PrivyNAS: Privacy-aware Neural Architecture Search for Split Computing in Edge-Cloud Systems

Mohanad Odema, *Student Member, IEEE* Mohammad Abdullah Al Faruque, *Senior Member, IEEE*

***Abstract*—Split Computing has become a prominent resource-efficient method to enable machine learning (ML) applications on user-constrained edge devices, where compute-intensive ML workloads can be delegated to remote cloud servers for processing. However, the exposure of data to the cloud service providers as such raises privacy alarms due to the possible leakage of sensitive user information. On a relevant note, typical deep neural network (DNN) design frameworks do not take into account model splitting and its complications at the early design stages of DNNs. Thus, a natural question arises on how to bridge this gap and optimize the DNN design process such that split computing operations can meet the requirements of accuracy, performance, and privacy. In this paper, we strive to address this question through adopting a *privacy-by-design* approach, where privacy is characterized either as a *constraint* or an *objective* to realize privacy-aware models tailored for split computing. Using the ϵ-differential privacy standard for our case study, we conduct intensive empirical analysis on the relation between architectural parameters and intrinsic privacy budgets, and propose *PrivyNAS* – a privacy-aware Neural Architecture Search framework for split computing. On the CIFAR-10 dataset, our approach has demonstrated promising results in providing DNN architectures that balance the required design trade-offs.**

***Index Terms*—Inference Privacy, Differential Privacy, Split Computing, Neural Architecture Search, Edge-Cloud.**

I. Introduction

With the advent of the Internet of Things (IoT) and Big Data era, Deep Neural Networks (DNNs) have become recognized standard solutions for numerous IoT applications, such as image recognition, Natural Language Processing, and healthcare monitoring, given their impressive capacity in handling large volumes of data and achieving remarkable performances [1]–[3]. In order to enable the deployment of compute intensive DNN models onto resource-constrained user end devices, recent research works have proposed to adopt a split computing approach in which DNN computations are divided between the user's edge device and a cloud server [4]–[7]. In this scheme, inference can be offered by a cloud provider as an online, on-demand remote inference service. Hence, the bulk (if not all) of the computational workloads from the user's model can be relayed for processing on the provider's cloud, enabling on-device execution of DNNs from the constrained user devices.

Nonetheless, the transmission of user's data to the cloud poses privacy concerns as users have no control over how the data is used once it has been made available to the provider. Previous incidents have already seen providers sharing users' personal information with other third parties (e.g., Facebook incident in 2018 [8]). Another study has demonstrated that aggregate user data at the cloud which are reused for model training are liable to model inversion attacks, mainly due to the model weights leaking users' sensitive attributes (as was demonstrated in a model inversion attack recovering images from a facial recognition system [9]). Even more so in the mobile health field, the transmission of time series physiological signals (e.g., ECG) for cloud processing can reveal user-private behavioral patterns (e.g., e.g., smoking, stress) sensitive to the user.

Addressing this, researchers have proposed numerous techniques to provide privacy guarantees under such remote inference model [10]–[13]. Despite their effectiveness, these methods were applied to models originally targeted for *single* platform deployment, that is, models whose design was not optimized for edge-cloud system operation. As works in [14], [15] have observed that varying a subset of architectural parameters can impact privacy guarantees, the inherent privacy-preserving capabilities of a model can be affected by the choice of architectural parameters. Hence, an argument can be made that architectural parameter choices can be optimized to enhance a DNN model's privacy preserving capabilities, giving rise to the following questions from a DNN model designer's perspective:

- How to assess candidate model architectural designs with regards to upholding inference privacy guarantees given a remote inference operational scheme?
- How to model the relationship between architectural design choices and the inherent privacy-preserving capabilities given a split computing model of computation?
- How to implement a design framework for non-monolithic DNN models balancing the underlying accuracy-performance-privacy trade-offs?

Figure 1 shows how application-level accuracy and hardware performance (e.g., execution latency) have been the standard objectives guiding the DNN design process to attain models that achieve the most balanced trade-offs. In this work, we aim to study the value of incorporating privacy as a design metric given a remote inference deployment scheme. To achieve this, a formal metric needs to be utilized for to quantify privacy as a design objective such as the rigorous Differential Privacy (DP) standard [16], [17] with its quantifiable privacy loss budget ϵ. In DP, noise is added to user's data for obfuscation and minimizing its

The authors are with the Department of Electrical Engineering & Computer Science, University of California, Irvine, CA, 92697 USA (email: modema@uci.edu; alfaruqu@uci.edu) (Corresponding author: Mohanad Odema)

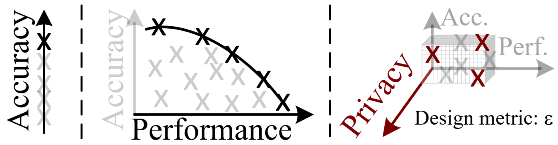

Fig. 1. The evolution of DNN design objectives. The goal from multi-objective optimization is to identify a Pareto-optimal set of model designs that balance design trade-offs. Here, we also consider privacy as a design objective.

likelihood of revealing sensitive information before transmission to the cloud. In other words, a differentially private algorithm with the right amount of noise perturbation operates on aggregate representations of users' data items to avoid leakage of their sensitive information within. In order to prevent noise injection from destroying the a model's utility, additive noise is sourced from Gaussian or Laplacian distributions that are calibrated to the data sensitivity (i.e., the max distance between any two data inputs). These well-defined distributions represent a standardized source of noise for differential privacy use-cases [10], [14], [18].

