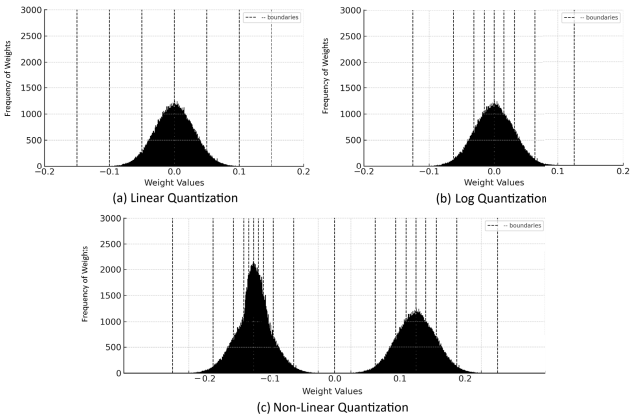

FIGURE 3. Comparison of quantization methods for neural network weights: (a) Linear Quantization, (b) Logarithmic Quantization, and (c) Non-Linear Quantization with clustering and fine-tuning [34].

April 2016, were both designed to operate using 8-bit integer arithmetic [34].

b: NON-LINEAR QUANTIZATION

By assigning quantization bins according to a logarithmic (base-2) scale rather than *evenly* spaced intervals, the distribution of weights and activations becomes more balanced, which significantly reduces quantization error–for example, 4-bit linear quantization may incur an accuracy drop of approximately 27.8%, whereas logarithmic quantization limits this loss to around 5%. Additionally, constraining weights to powers of two transforms multiplication operations into simple bit-shifts, further streamlining hardware implementation [34]. Nevertheless, the results in [34] for 4-bit linear quantization are outdated; however, they are presented in this context to compare linear and logarithmic quantization. Recent work on quantization [36] shows that with proper techniques (quantization-aware training, mixed-precision), 4-bit networks often lose only a few percentage points of accuracy–on the order of 2-5% in many cases. 8-bit is now standard, and 4-bit is emerging with minimal accuracy loss.

Fig. 3 compares (a) linear (uniform bins), (b) logarithmic (finer for small weights), and (c) non-linear (clustering/fine-tuning) quantization.

c: QUANTIZATION: WHEN DOES IT OCCUR?

There are two primary methods to apply quantization in a neural network. The first involves training the network using quantized values, and the second involves quantizing the parameters after the model has been trained with floating-point accuracy. The initial approach is known as *quantization-aware training (QAT)*, while the second is termed *post-training quantization (PTQ)*. PTQ can alter the model's parameters and affect the convergence point achieved during training with floating-point accuracy. Consequently, QAT is employed for the experiments in [33], and training is carried out with quantized model parameters.

Importantly, PTQ is not unique to TensorFlow Lite; comparable workflows exist across the main embedded deployment ecosystems. For models exported to ONNX, ONNX Runtime provides both *dynamic* quantization (weights quantized offline; activations quantized at runtime) and *static* quantization, where activation ranges are calibrated using a representative dataset [37]. For GPU-centric deployments, NVIDIA TensorRT supports reduced-precision inference (e.g., INT8) via calibrated post-training flows and/or explicit quantized graphs (e.g., Q/DQ-style representations), enabling hardware-efficient kernels while controlling accuracy loss [38]. Similarly, Intel's OpenVINO ecosystem supports post-training quantization through its compression toolchain (NNCF), including calibration-driven PTQ and accuracy-aware configurations [39]. More broadly, vendor-neutral toolkits such as Intel Neural Compressor expose PTQ pipelines (including calibration and automated tuning) that target multiple backends (CPU, GPU, and accelerators), highlighting that quantization is best viewed as a *deployment-stage co-design step* coupled to the target runtime and operator support [40].

TensorFlow Lite's post-training quantization, for example, compresses and accelerates pre-trained TensorFlow models by converting 32-bit weights and activations into 8-bit two's-complement integers (using per-axis weight and per-tensor activation encodings), typically with only a minor accuracy drop. Smaller networks may experience slightly larger degradations, so TensorFlow Hub provides fully quantized, pre-trained versions to mitigate this, and TensorFlow Lite includes tools to evaluate the resulting model accuracy [41].

2) SPARSITY

Sparsity leverages the fact that multiplications by zero can be skipped, saving computational resources and power [42]. Pruning and sparsity are often used interchangeably, and sometimes sparsity is seen as a subset of network pruning. However, in our view, sparsity is a broader concept, and pruning is a technique used to achieve it. Sparsity refers to a state where a model has many parameters (weights or connections) with a value of zero or very close to zero. Pruning, on the other hand, is a specific technique used to introduce sparsity into a neural network. It involves identifying and removing (setting to zero) the less important weights or connections in a trained neural network.

Activation sparsity arises naturally from ReLU nonlinearities, enabling runtime optimizations such as zero-skipping in the MAC pipeline and on-the-fly feature-map compression to save energy and memory. **Weight sparsity**, on the other hand, is achieved through network pruning, where unimportant connections (structured or unstructured) are removed, resulting in a smaller model and fewer operations. Together, these methods can yield substantial speedups and power reductions on embedded hardware.

Sparsity in neural networks can be categorized into *unstructured* or *structured* forms. Unstructured sparsity

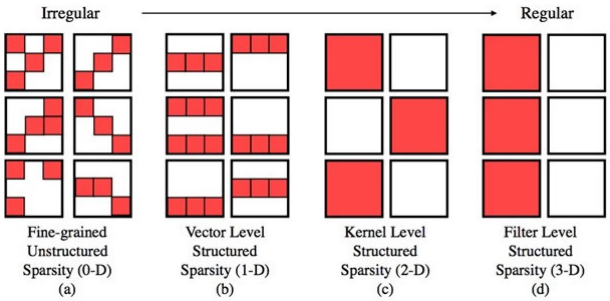

FIGURE 4. Spectrum of Sparsity in Neural Networks [42].

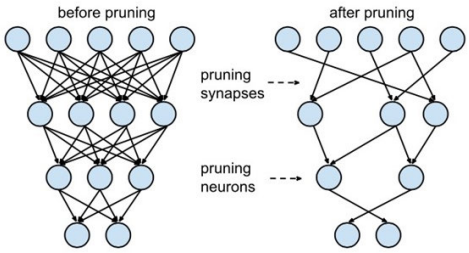

FIGURE 5. Neural Network Before and After Pruning [42].

refers to the random distribution of zeros across the network's connections, whereas structured sparsity implies a systematic arrangement, such as entire filters or channels being pruned. Fig. 4 illustrates the progression from irregular to regular forms of sparsity. By incorporating sparsity, a network's theoretical output–measured in terms of Tera Operations Per Second (TOPS)–can effectively increase. For instance, a network with an apparent capacity of 1 TOPS might achieve an effective throughput of 3 TOPS by not executing operations that involve zero, thus skipping redundant computations and enhancing performance [42].

a: EXPLOITING ACTIVATION STATISTICS

Modern DNNs frequently employ the ReLU nonlinearity, which sets all negative outputs to zero and thus produces inherently sparse activation maps. This sparsity can be leveraged in multiple ways: first, by compressing the feature maps before they are written to or read from off-chip DRAM–where each zero can be encoded in fewer bits–significantly reducing area and energy costs. Second, by augmenting the accelerator datapath to detect zero activations and skip both the corresponding weight fetch and MAC operation, studies have demonstrated up to a 45% reduction in energy consumption. Finally, further gains can be realized by pruning small-magnitude activations altogether; for example, discarding low-value activations can yield an additional 11% speedup or roughly a 2× reduction in power with negligible impact on accuracy [34].

b: NETWORK PRUNING

Networks are often over-parameterized to simplify training, resulting in many redundant weights that can be removed (i.e., set to zero) through a process known as network pruning. To preserve the original accuracy, aggressive pruning typically requires fine-tuning of the remaining weights. This concept was first introduced in 1989 with a technique called Optimal Brain Damage, which involved calculating the impact of each weight on the training loss, known as weight saliency. Weights with low saliency were eliminated, and the remaining weights were fine-tuned. This process was repeated until the network achieved the desired balance between weight reduction and accuracy [34]. For example, "this approach can reduce the number of weights in AlexNet by 9× and the number of MACs by 3×. Most of the weight reduction comes from the fullyconnected layers (9.9× for fully-connected layers versus 2.7× for convolutional layers)." [34]

Pruning removes weights that contribute minimally to a network's output, trimming connections or entire filters to yield sparser architectures. As established, *structured pruning* (removal of entire channels or filters) can directly translate to faster runtime on hardware accelerators or specialized libraries, whereas *unstructured pruning* achieves fine-grained sparsity but may require additional software support to realize speedups. In both cases, the trimmed model consumes less memory and allows inference to be completed with fewer operations.

The sparsity resulting from the pruning process can be leveraged to further reduce the computational load and the number of floating-point operations required. Fig. 5 illustrates the pruning effect. This sparsity facilitates model compression and can potentially improve inference latency [43], as it can also reduce the model size and memory footprint [44].

Recent research has investigated structured pruning, which prunes groups of weights rather than individual ones, to eliminate the need for custom hardware. This approach, known as coarse-grained pruning, offers two main advantages: (1) it aligns better with data-parallel architectures like single-instruction multiple-data (SIMD) in general-purpose hardware, leading to more efficient processing; and (2) it reduces the overhead of signaling non-zero weight locations, enhancing compression and lowering storage costs. However, pruning larger groups of weights, such as rows, columns, or entire filters, can lead to a greater loss in accuracy [34].

3) LOW-RANK FACTORIZATION

Neural networks often contain layers whose *weight* matrices exhibit redundancy. By decomposing these matrices into lower-rank approximations, one can replace large convolutional kernels or fully connected layers with smaller-dimension tensors. These factorized layers demand fewer multiply-accumulate operations and reduced memory transfers, thus lowering both computation time and power consumption.

4) WEIGHT SHARING

In weight sharing, multiple parameters are constrained to the same value, thereby reducing the total count of unique weights in a layer or filter. One common approach is to assign each weight to a bucket via a hash function and then use a single representative value per bucket. Alternatively, clustering methods such as k-means can group similar weights and replace each cluster with its centroid, yielding further memory and computational savings [34].

5) COMPACTING NETWORK ARCHITECTURES

The number of *weights* and operations in a network can be reduced by optimizing the architecture itself. One common method is to replace a large filter with a series of smaller filters, which collectively have fewer weights but maintain the same effective receptive field. This technique can be implemented either during the design phase before training or by decomposing filters in an already trained network. While the latter method avoids retraining from scratch, it is less flexible, as it generally allows only the decomposition of filters without introducing non-linearity between them [34].

a: BEFORE TRAINING

In modern deep neural network models, smaller filters are increasingly used because combining several smaller filters can achieve the same effect as a larger one. For instance, a single 5 × 5 convolution can be replaced by two 3 × 3 convolutions. Alternatively, an N × N convolution can be broken down into two 1-D convolutions, one 1 × N and one N × 1, which imposes the requirement that the 2-D filter be separable–a common approach in image processing. Similarly, a 3-D convolution can be replaced with a sequence of 2-D convolutions (applied on individual input channels) followed by 1 × 1 3-D convolutions, a method utilized in architectures like Xception and MobileNets. The sequence of applying 2-D and 1 × 1 3-D convolutions can also be rearranged. Reference [34] 1 × 1 convolutional layers can be employed to decrease the number of channels in the output feature map of a given layer. This reduction in the number of channels lowers the computational cost for the filters in the subsequent layer. This technique is commonly known as a 'bottleneck' layer [34].

SqueezeNet [34] leverages numerous 1 × 1 filters to significantly decrease the number of weights. It introduces a "fire module," which first 'squeezes' the network using 1 × 1 convolution filters and then 'expands' it with multiple 1 × 1 and 3 × 3 convolution filters. This approach results in a 50× reduction in the number of weights compared to AlexNet, while maintaining comparable accuracy. However, it is important to note that reducing the number of weights does not automatically lead to lower energy consumption; for example, SqueezeNet actually uses more energy than AlexNet.

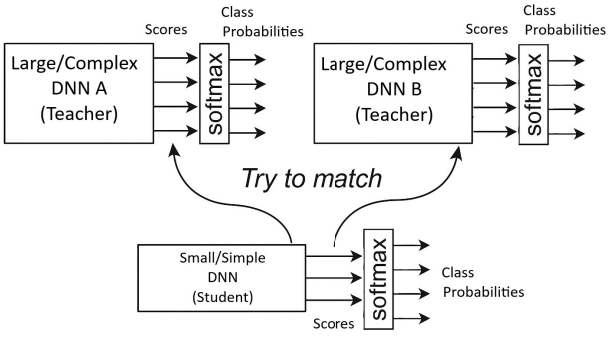

FIGURE 6. Example of knowledge distillation aligning class scores from a small DNN to an ensemble of larger DNNs [34].

b: AFTER TRAINING

In after training, tensor decomposition can be employed to break down filters in a trained network without affecting accuracy. This method treats the weights in a layer as a 4-D tensor and decomposes it into smaller tensors, effectively splitting the layer into several layers. To further increase compression, low-rank approximation can be applied, though this may degrade accuracy, which can subsequently be restored through fine-tuning of the weights [34].

6) KNOWLEDGE DISTILLATION

Knowledge Distillation is a technique where the knowledge gained by a complex model (teacher) is transferred to a simpler model (student). Although deep networks or ensembles of multiple models typically yield higher accuracy compared to a single shallow network, they also come with increased computational complexity. Knowledge distillation allows the student model to attain a level of accuracy that it would not have been able to achieve if trained on the same dataset independently [34].

Fig. 6 illustrates knowledge distillation via class-score alignment between a small DNN and an ensemble of larger DNNs.

The simplest method of knowledge distillation aligns the student's class scores with those produced by an ensemble of larger DNNs.

B. MODEL OPTIMIZATION (LIGHTWEIGHT ARCHITECTURES, TINYML APPROACHES)

Even after applying compression techniques, the foundational design of a neural network significantly influences its ability to run efficiently on edge devices like microsatellites. In recent years, a host of *lightweight architectures* and *TinyML approaches* have emerged, explicitly tailored for reduced complexity and power consumption. Key strategies include:

1) DEPTHWISE SEPARABLE CONVOLUTIONS

Networks such as MobileNet and ShuffleNet replace standard convolutions with depthwise separable convolutions to

reduce the total number of MAC operations. This yields a lower memory footprint and quicker inference speed without compromising too heavily on accuracy.

2) MICRO-ARCHITECTURAL REFINEMENTS

Designers can incorporate smaller kernel sizes (e.g., 3×3 instead of 5×5) and fewer channels per layer. Layer fusion, where adjacent operations (e.g., convolution followed by batch normalization) are consolidated into a single kernel, further reduces memory transfers and overhead. These optimizations are often implemented at the compiler or hardware-mapping level to ensure minimal resource usage.

3) EARLY-EXIT (ANYTIME) NETWORKS

Beyond static compression (e.g., pruning and quantization), *early-exit* or *anytime* networks reduce *average* inference cost by allowing inputs to terminate early when intermediate predictions are already confident. BranchyNet augments a backbone model with lightweight side classifiers (''exits'') attached to intermediate layers; during inference, an input exits at the first branch whose confidence exceeds a threshold, otherwise it continues to deeper layers [45]. This enables a tunable accuracy–latency–energy trade-off at run time: easy inputs are processed quickly with minimal energy, while hard or ambiguous inputs receive additional computation.

For onboard workloads dominated by ''easy negatives'' (e.g., routine images with no events of interest), early-exit policies can significantly reduce average latency and power draw without changing worst-case behavior. The approach is complementary to hardware offload and quantization: exits can be implemented using the same deployment toolchain, while thresholds can be tuned to match mission-driven constraints (available power, downlink window, or response time).

4) TinyML FRAMEWORKS

Initiatives such as TensorFlow Lite Micro and CMSIS-NN cater to microcontrollers with sub-megabyte memory capacities. These frameworks provide a curated set of operations, memory managers, and scheduling policies that ensure inference can run within stringent RAM limits. TinyML also emphasizes model compression techniques (such as quantization and sparsity) as first-class citizens, further improving performance on embedded platforms.

5) TASK-SPECIFIC DESIGNS

Instead of training large, general-purpose networks, microsatellite missions commonly adopt application-focused models tailored to specific tasks (e.g., binary classification for cloud/no-cloud detection). This allows developers to reduce the network's depth and breadth by focusing strictly on the domain knowledge and input characteristics needed. Data augmentation and custom loss functions can further refine model capacity for mission-critical features.

C. DATASET OPTIMIZATION

Squeezing a neural network into a power-limited flight computer is only half the battle–the *training data* that shapes that network can itself be ''right-sized'' for embedded deployment. Techniques such as representative-subset selection, curriculum learning, and active learning prune millions of redundant samples while preserving class balance and corner-case coverage, cutting both training time and on-orbit fine-tuning costs. Resolution and spectral-band down-selection align the input dimensionality with the sensor actually flown, lowering the first convolutional layer's memory footprint. In low-data regimes, physics-based augmentation and synthetic scene generation (e.g., domain-randomized renderings of cloud fields or debris targets) boost robustness without inflating the dataset. Finally, dataset quantization–storing training images in 8- or even 4-bit fixed-point formats–keeps onboard rehearsal or continual-learning buffers small enough for microcontroller-class RAM. Together, these data-centric strategies complement model compression and hardware acceleration, ensuring the whole ML pipeline, from pixels to predictions, remains compatible with the tight SWaP envelope of micro- and nanosatellites.

