2024 IEEE International Conference on Robotics and Automation (ICRA 2024)
May 13-17, 2024. Yokohama, Japan

On-device Self-supervised Learning of Visual Perception Tasks aboard Hardware-limited Nano-quadrotors

Elia Cereda[1], Manuele Rusci[2], Alessandro Giusti[1], and Daniele Palossi[13]

***Abstract*— Sub-50 g nano-drones are gaining momentum in both academia and industry. Their most compelling applications rely on onboard deep learning models for perception despite severe hardware constraints (i.e., sub-100 mW processor). When deployed in unknown environments not represented in the training data, these models often underperform due to domain shift. To cope with this fundamental problem, we propose, for the first time, on-device learning aboard nano-drones, where the first part of the in-field mission is dedicated to self-supervised fine-tuning of a pre-trained convolutional neural network (CNN). Leveraging a real-world vision-based regression task, we thoroughly explore performance-cost trade-offs of the fine-tuning phase along three axes: *i*) dataset size (more data increases the regression performance but requires more memory and longer computation); *ii*) methodologies (e.g., fine-tuning all model parameters vs. only a subset); and *iii*) self-supervision strategy. Our approach demonstrates an improvement in mean absolute error up to 30% compared to the pre-trained baseline, requiring only 22 s fine-tuning on an ultra-low-power GWT GAP9 System-on-Chip. Addressing the domain shift problem via on-device learning aboard nano-drones not only marks a novel result for hardware-limited robots but lays the ground for more general advancements for the entire robotics community.**

SUPPLEMENTARY MATERIAL

Experiment results video: https://youtu.be/blOid4iUFAM

I. INTRODUCTION

We pursue the vision of miniaturized flying robots, i.e., 10 cm in diameter, capable of learning and improving their artificial intelligence-based (AI) perception skills during the mission, exclusively relying on their onboard limited hardware. This ambitious goal arises from the unavoidable problem of *domain shift* [1], [2], which affects any vision-based machine learning (ML) model, such as those used aboard miniaturized unmanned aerial vehicles (UAVs) [3], [4], i.e., nano-UAVs. In robotics, domain shift can result from different sensing systems between the training and deployment domains, as well as from different environments, such as simulation-to-reality transfer [5], [6]. In all cases, the effect is a ML model, trained on data acquired from one domain, underperforms when applied to another.

This work has been partially funded by the Hasler Foundation (# 23059). The authors thank Mirko Nava and Davide Nadalini for their support.

[1]E. Cereda, A. Giusti, and D. Palossi are with the Dalle Molle Institute for Artificial Intelligence (IDSIA), USI-SUPSI, 6962 Lugano, Switzerland name.surname@idsia.ch

[2]M. Rusci is with the Department of Electrical Engineering, KU Leuven, Belgium.

[3]D. Palossi is also with the Integrated Systems Laboratory (IIS), ETH Zürich, 8092 Zürich, Switzerland

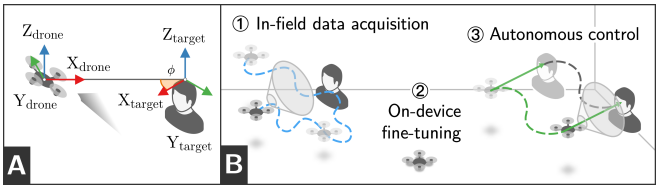

Fig. 1. A) Use case: human pose estimation with a tiny perception CNN aboard a nano-UAV. B) Self-supervised on-device learning introduces a fine-tuning phase during the mission to improve navigation performances.

The most straightforward solution to this fundamental problem would be to acquire new training samples (including ground-truths) from the deployment domain. Unfortunately, this option is often not feasible, i.e., in many real-life scenarios, the deployment domain is not known a priori. Moreover, even in the case of a known deployment domain, collecting and labeling new training data is a labor-intensive and costly activity that can also require expensive ad-hoc infrastructure, e.g., motion capture systems (mocap). This situation is further exacerbated by nano-drones, which can not host sophisticated and power-hungry sensors (e.g., LIDAR [7]), nor can they exploit powerful computational units, being constrained to micro-controller units (MCUs) that provide 100s MOp/s and 100s kB on-chip memory [8], [9].

To mitigate the domain shift problem on ultra-constrained MCU-class processors, on-device learning techniques have been recently proposed [10], [11], [12]. However, research has focused primarily on static sensor nodes [13], addressing mainly classification tasks and assuming readily available ground truths for the on-device training [12] (e.g., by interacting with the user to annotate new classes). Differently, in **our contribution**, we address a vision-based robotic regression task, i.e., human pose estimation from a nano-drone [3], assuming no ground-truths available. Therefore, considering on-device learning aboard nano-drones, this work revolves around the following research questions: *How to deal with i) limited on/off-chip memory, ii) limited compute power, and iii) the lack of precise labels?*

We answer these questions by presenting, for the first time, a thorough analysis of on-device learning for a real-world robotic task aboard nano-drones. In detail, we leverage the PULP-Frontnet [3] convolutional neural network (CNN) for human pose estimation, which predicts the 3D relative pose of a subject from the drone's camera. We investigate the impact of on-device fine-tuning during the mission, depicted in Figure 1, evaluating the trade-offs of different memory budgets to store new in-field data for on-device learning. We

also analyze five alternative training schemes that optimize different subsets of the network's parameters, varying in memory and computational requirements. Finally, we deal with the absence of ground truths through self-supervision from onboard odometry using a *state consistency* loss [14].