In this work, we study how DP for remote inference and potential split-computing can be considered from the early DNN architectural design stage. To asses our approach, we conduct extensive experiments on well-known models to empirically study the effect of varying architectural parameters and splitting point position on the privacy budget ϵ. Next, we implement a novel customized Neural Architecture Search (NAS) framework, namely PrivyNAS, through which we are able to specify privacy as either an objective or a constraint during the DNN design stage. Accordingly, the processes of designing, partitioning, and selecting additive noise levels for split DNN models are all automated within PrivyNAS framework. In summary, the paper's key contributions are as follows:

- We propose a methodology to design DNNs for remote inference applications in a privacy-aware fashion using the Differential Privacy (DP) standard.
- We conduct extensive empirical evaluations on the well-know VGG [19] and MobileNetv2 [20] DNN models to analyze how inference privacy budget ϵ can vary according to the choices of DNN architectural parameters.
- We develop a customized privacy-aware NAS framework for remote inference to search for optimal architectural designs with respect to accuracy, performance, and privacy. The privacy budget ϵ can be specified as either a design constraint or a minimization objective.
- Our experiments on the CIFAR-10 dataset have shown promising results in the sense that our customized NAS framework has provided a model that achieved 79.9% accuracy compared to 79.16% from the baseline ProxylessNAS framework [21]. Moreover, our customized framework was capable of realizing model designs that can balance the accuracy-performance-privacy trade-offs more effectively than conventional means.

II. Preliminaries

A. Differential Privacy (DP)

Differential Privacy [16], [17] has been established as a rigorous standard for providing quantifiable privacy guarantees on users' sensitive data. A formal definition for ϵ-DP is given:

Definition 1: *A randomized mechanism $\mathcal{A}$ is ϵ-differential private, iff for any adjacent inputs d and d', and any output $\mathcal{S}$ of $\mathcal{A}$,*

$$Pr[\mathcal{A}(d) = S] \leq e^{\epsilon} Pr[\mathcal{A}(d') = S]$$

where ϵ is a measurable privacy budget, whereas d and d' represent adjacent inputs differing by a single data item. d and d' are defined according to the application, where they can range from entire datasets differing by a single entry [14], or acquired signals instances differing in content by a single item (e.g., two sentences differing by at most i number of words) [10]. Generally, smaller ϵ values indicate stronger privacy guarantees.

For a deterministic function f, obtaining its ϵ-DP compliant randomized mechanism $\mathcal{A}_f$ entails the addition of noise calibrated to the global sensitivity of f. Global sensitivity, Δf, represents the maximum absolute distance $|f(d) - f(d')|$ for any adjacent input pairs d and d', and the additive noise can be incorporated as follows:

$$\mathcal{A}_f(d) = f(d) + u \tag{1}$$

where u represents the noise tensor which can be sampled from a Laplacian distribution $Lap(0, \frac{\Delta f}{\epsilon})$ of mean 0 and scale $\frac{\Delta f}{\epsilon}$ to attain a privacy budget of ϵ [17].

Added to its quantifiability, DP is characterized by the two essential properties of *postprocessing immunity* and *composition* [17], [22]. The first ensures that after the data has been processed through an ϵ-DP randomized mechanism to generate a specific output, further handling or processing of this output by other algorithms will *not* degrade the original privacy guarantee, i.e., the output would still remain ϵ-DP. Whereas the latter composition property characterizes aggregation of privacy losses when similar or neighboring data are processed by two DP algorithms [22], [23]. For instance, when k processing calls are made to an ϵ-DP algorithm, an upper-bound on the privacy drop (i.e., increase in the value of ϵ) equal to $k\epsilon$ cab expected as a result of this composition.

If noise perturbation is applied at the user's device, this resembles an instance of *local differential privacy* [23]. Owing to its inherent property of *immunity to post-processing*, local DP can ensure data privacy for remote inference models, where $\mathcal{A}_f$ represents the local model deployed on the user device, and the privacy guarantee is associated with making each data sample indistinguishable at the provider's side [10].

B. Neural Architecture Search

The typical approach to design DNN model architectures required considerable skill and expertise to effectively tune the various architectural design knobs. Alleviating this burden, neural architecture search (NAS) has emerged as a viable technique to automate the design of DNN architectures that are on-par or outperform their manually-crafted counterparts [21], [24]–[26]. In a nutshell, the purpose of NAS is to effectively navigate an enormous design space of neural architectural parameters to identify optimal candidate model architecture designs suited for the target objectives. Primarily, there are three foundational pillars to any NAS framework:

Search Space. Through the coalescence of various combinations of architectural design parameters, a pool of candidate designs can be constructed for the NAS engine to access. Thus, DNN architectural parameter choices such as the #layers, # channels, type of layer operation, etc can all be encoded into a single unified search string.