D. HARDWARE ACCELERATION

Although microcontrollers can execute small neural networks, typical microsatellite missions require real-time or near real-time inference and more energy-efficient execution than a typical general-purpose microcontroller can deliver. While microcontrollers perform well in various scenarios, specialized ML accelerators are optimized to significantly outperform microcontrollers in these specific contexts. The use of ML accelerators dates back to 1993, when DSPs were used to speed up DNNs for optical character recognition. Subsequently, GPUs became popular due to their parallel processing capabilities [46].

Hardware accelerators in the realms of embedded systems provide an appealing solution by exploiting parallel data processing, specialized arithmetic units, and bandwidth-optimized memory architectures. Several accelerator classes are relevant for microsatellites.

1) DIGITAL SIGNAL PROCESSOR (DSP)

Originally designed for real-time signal processing tasks, DSP cores can expedite operations such as convolutions or matrix multiplications in embedded contexts. Some specialized DSP-based microcontrollers feature hardware loops and multiply-accumulate pipelines, achieving meaningful speedups over general-purpose microcontroller execution. However, limited on-chip memory and narrower data paths may constrain the size and precision of feasible ML models.

2) GRAPHICS PROCESSING UNIT (GPU)

With thousands of cores optimized for parallel integer or floating-point operations, modern GPUs excel at accelerating large-scale neural networks. While originally more common

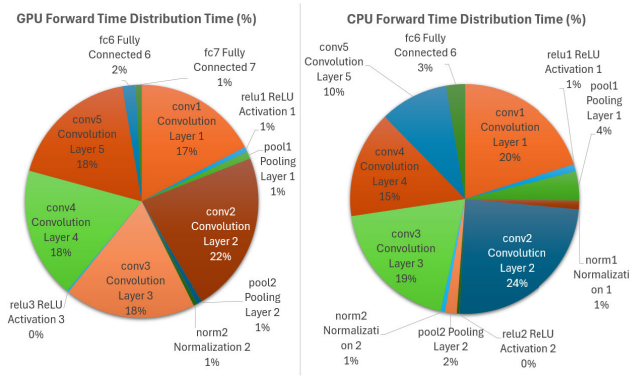

FIGURE 7. Layer-wise execution time comparison on a general GPU vs CPU (adapted from [47]).

in desktop or server environments, embedded GPUs (e.g., NVIDIA Jetson series) have been adopted in some larger microsatellite and small satellite missions. GPUs deliver impressive throughput but must be evaluated carefully for power constraints and heat dissipation in space. They are also more prevalent in 3U and 6U CubeSats, or larger, rather than minimal 1U designs, due to size and energy requirements.

Modern CPU and GPU architectures generally employ a degree of parallelization, such as single-instruction multiple-thread (SIMT) or single-instruction multiple-data (SIMD), that can handle MAC operations. By sharing control logic and memory across an array of arithmetic logic units (ALUs), these platforms can concurrently process the core computations of both fully connected and convolutional layers. Indeed, these layers are commonly recast as matrix multiplication tasks (i.e., kernel operations), allowing high-throughput parallel hardware to significantly accelerate DNN inference and training [34]. This makes GPUs widely utilized for both inference and training in ML due to their parallel architecture, which handles matrix operations efficiently. Companies like Nvidia and AMD dominate the market with CUDA Cores and Stream Processors, respectively.

Reference [47] benchmarked the forward pass of a model on a GPU and a CPU. Fig. 7 presents the resulting layer-wise time distributions. On the GPU, convolutional and fully connected layers consume a relatively balanced share of time (e.g. approximately 17–18% each for conv3–conv5 and fc6). On the CPU, the distribution is more uneven, with early convolutional layers dominating–conv2 alone accounts for 23.7% of the total. This contrast illustrates the parallel processing strengths of GPUs versus the more serialized execution patterns of CPUs and underscores why matrix-heavy layers benefit from GPU acceleration.

a: SOFT-GPU AND GPU-FOR-SPACE DIRECTIONS

Beyond COTS embedded GPUs, recent work is actively targeting GPU-like parallelism for space by (i) assessing embedded GPU roadmaps and space-relevant benchmarking (e.g., ESA's GPU4S initiative), and (ii) developing FPGA-based *soft-GPGPU* (General-Purpose GPU) overlays that expose an OpenCL-like programming model (e.g., GPU@SAT) to enable portable kernels on reconfigurable, potentially radiation-tolerant hardware. These approaches trade peak throughput for improved adaptability and a clearer path to space qualification while preserving the productivity benefits of the GPU software ecosystem [48], [49], [50], [51].

The high power consumption and high thermal output limit their use in space applications. Companies like ARM, AMD, and NVIDIA are developing embedded GPUs, but these devices still need to be evaluated for space use. Despite their adaptability, GPUs generally do not match the performance efficiency of an Application-Specific Integrated Circuit (ASIC) or FPGAs for ML tasks, as they are not specifically optimized for such applications [19].

3) FIELD-PROGRAMMABLE GATE ARRAY (FPGA)

FPGAs have long been favored in aerospace due to their *reconfigurability* and *customizable parallelism*. By implementing neural network operations as hardware circuits, an FPGA can efficiently map convolutional and fully connected layers into specialized pipelines, reducing the overhead of general-purpose processor instructions. This property makes FPGAs particularly appealing for real-time ML inference onboard microsatellites.

In an FPGA-based accelerator, developers can design a dataflow architecture that matches a neural network's dimensionality and layer connectivity. For instance, systolic arrays and deep pipelines can push large volumes of data through dedicated multiply-accumulate (MAC) logic in a streaming fashion. This reduces control overhead and memory bottlenecks, leading to substantial gains in throughput per watt relative to CPU- or GPU-based solutions.

FPGAs from Xilinx (AMD) and Intel often support partial or dynamic reconfiguration. This capability lets a mission upload new bitstreams or reconfigure sections of the FPGA while in orbit. If a network architecture evolves or if hardware faults are detected (e.g., from radiation events), the design can adapt without requiring a full system shutdown. Reconfiguration can also serve as a fault-mitigation measure, allowing for redundancy or fallback designs stored in the satellite's nonvolatile memory.

As noted in Section II, space radiation leads to SEUs and TID; for FPGAs, which are not inherently radiation-immune, some lines of devices (e.g., Xilinx's space-grade FPGAs, Intel's radiation-tolerant offerings) come with design-level mitigations for single-event upsets (SEUs). At the application level, triple modular redundancy (TMR) or error-correction strategies can harden the logic fabric. Coupled with partial reconfiguration, an FPGA can recover from localized faults, which is crucial for long-duration space missions.

FPGAs are prized for their reconfigurability, enabling developers to build custom hardware pipelines tuned to specific neural network layers. By implementing deep pipelines, systolic arrays, or parallel MAC engines, FPGAs

can drastically reduce inference latency and energy per operation. Additionally, partial reconfiguration allows an FPGA design to adapt to evolving mission needs or cope with radiation-induced faults. However, their deployment is slowed by the need for extensive design steps, including architecture design and model quantization [8]. Programming FPGAs for ML can be complex and typically requires domain expertise, high-level synthesis tools, and careful hardware/software partitioning.

FPGA-based solutions for ML applications are typically characterized by ''reduced inference time, and a higher possibility of customization, but at the cost of greater power consumption and a longer Time to Market'' [4]. On the other hand, they generally offer lower performance and higher power consumption compared to ASICs. Yet, modern FPGAs are increasingly used in space applications [19] and remain a strong option for neural network acceleration in space-edge ML applications. Originally prominent in digital prototyping, they have gained traction in defense and space contexts by effectively replacing custom ASICs in situations where production volumes are low and Time-to-Market must remain short. At the same time, their performance can rival that of alternative solutions. Recent advancements in vendor toolchains, combined with SoCs that integrate multicore processors alongside the FPGA-based accelerators, now enable deep neural network inference on reconfigurable hardware well suited to space missions [4].

Coarse-grained reconfigurable arrays (CGRAs) are increasingly viewed as a middle ground between fixed-function accelerators and fully general-purpose processors: they expose a programmable array of word-level processing elements and interconnects, aiming for ASIC-like energy efficiency while preserving software-level flexibility. Recent space-oriented work has started to evaluate CGRA fabrics specifically for onboard ML under strict SWaP and dependability constraints, including (i) highly parameterized CGRA templates to support design-space exploration for satellite ML workloads and (ii) low-power CGRA architectures targeting space-grade standard-cell libraries [52], [53]. In parallel, the supporting ecosystem is maturing through open toolflows that reduce adoption friction–e.g., unified CGRA modeling/evaluation frameworks and automated refinement loops that translate high-level kernels into CGRA implementations [54], [55]. As these toolchains and space-targeted prototypes mature, CGRAs (either as standalone accelerators or as CGRA-style overlays on radiation-tolerant FPGAs) are a plausible next step for missions needing more efficiency than embedded GPUs but more programmability than fixed accelerators.

4) VISION PROCESSING UNIT (VPU)

VPUs such as the Intel Movidius line include dedicated hardware for low-power image and video processing. They commonly feature optimized instructions for convolutions, pooling, and activation functions, making them well-suited for tasks like object detection or segmentation. VPUs also tend to have smaller footprints and more moderate power consumption compared to traditional GPUs, which can be a significant advantage for microsatellites. However, they may be less flexible if missions require non-vision tasks or frequent reprogramming of compute pipelines.

VPUs, such as Intel's Movidius Myriad series, enhance visual processing tasks like CNNs, SIFT, and Sobel filters. The Myriad 2 and Myriad-X VPUs use specialized Very Long Instruction Word (VLIW) processors called SHAVEs, which consume only a few watts, making them ideal for low-power applications. The Myriad 2 features 12 SHAVEs, while the Myriad-X includes 18, both providing superior performance over mobile CPUs or general-purpose low-power processors. The Myriad 2 has successfully passed radiation tests at CERN, confirming its suitability for space missions [8].

5) TENSOR PROCESSING UNITS (TPU)

TPUs are specialized hardware accelerators for tensor operations, which are fundamental in the TensorFlow framework by Google. The Coral Edge TPU, a notable example, excels in processing static images for tasks like CNN and Fully Convolutional Networks (FCN), achieving optimal performance using TensorFlow Lite and 8-bit integer quantization, despite potential impacts on model metrics. This TPU also supports in-hot reconfiguration, allowing updates to the model, weights, and hardware configurations through new GRAPH files [8].

6) NEURAL-PROCESSING UNIT (NPU)

To bridge the gap between ultra-low-power microcontrollers and more power-hungry GPUs/VPUs, Arm introduced the *Ethos-U65* neural-processing unit [56]. Designed as a licensable RTL block that pairs with Cortex-M33/M55 or even entry-level Cortex-A processors, the U65 delivers up to 1 TOPS within a sub-200 mW power envelope and occupies only a few mm^2 of silicon. Like its smaller sibling (Ethos-U55), it accelerates convolutions, depthwise separable filters, and activation functions common in CNNs, but scales to 128-512 MAC units and supports 8-/16-bit integers as well as mixed-precision execution. The tightly-coupled SRAM hierarchy enables zero-copy streaming of feature maps, while sparseness handling and weight compression reduce DRAM bandwidth–both critical for radiation-tolerant Low-Power Double Data Rate (LPDDR) devices typically used in small-sat avionics. For CubeSats and microsatellites, a U65 embedded in an SoC offers a middle ground: MCU-class determinism with an order-of-magnitude higher ML throughput, yet without the mass and power penalties of standalone VPUs or FPGAs.

7) ENERGY EFFICIENCY

Reference [19] analyzed the energy efficiency of a range of hardware architectures–quantified in GOP/J (Giga-Operations per Joule)–to guide the selection of embedded

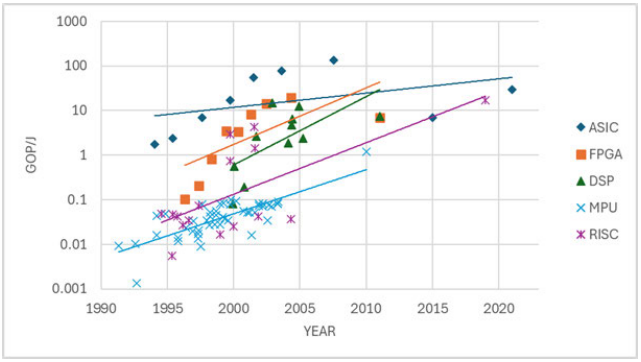

FIGURE 8. Energy efficiency (GOP per joule) of various hardware classes over time.

systems for spaceborne ML. Their results show that ASICs deliver the highest energy efficiency. FPGAs follow, typically around one order of magnitude lower, while DSPs occupy an intermediate position. General-purpose CPUs (MPUs) and RISC cores offer the greatest flexibility but at the cost of very low energy efficiency.

Fig. 8 summarizes energy efficiency trends across hardware classes (updated from [19]).

Fig. 8, updated from [19], shows the progression of energy efficiency (in GOP/J) across processor classes from 1990 through 2020. ASICs (solid diamonds) continue to demonstrate the most rapid and consistent growth, surpassing 100 GOP/J by the late 2000s. FPGAs (orange squares) show a strong upward trajectory as well, increasing from sub-1 GOP/J in the mid-1990s to over 10 GOP/J by the early 2000s and continuing to improve. DSPs (green triangles) follow a similar trend, clustering just below FPGA efficiency levels but rising more sharply after 2000. MPUs (blue crosses) and RISC cores (pink Xs) exhibit gradual improvement yet remain largely below 1 GOP/J throughout the two decades. The exponential trendlines reinforce the energy-efficiency gap between specialized accelerators (ASICs, FPGAs, DSPs) and general-purpose processors (MPUs, RISC), supporting the argument that domain-specific hardware offers substantial advantages in energy-constrained environments.

8) EMERGING TECHNOLOGIES FOR ACCELERATORS

Neuromorphic Computing is an emerging computing paradigm inspired by the brain's neural organization, aiming to deliver high efficiency (particularly in power and latency) compared with conventional ML accelerators. Neuromorphic hardware emulates neuron–synapse style computation and event-driven processing, which can reduce energy use and improve throughput for certain workloads. Although early research and industry prototypes have shown promising results, broad adoption remains at an early stage, partly because effective algorithms and design principles are still evolving alongside an incomplete understanding of biological intelligence [46].

Systolic arrays are a form of parallel architecture designed to execute high-throughput computations efficiently, making them particularly well-suited for the matrix and vector operations that are common in ML algorithms. They consist of a grid of processors, each passing data to their neighbors in a coordinated fashion similar to the human circulatory system, hence the name 'systolic'. This rhythmic data movement allows for the simultaneous processing of different data points, leading to highly efficient computation suitable for the repetitive and parallelizable tasks found in many deep learning models. Their structured data flow minimizes memory access costs, which in turn can lead to significant performance gains, especially in applications like convolutional neural networks. High Throughput and Energy Efficiency Systolic arrays are a form of dataflow architecture that pass data through a network of processing elements in a rhythmic, pulse-like manner. This design is highly efficient for matrix multiplications, providing high throughput and energy efficiency, which are critical for DNN processing [57]. When comparing SIMD and systolic array architectures, it is important to consider the trade-offs. SIMD architectures offer more flexibility and are easier to program, but they may not match the raw throughput and energy efficiency of systolic arrays. Systolic arrays excel in these areas but can be more challenging to implement and less flexible for different types of computations. [57]

The preceding subsections surveyed accelerator *classes* (DSP, GPU, FPGA, VPU, TPU, NPU). The following subsection maps these to *concrete COTS parts* and mission fit.

E. COMMERCIAL OFF-THE-SHELF (COTS) PROCESSING PARTS FOR ONBOARD ML

As outlined in Section II, microsatellites often favor COTS processors and accelerators for performance and cost within risk-mitigation strategies. We now survey mainstream hardware options–ranging from microcontrollers to integrated SoCs–that can host onboard ML workloads while respecting the tight SWaP margins of microsatellites.

1) MICROCONTROLLERS WITH ML EXTENSIONS

Microcontrollers (MCUs) have long been a mainstay in low-power embedded systems. The recent emergence of MCUs with dedicated *AI extensions* has dramatically expanded their potential for onboard inference. These specialized MCUs often include:

a: DSP/HELIUM INSTRUCTIONS

ARM Cortex-M processors (e.g., M4, M7, M33, M55) can integrate DSP extensions or the newer Helium vector instructions that accelerate matrix multiplications, convolutions, and other kernels ubiquitous in deep learning. When leveraged with carefully optimized firmware libraries (e.g., CMSIS-NN), such instructions significantly increase throughput without taxing the power budget.

b: INTEGRATED NN ACCELERATORS

Some MCUs go further by embedding small neural network co-processors or ML engines, each tailored for quantized operations (8-bit or lower). These co-processors can execute common layers (e.g., convolution, pooling, fully connected) at low precision, thereby boosting speed while minimizing energy usage. Examples include dedicated NPU blocks in select STM32 or NXP MCUs.

c: ON-CHIP MEMORY OPTIMIZATIONS

Given that MCUs have only a few hundred kilobytes to a couple of megabytes of RAM, ML-oriented variants often include features like tightly coupled memories (TCMs) and fast SRAM blocks that cut down on memory-access bottlenecks. The reduced latency and overhead can yield large performance gains for real-time ML inference tasks.

