Automatic Domain-Specific SoC Design for Autonomous Unmanned Aerial Vehicles

S. Krishnan, Z. Wan, K. Bhardwaj, P. Whatmough, A. Faust, S. Neuman, G. Wei, D. Brooks, V. J. Reddi

September 1, 2022

IEEE International Symposium on Microarchitecture (MICRO)
Chicago, IL, United States
October 1, 2022 through October 5, 2022

Automatic Domain-Specific SoC Design for Autonomous Unmanned Aerial Vehicles

Srivatsan Krishnan*, Zishen Wan[†], Kshitij Bhardwaj[‡], Paul Whatmough[§], Aleksandra Faust[¶], Sabrina Neuman*,
Gu-Yeon Wei*, David Brooks*, and Vijay Janapa Reddi*
* *Harvard University*
[†]*Georgia Institute of Technology*
[§]*Arm Research*
[‡]*Lawrence Livermore National Lab*
[¶]*Google Research, Brain Team*
Contact:srivatsan@seas.harvard.edu

***Abstract*—Building domain-specific accelerators is becoming increasingly paramount to meet the high-performance requirements under stringent power and real-time constraints. However, emerging application domains like autonomous vehicles are complex systems with constraints extending beyond the computing stack. Manually selecting and navigating the design space to design custom and efficient domain-specific SoCs (DSSoC) is tedious and expensive. Hence, there is a need for automated DSSoC design methodologies. In this paper, we use agile and autonomous UAVs as a case study to understand how to automate domain-specific SoCs design for autonomous vehicles. Architecting a UAV DSSoC requires consideration of parameters such as sensor rate, compute throughput, and other physical characteristics (e.g., payload weight, thrust-to-weight ratio) that affect overall performance. Iterating over several component choices results in a combinatorial explosion of the number of possible combinations: from tens of thousands to billions, depending on implementation details. To navigate the DSSoC design space efficiently, we introduce *AutoPilot*, a systematic methodology for automatically designing DSSoC for autonomous UAVs. AutoPilot uses machine learning to navigate the large DSSoC design space and automatically select a combination of autonomy algorithm and hardware accelerator while considering the cross-product effect across different UAV components. AutoPilot consistently outperforms general-purpose hardware selections like Xavier NX and Jetson TX2, as well as dedicated hardware accelerators built for autonomous UAVs. DSSoC designs generated by AutoPilot increase the number of missions on average by up to 2.25×, 1.62×, and 1.43× for nano, micro, and mini-UAVs, respectively, over baselines. Further, we discuss the potential application of AutoPilot methodology to other related autonomous vehicles.**

***Keywords*-Robotics; Domain-Specific Architectures; ML for Systems; Autonomous Machines; IoT and Edge Computing; Mobile Systems**

I. Introduction

Domain-specific SoC (DSSoC) architectures are increasingly popular due to their higher performance and energy efficiency than general-purpose processors. However, in DSSoCs, the trade-off of achieving higher efficiency is the loss of flexibility compared to general-purpose processors. Moreover, as the underlying domain can be complex and diverse, the lack of flexibility and scalability (for rapidly evolving domains) translates to increased cost, thus increasing the barrier to designing DSSoCs.

To reduce design cost and improve the scalability of domain-specific architectures, Firstly, we need systematic methodologies and tools to understand the domain and its unique constraints to design a balanced compute system. Secondly and more importantly, there is a need to define correct metrics to design and evaluate different domain-specific architectures. Lastly, there is a need for design automation for DSSoCs as domain requirements change rapidly. To that end, in this work, we aim to answer the following research question: *How do we systematically design DSSoCs for a rapidly evolving domain?*

To answer the question, we take autonomous UAVs as an example to show the need for a systematic methodology and evaluation metrics. Autonomous UAVs are an extraordinarily complex and diverse domain in which the traditional computing platform is just one component among many other components involving sensors, autonomy algorithms, onboard compute and the UAV platform itself. To achieve mission-level performance, we must understand the implications of other components have on the design of onboard compute. Moreover, in the context of autonomous systems, when the sensor configuration, compute, or body-dynamics changes, the optimal DSSoC design point can change dramatically, as we demonstrate in our evaluation. To this end, it becomes crucial to have flexible and agile DSSoC accelerator design methodologies.

In this paper, we introduce *AutoPilot* to provide a systematic methodology for automatically designing DSSoC for autonomous UAVs. We use the AutoPilot to demonstrate the shortcomings of existing methodologies and trends in designing DSSoC architectures. The high-level usage model is that given a high-level specification of autonomy task, UAV type, and mission goals, AutoPilot automatically navigates the large design space to perform full-system DSSoC co-design to generate a combination of autonomy algorithm and corresponding hardware accelerator to maximize UAV

[†]This work was done while the author was at Harvard University.
[‡]This work was done while the author was at Harvard University.

performance (e.g., number of missions).

AutoPilot has three main steps (Fig. 1): (1) Train several end-to-end autonomy algorithms for a given autonomy task (autonomous navigation) using reinforcement learning [43], or supervised learning [52], [71], and validate task-level functionality (① in Fig. 1). (2) Perform multi-objective algorithm-hardware co-design to maximize task success rate, compute performance and minimize power using Bayesian optimization [33] (② in Fig. 1). This ensures that we traverse the large design space rapidly and obtain several interesting design candidates. (3) Finally, we perform full-system co-design across UAV components using the F-1 model [46] to select the optimal compute and autonomy algorithm combination for a given UAV to maximize its mission performance, i.e., number of missions (③ in Fig. 1). This step accounts for the cross-product effect across the full-UAV stack and is critical to maximize UAV performance (as we demonstrate).

AutoPilot offers a complete solution for *automatically navigating* the UAV DSSoC design space and performing co-design across the entire system stack, including sensors, autonomy algorithms, onboard compute, and UAV dynamics. To demonstrate the scalability and generalizability of AutoPilot, we apply it to three different UAV types and three domain randomized environments (total nine combinations). These auto-generated domain randomized environments have varying degrees of obstacle densities and represent common deployment use cases for UAVs. For instance, autonomous navigation for a farming use case may be sparse (low-obstacle), whereas a search and rescue operation can be a dense-obstacle scenario.

The evaluation of AutoPilot generated designs reveal that the traditional design methodologies such as selecting a low-power design [60], [80], high throughput designs, or high efficiency (throughput/W) do not always improve mission-level performance. Instead, AutoPilot generated designs are more balanced as they account for the domain-specific interactions between various UAV parameters such as sensors, UAV physics, etc. As a result, the domain-specific AutoPilot methodology consistently outperforms general-purpose hardware selections like Xavier NX and Jetson TX2. It also outperforms dedicated hardware accelerators built for autonomous UAVs across diverse representative scenarios, as they do not take into account domain-specific characteristics.

More broadly, when we compare the results from AutoPilot to prior work, we come to the following general conclusions:

- *Evaluating DSSoCs needs the right set of metrics.* While evaluating DSSoCs, there is a tendency to use traditional isolated compute-system metrics, such as low-power, high-throughput, or high-compute efficiency, based on single kernel-level performance (e.g., SLAM [31], [60], [80]). But this can be misleading, resulting in under or overdesign. DSSoCs especially for SWaP constrained systems, must take into account domain-relevant constraints (e.g., weight) in evaluating proposed system capabilities.
- *Full-system co-design is important.* It is essential to look at the whole (U)AV (sensing, computation, actuation, body dynamics) to derive constraints for a DSSoC design. To achieve that, we must take into account all three phases: autonomy algorithms, intelligent hardware-software co-design, and specifically we must apply safety-constraints (e.g. stopping distance).
- *There is 'no one size fits all' DSSoC design.* Our results show that the optimal DSSoC design is sensitive to changes in sensing, computation, and actuation pipeline. As such, there is a need for customizing DSSoC design as UAVs come in various shapes and sizes. AutoPilot is an agile methodology to cope with the changes.

Prior Work	**End-to-End Autonomy?**	**Hardware Acceleration**	**UAV Components**		**Provides Design Methodology?**	**Automated?**
			Sensor	**UAV Physics**		
Navion [80]	✗	Only VIO	✗	✗	✗	✗
Hadidi et al [31]	✗	Only SLAM	✗	✗	✓	✗
RoboX [70]	✗	Only Motion Planning	✗	✓	✓	✓
MavBench [17]	✓	✗	✗	✗	✗	✗
PULP-DroNet [60]	✓	Full end-to-end stack	✗	✗	✗	✗
AutoPilot (This Work)	✓	Full End-to-End Stack	✓	✓ (Section V-C)	✓ (Section V-C)	✓

Table I: Comparison of prior work on autonomous UAVs. AutoPilot provides an automated methodology for co-design across the cyber-physical system stack for end-to-end UAV autonomy.

In summary, the goal of this work is to understand how to design a balanced DSSoC for an emerging application area, specifically UAVs, considering the whole system characteristics and SWaP constraints, as that affects the end DSSoC design. Our primary technical contributions are as follows:

- Introduce AutoPilot, a domain-specific system-on-chip (DSSoC) design automation methodology for AVs. We decompose the methodology into three stages: (1) *domain-specific, front-end task specification*, (2) *domain-agonstic multi-objective DSE* and a (3) *domain-specific backend.*
- We apply AutoPilot for autonomous UAVs. We quantitatively show that AutoPilot generates optimal DSSoC designs that increase the number of missions by 2.25×, 1.62×, and 1.43× for nano, micro, and mini-UAVs respectively, over general purpose hardware accelerators.
- We provide a detailed taxonomy based on AutoPilot to show how we can extend the methodology to other AV domains that can benefit from DSSoCs.

II. Background and Related Work

Prior work related to AutoPilot can be categorized into the following: autonomy algorithm design for UAVs, hardware accelerator design for autonomous UAVs, full-Stack UAV design, and accelerators design for other robots.

Autonomy Algorithms for UAVs. There are two main categories of autonomy algorithms, namely Sense-Plan-Act (SPA) and End-to-End (E2E) learning-based.

In *SPA*, the algorithm is broken into distinct stages: sensing, planning, and control. Specifically, sensor data is used to create a map [20], [23], [69] of the environment. Then, the planning stage [28], [40] processes the map to determine the best trajectory. Finally, the control stage uses the trajectory information, which actuates the rotor for the robot to follow the trajectory. Harris et al. [32] provides one of the early works on a co-simulation framework for characterizing the performance and power of embedded computers used in mobile UAVs. The work characterized image-processing kernels such as FAST [68] and BRIEF [18], building blocks for mapping and navigation tasks. However, their work [32] needs additional kernels and workloads to make the UAV fully autonomous. MAVBench [17] is another related work that provides a complete benchmarking suite for autonomous UAVs based on the SPA paradigm. MAVBench is one of the early works showing that 95% of the UAV battery power is spent on rotors, and only 5% of the power is spent on onboard electronics (including sensor and compute). Therefore, optimizing onboard computing to maximize the velocity can reduce the mission energy to improve mission performance.

E2E learning is an alternate paradigm where the algorithms process raw input sensor information (e.g., RGB, Lidar) and use a neural network model to produce output actions directly. The E2E learning methods do not require maps or separate planning stages and hence are computationally faster compared to the SPA paradigm [22], [52], [60]. The E2E models trained using supervised learning are DroNet [52] and Trailnet [67], [78]. The E2E model for UAVs trained using reinforcement learning includes CAD2RL [71] and source seeking application [22].

AutoPilot. In contrast to these prior works, where the general theme is to design one autonomy algorithm for a given task, AutoPilot trains several E2E autonomy algorithms. This gives greater flexibility in designing custom DSSoCs as the UAV type or deployment scenario evolves.