Search Controller. Due to the colossal size of typical DNN architectural search spaces, NAS frameworks employ search controllers adopting sophisticate search strategies to effectively balance the exploration/exploitation of the search space. In particular, search controllers learn to identify promising design subspaces from which they can sample superior population of architectural candidates, reducing the likelihood of considering sub-optimal designs in the interim. These controllers are typically adopt strategies that follow a learning-based approach (e.g., reinforcement learning based [21], [24]) or a metaheuristic approach (e.g., evolutionary algorithm [26]).

Performance Evaluation. In order to guide the search controller on promising architectural design sub-spaces, performance evaluations of the candidate model design on the target objectives (e.g., accuracy and execution latency) are fed back to the search controller for it to exploit optimizations around the top-performing candidates in a progressive manner.

Traditionally, classical NAS frameworks [6], [24] relied on *training* candidate models from scratch to determine the accuracy scores needed for comparison. However, this approach was deemed inefficient due to the substantial added timing overheads from training each candidate only to have all trained weights thrown away after evaluation. Addressing this, recent NAS approaches [21], [25], [26] proposed a *one-shot* approach in which all candidate models are trained simultaneously through the concepts of *supernet* and *shared weights*. Briefly, the idea relies on specifying the search space as a single, multi-path overparameterized network model (i.e., a supernet) that encapsulates all candidate architectural designs within (i.e., subnets). In this case, the sampling of a candidate model from the supernet is achieved through selecting for each potential layer position a particular path reflecting a specific architecture choice, as in choosing a path representing a 3×3 convolution. Then, the weights associated with the 3×3 convolution are loaded within the supernet, used for the candidate model evaluation, and are updated in the next udpate step. Our approach builds on this mature form of NAS frameworks detailed in Section V.

C. Related Work

Differential Privacy has been applied for privacy preservation across 3 fields: data aggregation, model training, and remote inference [18]. In data aggregation, the focus is on maintaining privacy of user responses when collecting data for computing statistics or dataset construction. Thus, techniques like randomized responses provide formal guarantees on users' replies in standardized systems, as Google's RAPPOR [27]. In model training, ϵ privacy guarantees are provided for each user data item contributing to a training dataset. [23], [28]. Consequently, the impact a single data item can have on a model's parameters is limited, preventing implicit storage of sensitive information which can be compromised if attacks were conducted on the model [14], [29], [30]. The last category is for remote inference privacy [10], which is discussed in detail throughout this paper.

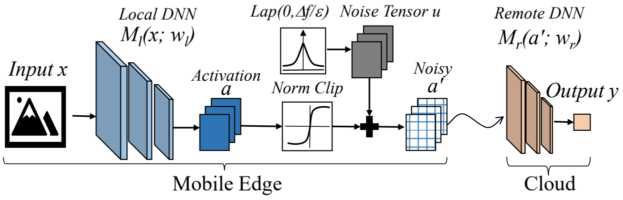

Fig. 2. Inference Privacy Model of Computation for DNNs using ϵ-DP.

Inference Privacy can be achieved through techniques such as homomorphic encryption [13], [31], [32] and Secure Multiparty Computation (SMC) [33], [34], allowing computation using encrypted data. However, encryption can be resource demanding making it prohibitive to execute on constrained edge devices. Alternatively, context-aware information theoretic approaches were proposed to reduce the information content of any data features that do not affect the accuracy of the primary inference task [11], [12], [35]–[37]. For a DP approach, it provides a formal measure on the degree of indistinguishability within the "mechanism", which is the inference model itself in this case. Therefore, by bringing ϵ-DP into the models' design process, we can obtain baseline privacy guarantees which can be further tightened through context-aware approaches if needed.

III. ϵ-DP Remote Inference Model

Figure 2 illustrates the basic inference privacy model of computation for deep neural network (DNNs) achieved through ϵ-DP. As shown, additional computing blocks are added on the user's mobile-edge device following the local model, $\mathcal{M}_l$, to incorporate ϵ guarantees prior to offloading. These computing blocks are described in further detail as follows:

Norm Clipping: Obtaining a precise estimate for global sensitivity, Δf, within DNN models can be a complicated process [14]. Alternatively, Δf is approximated by restricting the effect of each input to an absolute maximum threshold. As the privacy guarantee ϵ is to be associated with the local model's output $\mathbf{a}$, each $\mathbf{a}$ first needs to be bounded according to the Δf estimate at the data offloading point. Thus, each output $\mathbf{a}$ is scaled down to become $\mathbf{a} \leftarrow \mathbf{a}/max(1, \frac{\|a\|_\infty}{B})$, where $\|a\|_\infty$ is the infinity norm of $\mathbf{a}$ and B is a clipping threshold leading Δf to become $2B$. The Bound B can be approximated based on the median of infinity norms belonging to output activations of public training data samples [10], [14].