2) SoCs WITH INTEGRATED GPUs/VPUs (NVIDIA JETSON, INTEL MOVIDIUS, XILINX MPSoC)

System-on-chips (SoCs) that incorporate specialized graphics or vision processors have become an important stepping stone between basic microcontrollers and fully custom FPGA solutions. By bundling multiple CPU cores with on-die GPUs, VPUs, or ML-specific blocks, these SoCs offer significantly greater computational throughput for ML while attempting to maintain an embedded footprint. Several leading variants include:

a: NVIDIA JETSON FAMILY

Targeted at robotics and edge AI, Jetson modules (e.g., Nano, TX2, Xavier) feature a multi-core ARM CPU plus an NVIDIA GPU capable of high-throughput parallel operations. This enables real-time inference for convolutional neural networks, object detection, and other complex ML tasks. Although Jetsons typically demand more power than microcontrollers, their performance opens the door to advanced applications like real-time 3D reconstruction or onboard segmentation of hyperspectral data. Nonetheless, careful thermal and power management is essential, especially in smaller satellite form factors like 3U or 6U CubeSats.

b: INTEL MOVIDIUS MYRIAD VPUs

Intel's Movidius product line integrates specialized Vision Processing Units (VPUs) for low-power image analytics. These chips execute optimized hardware pipelines for convolutions, pooling, and activation functions, providing hardware acceleration without the overhead of a full GPU. Typically used for tasks such as onboard object recognition or face detection in consumer devices, Myriad VPUs are also attractive for space applications that need moderate throughput (e.g., 1-4 TOPS) while maintaining a strict power budget. Several CubeSat-scale prototypes have utilized Movidius for cloud detection and hyperspectral screening.

c: XILINX (AMD) MPSoC

The Zynq UltraScale+ MPSoC family combines ARM Cortex-A53 or R5 processors with reconfigurable FPGA logic and dedicated GPU-like engines in a single package. This architecture allows developers to partition their ML workloads between CPU cores, custom FPGA accelerators, or integrated graphics cores. The MPSoC approach is highly flexible–fine-grained hardware customization can be done on the FPGA side, while the ARM and GPU subsystems handle Linux-based applications. Power efficiency, however, depends heavily on project-specific implementation details (like clock gating, partial reconfiguration, and pipeline design).

Mobile-class SoCs (e.g., Qualcomm Snapdragon). Beyond VPUs and TPUs, mobile SoCs provide a compelling COTS route to onboard ML because they integrate heterogeneous compute (multi-core CPU, GPU, and DSP/NPU-class accelerators) with mature deployment stacks. Importantly, this class has demonstrated space relevance and heritage: NASA's Ingenuity Mars Helicopter employed a Qualcomm Snapdragon 801-based compute element to support vision-based navigation and autonomy, illustrating the feasibility of using mobile-class COTS silicon in extraterrestrial environments when risk is managed at the system level [58].

These SoC platforms have proven instrumental in reducing onboard inference times. However, their integration into microsatellites is more complex relative to microcontroller-only designs. Developers must consider:

- Additional power draw and cooling requirements for the GPU/VPU cores, which may not be feasible for 1U satellites.
- Radiation exposure and mitigation (see Section II).
- Board-level integration, including memory layout, high-speed interconnects, and software toolchains that support cross-compilation for ML inference.

Despite these challenges, SoCs with integrated GPUs or VPUs are a compelling option for larger CubeSats (3U, 6U, or beyond) that require on-orbit ML for tasks such as complex image classification, target tracking, or advanced scientific data analysis. Through tailored clock management and power gating, it is possible to capitalize on the high performance of these processors while still meeting the energy and thermal constraints of a small satellite environment.

3) FPGA-BASED SOLUTIONS (XILINX/AMD, INTEL, AND EUROPEAN SPACE-GRADE FAMILIES)

FPGAs offer extreme flexibility and customization, enabling tailored hardware accelerators while retaining post-launch reconfigurability. In space contexts, this flexibility is paired with availability of radiation-tolerant and radiation-hardened options from multiple vendors (e.g., AMD/Xilinx space-grade families, Microchip/Microsemi RT-class devices, and European space-qualified portfolios). These characteristics make FPGAs a recurring choice for onboard ML inference, where parallel MAC-heavy workloads benefit from custom datapaths and memory architectures, and where

fault-tolerance mechanisms (e.g., redundancy and configuration scrubbing) can be integrated at design time.

Modern **AMD/Xilinx** FPGAs dominate much of the published CubeSat ML accelerator literature, spanning COTS 7-series and UltraScale+ devices as well as space-grade parts. In particular, the Zynq family integrates ARM processors with FPGA fabric, supporting a mixed software/hardware approach where an OS and mission logic run on the CPU while ML kernels execute in hardware [14]. For higher-reliability missions, space-grade devices (e.g., Kintex UltraScale-class rad-tolerant parts) support on-orbit reconfiguration and sustained operation in radiation environments, enabling longer-lived deployments beyond short-duration COTS demonstrations [59], [60].

Intel (Altera) FPGA families (Cyclone, Arria, Stratix) also provide viable platforms for embedded ML, particularly via SoC variants that integrate ARM cores with programmable logic. While historically less common in flight demonstrations than AMD/Xilinx, they remain relevant for missions balancing performance-per-watt and development effort, and have been used in CubeSat-class experimentation for onboard image processing and classification [14], [60].

We additionally highlight *NanoXplore NG-Ultra / NG-Ultra RH*, which represents an increasingly prominent European FPGA/SoC direction. NG-Ultra RH is presented as a *rad-hard-by-design* FPGA/SoC platform (including an integrated processor subsystem) targeting space use [61]. In parallel, ESA has recently supported ecosystem maturation via an Open Space Innovation Platform (OSIP) activity explicitly aimed at adding *open-source* FPGA toolchain support for NanoXplore space-qualified devices, complementing NanoXplore's existing RTL-to-bitstream flow ("Impulse") and accelerating adoption by lowering tooling friction [62]. Collectively, these developments strengthen the case that FPGA-based onboard ML is shaped not only by raw compute density but also by *the availability of space-grade silicon and maintainable toolchains* across regional supply chains.

4) PROS AND CONS IN THE CONTEXT OF MICROSATELLITES

Selecting the most suitable computing platform for ML in a microsatellite environment requires balancing performance goals with stringent SWaP constraints, radiation reliability, and mission objectives. Table 3 presents a brief comparison of the major hardware categories.

In practice, mission teams frequently adopt a hybrid strategy: a low-power MCU for housekeeping, sensor interfacing, and emergency control, plus an accelerator (GPU, VPU, or FPGA) that only powers up when high-intensity ML tasks must run. The best choice hinges on factors like mission duration, target orbit, data volume, and the criticality of real-time processing.

TABLE 3. Comparison of hardware classes for enabling onboard ML in CubeSats and microsatellites.

Hardware Class	Pros	Cons	Typical Missions / Use Cases
MCUs with ML Extensions	Minimal power draw, low cost, relatively simple integration; sufficient for basic classification tasks (e.g., cloud or anomaly detection).	Limited memory and computational throughput; restricts model complexity and flexibility for advanced or evolving tasks.	Cloud detection, anomaly/event triggers, low-rate telemetry filtering in 1U-3U nanosatellites.
SoCs with Integrated GPUs/VPUs	Higher inference speeds on moderate-to-large models; supports advanced vision tasks; strong software ecosystem (CUDA for Jetson, OpenVINO for Movidius).	Greater power consumption and heat dissipation; typically requires at least a 3U CubeSat; robust thermal management is mandatory.	Proximity operations, visual navigation, image triage for 6U-12U microsatellites.
FPGA-Based Solutions	Customizable parallel pipelines; partial reconfiguration for adaptability or fault recovery; flexible numeric precision; strong performance/watt ratio if optimized.	Steep learning curve; longer design cycles; may require specialized rad-tolerant devices or SEU mitigation.	Onboard compression, cloud detection, meteor detection, real-time adaptive pipelines.
Radiation-Hardened or Tolerant Processors	Proven reliability in space; reduced risk of mission failure from hardware faults; often require minimal integration modifications.	Lag behind COTS devices in performance and efficiency; expensive and less available; limited access to cutting-edge ML accelerators.	Long-duration LEO/GEO missions, science payload controllers, high-reliability deep-space platforms.

F. SOFTWARE TOOLCHAINS AND FRAMEWORKS

Beyond algorithmic and hardware considerations, choosing the appropriate software stack is essential for effectively deploying ML models on constrained platforms. Over the past several years, numerous toolchains have emerged to streamline the creation, optimization, and deployment of neural networks in embedded contexts, including small satellites. Among the most notable:

1) TENSORFLOW LITE / TENSORFLOW LITE MICRO

Google's TensorFlow Lite (TFLite) is a popular solution for converting standard TensorFlow models into compressed representations suitable for edge devices. It supports post-training quantization, model optimization, and an interpreter specifically designed for low-power hardware. For microcontrollers with only a few hundred kilobytes of memory, TensorFlow Lite Micro further strips the runtime to a bare minimum set of core operations, allowing inference on devices as small as an ARM Cortex-M microcontroller.

TensorFlow is an open-source machine-learning framework widely used in various autonomous systems. TensorFlow Lite, introduced in late 2017, is a lightweight version designed for ML inference on embedded devices. The challenge for space applications is that these frameworks are typically used on powerful CPUs or GPUs on Earth, whereas microsatellites face significant constraints on size, weight, and power, which limits their processing capabilities and

makes it difficult to directly adopt these terrestrial designs for space use [17].

2) CMSIS-NN

Developed by Arm, the Cortex Microcontroller Software Interface Standard for Neural Networks (CMSIS-NN) provides highly optimized kernels for basic operations (e.g., convolution, fully connected layers, pooling) on Cortex-M processors. By leveraging SIMD instructions and memory-efficient layouts, CMSIS-NN achieves significant speedups and helps keep memory usage under tight constraints. Many modern frameworks and code generators can target CMSIS-NN backends seamlessly.

3) FLIGHT-SOFTWARE-INTEGRATED INFERENCE FRAMEWORKS

Most published onboard ML deployments remain mission-specific integrations of a given model into ad hoc hardware–software stacks. Polanco-Segovia et al. address this gap with the Neural Network Execution Framework (NNEF), a reusable software layer for generalized deep neural network inference atop standard flight software (demonstrated with NASA's Core Flight System, cFS). NNEF executes exported PyTorch/TensorFlow model scripts in C/C++ flight environments, supports ground upload of model updates, and integrates with spacecraft command and telemetry paths—aiming to decouple model development from platform-specific deployment on resource-constrained small satellites [63].

4) VENDOR-SPECIFIC FPGA TOOLCHAINS

For FPGAs, companies like Xilinx (AMD) and Intel offer high-level synthesis and specialized ML compilers (e.g., Xilinx Vitis AI, Intel OpenVINO) that can compile neural network models into hardware descriptions. These toolchains often incorporate quantization and pruning passes, automatically mapping computationally heavy layers into FPGA logic blocks or DSP slices for parallel speedups. While powerful, they generally require some hardware design knowledge, particularly if partial reconfiguration or custom IP blocks are to be integrated.

5) OTHER EMBEDDED ML FRAMEWORKS

Beyond vendor toolchains, several widely used *model-to-FPGA* and *ML-to-hardware* ecosystems have emerged that reduce development time and improve portability across SoCs and FPGA families.

Vendor-independent FPGA inference flows: Recent work such as *FPG-AI* provides a technology-independent framework that automates CNN deployment steps (e.g., quantization-aware model preparation and hardware-oriented integration), helping bridge the gap between ML development and FPGA implementation without locking the workflow to a single vendor stack [64].

Rapid prototyping via MATLAB/Simulink: In many aerospace/industrial workflows, MathWorks toolchains remain a common path for fast iteration and verification. In particular, *HDL Coder* and *Fixed-Point Designer* support fixed-point conversion and HDL-oriented design flows, and can be combined with deep-learning-to-hardware workflows (e.g., FPGA/SoC deployment paths) to accelerate prototyping and validation [65], [66], [67].

Automatic ML-to-HLS/RTL conversion: *hls4ml* is an established open-source workflow that translates trained ML models into FPGA-friendly HLS implementations, with hardware-aware optimizations (e.g., quantization, resource/latency tradeoffs) to support efficient inference on FPGA/SoC targets [68]. Complementing such flows, open-source compiler frameworks such as *Bambu*/*PandA* provide research-grade HLS infrastructure that can be integrated into automated hardware-generation pipelines [69], [70].

V. REVIEW OF THE STATE OF THE ART

The previous sections established why microsatellites stand at the forefront of efforts to deploy highly optimized ML in space. Yet, despite the commonalities in resource constraints, missions vary greatly in their technical approaches, algorithmic choices, and practical outcomes. To capture these details, we surveyed relevant publications, prototypical missions, and hardware demonstrations.

A. SELECTION CRITERIA AND PAPER CATEGORIZATION

To conduct our survey of the state-of-the-art literature on onboard ML for microsatellites, we initially performed systematic searches on prominent academic databases (Google Scholar, Scopus) using queries designed to capture the intersection of `"cubesat"` AND `"image processing"` with `"computer vision"` OR `"machine learning"`. We also recursively examined the references cited in each newly discovered paper, identifying additional studies if they met our broad interest in ML or computer-vision-based methods applicable to CubeSats or other microsatellites.

After assembling a preliminary set of publications, we defined five broad categories to organize the papers based on their application domains and hardware setups:

1) **Computer Vision on a Microcontroller (Generic Application)**: Papers focusing on resource-constrained microcontrollers running computer vision or ML tasks in domains not necessarily tied to space missions. This covers low-power, low-resource microcontroller units (MCUs) running Computer Vision (CV) algorithms for non-space use cases. Typical examples involve sensor nodes, robotics, or IoT devices.
2) **Computer Vision on a CubeSat (Space Classifiers)**: Studies directly applying CV or ML methods in orbit, typically running on hardware integrated into a CubeSat. Real or simulated flight scenarios fall under this category. In other words, these works

focus on deploying CV or ML algorithms *onboard an actual CubeSat or in a simulation directly tied to CubeSat missions*. They often aim to classify or filter imagery in flight (e.g., cloud detection, anomaly identification) to reduce downlink or enable some level of autonomy. The ''Space Classifiers'' label indicates that the classification tasks are directly performed in a space environment.

3) **Computer Vision on Ground Segment (Ground Classifiers)**: Works demonstrating computer vision or ML techniques for space imagery but executed on terrestrial computing resources, rather than onboard a satellite. Ground classifiers process data from satellites *on Earth*–either post-downlink or in near real time with a ground station.
4) **Image Processing Enabled Embedded Systems (Assisted Space Classifiers)**: Publications highlighting specialized accelerators used for ML or CV in a CubeSat or broader space context. Typically focuses on hardware and software integration for orbital or near-orbital processing. This category focuses on *space missions* that incorporate specialized accelerators (such as VPUs, TPUs, GPUs, or FPGAs) for onboard inference. These are the *''assisted''* or hardware-accelerated solutions, featuring more computational headroom than microcontrollers, thus allowing for more complex tasks.
5) **Image Processing Enabled Embedded Systems: VPU/TPU/GPU/FPGA (Generic Application)**: Generic demonstrations of hardware acceleration for ML or computer vision that, while not space-specific, provide relevant insights into performance, power consumption, or reliability on resource-constrained platforms. In contrast to Category 4, these are *non-space* uses of hardware accelerators for computer vision tasks in embedded environments–e.g., drones, autonomous vehicles, or industrial automation.

In essence, these five categories illustrated in Fig. 9 map out the boundary between generic embedded solutions and genuine ''space AI''–and between ''pure edge'' (onboard) vs. ''cloud or ground'' approaches. The **dark blue boxes** mark distinct subsets where *space missions* are in scope. Building on the platform scope of Section I, we restrict the literature below to studies that directly inform onboard ML for CubeSats and microsatellites. Section VI then orders these missions chronologically to show capability evolution. By establishing this taxonomy, the review systematically separates out less relevant works (e.g., purely ground-based or generic microcontroller CV) from those that directly inform the primary research goal: enabling ML on CubeSats.

In Fig. 9 the red-highlighted area of interest marks the subsets directly relevant to our *CubeSat edge AI scope*. The circles in the figure show the *number of papers* found for each category. A total of 37 articles were reviewed, of which only 26 were included in this study. In particular, Categories 2 (Space Classifiers) and 4 (Assisted Space Classifiers) represent the main inclusion criteria.

1) SCOPE BOUNDARIES AND SCREENING RULES (REPLICABILITY)

Consistent with the platform scope of Section I, this review targets *onboard* (space-edge) computer-vision and ML for CubeSats/microsatellites, with a focus on Category 2 (Space Classifiers) and Category 4 (Assisted Space Classifiers). We included studies written in English that (i) report *in-orbit*, *hardware-in-the-loop*, or *mission-tied simulation* results for CubeSat/microsatellite onboard processing, or (ii) present a space-relevant embedded accelerator integration intended for flight-like constraints (e.g., power/thermal/memory-limited payload computing). To ensure comparability, each included study had to report at least *two* of the following: target compute platform (MCU/SoC/VPU/NPU/TPU/GPU/FPGA), dataset/modality, performance metric (accuracy/F1/latency/throughput), or a stated SWaP-related constraint (power, memory footprint, duty cycle, or thermal envelope).

We excluded (i) purely ground-segment processing with no onboard deployment intent, (ii) generic embedded vision papers with no explicit space relevance, (iii) non-technical perspective pieces lacking reproducible implementation details, and (iv) duplicates/near-duplicates. Because reported metrics, datasets, and test conditions differ substantially across missions, we do not compute pooled effect size; instead, we standardize extraction fields (mission, application, model/toolchain, dataset/modality, metrics, and hardware) and present results as reported to enable transparent side-by-side comparison.