Hardware Accelerator for Autonomous UAVs. Domain-specific hardware accelerators for robots is an area of emerging interest. Recent work proposed a low-power accelerator [60] for the E2E paradigm, but it is only limited to nano-UAVs running DroNet [52]. Navion [80] is a specialized accelerator for improving Visual-Inertial-Odometry (VIO) in autonomous UAVs using the conventional SPA paradigm. VIO is just one of the many stages required for full autonomy in UAVs. In the context of motion planning for UAVs, RoboX [70] generates an accelerator for model predictive control from a high-level DSL. All three hardware accelerator approaches target low-power designs or higher compute efficiency (throughput/W) without considering other interactions of other UAV parameters on the onboard compute.

AutoPilot. Though the goal of AutoPilot is also to design custom DSSoCs for UAVs. Our work differs from prior work, which does not consider the effect of other UAV parameters on the computing platform. Table I captures these differences compared to prior work on designing hardware accelerators for autonomous UAVs. Instead, we show that choosing a balanced compute system and other UAV components is key to maximizing domain-specific metrics like mission-level performance. AutoPilot improves upon state-of-the-art by providing an automatic design methodology for generating DSSoCs given a UAV specification.

Autonomous UAVs Design Space Exploration. Recently, Hadidi et al. [31] quantified the enormity of the UAV design space and its various components needed for autonomous operations (e.g., UAV mechanical components, flight controllers, sensor, payload weight, etc.). However, despite quantifying the UAV design space, the benefits of hardware accelerator design are studied only for the mapping stage in the SPA pipeline. As we have described in the autonomy algorithms for UAV section above, a fully autonomous UAV requires the synthesis of many computationally-intensive kernels (e.g., mapping, motion planning) [17], [51], [54]. Park et al. [61] explored the design space of drone infrastructure for large-scale delivery services. However, this work focuses on the selection of batteries with no mention of compute or autonomous UAV operations, which is the focus of this work.

AutoPilot. By comparison, AutoPilot evaluates the effects of other components for a fully autonomous UAV. Compared to Hadidi et al. [31], which focuses only on one of the many components required for full autonomy (SLAM kernel), AutoPilot focuses on the end-to-end system. We come to a different conclusion from a holistic system perspective, where optimizing DSSoC compute based on only kernel speed-up numbers (high throughput, low-power, or higher compute efficiency) can be misleading. Compared to Park et al. [61], AutoPilot specifically focuses on the onboard computer design (DSSoCs) to enable autonomy in UAVs. AutoPilot provides a systematic automated solution to navigate the enormous design space to produce optimal DSSoCs for different UAV types and deployment scenarios while maximizing mission-level performance.

Other Robots. Outside of the domain of aerial robots, there has been work [55], [56], [58] showing the benefits of designing custom hardware accelerators for motion planning. In particular, Robomorphic Computing [58] provides a general methodology to synthesize custom hardware based on robot parameters (e.g., joint constraints). However, it focuses only on one of many stages required to achieve autonomy, i.e., motion planning for articulated robots (e.g., arms). By contrast, AutoPilot provides full end-to-end autonomy for UAVs.

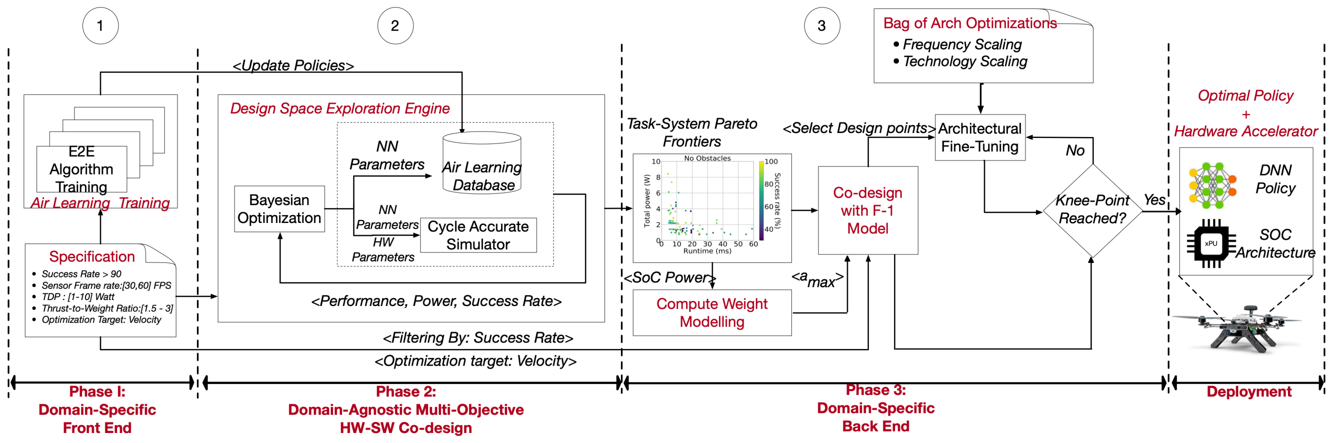


Figure 1: AutoPilot methodology for automating domain-specific SoCs for UAVs. The general methodology can be extended to other AVs (see Section VII). *Phase 1-Domain-Specific Front End:* Multiple E2E models trained using Air Learning [43] simulator based on high-level task specification. Success rates and hyper-parameters stored in a database. *Phase 2-Domain-Agnostic Multi-Objective HW-SW Co-design:* Multi-objective DSE using Bayesian optimization to find E2E model and accelerator designs that are optimal in success rate, and accelerator power and performance. *Phase 3-Domain-Specific Back End:* F-1 UAV tradeoff model [46] used to find the E2E model and accelerator that maximizes UAV mission performance.

III. AutoPilot

AutoPilot is a DSSoC methodology for automating HW-SW co-design, specifically in the context of UAVs. It consists of three stages (Fig. 1) to systematically perform the domain-specific system co-design; it leverages domain-specific insights to deliver SoC designs that are superior to more conventional architectural based methods. We make this conscious decision to perform DSSoC automation in AutoPilot in three phases instead of a single-phase optimization problem because: (1) Reuse the design as much as possible since architectural simulation and E2E model training can be slow processes; (2) A bad design point for one UAV type can be a balanced design that maximizes mission-level performance for another UAV. *Phase 1* is a domain-specific front end. The front end is designed to collect domain-specific information. In AutoPilot, the front end designs a collection of E2E-based autonomy algorithm implementations that are functionally correct for performing autonomous UAV tasks. It takes an input specification of the autonomous UAV tasks and trains several E2E autonomy algorithms for a given task using reinforcement learning [43]. *Phase 2* is domain-agnostic, focused on performing optimizations. AutoPilot performs multi-objective HW-SW co-design, or automated design space exploration (DSE) using Bayesian optimization [33] to find the Pareto frontier of E2E algorithms and hardware accelerators that are optimal in terms of task success rate, power, and runtime performance. Finally, *Phase 3* is a domain-specific back end that performs full-system UAV co-design based on its size, sensor characteristics (e.g., framerate and weight), and the design candidates (Pareto frontier designs) produced in Phase 2. Finally, a combination of the algorithm and an accelerator are automatically selected to maximize the number of missions.

A. Domain-Specific Front End for Task Specification

In this phase, the user provides a domain-specific *task-level specification* for an application (e.g., Source Seeking [22]), which includes some rough estimates about the environments such as obstacle densities. It also includes key metrics such as success rate, real-time latency constraints, etc. AutoPilot uses this information to configure the Air Learning [43] environment generator. Air Learning provides a high-quality implementation of RL algorithms that can be used to train a neural network navigation policy for the UAV. Air Learning includes a configurable environment generator [1] with domain randomization [83] that allows changing various parameters (e.g., the number of obstacles and size of the arena) to aid in generalizability. These parameters are configured based on the autonomy task specification. AutoPilot uses Air Learning to automatically train, evaluate and validate the E2E autonomy algorithms (Section II) for a given UAV task.

To determine the E2E model for each robot task (defined by environment complexity, i.e., obstacles), AutoPilot starts with the base multi-modal template used in Air Learning (Fig. 2a) and varies its hyperparameters (e.g., number of layers and filters) to create many candidate neural network (NN) policies. We start from a known template and vary the parameters inside the template because not all the layers within the E2E model improve UAV task-level performance. Using domain knowledge, we can seed the search process to explore regions that quickly give us desired results. For example, making the template layers deeper and wider gives a acceptable trade-off between the number of parameters and task-level success rate, as shown in Fig. 2b. The task-level success rate of 60% to 91% is comparable to autonomous navigation task success rate reported in robotics literature [27], [29], [71], [73], [78] for similar difficulty levels.

Based on these template parameters and the desired success rate, AutoPilot launches several Air Learning training instances. Each of the NN policies that achieve the required success rate is evaluated in a domain random environment [83], and its task-level functionality is validated. The validated NN policies are updated into an Air Learning database along with their success rates, which are then used by Bayesian optimization in the next DSE phase (Section III-B). It is important to note that this NN parameter seed selection may be inappropriate for a different task (similar to how ImageNet trained models might fail if applied to medical images). The goal of this phase is to have the flexibility to train several NN models, and there is no restriction on search space size.

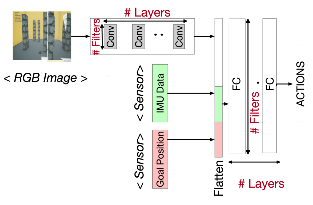
B. *Domain-Agnostic Multi-Objective DSE*

In this phase, AutoPilot performs a domain-agnostic automated multi-objective design space exploration (DSE) to find the Pareto frontier of the algorithms and DSSoC accelerator architectures that achieve optimal task success rate, performance, and power for a given autonomy task. Success rate is only affected by neural network hyperparameters (e.g., number of layers and filters). The runtime and power depend on the E2E model and accelerator architectural parameters.

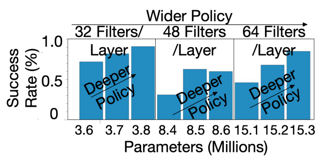
Success rates for the policies are accessed from the Air Learning database. At the same time, a cycle-accurate simulator is used to evaluate accelerator performance and power for the different policies and hardware configurations. Finally, we use Bayesian optimization to achieve rapid convergence to optimal solutions without performing an exhaustive search.

Air Learning Database. This database is used for storing the training results of various autonomy algorithms trained using Air Learning. Each entry in the database has an algorithm identifier, the hyperparameters used for training, and the success rate for the policy after validation.

DSSoC Template. We assume a basic UAV DSSoC that includes a parameterized template for hardware accelerator shown in Fig. 3a. The specification of various components in the DSSoC is tabulated in Table III. The onboard compute

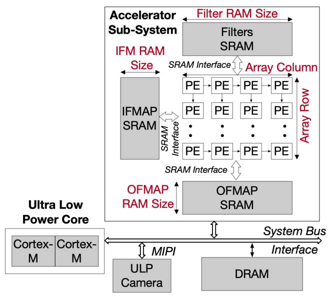


(a) E2E model template.

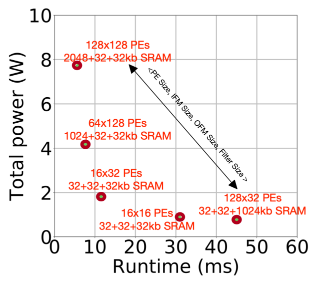


(b) Success Rate vs Model size.

Figure 2: (a) Parameterized End-to-End (E2E) model template. (b) E2E models parameters vs. task-level success rate.



(a) DSSoC architecture template.



(b) Power vs. performance.