Additive Noise: Next, perturbation is applied to the scaled down activation $\mathbf{a}$ proportional to the desired privacy budget ϵ (see equation 1). In particular, a noise tensor n of the same dimensions as $\mathbf{a}$ is populated with random samples from the distribution $Lap(0, \frac{\Delta f}{\epsilon})$, and added to $\mathbf{a}$ to generate the noisy representation $\mathbf{a}'$, which can then be transmitted to the cloud. In this scheme, the cloud-side DNN can be viewed as a *post-processing* stage for the local-side DNN given how local DP is applied at the client's part of the model. Accordingly, the DP *composition* property becomes primarily associated with the

TABLE I
THE VARIATION OF ϵ WITH ARCHITECTURE AND NOISE INJECTION LAYER.

Layer		DNN Architecture			
		VGG11	VGG13	VGG16	VGG19
MP_C	Bound	0.8	0.775	0.825	0.625
	Acc.	21.0	25.7	32.7	47.4
	ϵ@b=0.5	**3.2**	**3.1**	**3.3**	**2.5**
Conv_E1	Bound	0.175	0.15	0.15	0.275
	Acc.	19.7	19.9	20.6	12
	ϵ@b=0.5	0.7	0.6	**0.6**	**1.1**

local DNN side of the model, and from the cloud perspective, its outputs can typically be only traced back to the noisy intermediate representation a'.

DNN noisy retraining: As noise addition can lead the model's utility to deteriorate, retraining on publicly-available noisy data representations is beneficial. Typically, the cloud-side model $\mathcal{M}_r$ is retrained on perturbed representations of public data instances to enhance the model's resilience when dealing with noisy representations $\mathbf{a}'$. Meanwhile, local models $\mathcal{M}_l$ remain unaltered [10]. The retraining loss function for $\mathcal{M}_r$ can be as:

$$\mathcal{L}_{total}(w_r; a, a') = \lambda \mathcal{L}_{clean}(w_r; a) + (1-\lambda)\mathcal{L}_{noisy}(w_r; a') \quad (2)$$

where w_r represent M_r's weight parameters while λ trades off the contribution of the clean and noisy representations, taking values in the range of [0, 1].

It should be noted that there are additional techniques to further strengthen formal ϵ privacy guarantee, as the data nullification technique in [10]. However, our analysis in the following sections is conducted using the basic form of ϵ-DP remote inference model to analyze in the vanilla form the relation between a model's architectural build and privacy budget.

IV. Analysis of the Relation Between DNN Architectural Parameters and ϵ-DP

Through intensive empirical evaluations, we examine how the ϵ-DP guarantees for remote inference can vary according to the underlying DNN structure. The work in [15] similarly investigated such a relation to identify an optimal topology for $\mathcal{M}_l$ that enables cloud-based DNN training while protecting data privacy. Nonetheless, their work neither targets inference privacy nor is based on the DP standard. Throughout this paper, our analysis is established based on the benchmark CIFAR-10 image dataset for its role as a main evaluation dataset in numerous relevant DP works [10], [14], [15], [23], [29].

Proposition 1: ϵ **variability per layer.** ***Within a model****, the strength of ϵ-DP guarantee varies depending on the chosen noise injection layer.* ***Across model variants****, the noise injection layer capacity to influence ϵ varies based on its relative position within the computational graph.*

In this experiment, we analyze how the sensitivity bounds and the ϵ privacy guarantee vary when the position of noise injection layer changes *within* and *across* different model architectures. This first analysis is performed using pretrained models directly with no cloud-side retraining after noise injection (we leave that analysis for the immediate subsequent experiments). Briefly, the choice of offloading (noise injection) layer influences the amount of perturbation needed to achieve ϵ-DP guarantee, and in turn affects the DNN model's utility. As a motivational study, We analyze how ϵ would vary across 4 pre-trained variants of the VGG family of DNNs [19] under two potential injection layers: MP_C, which is the 3rd *Max Pooling* layer, and Conv_E1, the first *Conv* layer in the 5th block of the VGG architecture. We assume additive noise tensors are sampled from a Laplacian distribution with a scale of $b = 0.5$ (recall $b = \frac{\Delta f}{\epsilon}$ in (1)). As shown in Table I, we observe that not only do clipping bounds B differ based on the choice of injection layer but also across the distinct variants. This implies that ϵ budget computed would be different under the same b for different layers. For instance, the privacy guarantee in VGG16 at MP_C is $\epsilon = 3.3$ opposed to $\epsilon = 0.6$ at Conv_E1. Also for the same layer across different VGG variants, a stricter ϵ of 2.5 at MP_C can be attained for VGG19 compared to ϵ values from the other VGG variants. More interestingly, despite providing $\epsilon = 2.5$ budget at MP_C for VGG19, the model's utility does not degrade as much as that for the other variant models with looser ϵ budgets at MP_C. Contrarily, VGG19 model utility with $\epsilon = 1.1$ deteriorates in a worse fashion compared to other VGG models with tighter ϵ budgets. These observations show that based on how a model is structured (VGG variant) and partitioned (injection layer position), knowledge is maintained discordantly and so is its intrinsic resilience to noisy representations.