B. COMPUTER VISION ON A CUBESAT (SPACE CLASSIFIERS)

Under the *Space Classifiers* umbrella, we include studies that focus on actual or closely simulated onboard processing for CubeSat missions (listed in Table 4). These works typically validate computer vision or ML models in environments reflecting real orbital constraints–namely, limited power, radiation exposure, and intermittent communication.

Across the CubeSat-only corpus, two broad lines of work emerge. A first cluster targets *image triage for bandwidth relief* : the early IPEX flights proved that classical Random-Forest classifiers could already prioritize clear-sky scenes on a 400 MHz ARM9 CPU [24], [71]. Five years later, the AlAinSat-1 series transplanted far richer U-Net-style CNNs onto STM32 MCUs; after aggressive TensorFlow-Lite quantization, the models still achieve ∼85% accuracy while fitting in 2-4 MB of RAM [27], [72], [73]. OPS-SAT enlarges this thread by comparing four algorithms–from simple luminosity thresholds to U-Net–on a dual-core Cortex-A9, showing the classic accuracy-latency trade-off at the higher end of CubeSat power budgets [74].

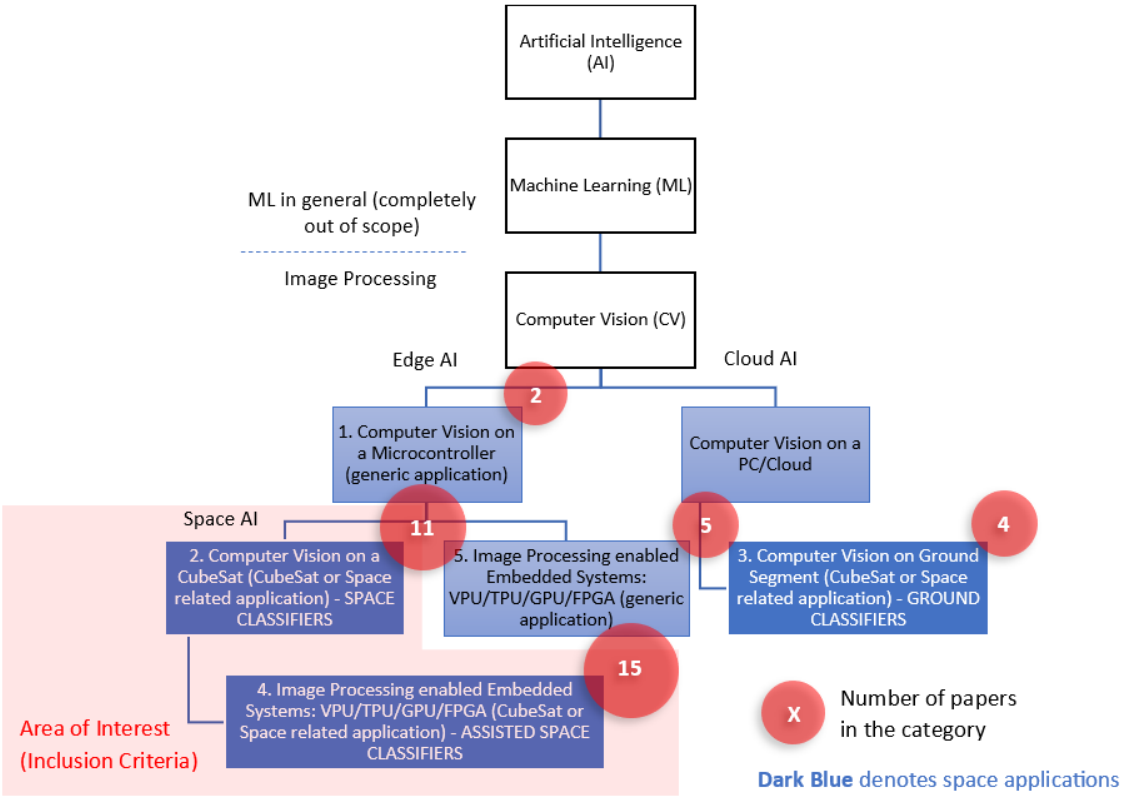

FIGURE 9. A hierarchical visualization of the categories used to classify the reviewed literature.

A second cluster is mission-driven autonomy. For navigation, Seeker-1 demonstrated 1 Hz bearing estimation of an uncooperative Cygnus target with a MobileNet Single Shot Detector (SSD) on an Intel Joule SBC [21], while synthetic-image training for Hera's Milani CubeSat shows CNNs outperforming centroid methods for asteroid pose estimation [22]. Optical-link pointing pushes the same idea earthward: a Raspberry Pi-based HSV/edge pipeline reliably finds beacon flashes to aid lasercom PAT subsystems [75]. Specialized science tasks also surface–OPS-SAT's plant-health module reaches 98% SVM accuracy on an Arduino-class controller [76], and a Jetson-TX2 proof-of-concept classifies ships at $\geq$98% while keeping power under 15 W for higher-end smallsats [15].

Common threads cut across these uses. All teams accept severe SWaP limits and therefore favor either tiny MCUs with integer quantization (AlAinSat-1 family) or modest ARM SoCs augmented by light optimization (OPS-SAT, Seeker-1). Accuracy above 85–90% is routinely reported; yet, the hardware split is stark: random forests and rule-based vision still dominate ultralow-power payload CPUs, whereas CNNs demand at least a mid-range ARM. Finally, almost every study stresses the same value proposition–local inference slashes downlink volume or enables time-critical decisions–confirming that onboard vision is no longer a novelty but an operational necessity for modern CubeSats.

C. IMAGE PROCESSING ENABLED EMBEDDED SYSTEMS: VPU/TPU/GPU/FPGA (ASSISTED SPACE CLASSIFIERS)

Under the *Assisted Space Classifiers* category, we consider studies that leverage specialized hardware accelerators–such as VPUs, TPUs, embedded GPUs, or FPGAs–to perform in-orbit ML tasks more efficiently than is possible with general-purpose CPUs or MCUs alone (listed in Table 5). While these systems share resource limitations with other CubeSats, they employ dedicated hardware logic to gain considerable improvements in inference speed, energy efficiency, or both. Recent work that augments CubeSat avionics with dedicated VPUs, GPUs, or FPGAs can be grouped by accelerator family.

VPU pipelines on PhiSat-1 show that an off-the-shelf Myriad-2 can execute a 2 MB CNN in 325 ms at 1.8 W, filtering hyperspectral cubes with 92% accuracy [8]. Follow-up benchmarking swaps that VPU for radiation-tolerant Zynq UltraScale+ and Kintex FPGAs: inference drops to 142 ms and energy per frame falls by 24%, at the cost of longer design cycles and slightly higher instantaneous power [4]. HyperScout-2's in-orbit demo confirms that the same VPU

TABLE 4. Category 2 (Space Classifiers): consolidated summary of onboard ML/CV studies (including mission-tied simulations when indicated).

Platform / status (year)	Task	Model / method	Data (modality)	Compute	Key result(s)	Key takeaway	Ref.
IPEX [Flight] (2015)	Visible-only cloud screening	Random Forest (texture-based)	Balloon train + ISS HDEV test (RGB)	Atmel ARM9 (+ Gumstix)	Targeting accuracy total: 79.2% (random forest) vs 36.5% (status quo)	Flight demo of RF cloud screening without calibrated multispectral data	[71]
IPEX [Flight] (2017)	Onboard triage + autonomy	RF/SVM + salience + planner	IPEX imagery (RGB; HS future)	Atmel ARM9 + Gumstix co-proc	Cls. acc. >95%	Early proof of onboard ML + autonomous ops under tight resources	[24]
N/A (SmallSat study) (2018)	Ship detection / classification	CNN pipeline	OpenCalifornia + Kaggle ships (RGB)	Jetson TX2 (SoC)	Acc. ∼98.3%; 7.5–15 W	COTS SoC enables higher-complexity onboard analytics within SWaP limits	[15]
Seeker-1 [Mission demo] (2020)	Proximity ops (bearing / target detect)	MobileNetV1-SSD (CNN)	Real + synthetic Cygnus (RGB)	Intel Joule 570x SBC	Acc. 96.9%; F1 0.972; 1 Hz	CNN vision nav is viable on CubeSat-class SBC for closed-loop assist	[21]
AlAinSat-1 ICU [Mission] (2021)	Cloud masking / triage	U-Net (quantized)	Landsat-8 + SPARCS (RGB)	STM32F746 MCU	Acc. ∼85% (fits ∼2 MB RAM)	CNN cloud masking on MCU is feasible (with accuracy trade-offs)	[27]
Hypothetical [Sim] (2022)	OGS beacon ID (lasercom PAT)	Classical CV (HSV + shape + Canny)	Lab photos (RGB)	Raspberry Pi 4 (ARM)	Acc. 86.6% (outdoors, 14:00); 93.3% (outdoors, 00:00)	Feasible low-cost beacon detection for OGS acquisition	[75]
AlAinSat-1 ICU payload [Mission] (2022)	Cloud mask + cloud/no-cloud	U-Net (quantized)	Landsat-8 + SPARCS (RGB)	STM32F746 (+ ext. RAM planned)	Acc. ∼84.5%; F1 ∼0.71	End-to-end ICU payload design for CubeSat-class downlink savings	[72]
OPS-SAT [Flight testbed] (2022)	Cloud segmentation (trade study)	U-Net / RF / K-means / threshold	OPS-SAT + Landsat (RGB)	Cortex-A9 (800 MHz)	U-Net: 7m13s, 89%, 90 MB; RF: 1m44s, 80%, 170 MB	Accuracy vs. time/memory is stark; motivates accelerators/lean models	[74]
Milani case study [Sim] (2022)	Asteroid optical nav	NN/CNN vs. centroid variants	10k synthetic Didymos (RGB)	Simulated	CNN best across CoF/PA errors (metrics in paper)	Data-driven methods outperform classical centroiding under hard geometry	[22]
AlAinSat-1 ICU impl. [Mission] (2024)	Cloud + image categorization	CNNs + U-Net; X-CubeAI path	Labeled sets (RGB; details n/s)	STM32F767 MCU	Categorization 4.5 s/img; decompr. 0.4 s (320×240)	Shows practical MCU deployment costs (latency dominated by pipeline)	[73]
N/A (CubeSat module) (2024)	Plant health monitoring	SVM + Otsu + HED	Ficus dataset (RGB)	Arduino Mega 2560	Acc. 98.2% (SVM)	Embedded CV can support life-science payload autonomy on MCU-class HW	[76]

can also drive VNIR+TIR analytics a few watts above baseline, proving the concept in flight [77].

GPU-class devices target data-rich missions that tolerate higher power. Jetson-TX1/TX2 boards accelerate dense SfM terrain reconstruction to map-level accuracy (<55 m RMS) within an hour for the 6U MOCI cubesat [25], push pattern-recognition kernels >14× faster than ARM cores in a 1-3U feasibility study [10], and support real-time object recognition for situational awareness [78]. Even a VideoCore IV on a Raspberry Pi can sustain beacon-tracking for lasercom pointing when algorithms are pared down to centroid extraction and color filtering [79].

FPGA accelerators dominate the literature because they marry custom precision with radiation resilience. A 4-bit FINN/Brevitas flow on a Zynq-7020 achieves 94% cloud-mask accuracy at 2.5 W and 15 fps [33], while ASRTU's MobU-Net couples an ARM9 with an FPGA JPEG2000 core to keep the entire segmentation-plus-compression chain below 1 MB and 38 ms Mpx $^{-1}$ [28]. OPS-SAT hosts several variants: a hybrid CNN-SNN that trades 12% fewer logic cells for a marginal latency penalty [80], highly-pruned C-FCN/UNet models that shrink parameters by $10^5\times$ yet sustain 95% cloud accuracy [81], and the SpaceMEDAL optical-flow kernel whose FPGA implementation cuts execution time by 5.2× compared with the ARM core [82]. Beyond single designs, ReCoN illustrates how partial re-configuration and TMR hardening can deliver 427.2× speed-ups and 210.6× energy gains over CPU baselines on Zynq MPSoCs while surviving neutron testing [23].

Finally, some investigations combine these themes: CubeSatNet proves that an MCU-only path is viable once extreme quantization shrinks a CNN to 104 kB with negligible accuracy loss, but such designs surrender throughput [26]. Overall, the literature converges on a clear hierarchy: GPUs offer raw throughput but challenge SWaP; VPUs provide turnkey low-power inference yet raise radiation concerns; FPGAs deliver the best latency-per-watt and fault tolerance when engineers can absorb lengthy HDL or HLS design cycles. Across all papers, quantization, depth-wise separable layers, or custom fixed-point formats are indispensable, ensuring that even accelerator-rich CubeSats stay within single-digit-watt power envelopes while meeting real-time orbital workloads.

D. CROSS-CUTTING THEMATIC ANALYSIS

To avoid repeating mission narratives across multiple sections, we re-map the reviewed studies onto nine **analytical themes** using a multi-label scheme (one study can contribute to multiple themes). The themes are grouped into mission-driven topics, algorithm/hardware levers, and system-integration concerns; Figs. 10 and 11 provide compact prevalence and evolution views.

TABLE 5. Category 4 (Assisted Space Classifiers): accelerator-enabled or compute-augmented onboard processing studies (mission-tied simulations marked where stated). Fields retained emphasize requirements, algorithm/hardware suitability, and headline in-orbit or prototype performance.

Platform / status (year)	Task	Model / method	Data (modality)	Compute	Key result(s)	Key takeaway	Ref.
N/A [Study] (2017)	SSA: small-sat object ID	CNNs (AlexNet/GoogLeNet)	ImageNet (RGB)	Jetson TX1 (GPU SoC)	mAP/FPS n/s	Early case for COTS GPU SoCs enabling onboard DL for SSA-class tasks	[78]
ASRTU [Sim] (2018)	Cloud mask + compression	MobU-Net + JPEG2000	Landsat-8 SPARCS (MS: RGB+NIR)	ARM9 + FPGA (JPEG2K)	Acc. 93.1%; 0.0378 s/MPix; 0.713 MB	Compression + lightweight DL can hit SWaP targets on embedded platforms	[28]
STP-H5/CSP (ISS) [Flight demo] (2018)	Onboard image filtering	MobileNetV1/V2 etc.	∼8000 ISS images (RGB)	Zynq-7020 (SoC/FPGA)	Top-1 ∼89.5%	TensorFlow/TFLite-class CNNs are practical on flight-like SmallSat compute	[17]
PhiSat-1 [Flight] (2019)	AI for VNIR/thermal EO (incl. cloud)	NN (details n/s)	S2/L8-derived synthetic (HS+TIR)	Dual-core CPU + Myriad2 VPU (+GPU)	n/s	Mission-level proof of onboard AI payload chain; results often reported qualitatively	[77]
OPS-SAT [Sim] (2019)	Low-power segmentation	C-FCN/C-UNet variants	38-Cloud, CloudPeru2, EOLearn (RGB)	Cyclone V SoC FPGA	Up to 95% (cloud), 83% (forest)	Co-designed compact nets make segmentation FPGA-feasible under tight budgets	[81]
N/A [Sim] (2019)	Hybrid SoC segmentation	SegNet + ReCoN accel	ISPRS Potsdam (RGB/IRRG)	Zynq7 / ZynqMP + FPGA	Acc. ∼90%; up to 427.2× speed; 210.6× energy	Reconfigurable CNN acceleration + fault-tolerance (TMR/rad testing) is a key path	[23]
Concept [Sim] (2019)	Onboard HPC for EO pipelines	Mixed (HPC kernels)	Mixed / n/s	Jetson TX1 (GPU SoC)	∼14× accel vs A57 CPU	Argues feasibility of GPU-class compute for EO preprocessing under CubeSat SWaP	[10]
BIRDS-3 [On-orbit data] (2020)	Downlink triage (good/bad)	Tiny CNN (CubeSatNet)	60k imgs + BIRDS-3 (RGB)	STM32H7 MCU	Acc. 90%; F1 0.92; 0.25 s/img	Ultra-light CNNs can run on MCU even in "assisted" mission stacks	[26]
PhiSat-1 / HyperScout-2 [Sim+deploy] (2020)	Cloud screening (HS)	CloudScout CNN	S2 → HyperScout-2 (HS via PCA)	Myriad2 VPU	Acc. 92%; 325 ms; 1.8 W; 2.1 MB	VPU-class edge AI enables watt-level onboard screening with deployable models	[8]
OPS-SAT [Sim] (2020)	FPGA image classification	Hybrid CNN+SNN	Custom 28×28 patches (RGB)	Cyclone V SoC FPGA	Acc. 87%; 43 μs/img; ∼1.19 W	Hybrid NN trades latency for logic savings; shows design-space beyond CNN-only	[80]
N/A (MOCI) [Sim] (2021)	3D recon / SfM (batch)	SIFT+SfM+BA (CUDA)	BlenderGIS/ASTER sims (RGB)	TX2/TX2i (GPU SoC)	35 min–1.4 h; 15–100 m recon error	High-end onboard analytics are feasible but episodic (minutes–hours)	[25]
N/A [Prototype] (2020)	Optical comms fine pointing	Classical CV tracking	Lab/field imgs (RGB)	Raspberry Pi 3B (+GPU)	n/s	Proof-of-concept for low-cost CV-based beacon tracking for lasercom	[79]
PhiSat-1 (CloudScout) [Sim] (2021)	CloudScout accel comparison	Quantized CloudScout CNN	S2-derived (MS: RGB+NIR)	FPGA (ZCU106/KU060) vs Myriad2	141 ms @1.65 W (FPGA) vs 346 ms @1.8 W (VPU)	Space-grade FPGA improves latency/energy per inf. vs COTS VPU; longer dev time	[4]
CloudSatNet-1 [Sim] (2022)	Cloud coverage cls. (≥70%)	Quantized CNN (4-bit)	L8 biome (RGB)	ARM A9 + Zynq-7020 FPGA	Acc. 94.4%; 15 fps; 2.5 W	Aggressive quantization + FPGA parallelism yields real-time cls. at low power	[33]
SOURCE [Mission] (2022)	Meteor detection (real-time)	Optical flow + blobs (CV)	ArtMESS + ISS meteor vids (RGB)	Dual-core A9 + FPGA	F1 0.86; 3× overall; 5.2× optical-flow accel	Classic CV + FPGA offload is effective for tight real-time onboard pipelines	[82]

The nine analytical themes used in the cross-cutting synthesis are defined as follows:

- **Onboard Image Triage and Cloud Detection:** Techniques for prioritizing, filtering, or discarding Earth-observation imagery onboard (often via cloud screening or quality assessment) to reduce downlink load and increase returned data utility.
- **Visual Navigation and Proximity Operations:** Vision-based estimation and guidance tasks (e.g., target detection/tracking, relative pose/position inference) supporting rendezvous, formation flying, docking, optical navigation, or proximity operations under challenging illumination and contrast conditions.
- **Radiation and Reliability Considerations:** Evidence and strategies addressing space-environment robustness (e.g., SEU/SEL effects, fault injection, resilience measures such as ECC/TMR, graceful degradation, or recovery mechanisms) for onboard ML and supporting compute hardware.
- **Resource-Aware ML Techniques:** Algorithmic and implementation methods that reduce memory, compute, and energy demands (e.g., pruning, compression, operator fusion, efficient architectures, embedded inference libraries) to meet stringent SWaP constraints.
- **Leveraging Hardware Parallelism:** Use of parallel compute substrates (e.g., VPUs, GPUs, FPGAs,

multi-core SoCs) to accelerate ML and vision primitives (convolutions, matrix operations) and enable higher throughput or more complex models within mission constraints.