Figure 3: (a) Parameterized accelerator template. (b) Varying the template parameters, as shown, allows us to generate Pareto frontier designs.

system consists of two ultra-low-power cores for running the flight controller, an accelerator sub-system (Systolic array-based), an external memory (DRAM), and an onboard RGB sensor connected to the system bus. The flight controller software stack is a Propotional, Integral, Derivative (PID) controller that runs bare-metal on the microcontroller unit (MCU), similar to Bitcraze Crazyflie [2], [22]. We also assume that the camera is interfaced with the system using a camera parallel interface [63] or MIPI [50] similar to this work [60] from which the accelerator sub-system can directly fetch the inputs to process the images. Also, we assume that the filter weights are loaded into the system memory as a one-time operation. To ensure that the DSSoC is adaptable and balanced for a given UAV and task, AutoPilot only changes the NN accelerator IP as per the change in deployment scenario or UAV types.

In the E2E paradigm (Section II), majority of the DSSoC time and energy is spent processing the NN policy [22], [52], [60], and its performance will dominate the overall throughput of the autonomy algorithm. For each frame from the sensor, the algorithm running on the accelerator sub-system generates high-level action commands that the flight controller interprets to generate low-level motor signals to control the UAV's physical rotors. Hadidi et al. [31] shows that the actuation cost (flight controller) is minimal compared to computation cost in typical scenarios. Hence, we focus our parameterization on the NN component although when we conduct our analysis and report results we consider the entire SoC shown in Figure 3a.

DSSoC Performance Estimation. AutoPilot uses SCALE-Sim, a configurable cycle-accurate systolic array-based NN accelerator simulator [72]. In addition, it exposes various architectural parameters such as array size (number of MAC units), scratchpad memory size for the input feature maps (ifmap), filters and output feature maps (ofmaps), dataflow mapping strategies, as well as system integration parameters, e.g., memory bandwidth. For example, enumerating the number of PE's, SRAM sizes give a acceptable trade-off

	Hyper-parameter	Potential Values
Neural Network	# Layers	[2, 3, 4, 5, 6, 7, 8, 9, 10]
	# Filter	[32, 48, 64]
Hardware	# PE Row	[8, 16, 32, 64, 128, 256, 512, 1024]
	# PE Column	[8, 16, 32, 64, 128, 256, 512, 1024]
	IFMAP/Filter/OFMAP SRAM Size (KB)	[32, 64, 128, 256, 512, 1024, 2048, 4096]

Table II: E2E model and architectural parameters tuned in AutoPilot for design of DSSoCs.

Components	Name	Peak Power	Throughput	Functionality	Parameters
ULP MCU	ARMv8-M [5]	0.38 mW [4]	100 MHz [4]	Flight Controller Stack, Driver stack	Fixed
Sensor	OV9755 [3]	100 mW [3]	30-90 FPS [3]	Sensor	Fixed
Sensor Interface	MIPI [50]	22 mW [53]	62.6 MHz	Camera Interface	Fixed
E2E NPU	Systolic Array	0.7 W to 8.24 W	22-200 FPS	E2E Model Processing Unit	Variable

Table III: The DSSoC spec used in AutoPilot for autonomous UAVs. The ultra low-power MCU and camera sensors are fixed.

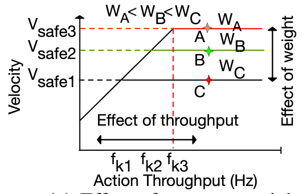
between performance and power as shown in Fig. 3b. Taking these architectural parameters, the SCALE-Sim generates the runtime latency.

DSSoC Power Estimation. To estimate the total DSSoC power, we add the power of individual components in the SoC as shown in Figure 3a. For estimating the power of the accelerator, we run a given NN policy on a cycle-accurate simulator. The cycle-accurate simulator produces SRAM traces, DRAM traces, number of read/write access to SRAM, number of read/write access to the DRAM. Using the SRAM and DRAM trace information, we model the SRAM power in CACTI [49] and DRAM power in Micron DRAM model [9]. For estimating the power for the systolic array, we multiply the array size with the PE energy. The PE power is modeled based on prior work [48].

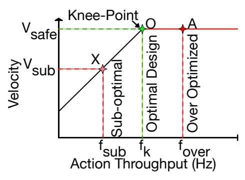
For the MCU cores, we use Cortex-M cores that implement the ARMv8-M ISA [5]. Each core consumes about 0.38 mW in a 28 nm process clocked at 100 MHz [4]. We also account for the power of the ultra-low-power core into our final power numbers. For the ULP camera, we assume it consumes 100 mW and form factor of 6.24 mm $\times$ 3.84 mm [3]. We account for the camera power in our overall power calculation.

Design Space. The searched design factors of both autonomy algorithms and DSSoC templates are shown in Table II. For autonomy algorithm space, we follow [43] and adopt the multi-modal template where all solutions are validated in AirLearning. For DSSoC space, we search PE array size and RAM size for input feature maps, filters, and output feature maps. More parameters can be exposed to the Bayesian optimization. Varying these parameters gives us wide range of performance/power profiles as tabulated in Table III.

Bayesian Optimization. AutoPilot uses Bayesian optimization [33] for algorithm-hardware design space exploration. Bayesian optimization is effective for optimizing black-



(a) Effect of compute weight.



(b) Optimal design.

Figure 4: (a) Mapping design candidates to F-1 to assess impact of compute weight and compute throughput on UAV performance. (b) Mapping design candidates to F-1 to select optimal design candidate for a UAV system.

box functions [75], [79] that are expensive to evaluate and cannot be expressed as closed-form expressions. But the bayesian optimization method can be replaced with reinforcement learning [81], evolutionary algorithms [88], simulated annealing [84] etc.

In AutoPilot, we use Bayesian optimization to optimize three objective functions: (i) task success rate, (ii) DSSoC power, and (iii) accelerator inference latency (runtime). A Pareto-optimal DSSoC design achieves maximum task success rate, minimum inference latency, and SoC power. The algorithm tunes NN policy hyper-parameters (such as the number of layers and filters) and accelerator hardware parameters (e.g., number of processing elements, SRAM sizes) to converge to Pareto-optimal NN policies and DSSoC designs.

Bayesian optimization initially evaluates the objective functions at random parameters, followed by intelligently selecting those that will optimize the objectives. The algorithm builds a Bayesian statistical model for each objective function using a Gaussian process (GP). These GP models are updated as the Bayesian optimization proceeds and samples new parameters. A GP distribution is defined by mean and covariance. The mean is the expected value of a function at some parameter value. The covariance, called the kernel, models the dependence between the function values at two distinct parameter values. In this paper, the widely-used squared exponential (SE) kernel is used due to its simplicity, leading to fast computation [65]. The right selection of parameter values is determined by an acquisition function computed using the GP-predicted objective values. The algorithm selects those inputs that maximize the acquisition function until all the optimal solutions are found.

In particular, the *S-Metric-Selection-based Efficient Global Optimization (SMS-EGO)* is used as the acquisition function. It has been shown to be highly effective for multi-objective optimization and handling a large design space compared to other acquisition strategies such as expected improvement [64]. SMS-EGO uses *hypervolume* to determine the degree to which a candidate point is optimal. A hypervolume is the volume enclosed between a candidate point and a fixed reference point in the Pareto space. Since the hypervolume of a Pareto-optimal point is higher than a non-optimal point, the approach maximizes the hypervolume until all the points

UAV Name	Base-UAV System (Fixed)					Autonomy Components (Custom Designed)		
	UAV Type	Battery Capacity (mAh)	Flight Controller	Base UAV Weight	Sensor	Sensor Framerate	Autonomy Algorithm	Onboard Compute
AscTec Pelican	*mini-UAV*	*6250 (fixed)*	*PID Controller 100 KHz*	*1650 g*	*RGB*	*30/60 FPS*	*E2E (custom)*	*Custom*
DJI Spark	*micro-UAV*	*1480 (fixed)*	*PID Controller 100 KHz*	*300 g*	*RGB*	*30/60 FPS*	*E2E (custom)*	*Custom*
Zhang et al [89]	*nano-UAV*	*500 (fixed)*	*PID Controller 100 KHz*	*50 g*	*RGB*	*30/60 FPS*	*E2E (custom)*	*Custom*

Table IV: In our experiments, we keep the base UAV system fixed (size, battery, sensor type) and focus on co-design of components needed for achieving autonomy. As the UAV type changes, the weight of the UAV also changes and this will impact the design of DSSoCs for autonomous UAV.

with the highest hypervolume (i.e, Pareto-optimal) are found.

However, not all Pareto-optimal DSSoC designs generated at this stage will result in optimal UAV performance (e.g., maximize the number of missions). Some of these designs will be over-provisioned or under-provisioned. Both these scenarios negatively affect the overall UAV performance. Hence, to determine which of the DSSoC designs is better suited for a given UAV, we need to perform full-system UAV co-design where we account for sensor, onboard compute, and its impact on UAV physics.

C. Domain-Specific Back End

To lower the DSSoC design to the target UAV platform, AutoPilot's Phase 2 prunes a large design space of $\approx 10^{18}$ designs to ≈100s of design candidates using domain-specific knowledge. These 100s of design candidates represent a sample of low-power, high-performance, or Pareto-optimal designs in terms of performance and power. However, enumerating 100s of design candidates is still tedious and requires a systematic way of selecting one of the designs for a UAV. Hence, the goal of Phase 3 is to determine the holistic evaluation of these design candidates and other UAV components.

Compute Weight Modelling. Since the payload weight affects the UAV physics, it is important to estimate the weight of the onboard computer. The onboard computer typically has two components: a motherboard where SoC is mounted and a passive heatsink for cooling the SoC. The heatsink weight is proportional to the TDP of the SoC.

For the motherboard weight, we assume that the final SoC is mounted on a PCB along with all electrical components weighing 20g (which per our analysis is typical for Ras-Pi [8], CORAL [6] like systems). For the heatsink weight, we use a heat sink calculator [7] which determines the heatsink volume required for cooling. The weight of the heatsink is then determined by multiplying the estimated volume with the density of aluminum (commonly used heatsink material).

Full-System Co-Design. A DSSoC design is a tailored SoC that maximizes the system efficiency, over its own efficiency metrics such as throughput, latency, power or area. AutoPilot filters the generated SoC designs with the highest success rate (based on the input specification) from the designs generated in Phase 2. It does this by mapping the filtered design candidates to the F-1 UAV tradeoff model [45], [46].

The F-1 [45], [46] is a roofline-like visual performance model built on top of a UAV safety model [51]. Each base UAV system has a unique F-1 plot that can gather insights about different bounds and bottlenecks. F-1 model can help identify if the system performance of a given UAV is bounded by sensor, compute or body dynamics, and it can aid a system architect in understanding the optimal compute design or selection for autonomous UAVs. The F-1 model plots the relationship between safe velocity and action throughput as shown in Fig. 4. The safe velocity is the maximum velocity at which the UAV can travel safely without colliding with an obstacle. Conversely, by relative motion (and switching the frame of reference), if the UAV is hovering, the same model can also determine the maximum velocity of an incoming object that the UAV can avoid before colliding. The action throughput is the decision-making rate, i.e., the output of the sensor-compute-control pipeline.

The F-1 model also considers the compute throughput and how the compute payload weight affects the UAV's physics, i.e., maximum acceleration, which can be determined by its thrust-to-weight ratio [57]. Prior work shows that UAV flight time can drop by 22% if the accelerator is not balanced with other UAV components [22], [60]. The F-1 model can also determine whether a combination of autonomy algorithm and onboard compute is over-provisioned, under-provisioned, or optimal for a given UAV system [46].