On the considered use case, we show up to a 56% MAE improvement against a non-finetuned baseline in the ideal scenario with ground-truth information and 30% with self-supervised fine-tuning. On the best-in-class GAP9 MCU, our approach requires just 22 s when fine-tuning only the batch-norm parameters (29% performance improvement against the baseline). Our promising results bring on-device learning aboard nano-drones as a viable way to address the critical domain shift problem and pave the way for more general advancements for the entire robotics community.

II. Related work

Like in any other deep learning field, deep learning-driven nano-drones suffer from the domain shift problem [1]. In robotics, the result is that AI models that perform well in the training domain often underperform in many real-life deployment environments not known a priori. Previous work focused on addressing this challenge, either by further exploiting limited real-world training data through task-specific data augmentations [15] or more recently with vast simulation-only training datasets [16], optionally combined with additional sensor modalities that are less affected by sim-to-real domain shift (e.g., depth sensors) [17].

On-device learning is an upcoming alternative to mitigate the domain shift problem [18], [19], [20]. However, its embodiment on nano-drones is still far out of reach due to the paramount challenge posed by their limited sensors and computational units aboard. Compared to a traditional kg-scale drone [21], nano-drones can leverage 1000× less memory and computational power [22], [4]. Lamers *et al.* [23] demonstrated self-supervised learning on a 19 g flapping-wing nano-drone, albeit on a very small neural network (single layer, 24 parameters) and with a simplified Widrow-Hoff learning algorithm. By comparison, we focus on a convolutional neural network with 9 layers, 300 k and fine-tune it with the ordinary backpropagation algorithm.

Focusing on CNNs trained with back-propagation algorithms, *TinyEngine* [12] applies gradient rescaling and sparse updates to gain fast 8-bit approximated backward passes. *PULP-TrainLib* [24] accelerates the back-propagation using parallel processing and software optimization using floating-point precision. More specific to robotics, *RLTools* [11] introduces a software package for on-device reinforcement learning. Despite the high portability, this framework specializes in fully connected models and control tasks, which are less complex than CNN-based perception tasks, like ours.

To reduce the workload of the computationally demanding learning steps, TinyTrain [25] introduced a task-adaptive sparse-update method that dynamically selects the layers or channels to update. When benchmarked on a set of image classification tasks, TinyTrain shows up to 5.0% accuracy gain, compared to vanilla fine-tuning and reduces the backward-pass memory and computation by up to 2.3× and 7.7×, respectively. Conversely, Mudrakarta *et al.* [26] propose to limit fine-tuning to just batch normalization layers, reducing the computation compared to full model fine-tuning. TinyTL [27] further limits to just the bias parameters and eliminates the need to store activation maps during the forward pass to compute the gradients of the weight parameters, reducing the memory requirements of fine-tuning by up to 6.5×. Our approach is based on the latter two techniques, which we extend to a real-world robotic regression problem where ground-truth labels are unavailable, which is the case in many robotics use cases. In contrast, previous methods are mainly tested on image classification problems and assume the availability of labeled data for on-device fine-tuning.

Self-supervised learning allows autonomous robots to learn perception models from self-collected data, with supervisory labels derived exclusively from onboard sensors instead of relying on external infrastructure, manual annotation, or user intervention. Approaches can be grouped into three categories. Many derive the desired labels through task-specific methods, for example, exploring an environment with a drone until it crashes [28] or while continuously measuring its distance from the surrounding environment [29].

Other approaches learn a secondary task, for which ground-truth information is available, as a pretext for learning the task of interest, for which ground truths are unknown. For example, predicting sound from a camera image has been used as a proxy for visual localization of a quadrotor [30]. Reconstructing images with a masked autoencoder can be used to learn robot manipulation tasks [31].

Finally, a third family of approaches improves a model's predictions by optimizing consistency with geometric constraints. Ensuring model predictions are consistent with the robot's ego-motion has proven successful in learning object pose estimation [32], while imposing transitive consistency improves visual odometry [33]. Minimizing image reprojection error has also been used to learn visual odometry [34], [35], monocular depth [36], and optical flow [37].

In comparison, our work exploits an ego-motion consistency loss combined with a task-specific approach to derive labels. While other approaches typically assume abundant self-supervised data, in this work, we face an extremely data-scarce scenario due to the limited memory available to store our dataset onboard an embedded system. In addition, while previous approaches assume self-supervised labels that are noisy but fully measurable (e.g., learning object pose estimation in the presence of odometry error, while the object remains still [32]), we are also faced with partially unknown labels, such as moving human subjects.

III. Background

Robotic platform: we employ the Bitcraze Crazyflie 2.1, a commercial off-the-shelf nano-UAV, extended by the pluggable AI-deck and Flow-deck boards. The Crazyflie relies on an STM32 single-core MCU for low-level flight control and can reach up to 7 min flight time on a single 380 mA h battery. The AI-deck extends onboard sensing and processing

TABLE I
GREENWAVES TECHNOLOGIES SOCS COMPARISON.

SoC	Cores	L1	L2	FPUs	CL freq.	Power
GAP8	8 (+1)	64 kB	512 kB	no	175 MHz	96 mW
GAP9	9 (+1)	128 kB	1.5 MB	4	370 MHz	66 mW

capabilities with a GreenWaves Technologies (GWT) GAP8 SoC and a Himax HM01B0 gray-scale QVGA camera, while the Flow-deck provides a time-of-flight altitude sensor and an optical flow sensor to improve the drone's stability. The GAP8 SoC features two power domains, a computationally capable 8-core cluster (CL) and a single-core fabric controller (FC), in charge of data orchestration for the CL's execution. All cores are based on the RISC-V instruction set architecture; the FC can reach up to 250 MHz, while the CL peaks at 175 MHz. The on-chip memories are organized in a fast 64 kB scratchpad L1 and a slower 512 kB L2 memory. Additionally, the AI-deck features 8 MB off-chip DRAM and 64 MB Flash. Finally, the GAP8 does not provide any hardware support for floating-point calculations, requiring either costly soft-float emulation ($10\times$ measured overhead on our workload) or fixed-point arithmetic through quantization.