- **Quantization and Custom Precision:** Reduced-precision inference and tailored numeric representations (e.g., INT8/INT4, mixed precision, custom fixed-point formats on FPGAs) employed to reduce memory traffic and power while maintaining acceptable accuracy.
- **Adaptive and Reconfigurable Designs:** Architectures that modify model/hardware behavior across operating modes or in-flight (e.g., FPGA partial/dynamic reconfiguration, switching between model variants, runtime resource scaling) to respond to power availability, mission phase, or fault conditions.
- **Use Cases and Mission Profiles:** The mission objectives and operational context driving onboard ML adoption (e.g., downlink reduction, event detection, terrain reconstruction, segmentation, continuous monitoring), including how performance requirements are shaped by orbit, payload, and concept of operations.
- **Integration Challenges:** Practical system-level constraints encountered during deployment (e.g., toolchain and software-stack complexity, verification and validation, thermal/power integration, memory and I/O bottlenecks, radiation-aware design choices, and interfacing with spacecraft avionics).

Fig. 10 shows that **hardware parallelism**, **mission-profile framing**, and **EO triage** dominate, while **radiation/reliability**, **adaptive reconfiguration**, and **vision-based proximity operations** remain sparse. The resulting imbalance is the central message: the field has progressed faster in capability demonstrations than in assurance-grade evidence.

Fig. 11 complements this view at the study level: earlier papers are narrower (often mission motivation + platform feasibility), while newer works combine more deployment-oriented themes. However, even in recent years, reliability/assurance themes are underrepresented relative to performance-centric themes.

Overall, the survey confirms that embedded ML is already transforming how CubeSats handle onboard data, from basic classification to sophisticated, real-time computer vision. Meanwhile, it underscores the ongoing need for robust, power-aware, and radiation-resistant solutions. The next section (Section VI) examines these findings through a timeline of flight missions that have piloted or operationalized these approaches.

VI. TIMELINE OF NOTABLE CubeSat MISSIONS WITH ML CAPABILITIES

Section V presented these missions by category; we order them here chronologically to show evolution. The missions in Table 6 reveal a clear progression driven by downlink bottlenecks, SWaP limits, and autonomy requirements. We group this into four overlapping phases: **(I)** classical ML and onboard triage on CPU-class processors; **(II)** lightweight CNNs on embedded CPUs/SBCs; **(III)** the accelerator era (VPU/FPGA/GPU); and **(IV)** multi-model, reconfigurable, and closed-loop autonomy. The entries below are milestones along that trajectory.

A. EARLY DEMONSTRATIONS OF ONBOARD PROCESSING

IPEX (2013): NASA's Early in-flight ML triage (random forest + salience on ARM9) for downlink prioritization [24], [71] (Phase I).

BIRDS (2017–2020): 1U CubeSats, lightweight NNs for good/bad image detection [26] (Phase II).

B. SIGNIFICANT BREAKTHROUGHS IN ONBOARD ML

Seeker-1 (2019): NASA's 3U CubeSat demonstrated autonomous proximity operations with Cygnus using a monocular camera and MobileNet SSD on Intel Joule 570x, achieving >96% detection at 1 Hz [21]—at the Phase II–III boundary.

OPS-SAT (2019): ESA's in-orbit testbed runs ML demos for cloud detection, anomaly detection, and attitude determination [74], [80], [81]; open access supports rapid iteration (Phase III–IV).

PhiSat-1 (2020): ESA's HyperScout-2 mission integrated a Myriad 2 VPU for real-time cloud detection on hyperspectral imagery, discarding cloud-covered frames before downlink—one of the first in-orbit CNNs on a dedicated ML chip [4], [8], [77] (Phase III).

SOURCE (2021): On the SOURCE 3U CubeSat, SpaceMEDAL offloaded optical-flow meteor detection to an onboard FPGA, achieving 5.2× optical-flow speedup and 3× pipeline acceleration; detection performance is reported as precision 94.68%, recall 78.76%, and F-measure 0.8599 [82] (Phase III).

C. RECENT AND PLANNED MISSIONS

Intuition-1 (2023): This 6U Polish nanosatellite uses KP Labs' Leopard (Xilinx-based SoC) for onboard hyperspectral analysis—cloud filtering, anomaly detection, and land-cover classification—before downlink [83] (late Phase III).

Milani (2024): ESA's 6U Hera CubeSat for Didymos proximity ops combines centroiding with CNNs for center-of-figure and phase-angle estimation; synthetic training showed CNNs outperforming classical methods [22], extending the trajectory beyond LEO.

PhiSat-2 (2024): ESA's 6U successor to PhiSat-1 carries a Myriad 2 VPU and runs modular "apps" (cloud/ship detection, compression, map generation) with in-orbit reconfigurable workflows [83] (Phase IV).

CogniSAT-6/HAMMER (2024): Dynamic Targeting flight tests reported autonomous look-ahead analysis and retargeting decisions without human involvement on CogniSAT-6/HAMMER-class demonstrations [84], [85] (Phase IV).

Pony Express 2 (2024): Lockheed Martin's pair of 12U CubeSats use UltraScale-class processors and a

TABLE 6. A chronological backbone for onboard ML in small satellites: missions cluster into phases reflecting a progression from classical triage to accelerator-enabled and reconfigurable multi-model autonomy.

Phase	Representative missions (this review)	Typical ML	Typical compute	Milestone contribution
I	IPEX (2013)	Classical ML (e.g., forests) + autonomy logic	CPU-class OBC	Onboard filtering/triage to reduce downlink and enable responsiveness
II	BIRDS (2017–2020), Seeker-1 (2019)	Lightweight CNNs / compact classifiers	Embedded CPU / SBC	Practical onboard vision autonomy under tight SWaP
III	PhiSat-1 (2020), SOURCE/SpaceMEDAL (2021), Intuition-1 (2023), MOCI (planned)	Quantized CNNs + HW/SW partitioning	VPU / FPGA / GPU offload	Step-change in throughput/latency making onboard DL operationally relevant
IV	PhiSat-2 (2024), CogniSAT-6/HAMMER (2024), Pony Express 2 (2024)	Multi-model pipelines, app frameworks	Heterogeneous compute + reconfiguration	Reconfigurable "AI services": multi-app autonomy, retargeting, distributed processing

software-defined platform aimed at in-orbit data analytics/AI and autonomous teaming or cloud-networking concepts [86] (Phase IV).

AlAinSat-1 (2025): The 3U UAE/Al Ain University CubeSat hosts an Image Classification Unit (ICU) for onboard cloud segmentation and filtering with lightweight ML [27], [72], [73] (Phase II/III diffusion).

MOCI (planned 2025): This mission will demonstrate GPU-accelerated structure-from-motion on a Jetson TX2 for onboard 3D terrain reconstruction and reduced downlink [25] (Phase III).

Taken together, these missions show onboard ML evolving from early triage through accelerator-enabled inference to reconfigurable multi-model and closed-loop autonomy; the remainder of the review uses them to contextualize techniques, hardware, and open challenges.

VII. DISCUSSION AND SYNTHESIS

This section synthesizes insights from the literature: design patterns, trade-offs, and open challenges. Missions show diverse hardware-software configurations; early work proved modest MCUs can enable useful autonomy with optimized models, while FPGAs and embedded GPUs enable demanding tasks (e.g. SfM, high-res segmentation) only when power, thermal, and integration are addressed up front.

A. DESIGN PATTERNS, ENABLERS, AND BOTTLENECKS FOR ONBOARD ML

Sections II and IV introduced constraints and levers; we synthesize recurring *design patterns*. Three dimensions dominate: algorithm choice, compute substrate, and system integration. **Algorithmic patterns:** Classical CV and shallow ML suit low-complexity, deterministic tasks; deep networks (CNNs) outperform for diverse conditions and remain deployable after compression. Many flight pipelines are hybrid: classical pre/post-processing around a compact learned core. **Compute substrate:** VPUs/NPUs offer the fastest path to watt-level inference but face operator support, toolchain, and (for COTS) radiation limits; embedded GPUs enable rapid prototyping and heavier models but need thermal/duty-cycle management; FPGAs give custom pipelines and radiation-aware design (scrubbing, redundancy) with high efficiency per watt at the cost of development time and verification. **Compression** (quantization, pruning, distillation, compact architectures) is the key enabler from research model to flight artifact; 8-bit and 4-bit often preserve accuracy within a few points. **System-level:** CONOPS must align with compute budgets and decision deadlines; thermal/radiation and fault mitigation must match decision criticality; toolchains must support real-time operation, coverage checks, and fault containment (watchdogs, checkpointing, fallbacks). Successful missions *co-design* algorithms, hardware, and mission objectives from the outset, treating SWaP and assurance as first-class requirements.

TABLE 7. Hardware-driven phases that shaped the evolution of onboard ML for small satellites.

Hardware phase	Typical onboard compute	What became feasible onboard	Dominant constraints
CPU-only triage	ARM9/LEON-class CPUs; simple SBC CPUs	Classical CV + classical ML (RF/SVM), saliency, basic autonomy and filtering	Tight RAM/CPU budgets; slow FP; limited real-time margin
SBC prototyping	Pi-class ARM/x86 SBCs; early embedded SoCs	Lightweight CNN inference at Hz-rate, early vision autonomy demos	Thermal + duty-cycle power; software robustness; limited radiation tolerance
COTS heterogeneous acceleration	VPUs/NPUs/DSP-class accelerators; mobile SoC engines	CNN segmentation/detection at watt-level power; multi-kernel pipelines	Operator/layer support; quantization/toolchains; component survivability
Space-grade heterogeneous compute	FPGA/MPSoC pipelines with ECC/TMR/scrubbing mixed-criticality partitioning	Sustained onboard analytics and adaptable workloads; higher TRL autonomy building blocks	Verification effort; integration complexity; cost/lead-time; qualification burden

B. TECHNOLOGICAL TURNING POINTS AND HARDWARE EVOLUTION

Onboard ML progressed via *hardware-enabled jumps*: (1) classical triage on CPUs (early 2010s); (2) TinyML and

TABLE 8. Order-of-magnitude requirement envelopes for onboard ML, anchored in published mission measurements. Ranges reflect spread across heterogeneous tasks and hardware (MCU-class TinyML to accelerator/SoC-class deployments), and should be interpreted as engineering bounds rather than universal constants.

Scenario (future-facing)	Typical input	Latency target	Throughput target	Memory / footprint	Power	Anchors (examples)
Downlink triage (binary cls.)	RGB patch/frame (224–227 px)	MCU: ∼0.25 s/image; Accel: O(10 ms) (task/model dependent)	MCU: ∼2–5 fps; Accel: up to O(10^2 fps) (heterogeneous workloads)	Model: 0.1–2 MB; RAM: 0.1–10 MB (mixed model/pipeline reports)	0.1–0.5 W (MCU); 1–2 W (VPU/DSP)	BIRDS/CubeSatNet; SBC-2 Snapdragon/Myriad [13], [26]
Cloud detection / masking (EO)	RGB/MS/HS tiles or frames (0.3–1 MP)	0.1–0.35 s per inference; ∼0.04 s/MPixel (tiling)	3–15 fps (tiles); full-frame decision $\lesssim$ seconds	Model: 0.7–2.1 MB; RAM: 1–100 MB (pipeline)	1.6–2.5 W	PhiSat-1 CloudScout; FPGA vs Myriad [4], [8]; JPEG+DL [28]
Onboard EO analytics (object/event det.)	RGB tiles/frames (224–640 px)	Task-dependent latency (illustrative envelope; workload dependent)	Task-dependent throughput (reported values vary by pipeline/model)	Model: 5–50 MB; RAM: 0.5–4 GB	7.5–15 W	Jetson/COTS GPU case studies [15], [78]
Vision nav / proximity ops (GNC-assist)	Monocular frames (VGA–MP)	End-to-end $\lesssim$1 s in demonstrated cases; tighter sub-second targets are design dependent	1–10 Hz (task dependent)	Model: 5–50 MB; RAM: 0.5–2 GB	5–15 W	Seeker-1 (1 Hz); Milani benchmarking [21], [22]
3D reconstruction / SfM (batch)	Tens–hundreds of images/scene	Minutes–hours per scene	Episodic/offline	GB-scale RAM + storage	TX2/TX2i-class operating envelope (mode dependent)	MOCI on TX2/TX2i [25]
Closed-loop autonomy (detect→retarget)	Wide-FOV look-ahead + high-res ROI	Decision-to-action <90 s (often ∼60–90 s in reported tests)	Episodic (per opportunity)	Multi-model/app framework; storage for logs/checkpoints	Mission/platform-dependent (power envelope not uniformly reported)	CogniSAT/HAMMER; PhiSat-2-class apps [84], [85]

efficient CNNs (mid–late 2010s); (3) COTS accelerators (VPU/NPU/TPU); (4) QAT and deployment toolchains; (5) space-grade FPGAs and partial reconfiguration; (6) multi-model, app-driven autonomy (early 2020s–present). Table 7 summarizes phases; Technology Readiness Level (TRL) is used in the table. The forward trend is *heterogeneous, updateable compute* with mixed-criticality isolation and deterministic fallbacks.

C. MISSION-DRIVEN DYNAMICS AND RECONFIGURABLE ONBOARD AI

Shorter build cycles and higher launch cadence [87] make in-orbit COTS and compressed-model demos viable and push toward update-friendly flight software (downlink → retrain → uplink) [16]. The field has shifted from single-purpose inference to *multi-purpose, reconfigurable* AI (PhiSat-2 app paradigm [88], [89]), requiring orchestration, fault containment, and lifecycle assurance. Dynamic targeting (e.g. ∼74 s look-ahead at 500 km LEO [18]) relies on rapid onboard analysis.

D. QUANTITATIVE REQUIREMENT ENVELOPES FOR ONBOARD ML APPLICATIONS

Table 8 summarizes *engineering requirement envelopes* for representative onboard-ML applications: feasibility is dominated by latency, throughput, memory, duty-cycle power, and decision criticality, not accuracy alone.

The envelopes are intentionally **order-of-magnitude** and heterogeneous: they aggregate measurements from different tasks, models, and hardware configurations. Unless a row is tied to one benchmarked pipeline (e.g., CloudScout family), ranges should be interpreted as design-space bounds rather than strict guaranteed operating points. We therefore distinguish measured mission values from platform-mode envelopes where papers report board mode/TDP but not full system-level power under identical workloads.

Autonomy is deadline-driven; outputs should be confidence-gated with watchdog supervision and a deterministic non-ML fallback.

E. UNIFIED CROSS-DOMAIN TRADE-SPACE AND DECISION FRAMEWORK

Table 9 provides the paper's signature synthesis artifact: a unified cross-domain and cross-layer trade-space matrix that maps each application class to its binding constraints, algorithm/hardware fit, and assurance implications under CubeSat constraints.

Table 9 and Fig. 12 constitute the main synthesis contribution of this review: the table captures **what fits where and why**, while the figure provides a reviewer-oriented **how-to-decide** flow for early-phase mission trade studies.

F. DERIVED DESIGN TRADE-OFFS FROM THE REVIEWED EVIDENCE

To make trade studies actionable, we distill the recurring *quantitative* and *architectural* tensions into

TABLE 9. Signature contribution of this review: unified cross-domain and cross-layer trade-space matrix for onboard ML in CubeSat/microsat systems, linking mission objective, binding constraints, ML strategy, compute substrate, bottlenecks, and assurance implications in a single design artifact.