Key Takeaway. *The noise injection layer position is to be optimized alongside the design process of DNN model architectures supporting ϵ-DP for inference privacy.*

Proposition 2: ***Depth.*** *For the same required ϵ budget, the likelihood of a model sustaining severe drop in accuracy decreases as the noise injection layer position tends towards the model's deeper layers.*

Depth: In this analysis, we assume a tight privacy budget requirement of $\epsilon = 2.8$ based on results from [10], [29]. We use two DNNs, VGG11 and VGG16 [19], trained to $\sim 94\%$ test accuracy on CIFAR-10 using the training hyperparameters in [38]. These two architectures resemble architectural depth variation since VGG16 possess one more *Conv* layer per each block than VGG11, but they share other architectural configuration parameters. Then for each layer, we evaluate how the overall DNN utility would degrade when the layer applies noise injection. We also retrain the cloud-side DNN on noisy perturbations in each case, and re-evaluate the overall utility. The two upper bar plots in Figure 3 demonstrate how the accuracy is impacted for every potential injection layer for both DNNs. At a first glance, we can observe that the deeper injection layers generally offer better overall accuracy under a specific ϵ. This is comprehensible given how they already deal with more abstract representations of data. We also notice that the required perturbation level b (shown in red) to achieve *2.8-DP* differs for each layer depending on the corresponding bounds, which are estimated using infinity norms' median at each respective layer. We also notice that the model's classification capability

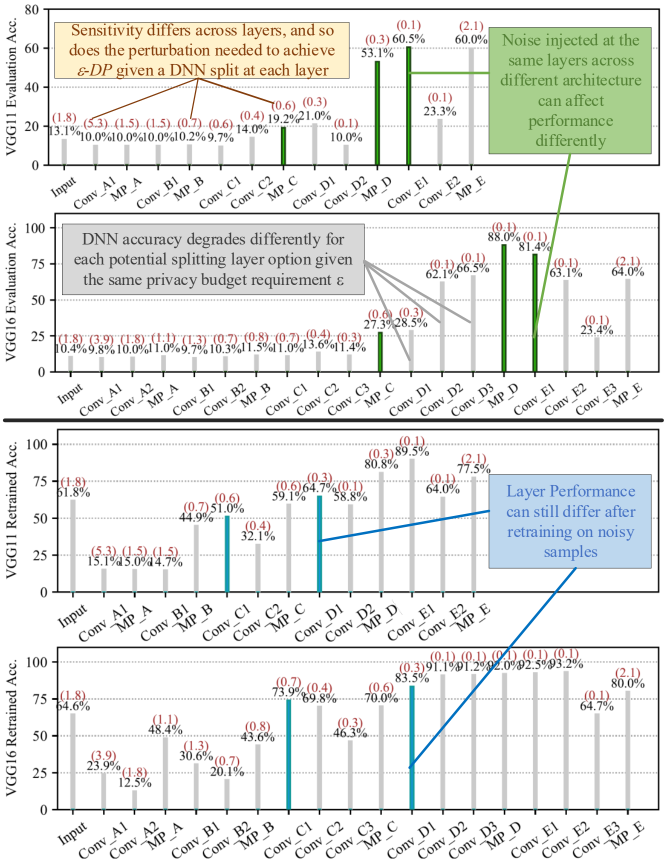

Fig. 3. The effect of varying architectural depth on utility illustrated through VGG11 and VGG16 before and after retraining for each potential injection layer (red values on top of the bars indicate b values).

suffers significantly when the injection layer is set prior to *MP_C*. On the other hand, we also note that the accuracy does not degrade as much for the VGG16 as its counterpart does across the latter layers for the same b requirement, indicating how the architectural build affects the resilience of a model. The other two bar plots show the accuracy of the DNNs after the noisy retraining of the cloud-side DNN. The key insight is that some of injection layers which initially led to a complete loss of utility (e.g., *Conv_C1*) can reach acceptable levels of accuracy after retraining. We observe that the deeper VGG16 layers generally sustain better recovery of utility after retraining.

Key Takeaway. *Retraining cloud-side DNNs on noisy representations can enhance the inference privacy model's accuracy. Thus, the chosen depth of noise injection presents a trade-off between privacy, accuracy, and computational complexity.*

> *Proposition 3:* ***Operation type.*** *The choice of kernel operations within a DNN model after ϵ-DP noise injection affects its degree of utility drop.*

Operation type: We compare the pretrained VGG16 against a variant which possess 5×5 *Conv* layers at blocks A and C (1^{st} and 3^{rd}) instead of the traditional 3×3 operations. As shown in the top two entries in Table II, same layers with disparate operations affect the model's utility differently. Specifically, some injection layers can degrade utility severely in one architecture but not the other. For example, when *MP_C* is set as the injection layer, performance drops significantly in VGG16 but not in its variant even after retraining. The opposite occurs when the injection layer is *Conv_D3*, showing how the operations' type influence utility under privacy constraints.