Our work also investigates the latest GWT GAP9 SoC, which marks strong improvements compared to GAP8, as shown in Table I. GAP9's CL includes four shared floating point units (FPUs), which execute floating-point instructions in a single clock cycle. FPUs are extremely valuable for on-device back-propagation, for which the basic primitives are implemented in the *PULP-TrainLib* software library [24]. On the forward and backward passes for convolutional layers, PULP-TrainLib achieves peak performance efficiencies of, respectively, 5.3 and 4.6 multiply-accumulate operations per clock cycle (MAC/cycle) on GAP9.

PULP-Frontnet: is a field-proven CNN for human pose estimation aboard nano-drones [3], [15], [17]. It takes gray-scale 160×96 px camera frames and estimates the subject's 4DOF pose relative to the drone frame, represented as 3D position (x, y, z) and rotation around the gravity z-axis, ϕ. This CNN is composed of eight convolutional layers based on the architectural template *conv, batch norm, ReLU* and a fully connected one, accounting for 304 k parameters in total. The total computational load is 14.1 MMAC/frame, which leads to a throughput of 48 Hz within only 96 mW on GAP8.

IV. IMPLEMENTATION

A. Self-supervised learning

We approach self-supervised learning using the state consistency loss introduced by Nava *et al.* [32]. We consider a fine-tuning flight sequence on which we minimize the loss:

$$\mathcal{L} = \mathcal{L}_{\text{task}} + \lambda_{\text{sc}}\mathcal{L}_{\text{sc}}, \tag{1}$$

composed of a task loss term $\mathcal{L}_{\text{task}}$ and a state-consistency loss term $\mathcal{L}_{\text{sc}}$. Figure 2 depicts the reference frames used to define the loss terms. We define $\mathbf{T}_A^B$ as the relative pose in

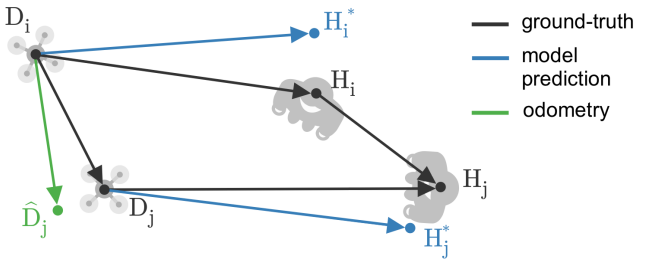

Fig. 2. Reference frames of drone D and subject H at two timesteps i and j and their estimates from model predictions and drone odometry.

SE(3) of reference frame B w.r.t. A and $\mathcal{T}$ as the set of all timesteps in the fine-tuning sequence.

The **task loss** is defined on individual timesteps i from a (possibly empty) subset $\mathcal{T}_{\text{t}} \subseteq \mathcal{T}$ with a known target $\mathbf{T}_{D_i}^{H_i}$:

$$\mathcal{L}_{\text{task}} = \frac{1}{|\mathcal{T}_{\text{t}}|} \sum_{i \in \mathcal{T}_t} \Delta(\mathbf{T}_{D_i}^{H_i^*}, \mathbf{T}_{D_i}^{H_i}) \tag{2}$$

where $\mathbf{T}_{D_i}^{H_i^*}$ represents the model estimation at time i of the relative pose of subject w.r.t. the drone. $\Delta(\mathbf{T_1}, \mathbf{T_2})$ is a distance function between relative poses, which we define as the L1 loss between 4DOF pose vectors (x, y, z, ϕ). L1 loss equally weighs a position error of 1 m and a rotation error of 1 rad, a reasonable heuristic in our setting. Further, L1 loss has been proven robust noise in the labels [38]. Angles are represented as radians, angle differences are computed on the circle group $\mathbb{S}^1$ to account for discontinuities at $\pm\pi$.

$\mathcal{L}_{\text{task}}$ is an ordinary supervised loss when $\mathbf{T}_{D_i}^{H_i}$ are ground-truth relative poses, but approximated relative poses can also be used. In our experiments, the relative pose is known at a time i, and the subject subsequently remains still, from which we define $\mathbf{T}_{\hat{D}_j}^{H_j} = \mathbf{T}_{\hat{D}_j}^{D_i}\mathbf{T}_{D_i}^{H_i}$. The relative pose $\mathbf{T}_{D_i}^{\hat{D}_j}$ indicates the (possibly noisy) odometry estimate of the drone's pose at time j w.r.t time i.

The **state-consistency loss** is defined on pairs of timesteps i and $j = i + dt$ at a fixed time delta (a hyper-parameter) sampled from the subset $\mathcal{T}_{\text{sc}} \subseteq \mathcal{T}$:

$$\mathcal{L}_{\text{sc}} = \frac{1}{|\mathcal{T}_{\text{sc}}|} \sum_{i \in \mathcal{T}_{\text{sc}}} \Delta(\mathbf{T}_{H_i^*}^{D_i}\mathbf{T}_{D_i}^{\hat{D}_j}\mathbf{T}_{D_j}^{H_j^*},\ \mathbf{T}_{H_j}^{H_i}), \tag{3}$$

where $\mathbf{T}_{H_j}^{H_i}$ the subject's relative pose at time j w.r.t time i.