To understand the role of F-1 in AutoPilot, consider three DSSoC designs, 'A,' 'B,' and 'C,' generated from the multi-objective DSE in Phase 2. Assume that all the designs achieve the same compute throughput but at different thermal design power (TDPs), with 'A' being the lowest and 'C' being the highest. Mapping these designs on the F-1 model (Fig. 4a) gives insight into which designs achieve better UAV mission-level performance. DSSoC design 'A' has the same action throughput as 'B' and 'C' but with the lowest power. Hence, it has the lowest heatsink weight. DSSoC design 'C' has the highest power; hence 'C' has the highest heatsink weight, thus translating to a higher compute payload weight. The onboard compute weight lowers the UAV's ability to move fast, and the lowering of ceilings captures this effect in Fig. 4a for 'B' and 'C.' Lowering the safe velocity will imply mission energy and the number of missions it can perform [17]. Thus, AutoPilot will select DSSoC 'A' over the other two designs.

Another utility of the F-1 model in AutoPilot is to determine if a DSSoC design candidate is balanced, over-provisioned, or under-provisioned. The minimum value of action throughput to maximize safe velocity (V_{safe}) is called the knee-point. For example, in Fig. 4b, design ‘O’ is optimal because it achieves the minimum action throughput to maximize V_{safe}. Likewise, ‘A’ is over-provisioned (achieves more throughput than required), and ‘X’ is under-provisioned. AutoPilot will choose design ‘O’ over the other two designs. The F-1 model bounds the DSE across sensing, compute, and actuation [45]. In a scenario where the sensing/compute stage is overly optimized and the flight controller becomes the bottleneck, the F-1 model will guide Autopilot to select a DSSoCs design balanced across the sensor, compute, and actuation pipeline. This balanced design across the sensing/compute/control stage will result in maximal UAV performance.

Architectural Fine-Tuning. In the case when no optimal design exists that achieves the knee-point, certain designs may require some architectural tuning to shift the design close to the knee-point. AutoPilot provides two options: (i) The design points can be user-defined, or (ii) the design point closest to the knee-point can be selected. Architectural tuning can be performed using various optimizations until the optimized design is at (or very close to) the base knee-point in the F-1 roofline. We seed the AutoPilot system with two optimization techniques for starters: frequency scaling and technology node scaling. Ideally, any optimization technique such as HW-SW co-design, microarchitecture, and device-level optimization can be a part of the architectural fine-tuning step.

Total Execution Time. One round of AutoPilot design flow takes 3 to 7 days. Phase-1 and Phase-2 take the most amount of total time, while Phase-3 time is negligible. However, Phase-1 can be parallelized using advances in massively distributed RL frameworks like ACME [35], QuaRL [44] or Seed-RL [24], which are designed to drastically reduce the training time.

IV. Experimental Setup

Autonomy Task. We train autonomous navigation tasks in Air Learning [43] for three different environments: low, medium, and dense obstacles. Autonomous navigation is one of the key building blocks in achieving autonomy and is used in many practical applications like search and rescue, source-seeking [22], and package delivery [86]. Each E2E model for autonomous navigation is trained for one million steps or until convergence. This is a standard training methodology for reinforcement learning, and the same methodology is used for building real autonomous UAV applications [22], [71].

Base UAV Systems and Autonomy Components. To show the scalability of AutoPilot methodology, we take a representative UAV from each size category: mini, micro, and nano-UAV (Table IV). The UAV system includes a frame, flight controller, battery, and rotors (all included in the base weight). The flight controller is solely dedicated to stabilizing and controlling the UAV. The flight controller firmware is computationally lightweight and is run on microcontrollers [11], [12] that are tightly integrated into the UAV platform. The flight controller uses onboard sensors, such as the Inertial Measurement Unit (IMU) [13] and GPS, to stabilize and control the UAV. To recover from unpredictable errors (sudden winds or damaged rotors), the inner-loop typically runs at closed-loop frequencies of up to 1 kHz [30], [41]. We keep the UAV system fixed and focus on designing the optimal DSSoC and autonomy algorithm to maximize the overall operational efficiency of the autonomous UAV system. More importantly, as the UAV type changes, its base weight also changes, which will impact the selection/design of DSSoCs.

Domain-Specific Evaluation Metrics. Unlike traditional computing systems, the important operational efficiency metric for autonomous UAVs is the *number of missions*, which captures how many times the UAV can complete similar missions on a single battery charge. For example, in a package delivery use case, a higher number of missions means more packages delivered with lower downtime spent recharging. This metric is affected by the choice of onboard compute combined with several other key UAV components.

For a given UAV, we define the number of missions as:

$$N_{missions} = \frac{E_{battery}}{E_{mission}}, \tag{1}$$

where $E_{battery}$ is the total energy available in the UAV (a function of battery mAh rating) and $E_{mission}$ is the total energy expended by the UAV per mission.

We can define $E_{mission}$ for a single mission as:

$$E_{mission} = (P_{rotors} + P_{compute} + P_{others}) * t_{mission}, \tag{2}$$

where P_{rotors}, $P_{compute}$, and P_{others} refer to the power consumption of the rotor propulsion, compute, and other electronic components (e.g., sensors, ESC) in the UAV, respectively. $t_{mission}$ is the time for completing the mission. Intuitively, Eq 2 suggests that the amount of energy expended in a mission corresponds to the duration of the time the UAV flies (mission time), and the total power dissipation of its components.

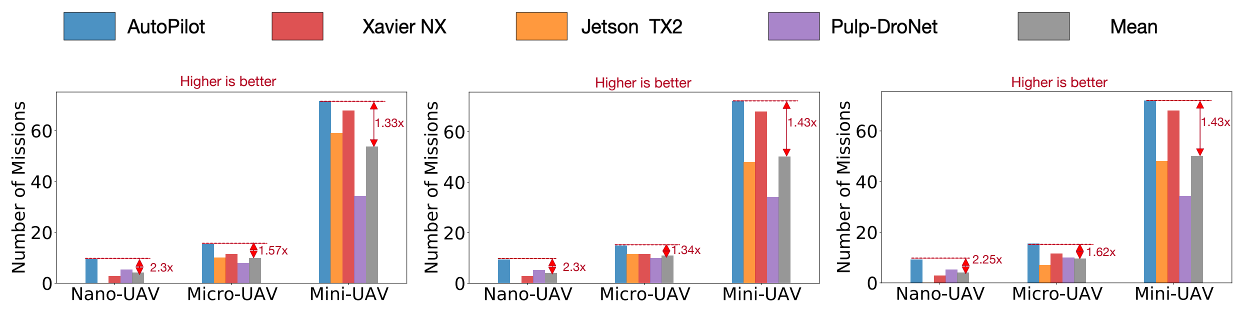
The mission time $t_{mission}$ depends upon the distance $D_{operation}$ and the UAV velocity. For a fixed distance, mission time is determined by how fast the UAV can travel through a dynamic environment filled with obstacles. The UAV needs to travel as quickly as possible while safely navigating around obstacles. We define this safe traveling speed as the safe velocity, V_{safe}. Maximizing V_{safe} lowers the mission time, increasing the total possible number of missions.

Using these terms, we can re-write Eq 2 as follows:

$$E_{mission} = (P_{rotors} + P_{compute} + P_{others}) * \frac{D_{operation}}{V_{safe}}, \tag{3}$$

Then, substituting Eq 3 in Eq 1, we get:

$$N_{missions} = \frac{E_{battery} * V_{safe}}{(P_{rotors} + P_{compute} + P_{others}) * D_{operation}}, \tag{4}$$



(a) Low-obstacle: Number of missions. **(b)** Some-obstacle: Number of missions. **(c)** Dense-obstacle: Number of missions.

Figure 5: Comparison of AutoPilot generated designs with other designs (TX2 and Xavier NX, PULP [60]) for three different deployment scenarios (low obstacle, medium obstacles, and dense obstacle environments) and three UAV categories (mini-UAV, micro-UAV, and nano-UAVs). (a), (b), and (c) denotes the number of missions *(higher is better)* on a single battery charge for three UAV types and three different deployment scenario. The degradation in the number of missions are compared with mean performance of TX2, NX, and PULP [60] and annotated in the plots with AutoPilot generated design as baseline. All the points except P-DroNet (PULP-DroNet) [60] runs the same policy. For P-DroNet, we use the numbers reported from their work.

According to Eq. 4, to maximize the number of missions, the optimization objective is to increase the UAV's safe velocity (V_{safe}) or increase the battery capacity ($E_{battery}$). Increasing the battery capacity is non-trivial since UAV size impacts the SWaP constraints. However, proper selection of various UAV components (compute, sensors, etc.) can maximize safe velocity. It is important to note that $E_{mission}$, V_{safe}, and $D_{operation}$ are not constants, and it depends upon the mission characteristics. But the fundamental relationship between them holds true.

Note that V_{safe} reported from AutoPilot is relative and measured from a frame of reference. If the UAV hovers, the safe velocity corresponds to the ability to dodge an incoming obstacle traveling at V_{safe}. If the UAV travels at V_{safe}, it corresponds to the velocity it can travel safely and stop to avoid the collision. For example, even in an inspection crawling "autonomous robot" in an unpredictable and dynamic environment, its sensing-compute-actuation pipeline must achieve a high safe velocity to avoid incoming dynamic obstacles or static obstacles when in motion. Hence, safe velocity is an important metric for autonomous UAVs.

V. Evaluation

This section evaluates AutoPilot's co-design results for different UAVs and deployment scenarios compared to baseline designs. We then compare AutoPilot methodology in selecting designs versus several traditional design strategies. Next, we characterize the effects of cyber-physical parameters such as sensor type and UAV agility on the compute co-design process. Finally, we analyze the cost of design specialization versus its impact on overall UAV mission efficiency.

The insights based on AutoPilot generated designs for autonomous UAVs are: (1) As the UAV type (e.g., mini-UAV vs. nano-UAVs) or deployment complexity changes (e.g., low clutter environment vs. densely cluttered environment), we must customize either the autonomy algorithm or DSSoC

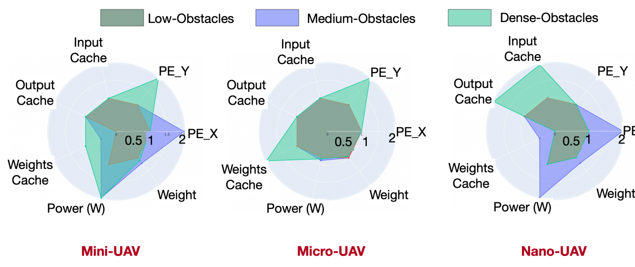
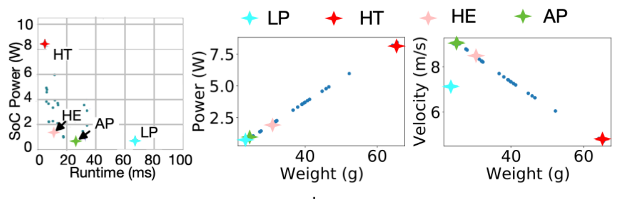


Figure 6: Visualization of DSSoC architectural parameter variations for nine scenarios (three UAVs and three deployment scenarios). The scales are normalized to the lowest value for each architectural parameter to show the variations requirements as the UAV components change. The variation of architectural parameters exemplifies the need for custom DSSoCs to maximize mission-level performance as per UAV type or deployment scenario.

to maximize the mission-level performance. This need for customization underscores the *necessity of DSSoCs for this domain*. Furthermore, since AutoPilot automates the DSSoC design, it rapidly generates *balanced* DSSoCs as the domain evolves, thus being a scalable and generalizable methodology; (2) We show that traditional methodology, in most cases, results in choosing a high throughput, low-power, or high-efficiency design. These traditional design methodologies do not consider domain-specific mission-level performance metrics. Instead, when designing DSSoCs using AutoPilot, domain-specific component (Phase 1 and Phase 3) play a critical role in maximizing mission-level performance; (3) When designing DSSoCs for a given UAV, its sensor performance and UAV's physics will greatly change the performance requirements for the DSSoCs thus the need for agile and scalable methodology like AutoPilot.