TABLE II
OPERATION AND WIDTH VARIATIONS EFFECT ON UTILITY FOR ϵ=2.8; ALL ARCHITECTURES ARE PRE-TRAINED TO $\sim$ 94% TEST ACCURACY.

Architecture	Layer	b	Eval. Acc.	Ret. Acc.
VGG16	MP_C	0.6	27.3%	**63.9%**
	Conv_D2	0.1	62.1%	91.1%
	Conv_D3	0.1	66.5%	**91.2%**
VGG16 5×5 ops.@ blocks A&C	MP_C	0.5	51.4%	**83.8%**
	Conv_D2	0.1	52.4%	90.2%
	Conv_D3	0.1	29.4%	**42.2%**
VGG16 0.5×#C_{in}@ blocks B&D	MP_C	0.6	26.5%	65.2%
	Conv_D2	0.2	68.6%	91.7%
	Conv_D3	0.1	27.9%	87.4%

Key Takeaway. *The inference privacy-by-design approach for DNN model architectures is to co-optimize the choice of DNN operators alongside the noise injection position.*

> *Proposition 4:* ***Width (# Channels).*** *Wider models with larger #channels can leverage computational redundancy to support inference privacy with tighter ϵ budgets.*

Width (# Channels): We provide a VGG16 variant containing half the number of output channels at its 2^{nd} and 4^{th} blocks, B and D. Despite being more concise, the variant's performance for different injection layers, shown in the last entry of Table II, is almost equivalent to that of the original VGG16 after retraining. In principal, this experiment is analogous to an analysis of the impact of channel pruning [39], except that the evaluation of performance degradation is performed on a mixture of clean and noisy samples. Thus, depending on the utility and privacy requirements, compact architecture designs can be chosen.

Key Takeaway. *Designing model architectures for ϵ-DP inference privacy is to enable compensating for the accuracy drop incurred from noise injection through including redundant computations along the width dimensions that aid in recovering utility.*

> *Proposition 5:* ***Residual Models.*** *To uphold the ϵ-DP inference privacy guarantee for residual models, the noise injection process* ***must*** *consider the model's multi-execution paths along its branches.*

Residual Models: Modern DNNs employ residual connections to counter the effects of vanishing/exploding gradients during training [40]. MobileNetV2 [20] is one such architecture designed for opertaion on mobile devices. We perform the same per-layer empirical performance analysis at $\epsilon = 2.8$, and evaluate the model's utility before and after retraining. For space considerations, we report the main takeaway from our analysis which was that partitioning and noise injection options should be restricted to the final concatenation/addition layer of every residual block, i.e., on an *inter-block* basis, because from an inference privacy perspective, partitioning on an intra-block basis (i.e., considering layers within residual block as potential partitioning options) would entail adding noise *twice* to both outputs from the layer and residual connection to satisfy ϵ-DP, which can significantly impact utility.

Key Takeaway. *The joining nodes that follow multi-path residual blocks represent the most adequate candidates for noise injection layers for a residual DNN model.*

Building on these observations, we investigate in the following Section how to parameterize privacy during the design stage of DNNs through implementing a customized NAS framework.

V. PrivyNAS: Privacy-Aware Neural Architecture Search Framework for Split Computing

We establish our customized PrivyNAS framework over the SOTA ProxylessNAS framework [21]. In brief, ProxylessNAS belongs to the class of NAS frameworks adopting the supernet and shared weights features as introduced in Section II-B. Originally, ProxylessNAS was implemented to design DNN model architectures suited for deployment on resource-constrained edge devices. In accordance, its supernet was constructed as a generalized form of the resource-efficient MobileNetv2 architecture [20] where the different architectural parameter choices are encoded at every residual block. Thus, contrary to the original MobileNetv2 model having fixed choices of kernel size and expansion ratios for its constituent layers, ProxylessNAS enables these architectural parameters to be variable across layers. To enable varying the depth of a candidate architecture, ProxylessNAS also includes a *skip* option in its pool of candidate operations for each MBConv layer. Regarding the controller's search strategy, a learnable gating mechanism is incorporated in order to control the sampling of architectural parameters whose combinations reflect diverse candidate architecture designs. As will be detailed in Section V-C, the search controller adopts a reinforcement learning approach to guide the gate selection strategy, in which the strategy is progressively updated via a reward function combining both accuracy and execution latency objectives for a target hardware platform. In the following, we breakdown the modifications and features we incorporate onto ProxylessNAS to support ϵ-DP within the NAS process.

A. Bounds estimation within NAS

Estimating the global sensitivity Δf through bounds B is key for determining the amount of additive noise needed to incorporate ϵ-DP (see (1)). In Section IV, bounds B were determined for a pre-trained model using the infinity norms of training data at the corresponding splitting layers. In a NAS framework however, the final model architecture is not known a priori, meaning that parameter weights – and accordingly B estimates – are susceptible to changes as the search progress. Moreover, it is impractical to train every candidate design in a search space just to obtain estimates of B.