Compared to [32], our loss formulation separates drone and subject movements, as the latter is generally unknown by the drone. In the experiments, they will be replaced by the identity matrix $\mathbf{I}$, i.e., assuming the subject remains still.

B. On-device fine-tuning

For our fine-tuning procedure, we adopt standard backpropagation-based learning: the update steps of the learnable parameters are obtained with a combination of forward (i.e., inference) and backward passes of the input data through the network. As a baseline setting, which we name *all (w+b)*, we update all parameters of every layer (i.e., both weights and biases). To implement this learning

TABLE II

AGGREGATED REGRESSION PERFORMANCE

Train on	Fine-tune on	MAE	R^2	cfr. Fig. 3
real world	nothing	0.50	-9.3	(A) SoA [3]
	environment	0.39	23.5	–
	subject	0.35	41.5	–
sim.	nothing	0.61	-55.0	(B) SoA [17]
	environment	0.38	29.1	(C)
	subject	**0.27**	**57.4**	(D)

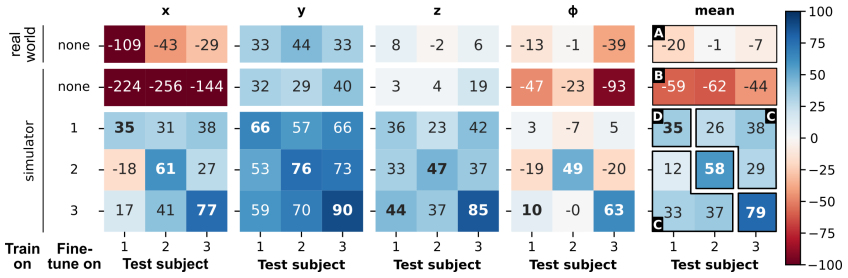

Fig. 3. R^2 scores [%] for all combinations of fine-tuning and test subjects.

scheme, all the activations computed during the forward pass must be preserved in memory to compute the backward gradients. We consider three memory-efficient learning strategies to reduce this memory overhead and decrease the compute requirements of the backward pass. *(a) Only the last fully-connected layer, fc (w+b).* All other parameters are frozen, avoiding backpropagation beyond the last layer. Cost is the lowest among these methods, as well as the expected benefit [27]. *(b) Only the batch-norm parameters, bn (w+b)* The batch normalization layers contain just 0.33% parameters of the entire model and can be updated with 50% of the baseline compute load, while still significantly impacting model performance [26]. *(c) Only the biases, all (b).* While the compute load is equivalent to *bn (w+b)*, memory usage decreases by 99%, because no activation tensor needs to be preserved to compute the weight gradients [27].

C. Experiment setup

Datasets: we use two human pose estimation datasets for the initial training, resulting in two baseline models. The first dataset [3] is acquired in the real world and has 2.6 k training samples coupled with mm-precise mocap-based labels. The second dataset [17] provides 75 k images and labels from the Webots simulator. For fine-tuning and testing, we use the real-world 4.7 k-sample test set from [15] with in-flight sequences from three distinct subjects, with different appearance and movements. Therefore, each dataset belongs to a unique domain, with different environments (real world or simulation), and subjects.

Furthermore, for our fine-tuning dataset, we provide additional noisy labels based on the onboard odometry of the nano-drone. We simulate odometry noise with a Gaussian random walk on x, y, and yaw [39, Sec. 5.2.4], and zero-mean Gaussian noise on z (due to the Crazyflie's altitude sensor). Error parameters are estimated from real-world flights up to 2 m/s. This overly pessimistic model does not account for Crazyflie's optical flow sensor, which reduces drift by tracking features on the ground.

Fine-tuning process: For each subject, the fine-tuning set is a random temporally-contiguous 128 s segment of the dataset (512 samples @ 4 Hz), while the rest is used as the test set. To provide unbiased measures of regression performance, we discard 100 contiguous samples (25 s) between fine-tuning and test segments, and we apply cross-validation, repeating 3 runs for each subject with different random fine-tuning segment (9 total experiment runs). At most, 75% of the samples from each subject are used as the fine-tuning set. As in previous work [3], we use the Adam optimizer with learning rate 10^{-3} to minimize L1 losses ($\mathcal{L}_{\text{task}}$ and/or $\mathcal{L}_{\text{sc}}$) for both initial training and fine-tuning. As [32], we set $\lambda_{\text{sc}} = 1$. Initial training lasts 100 epochs, selecting the model with the best validation performance; fine-tuning lasts 5 epochs, always selecting the final model.

Data augmentation: we apply common photometric data augmentations that can be efficiently implemented on an embedded device: exposure and contrast adjustment, Gaussian noise, box blurring, and vignetting. In addition, we randomly flip the image (and associated labels) horizontally to ensure a symmetric distribution along the y axis and yaw orientation in the fine-tuning set. When training with state-consistency loss, we perform time reversal on the image pairs to ensure that the relative poses $\mathbf{T}_{D_i}^{D_j}$ and $\mathbf{T}_{H_i}^{H_j}$ also follow a symmetric distribution centered on the identity.