A. DSSoC Scalability and Generalizability

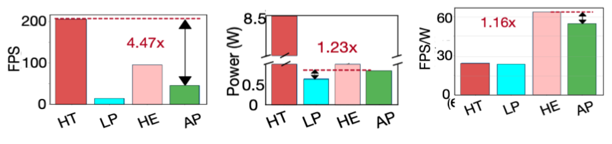
In this experiment, we demonstrate that the AutoPilot methodology is *scalable and generalizable* in generating



(a) (b) (c)

	Design Name	PE Size	Filter Cache	IFM Cache	OFM Cache
🟦	*Low Power (LP)*	8 x 16	32 kB	32 kB	32 kB
🟪	*High Efficiency (HE)*	32 x 128	32 kB	32 kB	32 kB
🟩	*AutoPilot (AP)*	16 x 16	64 kB	64 kB	32 kB
🟥	*High Throughput (HT)*	128 x 64	2 MB	2 MB	32 kB

(d)



(e) (f) (g)

Figure 7: (a) Phase 2 Pareto frontier designs with different performance and power profiles. (b) Relationship between weight and power for all designs. (c) Relationship between velocity and weight for all the generated designs. Various compute power and heatsink weights results in various safe velocity. (d) Hardware architecture. We show four designs: high throughput (HT), low power (LP), high perf/Watt compute efficiency (HE) and Autopilot (AP). (e) Compute throughput (FPS). (f) Power consumption. (g) Compute efficiency (FPS/W).

DSSoC designs that maximize the number of missions across three UAV types and three different deployment environment scenarios.

The *different classes of UAVs* evaluated are a mini-UAV (AscTec Pelican), a micro-UAV (DJI-Spark), and a nano-UAV (used in Zhang et al. [89]) whose specifications are in Table IV. The *different deployment scenarios* with varying levels of complexity evaluated are low, medium, and dense obstacles. For example, four obstacles with the goal position randomly change every training episode in the low-obstacle scenario. In the medium scenario, four fixed and up to three randomly-placed obstacles exist. The goal position also changes randomly during every training episode. In the dense obstacle scenario, there are four fixed obstacles in the dense scenario and up to five randomly placed obstacles. These auto-generated domain randomized environments have varying degrees of obstacle densities. However, they represent common deployment use cases for UAVs. For instance, autonomous navigation for a farming use case could be very sparse (low-obstacle). In contrast, a search and rescue operation in a forest or a racing UAV represents a dense-obstacle scenario.

The co-design goal is to choose an onboard DSSoC system and autonomy algorithm that maximizes the number of missions the UAV can perform on a single battery charge (Eq. 1 and Eq. 3).

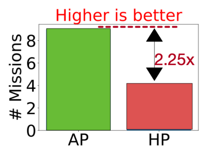
Co-Design Comparison Results. Fig. 5 shows the comparison in the number of missions between designs generated using AutoPilot methodology and two general-purpose systems (Jetson TX2 and Xavier NX) and PULP [60]. The motivation for PULP is designing a low-power chip for nano-UAVs. To that end, it achieves 6 FPS at 64mW. However, despite achieving low power, they report degradation of 22% in flight time, as they did not consider the effects of the weight of compute. They custom-designed PULP for one model (DroNet) and one UAV type (nano-UAV) to achieve low power. For PULP, we report the numbers as is, and this is an optimistic comparison for PULP. For instance, the AutoPilot generated E2E models have 109$\times$ to 121$\times$ larger than DroNet [52]. Running AutoPilot generated E2E models on PULP [60] will result in poorer performance than 6 FPS at 64 mW [60]. Even with an optimistic performance estimation for PULP running AutoPilot-generated E2E models (i.e., PULP producing the same 6 FPS at 64 mW for models that are 109 $\times$ larger than DroNet), we show that AutoPilot-generated DSSoCs consistently outperform PULP across different UAV types and deployment complexity in terms of mission-level performance.

On a fully charged battery, AutoPilot generated optimal designs achieve 1.33$\times$ to 1.43$\times$ more missions for mini-UAV compared to TX2, Xavier NX, and PULP [60]. For micro-UAVs, AutoPilot on average achieves 1.34$\times$ to 1.62$\times$. Likewise, for nano-UAV, AutoPilot on average achieves 2.3$\times$ more missions, thus enabling higher operational efficiency than an ad-hoc selection of general-purpose designs.

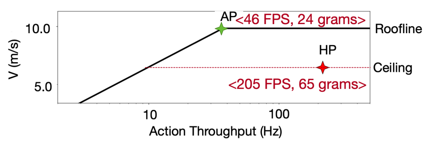
The AutoPilot DSSoC generation methodology for the onboard hardware and autonomy algorithm for these UAV types and deployment scenarios consistently outperforms general-purpose hardware (Jetson TX2 and Xavier NX) and a domain-specific hardware accelerator built for UAVs [60]. This demonstrates the flexibility of AutoPilot, compared to prior works that focus on a specific UAV type [22], [31], [60], [80].

DSSoC Architectural Parameter Variations. Visualizing the various DSSoC architectural parameters selections across all the nine scenarios (three UAVs and three deployment scenarios) to gain insights into the designs that AutoPilot generates is shown in Fig. 6. The deployment scenario affects the complexity of the models. For the low obstacle scenarios, a model with five layers and 32 filters achieves the highest success rate. A model with four layers and 48 filters achieves the highest success rate for the medium obstacle scenarios. Finally, a model with seven layers and 48 filters achieves the highest success rate for the dense obstacle scenarios.

The increasing complexity of the autonomy algorithm means the onboard compute for a specific UAV has to achieve similar performance while keeping the power and weight constant. Therefore, AutoPilot selects architectural parameters (PE size, cache size, etc.) intelligently to satisfy the performance, power, and weight constraints for the mini-

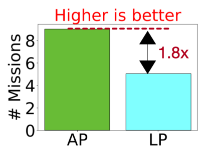


(a) # Missions.

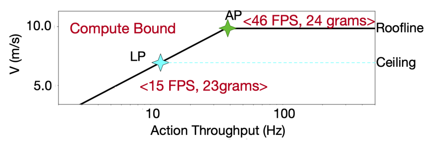


(b) Mapping AP and HT to the F-1 plot.

Figure 8: (a) Comparison of high performance (HP) and AutoPilot design (AP) in terms of the number of missions. (b) F-1 plot for nano-UAV (Table IV) to understand the degradation when using HT over AP.



(a) # Missions.



(b) Mapping LP and HT to the F-1 plot.

Figure 9: (a) Comparison of LP and AP in terms of missions. (b) F-1 plot for nano-UAV (Table IV) to understand the degradation when using LP over AP.

UAV as shown in Fig. 6. For instance, as the complexity of the task increases, the compute design becomes bigger (as seen by the larger spread in parameter space Fig. 6). But AutoPilot only chooses those parameters to keep the power and weight nearly identical (or never exceed a certain limit). Manually enumerating these points (isolated or one-size accelerator designs) would not maximize the mission performance as the UAV type or deployment scenario changes. However, automating the co-design space allows AutoPilot to consistently generate DSSoC designs that maximize mission performance, even when changing UAV types.

B. Pitfalls of Conventional Domain-Agnostic DSE

In the following experiments, we demonstrate the importance of having a balanced DSSoC for UAVs rather than selecting onboard compute based on traditional SoC design strategies: maximizing compute throughput alone; optimizing for low power alone, and even Pareto optimal selection with respect to both compute throughput and power (compute efficiency). Our results show that compared to high throughput (HT), low power (LP), or high compute efficiency (HE), a balanced DSSoC design by AutoPilot maximizes overall mission efficiency in autonomous UAVs.

1) HT, LP, HE vs. AutoPilot (AP) DSSoCs: Ideally, what differentiates between AutoPilot generated DSE versus traditional DSE (e.g., [66]) is the critical Phase 3 (See Fig. 1). Hence, we take an intermediate DSSoC design from Phase 2 of AutoPilot and show how a lack of full-system design degrades the mission performance of autonomous UAVs, even though on isolated compute metrics, the traditional designs outperform AutoPilot generated designs.

To demonstrate the degradation, we choose a nano-UAV whose specifications are in Table IV. The output of Phase 2 (from Fig. 1) for this task is shown in Fig. 7. Out of many Pareto optimal designs, we label three DSSoC designs in Fig. 7 as ‘HT’ (high throughput design), ‘LP’ (low-power design), and ‘HE’ (High efficiency) to highlight the design strategies outlined above. Additionally, we label the AutoPilot selected design (labeled as ‘AP’) that performs full-system co-design to select the onboard DSSoC design. The AP DSSoC is not Pareto optimal as it is neither high throughput, low-power, nor high compute efficiency (performance/watt) DSSoC design (but later, we show how it is, in fact, the best for this nano-UAV).

Fig. 7 results show that all traditional designs (HT, LP, and HE) outperform AP purely on isolated compute metrics. For instance, comparing high-throughput design (HT) with AutoPilot generated DSSoC design (AP) for the same E2E algorithm, we see from Fig. 7(c) that HT achieves 4.47× (due to large PE and memory capacity as shown in Fig. 7(b)) throughput than AP; hence should be able to process the algorithm faster to generate high-level decisions.

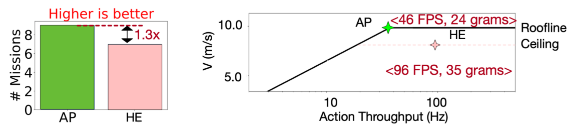
Likewise, Fig. 7(d) compares the low-power design and the AutoPilot DSSoC design. The low-power design (LP) consumes 1.23× less power compared (due to less PE’s and memory capacity as shown in Fig. 7(b)) to AutoPilot’s design (AP). Hence, the LP design should consume less energy from the battery to run the same E2E algorithm.

Lastly, Fig. 7(e) compares the high-efficiency design (HE) with AutoPilot design. The HE design achieves 96 FPS at 1.5 W (≈ 64 FPS/W) compared to AutoPilot generated design which achieves 46 FPS at 0.83 W (≈ 55 FPS/W) running the same E2E algorithm. Hence, a high-efficiency design should run the autonomy algorithm 1.16× efficiently compared AutoPilot design for the same power.

However, when the full UAV system and UAV mission-level performance are taken into account, we show that AutoPilot generated designs consistently outperform these traditional design choices in mission-level performance. Figures 8a-10a show that AP consistently performs better than HT, LP, and HE by 2.25×, 1.8×, 1.3×, respectively, in terms of the number of missions performed. In Section V-B2, Section V-B3, and Section V-B4 we perform a deep-dive analysis to understand why AutoPilot generated designs outperform traditional architectural DSE designs on mission-level metrics.

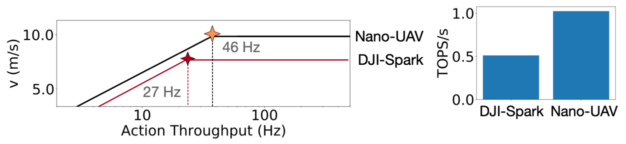
2) Pitfalls of High-Throughput (HT) DSSoC: A high compute throughput for a given autonomy algorithm allows the UAV to make decisions faster, which can be helpful in a highly dynamic environment. As the complexity of the autonomy algorithm increases, choosing high-throughput designs can be the traditional choice from an architectural DSE. However, traditional high throughput design does not directly translate to an overall increase in the number of missions. Furthermore, these design choices do not consider the domain-specific characteristics of UAVs and hence have pitfalls that require careful introspection.