Domain	Primary objective	Binding system constraint (typical)	Preferred ML strategy	Preferred compute substrate	Common bottleneck / constraint	Assurance / deployment implication
Cloud detection / masking (EO)	Discard low-value data; enable faster EO products	Energy per decision; memory movement; robustness across scenes	Compact CNN/U-Net; quantized segmentation with classical fallback thresholds	MCU for coarse triage; VPU/FPGA for segmentation throughput	Dataset shift; operator coverage; tiling/buffering costs	Define conservative confidence thresholds to avoid discarding high-value data
Downlink triage / quality filtering	Prioritize "good" frames for downlink	Model footprint + RAM; tolerance to moderate error	Tiny CNN or classical ML (RF/SVM) with lightweight heuristics	MCU/SBC; optional VPU	Overfitting to local scene statistics; limited labeled on-orbit data	Require policy-level thresholds and human-auditable reject criteria
Hyperspectral triage / onboard feature extraction	Reduce cubes; extract features before downlink	Memory bandwidth; data volume; latency under duty-cycle limits	Band reduction (e.g., PCA) + compact CNN/hybrid pipeline	VPU/FPGA; occasionally GPU SoC	DRAM traffic can dominate energy; preprocessing overhead	Benchmarking must include I/O and preprocessing, not model latency alone
Vision navigation / proximity operations	Relative bearing/pose cues for guidance	Deadline-driven latency; determinism; failure containment	Hybrid CV+ML detector/pose pipelines with bounded depth	SBC/SoC with strict scheduling; FPGA for deterministic segments	Safety criticality and worst-case timing dominate	Requires confidence gating, watchdog supervision, and deterministic fallback
Anomaly detection / health monitoring (payload or bus)	Detect off-nominal behavior to trigger response	Very low false-alarm rates; interpretability; continuous operation	Lightweight classifiers, statistical+ML hybrids, or compact autoencoders	MCU/CPU; sometimes FPGA	Data drift; threshold tuning; sparse fault labels	Needs calibration/drift monitoring and update governance before autonomy use
Closed-loop autonomy (detect→retarget)	Improve yield via onboard "observe–decide–act"	End-to-end decision time; multi-model orchestration	Multi-model pipelines; app frameworks; confidence gating	VPU/SoC; sometimes FPGA/GPU	System integration: scheduling, thermal duty-cycle, assurance	Verification burden shifts to system-level behavior under off-nominal cases
3D reconstruction / heavy analytics (batch)	Generate products onboard; reduce downlink load	Total energy/time per scene; storage/DRAM	Feature pipelines/SfM; learned features in some cases	GPU SoC	Thermal + power duty cycle; checkpoint/restart required	Treat as opportunistic workload isolated from mission-critical control loops

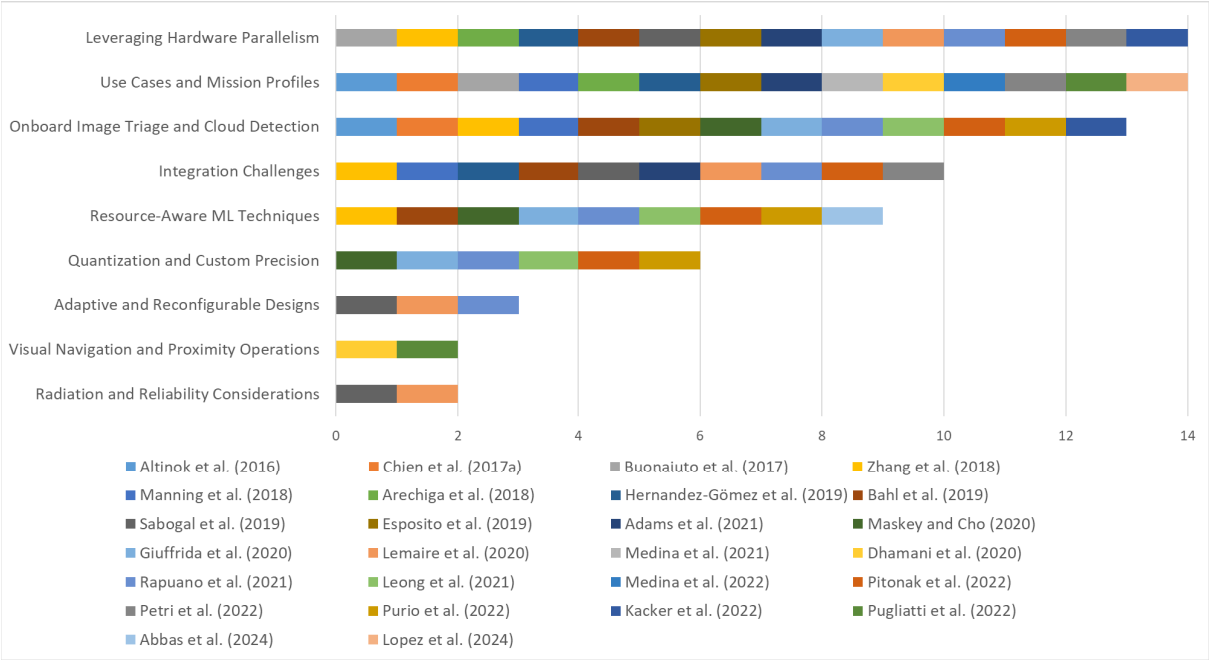

FIGURE 10. Thematic coverage across the reviewed studies mapped onto nine cross-cutting themes. Each colored block corresponds to one paper (see legend); the total bar length indicates how frequently a theme is addressed.

reviewer-auditable statements. These are not universal constants; they are source-anchored tendencies across heterogeneous missions and testbeds.

Trade-off A: latency margin vs deployment simplicity. MCU-class deployments can satisfy low-rate triage workloads (e.g., ∼0.25 s/image in CubeSatNet), but tighter reaction loops generally require accelerator support (e.g.,

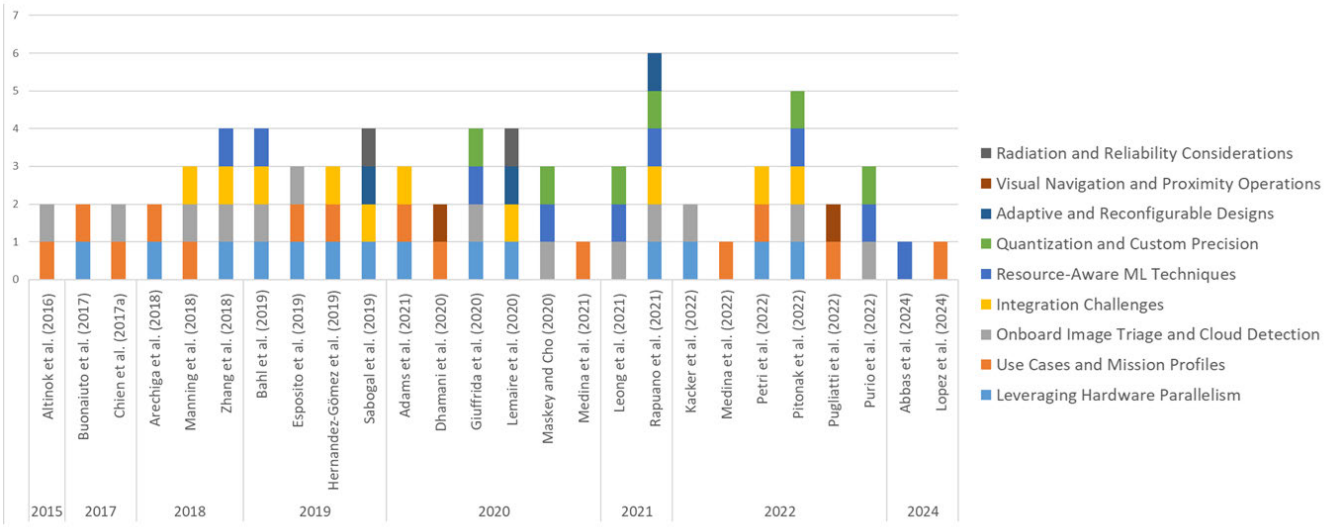

FIGURE 11. Per-study thematic breadth and composition over time. Each vertical stacked bar corresponds to one reviewed paper (x-axis, grouped by publication year), with colored segments indicating which of the nine themes are addressed. The bar height gives the total number of themes covered by that paper.

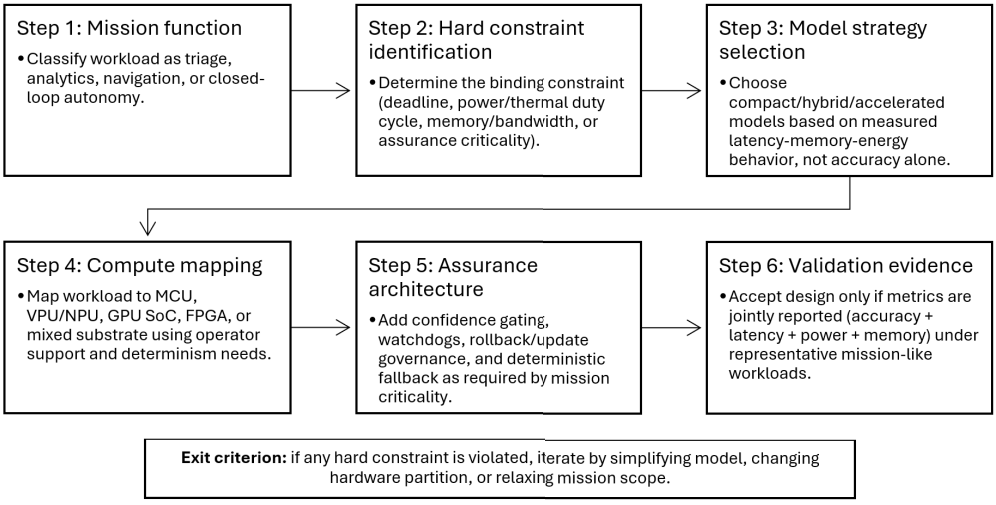

FIGURE 12. Reviewer-oriented decision flow for translating mission needs into deployable onboard-ML architectures. The flow is grounded in recurring patterns across the surveyed mission demonstrations and benchmark studies.

O(10 ms) classifier latency on SBC-2 benchmarks) [13], [26]. The practical implication is that ''software simplicity'' and ''latency margin'' usually move in opposite directions once workloads exceed coarse filtering.

Trade-off B: throughput-per-watt vs assurance burden. Dedicated accelerators and FPGA pipelines improve throughput per watt for cloud/segmentation-style workloads (e.g., CloudScout and CloudScoutBench results), but qualification, operator support, and fault-mitigation evidence become first-order constraints [4], [8], [80]. Higher performance therefore shifts risk from arithmetic throughput to verification and lifecycle assurance.

Trade-off C: model complexity vs memory-traffic energy. Across EO analytics pipelines, model selection is bounded not only by parameter count but also by activation movement and buffering overhead; this is visible in the spread between compact edge models and heavier GPU-oriented deployments [15], [25], [28]. In practice, memory and I/O pressure can dominate total energy even when nominal compute utilization is acceptable.

Trade-off D: autonomy gain vs failure consequence. Closed-loop ''observe–decide–act'' pipelines increase mission yield (e.g., sub-90 s dynamic retargeting demonstrations), but they require explicit confidence gating, watchdogs,

and deterministic fallback paths because wrong decisions can directly affect spacecraft behavior [84], [85]. As autonomy criticality increases, acceptable uncertainty shrinks faster than achievable model accuracy improves.

These four trade-offs explain why top-line model accuracy is an insufficient design objective. For publishable and reusable reporting, studies should always co-report accuracy, latency, memory, and power under the *same* workload and operating mode, together with the corresponding assurance architecture.

G. IDENTIFIED GAPS, LIMITATIONS, AND TRADE-OFFS

Despite the significant progress in the realm of space-edge AI, the survey also exposes key gaps:

Assurance and certifiability. As onboard ML moves from advisory triage to state-affecting autonomy, verification and product assurance (e.g. ECSS, NASA [90], [91], [92], [93]) become central.

Missions need evidence on **decision reliability** (false-alarm/miss rates, calibration, uncertainty for confidence gating), **robustness** (domain shift, out-of-distribution (OOD)), **real-time determinism** (bounded latency, contention), and **update governance** (versioning, rollback, isolation) so model updates do not jeopardize platform functions [88].

Radiation/fault verification: Many demos use COTS without quantifying SEU sensitivity or mitigation; mission-critical use needs explicit plans (ECC, watchdogs, self-tests, TMR, FPGA scrubbing with fault-injection or irradiation evidence [91], [94], [95]). **Limited on-orbit validation:** Many results remain simulation or lab; in-orbit demos are growing but still fewer.

1) ABSENCE OF ON-DEVICE OR IN-ORBIT TRAINING

The prevailing workflow remains ‘‘train on ground or in the cloud, infer at the space edge.’’ Adaptation or retraining on the satellite is largely unexplored; current edge accelerators focus on inference; future ones may support edge training and federated learning [46].

Inconsistent metrics: Reporting varies; a unified framework for latency, power, memory, and resource use under CubeSat constraints is lacking. The following subsection outlines benchmarking requirements and existing suites.

H. BENCHMARKING MACHINE-LEARNING ACCELERATORS

Performance in the reviewed studies is typically reported in three dimensions, but with inconsistent depth across papers:

- **Accuracy and F1 Scores:** Frequently reported, but often without calibration or uncertainty quality needed for operational decision thresholds.
- **Inference Time and Throughput:** Usually available, yet not always tied to full pipeline conditions (I/O, preprocessing, and scheduling overheads).
- **Power and Resource Use:** Commonly partial (board mode/TDP or accelerator-only), with fewer studies reporting full system energy per inference and memory traffic.

TABLE 10. Example per-engine inference latency on Snapdragon 855 and Myriad X (ISS Spaceborne Computer-2), including both mission workloads and standard ImageNet classifiers [13].

Workload	Input	CPU	GPU	DSP/AIP	Myriad X
HiRISENet (classifier)	227×227	87.8 ms	16.3 ms	7.6 ms	16.2 ms
NavCam DeepLabv3 (seg.)	513×513	6235 ms	2233 ms	192 ms	N/A
UAVSAR U-Net (seg.)	64×64 patch	50.0 ms	6.17 ms	2.56 ms	5.99 ms
MobileNet (ImageNet)	224×224	200 ms	14 ms	60 ms	20 ms
ResNet50 (ImageNet)	224×224	388 ms	43 ms	10 ms	41 ms
VGG19 (ImageNet)	224×224	1496 ms	176 ms	31 ms	220 ms

MNIST remains common in benchmarks [19], but space-edge AI needs metrics that reflect algorithmic quality and resource constraints. Four categories are essential [5], [34]: **Accuracy & robustness**—common datasets and reference models (e.g. ImageNet, AlexNet/GoogLeNet) to isolate hardware effects; **energy & power**—average and peak system power including off-chip DRAM; **latency & throughput**—end-to-end inference and sustained throughput for target batch sizes; **hardware cost**—memory, LUT/FF/DSP/BRAM (FPGA), or area (ASIC). A credible study reports accuracy, system energy per inference, latency/throughput under the same bandwidth, and area or resource usage. No single open benchmark yet unifies MCU, GPU, FPGA, and VPU/TPU/NPU under space-relevant SWaP and radiation.

1) WORKED EXAMPLE (SPACE-TESTED COTS)

Reference [13] benchmarked Snapdragon 855 (CPU/GPU/DSP/NPU) and Movidius Myriad X on Spaceborne Computer-2 (ISS), reporting per-engine inference latency across representative vision workloads.

Table 10 shows DSP/NPU often dominate latency (e.g. ∼11–32× vs CPU); the best engine depends on model structure and operator support [13]. Conclusions must be workload-dependent. Cross-*family* benchmarks (e.g. [4] for CloudScout on Myriad 2 vs FPGAs) remain rare; most studies benchmark within a platform or mission. Benchmarks can be categorized by focus, phase, and granularity [5]. Section VII-H2 lists edge-ML suites. The community still lacks a CubeSat-specific suite (datasets, reference networks, power/thermal envelopes) for apples-to-apples comparison.

2) EXISTING ML BENCHMARK SUITES

Although the space-AI community has yet to adopt a dedicated CubeSat benchmark, several open-source suites do cover low-power or edge-oriented inference. These frameworks provide reference workloads, input data, and scoring rules that can be adapted to on-orbit constraints and thus serve as a starting point for reproducible evaluation:

1) **MLPerf Tiny** - A branch of MLPerf aimed at *TinyML* devices (microcontrollers, Raspberry Pi-class SBCs). It supplies sensor-driven workloads–keyword spotting,

TABLE 11. Open-source benchmark suites relevant to edge-ML on CubeSats.

Benchmark	Taxonomy	Granularity	Phase	Notes
MLPerf Tiny	System	Macro / E2E	I	Runs complete reference models (KW-Spotting, Visual Wake-Word, etc.) under strict MCU RAM/latency/energy caps.
EEMBC ML-Mark	System	Macro	I	Throughput + energy for full vision/audio nets; scores normalised for voltage/clock.
AI Benchmark	System	Macro	I	CNN/RNN suite for mobile/edge SoCs (e.g., Jetson, Movidius).
AI Matrix	System	Macro / E2E	T & I	Cross-platform workloads covering both training and inference; modules can be trimmed for resource-limited hardware.

visual wake-word, image classification, and anomaly detection–together with strict memory, latency, and energy budgets.

2) **EEMBC MLMark** - A consortium benchmark that reports both throughput and energy efficiency for embedded processors running pre-defined vision and audio models; scores are normalised to account for different clock rates and supply voltages.
3) **AI-Benchmark** - Originally developed for Android handsets, this suite nevertheless stresses higher-end edge platforms (e.g., Jetson, Movidius) with a mix of CNN and RNN tasks, making it useful when CubeSat payloads incorporate powerful SoCs.
4) **AI Matrix** - Provides cross-platform tests covering training and inference on CPUs, GPUs, and specialised accelerators; while not tailored to embedded devices, its modular design allows selective inclusion of lightweight kernels for resource-limited hardware.