Hence, we propose a *successive refinement* approach for the B estimates to enable the application of fine updates to the Laplacian noise distributions as the search progresses. As illustrated in Figure 4, this is motivated by the observed pattern in which infinity norm estimates for B change during the training of a MobileNetV2 architecture on CIFAR10. In particular, we notice that the norm values across the various layers progress in the exact opposite manner of training accuracy, where the initial sharp increase in accuracy is mirrored by a sharp decline in infinity norms. Additionally, once training enters the fine-tuning phase (after $\sim$ epoch 10), the rate of change of infinity norm values remarkably drops until the training concludes, as shown in the Figure through how the median and mean estimates changes. Analogously, NAS approaches like ProxylessNAS with the supernet and shared weights features can leverage this proposed successive bounds refinement technique as the supernet represent a generalized DNN model. Thus, all of its weights are trained and updated simultaneously providing the same progression pattern of infinity norms as in Figure 4.

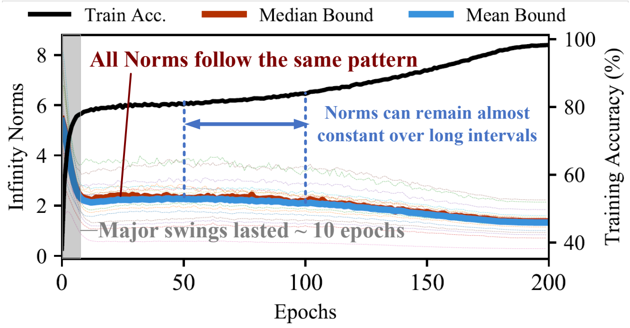

Fig. 4. The change in infinity norms (i.e., sensitivity bounds) and the training accuracy for a MobileNetV2 network on the CIFAR-10 dataset.

From here, we implement our approach to entail an initial warmup training phase using clean data representations until the large swings in the early training stage is bypassed. Afterwards, preliminary B estimates are computed to construct Laplacian noise distributions that meet ϵ for every candidate operation in the supernet. As the search progresses, noise tensors are sampled to be injected at a selected noise injection layer for a candidate architecture (subnet). An additional periodic pass on the clean data representations is invoked for the successive bounds refinement process in order to update B and the Laplacian scale values. In our experiments, a uniform periodic invocation frequency every 30 epochs was sufficient.

B. Joint Training

For inference privacy, cloud-side DNN retraining on noisy data representations is needed to maintain a model's utility (see Section III). In NAS, this is infeasible as neither the architecture nor the splitting layer are known beforehand. Alternatively, we propose to jointly train the supernet from the start on clean and noisy data representations where the injection layer is to be chosen dynamically as the search progresses. This is feasible in one-shot NAS techniques as all candidate architectures are trained simultaneously, and noisy representations can be provided through pre-specified ϵ budgets. Hence, the formulation of the training loss function in equation 2 is to be modified to include the entire weights of the supernet w rather than only the cloud-side parameters.

TABLE III
TRAINING AND TEST ACCURACY (%) ACROSS MOBILENETV2 RETRAINING TECHNIQUES AT ϵ=2.8 AND λ=0.5 OVER 200 EPOCHS.

Technique	Train$_{total}$	Test$_{total}$	Test$_{clean}$	Test$_{noisy}$
Fixed MB5	92.48	**88.65**	**89.7**	87.6
Inter-Block	85.72	**76.92**	**88.7**	65.1

As the final position of the noise injection layer is also not known aprioi, We examine how the training of a model under a dynamic assignment of the noise injection layer position can affect its accuracy. We use the MobileNetV2 architecture with 92.5% test accuracy on clean data representations. The dynamic assignment approach selects injection layers from the set of inter-block positions (recall the residual blocks in Section IV), and is compared to a fixed assignment approach with noise injection applied at the 5^{th} residual Block, namely MB5. As shown in Table III, the fixed assignment approach sustains an overall test accuracy after retraining on a clean/nosiy data mix of 88.65%, whereas the mean accuracy from training under all potential injection layers is 76.92%. The drop is comprehensible as we have shown in Section IV that different injection layer candidates do not preserve utility equally. Consequently, the NAS search would need to *"learn how to filter out"* such underperforming candidates, and concentrate the training effort to be conducted uner the most promising injection candidates. For concreteness, we also notice in Table III that the clean test accuracy suffers $< 4\%$ degradation, which represents the trade-off cost for training a MobileNetV2 at inference DP ϵ=2.8.

C. Search Parameters Setup

The processing of an input through the layers of a DNN model can be depicted as a directed acyclic graph (DAG) with the source node at the input and the sink node at the final output prediction. In between, each directed edge within the DAG represents an operation (e.g., convolution, pooling, etc) applicable to the intermediate feature representations at a particular layer position. Formally, the DNN model can be denoted as $\mathcal{M}(e_1, ..., e_n)$, with each edge, e_k, representing the k^{th} layer operation. In the context of NAS, the supernet represents a more generalized form of the DNN model, where instead of a singular operation, a set of all possible N operations, $\mathcal{O} = \{o_i\}$, becomes associated with each edge at layer k. Thus, e_k represents a mixed operation edge, $m^k_{\mathcal{O}}$, with N parallel paths. Hence, the overall supernet DAG can be characterized as $\mathcal{M}(m^1_{\mathcal{O}}, ..., m^n_{\mathcal{O}})$.