While Fig. 7(c) shows that HT achieves 4.47× higher



(a) # Missions. **(b)** Mapping LP and HE to the F-1 plot.

Figure 10: (a) Comparison of HE and AP in terms of the number of missions. (b) F-1 plot for nano-UAV (Table IV) to understand the degradation when using HE over AP.



(a) Number of missions. **(b)** DSSoC requirement.

Figure 11: UAV agility increases compute throughput requirement.

throughput than AutoPilot's AP design, AP outperforms HT by 2.25× in Figure 8a for mission metrics. HT achieves a throughput of 205 FPS while consuming 8.24 W (65 g), whereas AP achieves a throughput of 46 FPS at 0.7 W (24 g). Comparing these two designs purely on isolated metrics, HT is more throughput than AP; hence should be able to process the E2E autonomy algorithm faster. But the AP DSSoC's lighter compute payload weight has less impact on the UAV's body dynamics while delivering enough compute throughput that maximizes the safe velocity. Thanks to the full-system UAV co-design in Phase 3 of AutoPilot, AP outperforms HP in the number of missions by 2.25 ×.

To provide deeper insights we plot these DSSoC candidates on the F-1 model [46] for the nano-UAV (Table IV). Recall from Section III-C that the F-1 model is used to understand an optimal DSSoC design for a given UAV and also shows the effects of the performance of the sensor-compute-control pipeline and how the weight of the payload affects the UAV physics. Since the high throughput design (HT) consumes 11.7 × more power than AutoPilot design (AP), the heatsink needed to cool HT is also larger than what is required to cool the AP system. Larger heatsink weight adds to the overall weight of the HT system, which lowers the UAV's ability to move faster (and safely). This effect is captured in F-1 plot shown in Fig. 8b. The reduction in safe velocity affects the UAV's mission time, energy, and the number of missions (Eq. 4).

3) Pitfalls of Low Power (LP) DSSoC: A low-power design (LP) does not always reduce the mission energy. Fig. 9a shows the mission metrics for LP and AP. AP achieves 1.8× more missions compared to LP.

While conventional wisdom recommends DSE to select a low-power DSSoC design as UAVs have a limited onboard battery, AutoPilot's DSSoC methodology balances the DSSoC power with other domain-specific components to maximize UAV's safe velocity. Fig. 9b shows the F-1 roofline plot for LP and AP. The LP design achieves an action throughput of 18.4 Hz, which is 2.5× lower than what the UAV's physics can allow. Due to the lower decision-making rate for LP, the UAV must lower its safe flying velocity to fly safely. Lowering safe velocity increases the mission flight time, thus increasing the mission's total energy consumption (Eq. 2).

Therefore, we conclude that choosing a low-power onboard compute does not necessarily lower the mission energy since the decision-making rate (action throughput) also plays a role in the UAV's ability to fly faster, which can lower mission time and overall mission energy. Thus, compromising performance for low pow consumption can degrade mission performance, and full-system UAV co-design is necessary to ensure optimal onboard compute is selected for a given UAV system.

4) Pitfalls of High-Efficiency (HE) DSSoC: General design methodologies co-designing HW-SW together [66] focus on selecting higher efficiency designs (i.e., throughput/W). However, when designing DSSoCs for UAVs, these higher compute efficiency designs do not always maximize UAV's mission-level performance. To demonstrate this, Fig. 10a shows the mission level metrics of HE and AP designs. We observe that AP achieves 1.3× more number of missions compared to HE. However, despite HE achieving higher energy efficiency (64 FPS/W) compared to AP (55 FPS/W), we still observe degradation in HE's mission-level metrics.

Fig. 10b shows the F-1 plot with AutoPilot design and high-efficiency (HE). The knee-point for the nano-UAV is around 46 FPS, whereas the HE achieves a throughput of 96 FPS at 1.5 W (over-provisioned by ∼2×). The over-provisioned HE design also consumes relatively higher power. Thus, the heatsink required to cool HE design will be higher than AP, thus increasing the compute weight for the HE design. The increasing payload weight lowers the nano-UAV's ability to move faster (Fig. 4a), thus this lowers the safe velocity, captured by the F-1 model's ceilings, which in turn lowers the mission energy and the number of missions (Eq 4).

C. UAV Agility's Impact on DSSoC Design

In this section, we demonstrate how UAV's need for agility impacts the DSSoCs design requirements needed to run the autonomy algorithms. UAV's agility is typically characterized by its maximum acceleration (which depends upon the UAV's thrust-to-weight ratio) [25], [57]. However, it is important to note that the increase in payload weight (onboard compute, heatsink, etc.) lowers the thrust-to-weight ratio (lowers maximum acceleration), making the UAV less agile. Hence, there is a *need to account for these physical effects when designing onboard compute* for these different classes of UAVs.

To demonstrate the increase in compute requirement with UAV's agility, we take two UAV's namely DJI-spark and

	Single DSSoC Design			General Purpose Designs	
	Knee-Point (Low Obs.)	**Knee-Point (Med. Obs.)**	**Knee-Point (Dense Obs.)**	**Nvidia TX2**	**Intel NCS**
# of Missions Degradation	30%	0%	27%	30%	67%
Comments	Compute Bound lowers Vsafe	Optimal Design	Weight lowers the roofline	Weight lowers the roofline	Compute Bound lowers Vsafe

Table V: Design Trade-off comparisons.

nano-UAV [89]. The specification of these UAVs is in Table IV. The nano-UAV has a more thrust-to-weight ratio than DJI-Spark. We assume that both UAVs are equipped with 60 FPS sensors (to avoid being sensor-bound).

Fig. 11a shows the mapping of DJI-Spark and nano-UAVs to the F-1 model [46]. The optimal compute throughput required to maximize the mission performance for the DJI-spark is around 27 Hz, whereas, for the nano-UAV, it is 46 Hz. The respective knee-points suggest that the sensor-compute-control pipeline needs to make decisions at 27 FPS for DJI-spark and 46 FPS for nano-UAV to maximize the safe velocity.

AutoPilot's DSSoC methodology performs full-system UAV co-design to select the designs closer to the optimal point in the F-1 model. For example, for the nano-UAV, AutoPilot chooses design 2$\times$ more compute throughput than the design point it chooses for DJI-spark without affecting the UAV physics as shown in Fig. 11b.

As UAVs become more agile and smaller, the need for balanced and high-performance DSSoCs increases. In such scenarios, the AutoPilot methodology is advantageous since it automates the full-system UAV co-design to generate DSSoC that maximizes mission performance without impacting the UAV's physics.

VI. Specialization Cost vs. Mission Efficiency

Our results demonstrate that when the UAV components or type changes, we see the need for re-optimizing (or a new custom design). Customization for UAVs is a trade-off between operational efficiency and design effort (cost). We quantify the trade-off between using single designs versus domain-specific hardware.

In this case, we take the AutoPilot design point for a medium obstacle density scenario for a mini-UAV. We compare this design point against general-purpose design (e.g., TX2 and Intel NCS) and optimal specialized HW accelerators designed for other scenarios (e.g., the AutoPilot generated designs for low obstacle and dense-obstacle scenarios but reused medium obstacles scenario). Table V tabulates the cost of these trade-offs. Compared to the deployment-specific hardware specialization, we observe 27% to 67% degradation in the number of missions possible depending upon the choice of onboard compute. Hence, deployment-specific hardware specialization is key to achieving that goal if operational efficiency is important. However, if cost is critical, then general-purpose designs (or reusing single designs) can save design costs with a 27% to 67% reduction in the number of missions. However, it is important to note that this can also increase the operational cost of grounding and recharging the UAVs frequently.

In summary, the trade-off between mission efficiency and the cost of computing exists, but the AutoPilot DSSoC methodology can reduce the DSSoC design complexity (cost).

VII. Methodology Generalization Discussion

AutoPilot methodology splits the DSSoC design problem for UAVs into three phases namely, *Domain-Specific Front End*, *Domain-Agnostic Multiobjective HW-SW Co-Design* and the *Domain-Specific Back End*. Our results consistently demonstrate the efficacy of our methodology in designing DSSoCs for autonomous UAVs. To that end, in this section, we provide a qualitative discussion on how AutoPilot methodology (Fig. 1) can extend to other closely related domains where the onboard computer is just of the many components. Since autonomous UAVs by themselves are quite complex, and the relationship between various UAV components is quite nuanced, we primarily focus on one domain and quantitatively demonstrate how the AutoPilot DSSoC methodology can automatically generate many DSSoCs for various UAVs and deployment scenarios. Other autonomous vehicle domains, such as self-driving cars, are much more complex due to the driving rules (speed limit, traffic signs, road topology). Our learnings from designing DSSoCs for autonomous UAVs provide a generalizable template to extend to other robotic domains. Nonetheless, we strongly believe quantitative validation is necessary to fully discern various trade-offs and potential gains for individual domains.

Table VI provides a generic high-level taxonomy of AutoPilot methodology for other closely related domains such as self-driving cars and articulated robots (i.e., robotic arms). Our experience suggests an opportunity to design characterization tools like F-1 that couples physics, sensing, compute latency, and actuation latency for articulated robots. Also, more importantly, similar to UAVs, we believe we should not optimize for isolated compute metrics when targeting articulated robots; rather, we must think of the holistic autonomous system. Next, we explain how we would extend the AutoPilot methodology to each of these domains individually.

UAV with SPA Autonomy Algorithms. *Phase 1.* In the domain-specific front end, the Air Learning simulator [43] used in this work can be replaced with AirSim [74] or MAVBench [17] as the robot simulator to develop and validate SPA based autonomy algorithm. *Phase 2.* In domain-agnostic multi-objective HW-SW co-design, the systolic array template used in this work can be replaced by hardware templates for SLAM [80], OctoMAP [37], motion-planning accelerators [70]. *Phase 3.* In the domain-specific back end, we can still use the F-1 model [46] to evaluate the impact of other UAV parameters on the DSSoC design.

Domain	Autonomy Algorithm Paradigm	Phase 1 Domain-Specific Front End	Phase 2 Domain-Agnostic Multiobjective HW-SW DSE		Phase 3 Domain-Specific Back End
		Robot Simulator Environment	**HW-SW Co-Design**	**ML-Based Method**	**AV Safety Model**
UAV (Our Work)	*E2E (Section 2)*	*Air Learning [43]*	*Systolic Arrays [72]*	*BO [33]*	*F-1 Model [45]*
UAVs	*E2E (See Section 2)*	*PEDRA [14], AirSim [74], Gym-FC [41]*	*Systolic Arrays [26], [72], Simba [77], Edge-TPUs [38], Eyeriss [19], Mapping Optimization [34], [47], Movidius [10], MCU [22], PULP [60], Magnet [85]*	*BO [33], RL [81], GA [88], SA [84]*	*F-1 Model [45]*
	SPA (Section 2)	*MavBench [17]*	*Perception: SLAM [80], Octomap [37] Motion Planning: Robox [70]*	*BO [33], RL [81], GA [88], SA [84]*	
Self-Driving Cars	*Hybrid (PPC+NN)*	*CARLA [21], Appollo [62], AirSim [74],*	*Systolic Arrays [26], [72], Simba [77], Eyeriss [19], EyeQ [36], Tesla FSD [82], Magnet [85]*	*BO [33], RL [81], GA [88], SA [84]*	*Intel RSS [76], Nvidia SFF [59]*
Articulated Robots	*End-to-End Learning (NN-Based)*	*Robot Farms (Qt-OPT), Gazebo*	*Systolic Arrays [26], [72], Simba [77], Eyeriss [19], EyeQ [36], Tesla FSD [82], Magnet [85]*	*BO [33], RL [81], GA [88], SA [84]*	*ANYpulator [87]*
	SPA	*Gazebo [42]*	*Perception: SLAM [80], Octomap [37] Motion Planning: Murray et al. [55], Robomorphic Computing [58], RACOD [15]*		

Table VI: This work (cells highlighted in green) uses a specific set of frameworks to build three phases (Domain-specific Front End, Domain-Agnostic Multiobjective HW-SW DSE, and Domain-Specific Back End) as laid out by the AutoPilot methodology for DSSoC design for UAVs. Our evaluation (Section V) shows that it is important to consider these three phases as it improves the mission-level performance of UAVs. Nonetheless, future work can construct a similar methodology for other autonomous vehicles using other components which serve a similar purpose. To that end, we provide a taxonomy for extending AutoPilot methodology to other closely related domains, such as self-driving cars and articulated robots (cells highlighted in gray). In the ML-based methods, BO refers to Bayesian Optimization, RL refers to Reinforcement Learning, GA refers to Genetic Algorithms, and SA refers to Simulated Annealing.