A quick classification of the four suites is presented in Table 11. (The first tag refers to *what* the benchmark measures; the second tag to *granularity*; "T/I" indicates whether workloads are primarily Training, Inference, or both.)

All four are system-level, macro-scale; none capture full space-specific constraints but offer reusable workloads extendable to a CubeSat-oriented benchmark.

VIII. CONCLUSION

ML for embedded space systems has evolved from feasibility studies to a mission-driven capability under stringent SWaP, thermal, and reliability constraints. *Onboard inference* is increasingly required to manage downlink bottlenecks and enable timely autonomy, especially beyond LEO where latency and limited contact restrict ground-in-the-loop operations.

A. KEY CONCLUSIONS FROM THE LITERATURE

- **Resource-aware ML is practical when evaluated with mission-relevant metrics.** Latency, memory, energy per inference, and robustness to faults drive feasibility and model choice; accuracy alone is insufficient.
- **TinyML enables meaningful onboard analytics within memory and duty-cycle limits.** Quantization and compression make MCU-class deployments viable for triage, event detection, and basic scene understanding when workloads are designed around tight envelopes.
- **Hardware acceleration shifts risk from computation to integration and assurance.** VPUs, GPUs, and FPGAs yield throughput gains but add power–thermal, toolchain, and verification burden; determinism and fault mitigation can be more limiting than peak TOPS.
- **Mission value is demonstrated through downlink reduction and shorter decision loops.** Selective downlink, filtering, compression, and feature extraction improve data utility per contact and support faster response.
- **The field is moving toward multi-purpose, reconfigurable onboard AI.** Recent missions treat AI as a platform capability (uploadable models, multi-application operation), requiring scheduling, isolation, and lifecycle management.
- **A reusable design artifact emerges from the literature synthesis.** Table 9, together with Fig. 12, provides an explicit mission-to-architecture mapping intended for early-phase trade studies and defensible hardware/software partitioning decisions.

B. REMAINING BARRIERS TO MISSION-CRITICAL ADOPTION

Unresolved barriers for mission-critical adoption include: **(1) Dependable computing under radiation and faults**—lightweight assurance stacks integrating fault mitigation with ML robustness are still lacking. **(2) Comparable benchmarking**—inconsistent metrics and deployment context hinder cross-platform comparison; a community suite coupling task difficulty with latency, power, memory, and reliability would help. **(3) Operational frameworks for model lifecycle and updates**—governance of updates, configuration control, and verification and validation (V&V) for mission-critical ML remain open. **(4) Sustained performance under domain shift**—higher-resolution and multi-modal sensors increase workload while orbital/seasonal variation induces shift; strategies with limited compute and data are still needed.

Declaration of generative AI and AI-assisted technologies in the manuscript preparation process

During manuscript preparation, the authors used OpenAI's ChatGPT to re-phrase the authors' original notes, to paraphrase them, and for summarization of that draft text. The authors reviewed all outputs, verified technical accuracy and references, and take full responsibility for the final content.

REFERENCES

[1] G. Furano, G. Meoni, A. Dunne, D. Moloney, V. Ferlet-Cavrois, A. Tavoularis, J. Byrne, L. Buckley, M. Psarakis, K.-O. Voss, and L. Fanucci, "Towards the use of artificial intelligence on the edge in space systems: Challenges and opportunities," *IEEE Aerosp. Electron. Syst. Mag.*, vol. 35, no. 12, pp. 44–56, Dec. 2020.

[2] G. Mystkowska, M. Monopoli, P. Nannipieri, L. Zulberti, D. M. Codinachs, and L. Fanucci, "Hardware platforms enabling edge AI for space applications: A critical review," *IEEE Access*, vol. 13, pp. 143939–143956, 2025.

[3] H.-T. Pham, M.-A. Nguyen, and C.-C. Sun, "AIoT solution survey and comparison in machine learning on low-cost microcontroller," in *Proc. Int. Symp. Intell. Signal Process. Commun. Syst. (ISPACS)*, Dec. 2019, pp. 1–2.

[4] E. Rapuano, G. Meoni, T. Pacini, G. Dinelli, G. Furano, G. Giuffrida, and L. Fanucci, "An FPGA-based hardware accelerator for CNNs inference on board satellites: Benchmarking with Myriad 2-based solution for the CloudScout case study," *Remote Sens.*, vol. 13, no. 8, p. 1518, Apr. 2021. [Online]. Available: https://www.mdpi.com/2072-4292/13/8/1518

[5] V. J. Reddi, "Benchmarking AI," in *Machine Learning Systems With TinyML*, V. J. Reddi, Ed., Cambridge, MA, USA: Harvard Univ. Press, 2024, ch. 4, pp. 1–20.

[6] A. Sharma, S. Dhanka, A. Kumar, M. Nain, B. Dhanka, V. K. Bhardwaj, S. Maini, and A. S. Arora, "A systematic review on machine learning intelligent systems for heart disease diagnosis," *Arch. Comput. Methods Eng.*, vol. 32, no. 7, pp. 4303–4329, Oct. 2025.

[7] S. Dhanka, A. Kumar, A. Sharma, H. Vundavilli, S. Maini, and E. Rajasekar, "Advances in machine learning and deep learning for hormonal disorder diagnosis: An exhaustive review on PCOS, thyroid, and optimization techniques," *Arch. Comput. Methods Eng.*, vol. 33, no. 2, pp. 2785–2829, Sep. 2025, doi: 10.1007/s11831-025-10380-y.

[8] G. Giuffrida, L. Diana, F. de Gioia, G. Benelli, G. Meoni, M. Donati, and L. Fanucci, "CloudScout: A deep neural network for on-board cloud detection on hyperspectral images," *Remote Sens.*, vol. 12, no. 14, p. 2205, Jul. 2020. [Online]. Available: https://www.mdpi.com/2072-4292/12/14/2205

[9] A. Sharma, S. Dhanka, A. Kumar, M. Nain, B. Dhanka, V. K. Bhardwaj, S. Maini, and A. S. Arora, "View on machine learning intelligent systems for arrhythmia classification: A comparative analysis," *Eng. Res. Exp.*, vol. 6, Jan. 2024, Art. no. 035209, doi: 10.1088/2631-8695/ad5d51.

[10] J. J. Hernández-Gómez, G. A. Yañez-Casas, A. M. Torres-Lara, C. Couder-Castañeda, M. G. Orozco-Del-Castillo, J. C. Valdiviezo-Navarro, I. Medina, A. Solís-Santomé, D. Vázquez-Álvarez, and P. I. Chávez-López, "Conceptual low-cost on-board high performance computing in CubeSat nanosatellites for pattern recognition in Earth's remote sensing," in *Proc. 1st Int. Conf. Geospatial Inf. Sci.*, vol. 13, 2019, pp. 114–122.

[11] D. Lüdtke, T. Firchau, C. G. Cortes, A. Lund, A. M. Nepal, M. M. Elbarrawy, Z. H. Hammadeh, J.-G. Meß, P. Kenny, F. Brömer, M. Mirzaagha, G. Saleip, H. Kirstein, C. Kirchhefer, and A. Gerndt, "ScOSA on the way to orbit: Reconfigurable high-performance computing for spacecraft," in *Proc. IEEE Space Comput. Conf. (SCC)*, Jul. 2023, pp. 34–44.

[12] J. M. Badia, G. Leon, M. Garcia-Valderas, J. A. Belloch, A. Lindoso, and L. Entrena, "Analysing the radiation reliability, performance and energy consumption of low-power SoC through heterogeneous parallelism," *Sustain. Comput., Informat. Syst.*, vol. 44, Dec. 2024, Art. no. 101049.

[13] E. R. Dunkel, J. Swope, A. Candela, L. West, S. A. Chien, Z. Towfic, L. Buckley, J. Romero-Cañas, J. L. Espinosa-Aranda, E. Hervas-Martin, and M. R. Fernandez, "Benchmarking deep learning models on myriad and snapdragon processors for space applications," *J. Aerosp. Inf. Syst.*, vol. 20, no. 10, pp. 660–674, Oct. 2023, doi: 10.2514/1.i011216.

[14] A. D. George and C. M. Wilson, "Onboard processing with hybrid and reconfigurable computing on small satellites," *Proc. IEEE*, vol. 106, no. 3, pp. 458–470, Mar. 2018.

[15] A. P. Arechiga, A. J. Michaels, and J. T. Black, "Onboard image processing for small satellites," in *Proc. IEEE Nat. Aerosp. Electron. Conf.*, Jul. 2018, pp. 234–240.

[16] M. Eshaq, M. S. Zitouni, S. Atalla, S. Al-Mansoori, and M. Macdonald, "CubeSat flight software: Insights and a case study," *J. Spacecraft Rockets*, vol. 62, no. 4, pp. 1328–1345, Jul. 2025, doi: 10.2514/1.a35882.

[17] J. Manning, D. Langerman, B. Ramesh, E. Gretok, C. Wilson, A. George, J. MacKinnon, and G. Crum, "Machine-learning space applications on smallsat platforms with tensorflow," in *Proc. 32nd Annu. AIAA/USU Conf. Small Satell.*, 2018, pp. 1–12. [Online]. Available: https://digitalcommons.usu.edu/smallsat/2018/all2018/458

[18] S. Chien, I. Zilberstein, A. Candela, D. Rijlaarsdam, T. Hendrix, A. Dunne, A. Oriol, and M. J. Puig, "Flight of dynamic targeting on the CogniSAT-6 spacecraft," in *Proc. Int. Symp. Artif. Intell., Robot. Automat. Space (ISAIRAS)*, Nov. 2024, pp. 1–8.

[19] R. Dengel, "Machine learning for space applications on embedded systems," Master's thesis, Dept. Comput. Sci., Elect. Space Eng., Space Technol., Luleå Univ. of Technology, Luleå, Sweden, Dec. 2021.

[20] R. M. Millan, R. von Steiger, M. Ariel, S. Bartalev, M. Borgeaud, S. Campagnola, J. C. Castillo-Rogez, R. Fléron, V. Gass, A. Gregorio, D. M. Klumpar, B. Lal, M. Macdonald, J. U. Park, V. Sambasiva Rao, K. Schilling, G. Stephens, A. M. Title, and J. Wu, "Small satellites for space science: A COSPAR scientific roadmap," *Adv. Space Res.*, vol. 64, no. 8, pp. 1466–1517, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0273117719305411

[21] N. Dhamani, G. Martin, C. Schubert, P. Singh, N. Hatten, and M. R. Akella, "Applications of machine learning and monocular vision for autonomous on-orbit proximity operations," in *Proc. AIAA Scitech Forum*, 2020, pp. 1–18. [Online]. Available: https://arc.aiaa.org/doi/abs/10.2514/6.2020-1376

[22] M. Pugliatti, V. Franzese, and F. Topputo, "Data-driven image processing for onboard optical navigation around a binary asteroid," *J. Spacecraft Rockets*, vol. 59, no. 3, pp. 943–959, May 2022, doi: 10.2514/1.a35213.

[23] S. Sabogal, A. George, and G. Crum, "ReCoN: A reconfigurable CNN acceleration framework for hybrid semantic segmentation on hybrid SoCs for space applications," in *Proc. IEEE Space Comput. Conf. (SCC)*, Jul. 2019, pp. 41–52.

[24] S. Chien, J. Doubleday, D. R. Thompson, K. L. Wagstaff, J. Bellardo, C. Francis, E. Baumgarten, A. Williams, E. Yee, E. Stanton, and J. Piug-Suari, "Onboard autonomy on the intelligent payload experiment CubeSat mission," *J. Aerosp. Inf. Syst.*, vol. 14, no. 6, pp. 307–315, Jun. 2017. [Online]. Available: https://api.semanticscholar.org/CorpusID:26299801

[25] C. Adams, J. Parker, and D. Cotten, "A hardware accelerated computer vision library for 3D reconstruction onboard small satellites," in *Proc. IEEE Aerosp. Conf.*, Mar. 2021, pp. 1–14.

[26] A. Maskey and M. Cho, "CubeSatNet: Ultralight convolutional neural network designed for on-orbit binary image classification on a 1U CubeSat," *Eng. Appl. Artif. Intell.*, vol. 96, Nov. 2020, Art. no. 103952. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0952197620302700

[27] T. I. Leong, Y. M. O. Abbas, M. A. C. Purio, and H. A. Elmegharbel, "Image classification unit: A U-Net convolutional neural network for on-orbit cloud detection aboard CubeSats," in *Proc. IEEE Int. Geosci. Remote Sens. Symp. IGARSS*, Jul. 2021, pp. 2807–2810.

[28] Z. Zhang, G. Xu, and J. Song, "CubeSat cloud detection based on JPEG2000 compression and deep learning," *Adv. Mech. Eng.*, vol. 10, no. 10, Oct. 2018, Art. no. 1687814018808178, doi: 10.1177/1687814018808178.

[29] M. Vasile, L. Walker, R. D. Dunphy, J. Zabalza, P. Murray, S. Marshall, and V. Savitski, "Intelligent characterisation of space objects with hyperspectral imaging," *Acta Astronautica*, vol. 203, pp. 510–534, Feb. 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0094576522006464

[30] J. Ballé, D. Minnen, S. Singh, S. J. Hwang, and N. Johnston, "Variational image compression with a scale hyperprior," 2018, *arXiv:1802.01436*.

[31] S. Xu, J. Zhang, Z. Zhao, K. Sun, J. Liu, and C. Zhang, "Deep gradient projection networks for pan-sharpening," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2021, pp. 1366–1375.

[32] X. Wang, Q. Hu, Y. Cheng, and J. Ma, "Hyperspectral image super-resolution meets deep learning: A survey and perspective," *IEEE/CAA J. Autom. Sinica*, vol. 10, no. 8, pp. 1668–1691, Aug. 2023, doi: 10.1109/JAS.2023.123681.

[33] R. Pitonak, J. Mucha, L. Dobis, M. Javorka, and M. Marusin, "CloudSatNet-1: FPGA-based hardware-accelerated quantized CNN for satellite on-board cloud coverage classification," *Remote Sens.*, vol. 14, no. 13, p. 3180, Jul. 2022. [Online]. Available: https://www.mdpi.com/2072-4292/14/13/3180

[34] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proc. IEEE*, vol. 105, no. 12, pp. 2295–2329, Dec. 2017. [Online]. Available: https://api.semanticscholar.org/CorpusID:3273340

[35] Z. Liu, Z. Shen, M. Savvides, and K.-T. Cheng, "ReActNet: Towards precise binary neural network with generalized activation functions," in *Computer Vision—ECCV 2020* (Lecture Notes in Computer Science), vol. 12359, A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, Eds., Cham, Switzerland: Springer, 2020, pp. 143–159.

[36] Zaoyang. (2025). *4-Bit vs 8-bit Quantization: Key Differences*. Newline. Accessed: Nov. 18, 2025. [Online]. Available: https://www.newline.co/@zaoyang/4-bit-vs-8-bit-quantization-key-differences--842272c7

[37] ONNX Runtime Team. *ONNX Runtime Quantization*. Accessed: Mar. 4, 2026. [Online]. Available: https://onnxruntime.ai/docs/performance/quantization.html

[38] NVIDIA Corporation. (2026). *NVIDIA TensorRT Quantization*. Accessed: Mar. 4, 2026. [Online]. Available: https://docs.nvidia.com/deeplearning/tensorrt/latest/inference-library/work-quantized-types.html

[39] Intel Corporation. (2025). *OpenVINO Neural Network Compression Framework (NNCF)—Post-Training Quantization*. Accessed: Mar. 4, 2026. [Online]. Available: https://docs.openvino.ai/2025/openvino-workflow/model-optimization-guide/quantizing-models-post-training.html

[40] Intel Corporation. (2026). *Intel Neural Compressor—Quantization*. Accessed: Mar. 4, 2026. [Online]. Available: https://intel.github.io/neural-compressor/latest/docs/source/quantization.html

[41] T. Lite. (2023). *Post-Training Quantization*. [Online]. Available: https://ai.google.dev/edge/litert/models/posttrainingquantization

[42] P. McLellan. (Aug. 2018). *AI Processing for On-Device Inference*. [Online]. Available: https://community.cadence.com/cadenceblogs8/b/breakfast-bytes/posts/ai-processing

[43] TensorFlow Model Optimization Team. (2022). *Trim Insignificant Weights*. [Online]. Available: https://www.tensorflow.org/modeloptimization/guide/pruning

[44] M. Suryavansh. (Sep. 2019). *How to Make Your Own Deep Learning Accelerator Chip!* Towards Data Sci. [Online]. Available: https://towardsdatascience.com/how-to-make-your-own-deep-learning-accelerator-chip-1ff69b78ece4

[45] S. Teerapittayanon, B. McDanel, and H. T. Kung, "BranchyNet: Fast inference via early exiting from deep neural networks," in *Proc. 23rd Int. Conf. Pattern Recognit. (ICPR)*, Dec. 2016, pp. 2464–2469.

[46] W. Li and M. Liewig, "A survey of AI accelerators for edge environment," in *Advances in Intelligent Systems and Computing*. Cham, Switzerland: Springer, 2020, pp. 35–44.