V. RESULTS

A. Baseline performance

Baseline models obtain a lower bound on test regression performance with no fine-tuning. An upper bound is obtained by fine-tuning in the best-case scenario: we assume perfect knowledge of the drone and subject poses, we optimize exclusively the task loss (i.e., $\mathcal{T}_{\text{t}} = \mathcal{T}$ and $\mathcal{T}_{\text{sc}} = \varnothing$), and we fine-tune all model parameters. We report the mean absolute error (MAE) and the R^2 score in Table II. The R^2 is a normalized regression metric, where a perfect model achieves a R^2 score of 100%, while a dummy one, always predicting the test set mean, would score 0%. Worse models can score negative, e.g., due to systematic bias.

Figure 3 breaks the R^2 down by test subject, both on the four individual regression outputs and as an average. Figure 3-A shows the performance of the original PULP-Frontnet (real-world training and no fine-tuning), while Figure 3-B is trained on simulated data (no fine-tuning) and marks our lower bound performance. The results for the remaining three fine-tuned models are aggregated in two groups. Figure 3-C tests models on a different subject w.r.t. the fine-tuning set (but the same environment), while Figure 3-D tests on the both same subject and the same environment as the fine-tuning set. Table II shows that, without fine-tuning, training on real-world data is the best approach. However, fine-tuning is always beneficial, in particular when pre-training on simulated data. Therefore, we focus on scenario (*D*) in the following experiments.

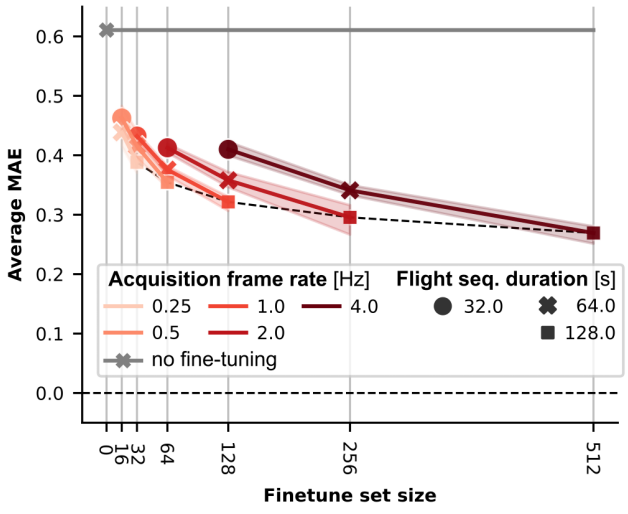

Fig. 4. Fine-tuning set acquisition. Longer flight duration impacts performance more than a higher acquisition frame rate.

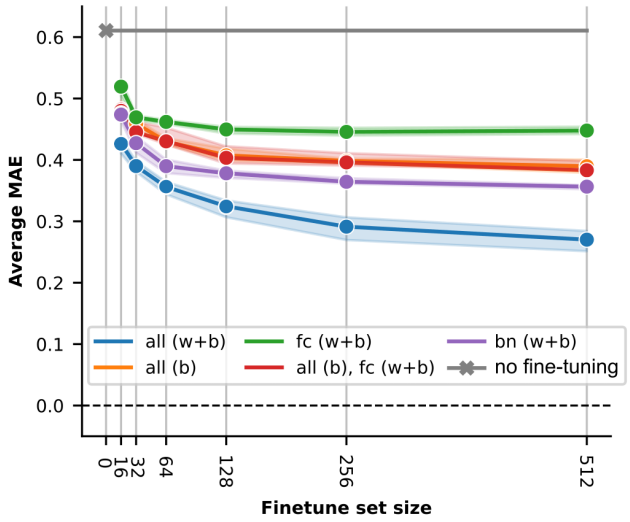

Fig. 5. Comparison of fine-tuning methods. All methods improve w.r.t. the baseline, with consistent behavior across fine-tuning set sizes.

B. *Fine-tuning set acquisition*

In the proposed on-device setup, the acquisition of fine-tuning carries a significant cost along two axes which we explore in Figure 4: the flight time needed to acquire the data and the memory required to store it on-device (i.e., the number of fine-tuning samples). For each line, we sample frames in order from the 128-s fine-tuning sequences at the given acquisition rate until the dataset reaches the desired size (horizontal axis). The markers on each line represent a different flight duration, while the shaded area represents the 95% confidence interval over the 3 cross-validation folds. The gray horizontal line marks the baseline performance before fine-tuning. The plot highlights that, the more data available for fine-tuning, the higher the fine-tuned model's performance. At the same time, for any given fine-tune set size, longer flight times at a lower frame rate are preferable to shorter flight times at a higher frame rate: to reduce the size from 512 to 256 samples, we incur in a +26.7% MAE penalty if we shorten the flight time to 64 s, while only +9.8% if we reduce the acquisition frame rate to 2 Hz. Further, we notice a good trade-off corresponding to 128 samples@1 Hz, below which performance sharply decreases.

C. *On-device fine-tuning*

In Figure 5, we explore the effectiveness of methods that reduce the fine-tuning workload, by limiting the subset of model parameters to update. Full fine-tuning, named *all (w+b)*, sets the lower bound at a MAE of 0.27 (−56 % w.r.t. the non-finetuned baseline, gray). Optimizing only the batch-norm layers, *bn (w+b)*, is second best, followed closely by fine-tuning the biases, *all (b)*. Fine-tuning the final fully connected layer, *fc (w+b)*, performs the worst but notably still shows a MAE improvement of up to −26 %. Compared to *all (w+b)*, other methods also take less advantage of the available fine-tuning samples, peaking at lower set sizes (e.g., 128 samples for *all (b)*).