Self-Driving Cars. In *Phase 1*, Air Learning can be replaced with Apollo [62], CARLA [21] or AirSim [74] to validate the autonomy algorithms. In *Phase2* custom hardware accelerators [19], [26], [36], [77], [82], [85] to run these multiple neural networks efficiently can replace the systolic array-based hardware template in phase 2. In *Phase 3*, the F-1 model can be replaced with Intel RSS [76], Nvidia SFF [59] or its derivatives as the safety model.

Robotic Arms and Other Articulated Robots. For robotic arms, the Air Learning simulator in phase 1 can be replaced with Gazebo [42] or QT-OPT [39] with the articulated robot model. In Phase 2, the SCALE-Sim [72] simulator used in this work can be replaced with any architectural DSE tool or hardware accelerator template. For SPA, the hardware accelerator template should include perception and mapping [37], motion-planning [55], [58]. For E2E-based autonomy algorithms for these robots, any NN hardware accelerator template [19], [26], [34], [38], [47], [85] can be used instead of SCALE-Sim. In phase 3, the F-1 model [46] used in this work can be replaced with safety model for the robot arm [16], [87]. In general, there is an opportunity to develop a unified characterization and bottleneck analysis tools like F-1 [46], RSS [76], and SFF [59], which tie the sensor, compute, and robot physics interactions to evaluate their impact on onboard compute design for these articulated robots.

Choice of ML-Based Optimizer. Lastly, irrespective of the domain, Bayesian optimization used in phase 2 of this work can be replaced with the genetic algorithms [88], reinforcement learning [81], or simulated annealing [84] to perform multi-objective HW-SW DSE.

VIII. Conclusion

AutoPilot is a DSSoC design methodology that automatically generate an optimal autonomy algorithm (E2E Model) and its hardware accelerator from a high-level user specification for autonomous UAVs. It can be adapted to perform full-system co-design for the Sense-Plan-Act (SPA) paradigm. The only requirement for AutoPilot is that the SPA-based algorithm and hardware templates be parameterizable. Moreover, the general methodology we have developed for AutoPilot, such as full-stack UAV co-design for identifying the balanced design point, architectural fine-tuning, and selecting the optimal design points by showing how it affects the overall mission can also be adapted to other types of autonomous vehicles such as self-driving cars and "ground" drones.

Acknowledgment

The authors would like to thank anonymous reviewers for their feedback on improving the manuscript's quality. The authors would also like to thank Dr. Gage Hills (Harvard) for providing constructive feedback during several iterations of this paper. The work was sponsored in part by IARPA award 2022-21100600004 and 2022-21102100013. Part of this work was performed under the auspices of the U.S. Department of

Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

References

[1] https://github.com/harvard-edge/airlearning, title = Air Learning Environment Generator, year = 2020.

[2] https://www.bitcraze.io/products/crazyflie-2-1/, title = CrazyFlie 2.1 Product Page, year = 2020.

[3] https://www.ovt.com/sensors/OV9755, title = OV9755,Color CMOS 720p (1280x720) HD Image Sensor with OmniPixelÂ®3-HS Technology, year = 2020.

[4] Arm Cortex-M33. [Online]. Available: https://developer.arm.com/Processors/Cortex-M33

[5] ARMv8-M Technical Reference manual. [Online]. Available: https://static.docs.arm.com/ddi0553/a/DDI0553A_e_armv8m_arm.pdf

[6] "Coral-SoM-datasheet." [Online]. Available: https://coral.ai/docs/som/datasheet/

[7] "Heat sink size calculator," https://celsiainc.com/resources/calculators/heat-sink-size-calculator/, (Accessed on 01/29/2020).

[8] "Raspberry pi," https://www.pololu.com/blog/598/new-product-raspberry-pi-3-model-b.

[9] "Micron ddr4 power calculator." https://media-www.micron.com/-/media/client/global/documents/products/power-calculator/ddr4_power_calc.xlsm?rev=a8a5e30d8a7e41c4adcaad2df73934b4, 2016.

[10] "News announcements," https://www.movidius.com/news/intel-movidius-myriad-2-vpu-enables-advanced-computer-vision-and-deep-learn, May 2017.

[11] "All about multirotor drone fpv flight controllers." https://www.getfpv.com/learn/new-to-fpv/all-about-multirotor-fpv-drone-flight-controller/, 2020.

[12] "Pixhawk 1 flight controller." https://docs.px4.io/v1.9.0/en/flight_controller/pixhawk.html, 2020.

[13] M. Achtelik, T. Zhang, K. Kuhnlenz, and M. Buss, "Visual tracking and control of a quadcopter using a stereo camera system and inertial sensors," in *2009 International Conference on Mechatronics and Automation*. IEEE, 2009, pp. 2863–2869.

[14] A. Anwar and A. Raychowdhury, "Autonomous navigation via deep reinforcement learning for resource constraint edge nodes using transfer learning," *IEEE Access*, vol. 8, pp. 26 549–26 560, 2020.

[15] M. Bakhshalipour, S. B. Ehsani, M. Qadri, D. Guri, M. Likhachev, and P. B. Gibbons, "Racod: algorithm/hardware co-design for mobile robot path planning," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, 2022, pp. 597–609.

[16] K. Bodie, C. D. Bellicoso, and M. Hutter, "Anypulator: Design and control of a safe robotic arm," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016, pp. 1119–1125.

[17] B. Boroujerdian, H. Genc, S. Krishnan, W. Cui, A. Faust, and V. J. Reddi, "Mavbench: Micro aerial vehicle benchmarking," in *Proceedings of the 51st Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-51. IEEE Press, 2018, pp. 894–907. [Online]. Available: https://doi.org/10.1109/MICRO.2018.00077

[18] M. Calonder, V. Lepetit, M. Ozuysal, T. Trzcinski, C. Strecha, and P. Fua, "Brief: Computing a local binary descriptor very fast," *IEEE transactions on pattern analysis and machine intelligence*, vol. 34, no. 7, pp. 1281–1298, 2011.

[19] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE journal of solid-state circuits*, vol. 52, no. 1, pp. 127–138, 2016.

[20] M. G. Dissanayake, P. Newman, S. Clark, H. F. Durrant-Whyte, and M. Csorba, "A solution to the simultaneous localization and map building (slam) problem," *IEEE Transactions on robotics and automation*, vol. 17, no. 3, pp. 229–241, 2001.

[21] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "Carla: An open urban driving simulator," in *Conference on robot learning*. PMLR, 2017, pp. 1–16.

[22] B. P. Duisterhof, S. Krishnan, J. J. Cruz, C. R. Banbury, W. Fu, A. Faust, G. C. H. E. de Croon, and V. J. Reddi, "Tiny robot learning (tinyrl) for source seeking on a nano quadcopter," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2021.

[23] A. Elfes, "Using occupancy grids for mobile robot perception and navigation," *Computer*, vol. 22, no. 6, pp. 46–57, 1989.

[24] L. Espeholt, R. Marinier, P. Stanczyk, K. Wang, and M. Michalski, "Seed rl: Scalable and efficient deep-rl with accelerated central inference," *arXiv preprint arXiv:1910.06591*, 2019.

[25] D. Falanga, S. Kim, and D. Scaramuzza, "How fast is too fast? the role of perception latency in high-speed sense and avoid," *IEEE Robotics and Automation Letters*, vol. 4, pp. 1884–1891, 2019.

[26] H. Genc, S. Kim, A. Amid, A. Haj-Ali, V. Iyer, P. Prakash, J. Zhao, D. Grubb, H. Liew, H. Mao *et al.*, "Gemmini: Enabling systematic deep-learning architecture evaluation via full-stack integration," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 769–774.

[27] A. Giusti, J. Guzzi, D. C. Cireşan, F. He, J. P. Rodriguez, F. Fontana, M. Faessler, C. Forster, J. Schmidhuber, G. D. Caro, D. Scaramuzza, and L. M. Gambardella, "A machine learning approach to visual perception of forest trails for mobile robots," *IEEE Robotics and Automation Letters*, vol. 1, no. 2, pp. 661–667, 2016.

[28] D. Gonzalez, J. Perez, V. Milanes, and F. Nashashibi, "A review of motion planning techniques for automated vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 4, pp. 1135–1145, 2016.

[29] T. Guo, N. Jiang, B. Li, X. Zhu, Y. Wang, and W. Du, "Uav navigation in high dynamic environments: A deep reinforcement learning approach," *Chinese Journal of Aeronautics*, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1000936120302247

[30] D. Gurdan, J. Stumpf, M. Achtelik, K.-M. Doth, G. Hirzinger, and D. Rus, "Energy-efficient autonomous four-rotor flying robot controlled at 1 khz," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*. IEEE, 2007, pp. 361–366.

[31] R. Hadidi, B. Asgari, S. Jijina, A. Amyette, N. Shoghi, and H. Kim, "Quantifying the design-space tradeoffs in autonomous drones," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS 2021. New York, NY, USA: Association for Computing Machinery, 2021, pp. 661–673. [Online]. Available: https://doi.org/10.1145/3445814.3446721

[32] C. B. Harris and R. I. Bahar, "A research tool for the power and performance analysis of sensor-based mobile robots," in *2017 New Generation of CAS (NGCAS)*, 2017, pp. 25–28.

[33] M. Havasi and J. M. Lobato, "Bayesian optimization," https://github.com/cambridge-mlg/gem5-aladdin/tree/master/bo_script, 2018.

[34] K. Hegde, P.-A. Tsai, S. Huang, V. Chandra, A. Parashar, and C. W. Fletcher, "Mind mappings: enabling efficient algorithm-accelerator mapping space search," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2021, pp. 943–958.

[35] M. Hoffman, B. Shahriari, J. Aslanides, G. Barth-Maron, F. Behbahani, T. Norman, A. Abdolmaleki, A. Cassirer, F. Yang, K. Baumli *et al.*, "Acme: A research framework for distributed reinforcement learning," *arXiv preprint arXiv:2006.00979*, 2020.

[36] Intel, "Mobileye and nio partner to bring level 4 autonomous vehicles to consumers in china and beyond." https://newsroom.intel.com/news/mobileye-nio-partner-bring-level-4-autonomous-vehicles-consumers-china-beyond/#gs.3l52m2.