[47] Y. Jia, "Learning semantic image representations at a large scale," Dept. Elect. Eng. Comput. Sci., Univ. of California at Berkeley, Berkeley, CA, USA, Tech. Rep. UCB/EECS-2014-93, May 2014. [Online]. Available: https://www2.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-93.pdf

[48] I. Rodriguez, L. Kosmidis, J. Lachaize, O. Notebaert, and D. Steenari, "GPU4S Bench: Design and implementation of an open GPU benchmarking suite for space on-board processing," Dept. Comput. Archit., Universitat Politècnica de Catalunya, Barcelona, Spain, Tech. Rep. UPC-DAC-RR-CAP-2019-1, 2019. [Online]. Available: https://www.ac.upc.edu/app/research-reports/html/RR/2019/7.pdf

[49] G. Benelli, G. Giuffrida, R. Ciardi, D. Davalle, G. Todaro, and L. Fanucci, "GPU@SAT, the AI enabling ecosystem for on-board satellite applications," in *Proc. Eur. Data Handling Data Process. Conf. (EDHPC)*, Oct. 2023, pp. 1–4.

[50] G. Benelli, G. Todaro, M. Monopoli, G. Giuffrida, M. Donati, and L. Fanucci, "GPU@SAT DevKit: Empowering edge computing development onboard satellites in the space-IoT era," *Electronics*, vol. 13, no. 19, p. 3928, Oct. 2024. [Online]. Available: https://www.mdpi.com/2079-9292/13/19/3928

[51] W. Powell, M. Campola, T. Sheets, A. Davidson, and S. Welsh, "Commercial off-the-shelf GPU qualification for space applications," in *Space Technology Mission Directorate (STMD) Annual Program Review (APR)*. Cleveland, OH, USA: NASA Goddard Space Flight Center, Sep. 2018. [Online]. Available: https://ntrs.nasa.gov/citations/20180006906

[52] L. Zulberti, M. Monopoli, P. Nannipieri, L. Fanucci, and S. Moranti, "Highly parameterised CGRA architecture for design space exploration of machine learning applications onboard satellites," in *Proc. Eur. Data Handling Data Process. Conf. (EDHPC)*, Oct. 2023, pp. 1–6.

[53] L. Zulberti, M. Monopoli, P. Nannipieri, S. Moranti, G. Thys, and L. Fanucci, "Efficient coarse-grained reconfigurable array architecture for machine learning applications in space using DARE65T library platform," *Microprocessors Microsystems*, vol. 113, Mar. 2025, Art. no. 105142.

[54] C. Tan, C. Xie, A. Li, K. J. Barker, and A. Tumeo, "OpenCGRA: An open-source unified framework for modeling, testing, and evaluating CGRAs," in *Proc. IEEE 38th Int. Conf. Comput. Design (ICCD)*, Oct. 2020, pp. 381–388.

[55] C. Tan, C. Xie, A. Li, K. J. Barker, and A. Tumeo, "AURORA: Automated refinement of coarse-grained reconfigurable accelerators," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Feb. 2021, pp. 1388–1393.

[56] A. Tyagi, R. Jeyapaul, C. Zhou, P. Whatmough, and Y. Zhu, "Characterizing soft-error resiliency in Arm's ethos-U55 embedded machine learning accelerator," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw. (ISPASS)*, May 2024, pp. 96–108.

[57] R. Das and T. Krishna. (Sep. 2018). *DNN Accelerator Architecture—SIMD or Systolic?* SIGARCH. [Online]. Available: https://www.sigarch.org/dnn-accelerator-architecture-simd-or-systolic/

[58] V. Cretu, M. Ogut, J. Swope, E. Dunkel, F. Mirza, S. Chien, A. Jongeling, D. Sheldon, J. Sauvageau, D. Ogbe, and Z. J. Towfic, "Benchmarking and testing of qualcomm snapdragon system-on-chip for JPL space applications and missions," Jet Propulsion Laboratory, National Aeronautics and Space Administration, Washington, DC, USA, Tech. Rep. NASA/TM–20230006988, Mar. 2022. [Online]. Available: https://ntrs.nasa.gov/citations/20230006988

[59] Advanced Micro Devices, Inc. (2025). *Space-grade AMD Kintex Ultrascale XQR FPGA Family*. Accessed: Nov. 18, 2025. [Online]. Available: https://www.amd.com/en/products/adaptive-socs-and-fpgas/fpga/kintex-ultrascale-xqr.html

[60] P. Antunes and A. Podobas, "FPGA-based neural network accelerators for space applications: A survey," 2025, *arXiv:2504.16173*.

[61] NanoXplore. (2025). *NG-Ultra RH: Rad-Hard by Design FPGA/SoC (product Page)*. Accessed: Jan. 26, 2026. [Online]. Available: https://www.nanoxplore.com/space/solutions/ngultra-rh/

[62] European Space Agency. (2023). *An Open Source FPGA Toolchain for a European Space-Grade FPGA*. Accessed: Jan. 26, 2026. [Online]. Available: https://activities.esa.int/4000141380

[63] R. Polanco-Segovia, S. A. Zemerick, and P. M. Mehta, "A generalized software framework for deep neural network inference in space applications," *IEEE Aerosp. Electron. Syst. Mag.*, vol. 40, no. 12, pp. 56–68, Dec. 2025.

[64] T. Pacini, E. Rapuano, and L. Fanucci, "FPG-AI: A technology-independent framework for the automation of CNN deployment on FPGAs," *IEEE Access*, vol. 11, pp. 32759–32775, 2023.

[65] MathWorks. (2026). *HDL Coder (Product/Documentation Page)*. Accessed: Jan. 27, 2026. [Online]. Available: https://www.mathworks.com/products/hdl-coder.html

[66] MathWorks. *Fixed-Point Designer(Product/Documentation Page)*. Accessed: Jan. 27, 2026. [Online]. Available: https://www.mathworks.com/products/fixed-point-designer.html

[67] MathWorks. (2026). *FPGA/ASIC/SoC Development (Overview of HDL and Deep-Learning Deployment Workflows)*. Accessed: Jan. 27, 2026. [Online]. Available: https://www.mathworks.com/solutions/fpga-asic-soc-development.html

[68] F. Fahim et al., "Hls4ml: An open-source codesign workflow to empower scientific low-power machine learning devices," 2021, *arXiv:2103.05579*.

[69] F. Ferrandi, V. G. Castellana, S. Curzel, P. Fezzardi, M. Fiorito, M. Lattuada, M. Minutoli, C. Pilato, and A. Tumeo, "Invited: Bambu: an open-source research framework for the high-level synthesis of complex applications," in *Proc. 58th ACM/IEEE Design Autom. Conf. (DAC)*, Dec. 2021, pp. 1327–1330.

[70] P. Developers. (2026). *PandA-Bambu (Framework and Releases)*. Accessed: Jan. 27, 2026. [Online]. Available: https://github.com/ferrandi/PandA-bambu

[71] A. Altinok, D. R. Thompson, B. Bornstein, S. A. Chien, J. Doubleday, and J. Bellardo, "Real-time orbital image analysis using decision forests, with a deployment onboard the IPEX spacecraft," *J. Field Robot.*, vol. 33, no. 2, pp. 187–204, Mar. 2016. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.21627

[72] M. A. C. Purio, T. I. Leong, Y. M. O. Abbas, H. A. Elmegharbel, K. Hiraki, and M. Cho, "On-board image classification payload for a 3U CubeSat using machine learning for on-orbit cloud detection," in *Proc. 4th Symp. Space Educ. Activities*, Apr. 2022, pp. 1–6.

[73] Y. M. O. Abbas, E. Edwar, M. A. C. Purio, and A.-H. Jallad, "Onboard image classification unit implementation for AlAinSat-1 CubeSat," in *Proc. IEEE Int. Geosci. Remote Sens. Symp.*, Jul. 2024, pp. 7773–7776.

[74] S. Kacker, A. Meredith, K. Cahoy, and G. Labrèche, "Machine learning image processing algorithms onboard OPS-SAT," in *Proc. 36th Annu. AIAA/USU Conf. Small Satell.*, 2022, pp. 1–10. [Online]. Available: https://digitalcommons.usu.edu/smallsat/2022/all2022/65/

[75] I. Medina, L. Santiago, J. J. H. Gómez, R. Castillo, and C. C. Castañeda, "Artificial vision algorithm for Earth station location from CubeSats in narrow beam optical communications," *J. Phys., Conf. Ser.*, vol. 2307, no. 1, Sep. 2022, Art. no. 012023, doi: 10.1088/1742-6596/2307/1/012023.

[76] A. Lopez, O. Olugbenle, and M. C. F. Bazzocchi, "A computer vision approach to plant growth monitoring in an embedded CubeSat module," in *Proc. IEEE Aerosp. Conf.*, Mar. 2024, pp. 1–9.

[77] M. Esposito, S. S. Conticello, M. Pastena, and B. C. Domínguez, "In-orbit demonstration of artificial intelligence applied to hyperspectral and thermal sensing from space," *Proc. SPIE*, vol. 11131, Feb. 2019, Art. no. 111310C, doi: 10.1117/12.2532262.

[78] N. Buonaiuto, M. Louie, J. Aarestad, R. Mital, D. Mateik, R. Sivilli, A. Bhopale, C. Kief, and B. Zufelt, "Satellite identification imaging for small satellites using NVIDIA," in *Proc. 31st Annu. AIAA/USU Conf. Small Satell.*, 2017, pp. 1–12. [Online]. Available: https://digitalcommons.usu.edu/smallsat/2017/Alternates/5/

[79] I. Medina, J. J. Hernández-Gómez, C. R. T.-S. Miguel, L. Santiago, and C. Couder-Castañeda, "Prototype of a computer vision-based CubeSat detection system for laser communications," *Int. J. Aeronaut. Space Sci.*, vol. 22, no. 3, pp. 717–725, Jun. 2021, doi: 10.1007/s42405-020-00320-4.

[80] E. Lemaire, M. Moretti, L. Daniel, B. Miramond, P. Millet, F. Feresin, and S. Bilavarn, "An FPGA-based hybrid neural network accelerator for embedded satellite image classification," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Oct. 2020, pp. 1–5.

[81] G. Bahl, L. Daniel, M. Moretti, and F. Lafarge, "Low-power neural networks for semantic segmentation of satellite images," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. Workshop (ICCVW)*, Oct. 2019, pp. 2469–2476.

[82] J. Petri, S. Klinkner, and J. Zink, "Hardware accelerated onboard image processing for space-based meteor observation: Concept and implementation of spacemedal," in *Proc. 4S Symp. Inf. Made Space*, May 2022, pp. 1–8.

[83] J. Nalepa and M. Cwiek. (2025). *How on-Board Data Processing is Reshaping Space Missions*. Accessed: Nov. 21, 2025. [Online]. Available: https://www.kplabs.space/news/how-on-board-data-processing-is-reshaping-space-missions

[84] NASA. (2025). *How NASA is Testing Ai to Make Earth-Observing Satellites Smarter*. Accessed: Nov. 21, 2025. [Online]. Available: https://www.nasa.gov/science-research/earth-science/how-nasa-is-testing-ai-to-make-earth-observing-satellites-smarter/

[85] E. Kulu. (2024). *Hammer (IOD6) Spacecraft*. Accessed: Nov. 21, 2025. [Online]. Available: https://www.nanosats.eu/sat/hammer

[86] E. Kulu. (2024). *Pony Express 2 Spacecraft*. Accessed: Nov. 21, 2025. [Online]. Available: https://www.nanosats.eu/sat/pony-express-2

[87] National Academies of Sciences, Engineering, and Medicine, *Achieving Science With CubeSats: Thinking Inside the Box*. Washington, DC, USA: National Academies Press, 2016, doi: 10.17226/23503.

[88] N. Longépé, N. Melega, V. Marchese, A. Paskeviciute, I. Babkina, I. Petrelli, M. Casaburi, D. Peressutti, and N. O. Kadunc. (Feb. 2023). *Φsat-2 Mission Overview for the #ORBITALAI Challenge*. European Space Agency (ESA). [Online]. Available: https://ai4eo.eu/wp-content/uploads/2023/02/Phisat-2MissionOverviewWeb.pdf

[89] GIM International. (Aug. 2024). *AI-Powered Phisat-2 Launched by Open COSMOS and ESA*. [Online]. Available: https://www.gim-international.com/content/news/ai-powered-phisat-2-launched-by-open-cosmos-and-esa

[90] *Space Engineering—Software*, Standard ECSS-E-ST-40C, European Cooperation for Space Standardization (ECSS), Mar. 2009. [Online]. Available: https://ecss.nl/standard/ecss-e-st-40c-space-engineering-software/

[91] *NASA Software Engineering Requirements*, Standard NPR 7150.2D, National Aeronautics and Space Administration (NASA), Mar. 2022. [Online]. Available: https://nodis3.gsfc.nasa.gov/displayDir.cfm?InternalID=NPR7150002D

[92] *Space Product Assurance—Software Product Assurance*, Standard ECSS-Q-ST-80C Rev.2, ECSS, Apr. 2025. [Online]. Available: https://ecss.nl/standard/ecss-q-st-80c-rev-2-space-product-assurance-software-product-assurance/

[93] *Software Safety Standard*, Standard NASA-STD-8719.13, 2013. [Online]. Available: https://standards.nasa.gov/standard/NASA/NASA-STD-8719.13

[94] *Space Product Assurance—Radiation Hardness Assurance—EEE Components*, Standard ECSS-Q-ST-60-15C, European Cooperation for Space Standardization (ECSS), Oct. 2012. [Online]. Available: https://escies.org/download/webDocumentFile?id=59672

[95] *Space Product Assurance—Electrical, Electronic and Electromechanical (EEE) Components*, Standard ECSS-Q-ST-60C Rev.4, European Cooperation for Space Standardization (ECSS), Apr. 2025. [Online]. Available: https://ecss.nl/standard/ecss-q-st-60c-rev-4-electrical-electronic-and-electromechanical-eee-components/

MOHAMMED ESHAQ (Member, IEEE) received the B.Sc. and M.Sc. degrees in computer engineering from American University of Sharjah, United Arab Emirates, and the Certificate in University Teaching from The University of Hong Kong. He is currently pursuing the Ph.D. degree in electronics and electrical engineering with the University of Strathclyde, Glasgow, U.K., where his research focuses on embedded vision and artificial intelligence for CubeSat and microsatellite autonomy.

He is also a Senior Flight Software Engineer with the Mohammed Bin Rashid Space Centre (MBRSC), Dubai, United Arab Emirates, where he works on on-board computer systems and flight software for satellite missions. He is also a Senior Embedded Engineer with MBRSC Laboratory, University of Dubai, and a Ph.D. Researcher with the Applied Space Technology Laboratory (ApSTL), University of Strathclyde. His research interests include space systems engineering, embedded and real-time software, and machine learning for space applications.

M. SAMI ZITOUNI (Senior Member, IEEE) received the M.Sc. and Ph.D. degrees in electrical and computer engineering from Khalifa University (KU), Abu Dhabi, United Arab Emirates, in 2015 and 2019, respectively. He completed his studies with KU Center for Autonomous Robotic Systems (KUCARS) and the Visual Signal Analysis and Processing Center (VSAP), focusing on video analysis, detection and tracking, crowd modeling, social and cognitive behavior analysis, and scene understanding. Before joining the University of Dubai as an Assistant Professor, he was a Postdoctoral Fellow of Biomedical Engineering with KU, KU–KAIST Joint Research Center, where he worked on physiological signal analysis, affective-state recognition, mental-health monitoring, and emotionally aware human–machine interaction. He is currently the Director of the Future Foresight Scenario Laboratory, a Coordinator of the Mohammed Bin Rashid Space Center (MBRSC) Laboratory, University of Dubai, and a Visiting Professor with the Department of Earth and Environmental Sciences, University of Milano-Bicocca, Milan, Italy. His research interests include artificial intelligence, machine learning, remote sensing, Earth observation, computer vision, affective computing, and embedded systems.

JAIME ZABALZA (Member, IEEE) received the M.Eng. degree in industrial engineering from Universitat Jaume I, Castellón de la Plana, Spain, in 2006, and the M.Sc. and Ph.D. degrees in electronic and electrical engineering from the University of Strathclyde, Glasgow, U.K., in 2012 and 2015, respectively. He is currently a Chancellor's Fellow (a Lecturer) with the Advanced Nuclear Research Centre, Department of Electronic and Electrical Engineering, University of Strathclyde. His research interests include hyperspectral data analysis and signal and image processing in a wide range of applications. He was a recipient of IET Image and Vision Section Prize for Best Ph.D. thesis for his work in hyperspectral remote sensing.

MALCOLM MACDONALD received the Bachelor of Engineering degree in aeronautical engineering from the University of Glasgow, in 2000, and the Ph.D. degree, in 2002. He is currently an Engineer, an Academician, and the Director working at the interface between academia, industry, and government. He is also the Chair of the Applied Space Technology, University of Strathclyde, Glasgow, U.K., and a Visiting Professor with the Centre for Space Research, University College Dublin, Ireland. He was the Director of Scottish Centre of Excellence in Satellite Applications, from 2014 to 2020, and a Non-Executive Board Member of U.K. Space Agency, from 2017 to 2020. He is the Vice-Chair of that Agency's Space Technology Advisory Committee. Thereafter, he was a Research Assistant, from 2003 to 2004, before joining SCISYS (U.K.) Ltd. (now CGI), from 2005 to 2008, to work on ADM-Aeolus and LISAPathfinder, after which he joined the University of Strathclyde. His research aims to enable and develop new space-derived services through advancing a range of new technologies, systems, and concepts to challenge conventional ideas by working at the interface between disciplines to advance new concepts in the exploration and exploitation of space.