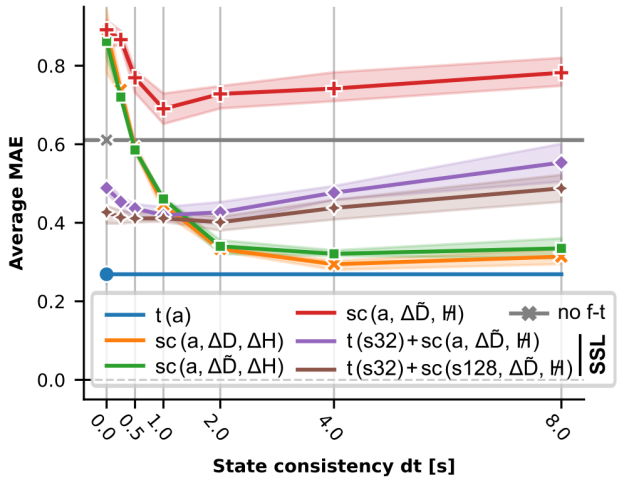

Fig. 6. Self-supervised loss functions. Losses marked "SSL" can be computed entirely from the drone's onboard sensors and achieve up to 50% of the MAE improvement of the ideal loss $t(a)$.

D. *Self-supervised learning*

In Figure 6, we report our fine-tuning results obtained when ground-truth labels are unavailable. We consider three setups for the drone pose – *perfect absolute pose* (D), *perfect odometry* (ΔD), and *uncertain odometry* ($\Delta\widetilde{D}$) – and three for the pose of the human subject – *perfect absolute pose* (H), *perfect odometry* (ΔH), and *unknown* ($\not{H}$).

When perfect absolute poses are known for both the drone and the subject, we have the ground-truth information to fine-tune using $\mathcal{L}_\mathrm{t}$ (i.e., regular supervised learning). The *ideal* case where these are known for all samples is named $\mathrm{t}(a, D, H)$, in short $\mathrm{t}(a)$, and was considered in all previous experiments, where it reaches MAE 0.27.

To reduce our reliance on privileged information, we assume only odometry is known, i.e., relative poses between

TABLE III

COMPARISON OF FINE-TUNING METHODS BY PERFORMANCE AND WORKLOAD. MAE WITH IDEAL AND SELF-SUPERVISED LOSSES (SSL).

Method	**Fine-tune set size**	**Params** [k]	**Activations** [kB/frame]	**Train step** [MMAC/frame]	**Fine-tuning time**, 5 epochs [s]		**MAE**	
					GAP9 @ 370 MHz	**GAP8** @ 175 MHz	*Ideal*	*SSL*
No fine-tuning		–	–	–	–	–	0.61	
all	512	304.4	217.5	53.1	2:03	86:51	**0.27**	0.43
(w+b)	128	(100%)	(100%)		0:31	21:43	0.32	**0.41**
bn	512	1.0	146.2	38.8	1:29	62:46	0.36	0.43
(w+b)	128	(0.33%)	(67%)		0:22	15:42	0.38	0.42
fc	512	7.7	1.9	**14.3**	0:32	22:40	0.45	0.47
(w+b)	128	(2.5%)	(0.09%)		**0:08**	**5:40**	0.45	0.47
all	512	**0.5**	**0.8**	38.7	1:29	62:31	0.39	0.46
(b only)	128	(0.15%)	(0.05%)		0:22	15:38	0.41	0.45

two instants in time. When odometry is perfect for both drone and subject on all samples, $\mathrm{sc}(a, \Delta D, \Delta H)$, we can fine-tune using the state-consistency loss $\mathcal{L}_{\mathrm{sc}}$ and achieve a MAE 0.29 (93% of the ideal improvement w.r.t. the baseline). Uncertain odometry, $\mathrm{sc}(a, \Delta\widetilde{D}, \Delta H)$, also has limited impact on performance and achieves 85% of ideal. Higher state-consistency time deltas dt (horizontal axis) are beneficial, as samples farther in time carry more information. On the other hand, performance is drastically reduced with unknown subject poses, $\mathrm{sc}(a, \Delta\widetilde{D}, \mathcal{H})$. In this case, we compute $\mathcal{L}_{\mathrm{sc}}$ assuming that $\mathbf{T}_{H_i}^{H_j} = \mathbf{I}$, i.e., the subject is always still. The time reversal augmentation ensures this holds on average, i.e., the fine-tuning set has $\mathbb{E}[\mathbf{T}_{H_i}^{H_j}] = \mathbf{I}$ by design, but the model still degenerates to a dummy constant predictor. Perfect drone odometry $\mathrm{sc}(a, \Delta D, \mathcal{H})$ performs the same.

In a different experiment, we envision a cooperative scenario where subjects stand still in a known position, and the drone moves around to acquire several images. The procedure is repeated at different locations in the environment to acquire highly diverse fine-tuning data. We test the scenario by identifying the subset of frames in which the subject stands still (i.e., speed $\leq 0.1\,\mathrm{m/s}$ for $\geq 1\,\mathrm{s}$) and selecting 32 random samples on which we optimize $\mathcal{L}_t$. This scenario, $\mathrm{t}(s32) + \mathrm{sc}(a, \Delta\widetilde{D}, \mathcal{H})$, relies on realistic in-field infrastructure-free data acquisition, achieving a significant improvement, up to 39% of the ideal case.

In our last experiment, we only consider a subject-still subset for the state-consistency loss term, $\mathrm{t}(s32) + \mathrm{sc}(s128, \Delta\widetilde{D}, \mathcal{H})$. As odometry drift does not impact $\mathcal{L}_{\mathrm{sc}}$, we are not limited in how much time we can exploit state consistency. We thus choose a larger set of 128 samples (limited by the number of subject-still samples available for all three subjects) and reach an improvement of 50% of the ideal case. The latter two scenarios are taken as the *self-supervised* loss function in the next experiment (for 512 and 128 samples), with the best-performing $dt = 2\,\mathrm{s}$.