[37] T. Jia, E.-Y. Yang, Y.-S. Hsiao, J. Cruz, D. Brooks, G.-Y. Wei, and V. J. Reddi, "Omu: A probabilistic 3d occupancy mapping accelerator for real-time octomap at the edge," in *Design, Automation and Test in Europe (DATE)*, Mar 2022.

[38] A. Jones, A. Yazdanbakhsh, B. Akin, C. Angermueller, J. P. Laudon, K. Swersky, M. Hashemi, R. Narayanaswami, S. Chatterjee, and Y. Zhou, "Apollo: Transferable architecture exploration," in *ML for Systems Workshop at NeurIPS 2020*, 2020.

[39] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke *et al.*, "Scalable deep reinforcement learning for vision-based robotic manipulation," in *Conference on Robot Learning*. PMLR, 2018, pp. 651–673.

[40] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The international journal of robotics research*, vol. 30, no. 7, pp. 846–894, 2011.

[41] W. Koch, R. Mancuso, and A. Bestavros, "Neuroflight: Next generation flight control firmware," *arXiv preprint arXiv:1901.06553*, 2019.

[42] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, vol. 3, 2004, pp. 2149–2154 vol.3.

[43] S. Krishnan, B. Boroujerdian, W. Fu, A. Faust, and V. J. Reddi, "Air learning: a deep reinforcement learning gym for autonomous aerial robot visual navigation," in *Machine Learning (Special Issue on Reinforcement Learning for Real Life)*. Springer, 2021, pp. 1–40. [Online]. Available: https://doi.org/10.1007/s10994-021-06006-6

[44] S. Krishnan, M. Lam, S. Chitlangia, Z. Wan, G. Barth-maron, A. Faust, and V. J. Reddi, "QuaRL: Quantization for fast and environmentally sustainable reinforcement learning," *Transactions on Machine Learning Research*, 2022. [Online]. Available: https://openreview.net/forum?id=xwWsiFmUEs

[45] S. Krishnan, Z. Wan, K. Bhardwaj, A. Faust, and V. J. Reddi, "Roofline model for uavs: A bottleneck analysis tool for designing compute systems for autonomous drones," *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2022.

[46] S. Krishnan, Z. Wan, K. Bhardwaj, P. Whatmough, A. Faust, G.-Y. Wei, D. Brooks, and V. J. Reddi, "The sky is not the limit: A visual performance model for cyber-physical co-design in autonomous machines," *IEEE Computer Architecture Letters*, vol. 19, no. 1, pp. 38–42, 2020.

[47] H. Kwon, P. Chatarasi, V. Sarkar, T. Krishna, M. Pellauer, and A. Parashar, "Maestro: A data-centric approach to understand reuse, performance, and hardware cost of dnn mappings," *IEEE micro*, vol. 40, no. 3, pp. 20–29, 2020.

[48] H. Li, M. Bhargav, P. N. Whatmough, and H. . Philip Wong, "On-chip memory technology design space explorations for mobile deep neural network accelerators," in *2019 56th ACM/IEEE Design Automation Conference (DAC)*, June 2019, pp. 1–6.

[49] S. Li, K. Chen, J. H. Ahn, J. B. Brockman, and N. P. Jouppi, "Cacti-p: Architecture-level modeling for sram-based structures with advanced leakage reduction techniques," in *Proceedings of the International Conference on Computer-Aided Design*. IEEE Press, 2011, pp. 694–701.

[50] K. Lim, G. S. Kim, S. Kim, and K. Baek, "A multi-lane mipi csi receiver for mobile camera applications," *IEEE Transactions on Consumer Electronics*, vol. 56, no. 3, pp. 1185–1190, Aug 2010.

[51] S. Liu, M. Watterson, S. Tang, and V. Kumar, "High speed navigation for quadrotors with limited onboard sensing," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 1484–1491.

[52] A. Loquercio, A. I. Maqueda, C. R. Del-Blanco, and D. Scaramuzza, "Dronet: Learning to fly by driving," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 1088–1095, 2018.

[53] Y.-C. Lu, Z.-Y. Chen, and P.-C. Chang, "Low power multi-lane mipi csi-2 receiver design and hardware implementations," in *2013 IEEE International Symposium on Consumer Electronics (ISCE)*, 2013, pp. 199–200.

[54] K. Mohta, M. Watterson, Y. Mulgaonkar, S. Liu, C. Qu, A. Makineni, K. Saulnier, K. Sun, A. Zhu, J. Delmerico *et al.*, "Fast, autonomous flight in gps-denied and cluttered environments," *Journal of Field Robotics*, vol. 35, no. 1, pp. 101–120, 2018.

[55] S. Murray, W. Floyd-Jones, Y. Qi, D. J. Sorin, and G. D. Konidaris, "Robot motion planning on a chip." in *Robotics: Science and Systems*, 2016.

[56] S. Murray, W. Floyd-Jones, Y. Qi, G. Konidaris, and D. J. Sorin, "The microarchitecture of a real-time robot motion planning accelerator," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2016, pp. 1–12.

[57] NASA, "Thrust to weight ratio." https://www.grc.nasa.gov/www/k-12/airplane/fwrat.html.

[58] S. M. Neuman, B. Plancher, T. Bourgeat, T. Tambe, S. Devadas, and V. J. Reddi, "Robomorphic computing: A design methodology for domain-specific accelerators parameterized by robot morphology," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS 2021. New York, NY, USA: Association for Computing Machinery, 2021, pp. 674–686. [Online]. Available: https://doi.org/10.1145/3445814.3446746

[59] D. Nistér, H.-L. Lee, J. Ng, and Y. Wang, "The safety force field," *NVIDIA White Paper*, 2019.

[60] D. Palossi, A. Loquercio, F. Conti, E. Flamand, D. Scaramuzza, and L. Benini, "A 64mw dnn-based visual navigation engine for autonomous nano-drones," *IEEE Internet of Things Journal*, 2019.

[61] S. Park, L. Zhang, and S. Chakraborty, "Design space exploration of drone infrastructure for large-scale delivery services," in *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. ACM, 2016, pp. 1–7.

[62] Z. Peng, J. Yang, T.-H. P. Chen, and L. Ma, *A First Look at the Integration of Machine Learning Models in Complex Autonomous Driving Systems: A Case Study on Apollo*. New York, NY, USA: Association for Computing Machinery, 2020, pp. 1240–1250. [Online]. Available: https://doi.org/10.1145/3368089.3417063

[63] P. P.Lee, L. J. Bernstein, R. M. Guidash, and T.-H. Lee, "Integrated cmos active pixel digital camera," Patent, 2004.

[64] W. Ponweiser *et al.*, "Multiobjective optimization on a limited budget of evaluations using model-assisted S-Metric selection," in *PPSN*, 2008, pp. 784–794.

[65] C. Rasmussen and C. Williams, "Gaussian processes for machine learning," *MIT press, Cambridge, MA*, 2005.

[66] B. Reagen, J. M. Hernández-Lobato, R. Adolf, M. Gelbart, P. Whatmough, G.-Y. Wei, and D. Brooks, "A case for efficient accelerator design space exploration via bayesian optimization," in *2017 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. IEEE, 2017, pp. 1–6.

[67] S. Ross, N. Melik-Barkhudarov, K. S. Shankar, A. Wendel, D. Dey, J. A. Bagnell, and M. Hebert, "Learning monocular reactive uav control in cluttered natural environments," in *2013 IEEE international conference on robotics and automation*. IEEE, 2013, pp. 1765–1772.

[68] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," in *Computer Vision – ECCV 2006*, A. Leonardis, H. Bischof, and A. Pinz, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 430–443.

[69] R. B. Rusu and S. Cousins, "3d is here: Point cloud library (pcl)," in *2011 IEEE international conference on robotics and automation*. IEEE, 2011, pp. 1–4.

[70] J. Sacks, D. Mahajan, R. C. Lawson, and H. Esmaeilzadeh, "Robox: an end-to-end solution to accelerate autonomous control in robotics," in *Proceedings of the 45th Annual International Symposium on Computer Architecture*. IEEE Press, 2018, pp. 479–490.

[71] F. Sadeghi and S. Levine, "Cad2rl: Real single-image flight without a single real image," in *Robotics: Science and Systems Conference*. IEEE, 2017.

[72] A. Samajdar, J. M. Joseph, Y. Zhu, P. Whatmough, M. Mattina, and T. Krishna, "A systematic methodology for characterizing scalability of dnn accelerators using scale-sim," in *2020 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2020, pp. 58–68.

[73] N. J. Sanket, C. M. Parameshwara, C. D. Singh, A. V. Kuruttukulam, C. FermÃ¼ller, D. Scaramuzza, and Y. Aloimonos, "Evdodgenet: Deep dynamic obstacle dodging with event cameras," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 10 651–10 657.

[74] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "Airsim: High-fidelity visual and physical simulation for autonomous vehicles," in *FSR*, 2017.

[75] B. Shahriari *et al.*, "Taking the human out of the loop: a review of Bayesian optimization," *Proceedings of the IEEE*, pp. 148–175, 2016.

[76] S. Shalev-Shwartz, S. Shammah, and A. Shashua, "On a formal model of safe and scalable self-driving cars," *arXiv preprint arXiv:1708.06374*, 2017.

[77] Y. S. Shao, J. Clemons, R. Venkatesan, B. Zimmer, M. Fojtik, N. Jiang, B. Keller, A. Klinefelter, N. Pinckney, P. Raina *et al.*, "Simba: Scaling deep-learning inference with multi-chip-module-based architecture," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 14–27.

[78] N. Smolyanskiy, A. Kamenev, J. Smith, and S. Birchfield, “Toward low-flying autonomous mav trail navigation using deep neural networks for environmental awareness,” in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 4241–4247.

[79] J. Snoek, H. Larochelle, and R. P. Adams, “Practical Bayesian optimization of machine learning algorithms,” in *NIPS*, 2012, pp. 2960–2968.

[80] A. Suleiman, Z. Zhang, L. Carlone, S. Karaman, and V. Sze, “Navion: A 2-mw fully integrated real-time visual-inertial odometry accelerator for autonomous navigation of nano drones,” *IEEE Journal of Solid-State Circuits*, vol. 54, no. 4, pp. 1106–1119, 2019.

[81] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[82] Tesla, “Tesla‘s autopilot,” https://www.tesla.com/autopilotAI.

[83] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, “Domain randomization for transferring deep neural networks from simulation to the real world,” in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 23–30.

[84] P. J. Van Laarhoven and E. H. Aarts, “Simulated annealing,” in *Simulated annealing: Theory and applications*. Springer, 1987, pp. 7–15.

[85] R. Venkatesan, Y. S. Shao, M. Wang, J. Clemons, S. Dai, M. Fojtik, B. Keller, A. Klinefelter, N. Pinckney, P. Raina *et al.*, “Magnet: A modular accelerator generator for neural networks,” in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2019, pp. 1–8.

[86] Z. Wan, B. Yu, T. Y. Li, J. Tang, Y. Zhu, Y. Wang, A. Raychowdhury, and S. Liu, “A survey of fpga-based robotic computing,” *IEEE Circuits and Systems Magazine*, vol. 21, no. 2, pp. 48–74, 2021.

[87] M. Wassink and S. Stramigioli, “Towards a novel safety norm for domestic robotics,” in *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2007, pp. 3354–3359.

[88] D. Whitley, “A genetic algorithm tutorial,” *Statistics and computing*, vol. 4, no. 2, pp. 65–85, 1994.

[89] X. Zhang, B. Xian, B. Zhao, and Y. Zhang, “Autonomous flight control of a nano quadrotor helicopter in a gps-denied environment using on-board vision,” *IEEE Transactions on Industrial Electronics*, vol. 62, no. 10, pp. 6392–6403, 2015.