E. Discussion

We summarize our regression performance findings in Table III and complement them by analyzing the workload when deployed aboard the nano-drone. We estimate the number of MACs required to perform a training step on one input frame (forward + backward) and the runtime of the whole fine-tuning process when implemented with *PULP-TrainLib* on GAP9 and GAP8 (soft-float).

Full fine-tuning, *all (w+b)*, with ideal labels reaches the best MAE of 0.27 but is also the most expensive method at 53 MMAC per frame. In addition, it requires storing all activations for the backward pass, which, for 32-sample batches, amounts to 57% of the AI-deck's 8 MB DRAM. This takes up valuable memory that could store a larger fine-tuning dataset. Fine-tuning the batch-norms, *bn (w+b)*, has the next best performance (MAE 0.36) and reduces workload (-25%), but still stores 146 kB/frame. In contrast, *all (b)* strikes the best trade-off with comparable performance (MAE 0.39) and workload while storing only 800 B/frame.

All four methods achieve a similar score when fine-tuning with the self-supervised loss. However, all four achieve a solid improvement upon the baseline, even with just 128 fine-tuning images. This reduces fine-tune times by $4\times$ compared to 512 images, making even full fine-tuning manageable on GAP9 (0:31, viable for real use cases). With *fc (w+b)*, improvements are achievable in a reasonable time frame (5:40) even on current-generation GAP8 SoC with soft-float emulation. In the envisioned use case, the drone saves energy by landing and computing on the ground. Otherwise, fine-tuning would take up most of the drone's 7 min flight time.

VI. CONCLUSION

We present on-device learning aboard nano-drones as a solution for the domain shift problem in visual perception tasks, with a self-supervised fine-tuning phase at the beginning of the nano-drone's mission. In our use case, we show up to a 56% MAE improvement against a non-finetuned baseline, in the ideal scenario with ground-truth information, and 30% with self-supervised fine-tuning. On the best-in-class GAP9 MCU, our approach requires just 22 s when fine-tuning only the batch-norm parameters. Our work demonstrates that on-device self-supervised learning is a viable option also for highly hardware-constrained robots, pushing forward the nano-robotics state of the art.

REFERENCES

[1] J. Wang, C. Lan, C. Liu, Y. Ouyang, T. Qin, W. Lu, Y. Chen, W. Zeng, and P. S. Yu, "Generalizing to unseen domains: A survey on domain generalization," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 8, pp. 8052–8072, 2023.

[2] C. Heinze-Deml and N. Meinshausen, "Conditional variance penalties and domain shift robustness," *Machine Learning*, vol. 110, no. 2, pp. 303–348, 2021.

[3] D. Palossi, N. Zimmerman, A. Burrello, F. Conti, H. Müller, L. M. Gambardella, L. Benini, A. Giusti, and J. Guzzi, "Fully onboard AI-powered human-drone pose estimation on ultralow-power autonomous flying nano-UAVs," *IEEE Internet of Things Journal*, vol. 9, no. 3, pp. 1913–1929, 2021.

[4] L. Lamberti, V. Niculescu, M. Barciś, L. Bellone, E. Natalizio, L. Benini, and D. Palossi, "Tiny-PULP-Dronets: Squeezing neural networks for faster and lighter inference on multi-tasking autonomous nano-drones," in *2022 IEEE 4th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, 2022, pp. 287–290.

[5] K. Kelchtermans and T. Tuytelaars, "RARA: Zero-shot Sim2Real visual navigation with following foreground cues," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2022, pp. 1704–1710.

[6] L. Lamberti, E. Cereda, G. Abbate, L. Bellone, V. J. K. Morinigo, M. Barciś, A. Barciś, A. Giusti, F. Conti, and D. Palossi, "A sim-to-real deep learning-based framework for autonomous nano-drone racing," pp. 1899–1906, 2024.

[7] S. Jung, S. Cho, D. Lee, H. Lee, and D. H. Shim, "A direct visual servoing-based framework for the 2016 IROS Autonomous Drone Racing Challenge," *Journal of Field Robotics*, vol. 35, no. 1, 2018.

[8] K. Mcguire, C. De Wagter, K. Tuyls, H. Kappen, and G. Croon, "Minimal Navigation Solution for a Swarm of Tiny Flying Robots to Explore an Unknown Environment," *Science Robotics*, vol. 4, p. eaaw9710, Oct. 2019.

[9] S. Chen *et al.*, "Towards specialized hardware for learning-based visual odometry on the edge," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2022, pp. 10 603–10 610.

[10] T. Lesort, V. Lomonaco, A. Stoian, D. Maltoni, D. Filliat, and N. Díaz-Rodríguez, "Continual learning for robotics: Definition, framework, learning strategies, opportunities and challenges," *Information Fusion*, vol. 58, pp. 52–68, 2020.

[11] J. Eschmann, D. Albani, and G. Loianno, "RLtools: A fast, portable deep reinforcement learning library for continuous control," 2023.

[12] J. Lin, L. Zhu, W.-M. Chen, W.-C. Wang, C. Gan, and S. Han, "On-device training under 256kB memory," *Advances in Neural Information Processing Systems*, vol. 35, pp. 22 941–22 954, 2022.

[13] C. Profentzas, M. Almgren, and O. Landsiedel, "MiniLearn: On-device learning for low-power IoT devices," in *International Conference on Embedded Wireless Systems and Networks*, 2022.

[14] M. Nava, L. M. Gambardella, and A. Giusti, "State-consistency loss for learning spatial perception tasks from partial labels," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 1112–1119, 2021.

[15] E. Cereda, D. Palossi, and A. Giusti, "Handling pitch variations for visual perception in MAVs: Synthetic augmentation and state fusion," in *13^{th} International Micro Air Vehicle Conference (IMAV)*, G. de Croon and C. D. Wagter, Eds., Sep 2022, pp. 59–65.

[16] A. Moldagalieva and W. Hönig, "Virtual omnidirectional perception for downwash prediction within a team of nano multirotors flying in close proximity," 2023.

[17] L. Crupi, E. Cereda, A. Giusti, and D. Palossi, "Sim-to-real vision-depth fusion CNNs for robust pose estimation aboard autonomous nano-quadcopters," in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2023, pp. 7711–7717.

[18] M. Pavan, E. Ostrovan, A. Caltabiano, and M. Roveri, "TyBox: an automatic design and code-generation toolbox for TinyML incremental on-device learning," *ACM Transactions on Embedded Computing Systems*, 2023.

[19] H. Ren, D. Anicic, and T. A. Runkler, "TinyOL: TinyML with online-learning on microcontrollers," in *2021 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2021, pp. 1–8.

[20] D. P. Pau, P. K. Ambrose, and F. M. Aymone, "A quantitative review of automated neural search and on-device learning for tiny devices," *Chips*, vol. 2, no. 2, p. 130–141, May 2023. [Online]. Available: http://dx.doi.org/10.3390/chips2020008

[21] C. D. Wagter, F. Paredes-Vallé, N. Sheth, and G. de Croon, "The sensing, state-estimation, and control behind the winning entry to the 2019 Artificial Intelligence Robotic Racing Competition," *Field Robotics*, vol. 2, no. 1, pp. 1263–1290, mar 2022.

[22] R. J. Bouwmeester, F. Paredes-Vallés, and G. C. De Croon, "NanoFlowNet: Real-time dense optical flow on a nano quadcopter," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 1996–2003.

[23] K. Lamers, S. Tijmons, C. De Wagter, and G. de Croon, "Self-supervised monocular distance learning on a lightweight micro air vehicle," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016, pp. 1779–1784.

[24] D. Nadalini, M. Rusci, L. Benini, and F. Conti, "Reduced precision floating-point optimization for deep neural network on-device learning on microcontrollers," *Future Generation Computer Systems*, vol. 149, pp. 212–226, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X23002728

[25] Y. D. Kwon, R. Li, S. I. Venieris, J. Chauhan, N. D. Lane, and C. Mascolo, "TinyTrain: Deep neural network training at the extreme edge," *arXiv preprint arXiv:2307.09988*, 2023.

[26] P. K. Mudrakarta, M. Sandler, A. Zhmoginov, and A. Howard, "K for the price of 1: Parameter efficient multi-task and transfer learning," in *International Conference on Learning Representations*, 2019.

[27] H. Cai, C. Gan, L. Zhu, and S. Han, "TinyTL: Reduce memory, not parameters for efficient on-device learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 11 285–11 297, 2020.

[28] D. Gandhi, L. Pinto, and A. Gupta, "Learning to fly by crashing," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE Press, 2017, p. 3948–3955. [Online]. Available: https://doi.org/10.1109/IROS.2017.8206247

[29] A. Kouris and C.-S. Bouganis, "Learning to fly by myself: A self-supervised cnn-based approach for autonomous navigation," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE Press, 2018, p. 1–9. [Online]. Available: https://doi.org/10.1109/IROS.2018.8594204

[30] M. Nava, A. Paolillo, J. Guzzi, L. M. Gambardella, and A. Giusti, "Learning visual localization of a quadrotor using its noise as self-supervision," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 2218–2225, 2022.

[31] I. Radosavovic, T. Xiao, S. James, P. Abbeel, J. Malik, and T. Darrell, "Real-world robot learning with masked visual pre-training," *CoRL*, 2022.

[32] M. Nava, A. Paolillo, J. Guzzi, L. M. Gambardella, and A. Giusti, "Uncertainty-aware self-supervised learning of spatial perception tasks," *IEEE Robotics and Automation Letters*, vol. 6, no. 4, pp. 6693–6700, 2021.

[33] G. Iyer, J. Krishna Murthy, G. Gupta, M. Krishna, and L. Paull, "Geometric consistency for self-supervised end-to-end visual odometry," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2018.

[34] J. Bian, Z. Li, N. Wang, H. Zhan, C. Shen, M.-M. Cheng, and I. Reid, "Unsupervised scale-consistent depth and ego-motion learning from monocular video," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32. Curran Associates, Inc., 2019.

[35] T. Zhou, M. Brown, N. Snavely, and D. G. Lowe, "Unsupervised learning of depth and ego-motion from video," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1851–1858.

[36] C. Godard, O. Mac Aodha, M. Firman, and G. J. Brostow, "Digging into self-supervised monocular depth estimation," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

[37] P. Liu, M. Lyu, I. King, and J. Xu, "SelFlow: Self-supervised learning of optical flow," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[38] A. Ghosh, H. Kumar, and P. S. Sastry, "Robust loss functions under label noise for deep neural networks," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 31, no. 1, 2017.

[39] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, *Introduction to autonomous mobile robots*. MIT Press, 2011.