To sample a candidate model design (i.e., subnet) from within the supernet, a single path is to be selected out of the $|\mathcal{O}|$ choices from $m^k_{\mathcal{O}}$ for every k^{th} layer to form a subnet. To guide the selection of paths, recent works [21], [41], [42] have adopted a learning-based approach where for each k^{th} layer, N learnable architectural parameters, $\{\alpha_i\}$, are specified for the N edge paths in order to guide the selection of o_i, and ultimately provide the output features from the mixed operation $m^k_{\mathcal{O}}$. To elaborate, the application of $m^k_{\mathcal{O}}$ on an input x can be seen as $m^k_{\mathcal{O}}(x) = \sum^N_{i=1} g_i o_i(x)$, where $g_i \in \{0, 1\}$ represents a binary gate associated with each o_i, and only one g_i can be active at a time. In this case, α_i is used to determine the probability of g_i being the active gate through the Softmax activation probability given by $p_i = \frac{exp(\alpha_i)}{\sum^N_j exp(\alpha_j)}$. From here, through the designated search strategy and defined performance evaluation scheme, the NAS search progresses to update architectural parameter weights, α_i, each iteration to increase the sampling probabilities of the top performing operations at each $m^k_{\mathcal{O}}$ (which accordingly leads to better performing subnets).

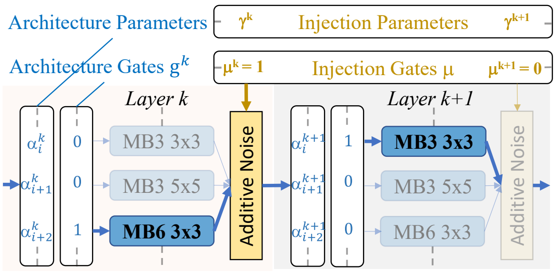

Fig. 5. Sampling of operations and the injection layer. *MBx y×y* are candidate operations from our experiments' search space inspired by [21]

In the context of our edge-cloud inference privacy use-case, the NAS search is also to be responsible for identifying the optimal splitting/noise injection layer between the edge and cloud. Similar to candidate architecture sampling, we define additional customized K injection parameters, denoted as $\{\gamma_k\}$, that are associated with the K possible injection layer positions in the supernet. Hence, the activation of an injection layer becomes also governed by a separate set of gates, $\{\mu_k\}$, where only a single μ_k can be active at any time with a Softmax probability $q_k = \frac{exp(\gamma_k)}{\sum^K_j exp(\gamma_j)}$. Given the active noise injection layer at position k, the output from $m^k_{\mathcal{O}}$ is perturbed via a noise addition function, ρ_k as follows:

$$\hat{m}^k_{\mathcal{O}} = \rho_k(m^k_{\mathcal{O}}) = m^k_{\mathcal{O}} + u_k \tag{3}$$

where u_k represents a noise tensor to be added to the output of $m^k_{\mathcal{O}}$ at the k^{th}. For concreteness, we depict the full operational sequence in Figure 5 as follows: a candidate submodel is sampled within the NAS with layer k as the noise injection layer and operation o^k_i as the k_{th} layer active operation. As shown, additive noise injection is applied through the noise tensor u_k sampled from a Laplacian distribution $Lap(0, \frac{\Delta f_{ki}}{\epsilon})$ constructed for operation o^k_i, where Δf_{ki} represents the model's global sensitivity when the i_{th} operation and the k_{th} layer are selected for the layer's operation and noise injection, respectively (recall Section V-A). Hence, the evaluation of candidate submodels under given ϵ-*DP* inference privacy guarantees becomes feasible.

D. Architecture and Injection Parameters Updates

The main objective from this customized NAS implementation is to jointly optimize the supernet's architecture α, injection γ, and weight w parameters. Consequently, this bi-level optimization problem can be formulated as follows:

$$\min_{\alpha,\gamma} \mathcal{L}_{val}(w^*(\alpha, \gamma), \alpha, \gamma) \tag{4}$$

$$s.t. \; w^* = argmin_w \; \mathcal{L}_{train} \tag{5}$$

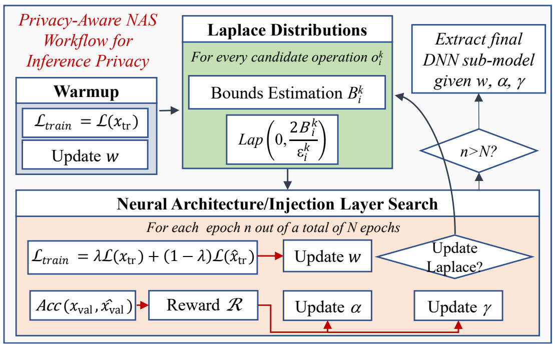
Privacy-Aware NAS Workflow for Inference Privacy
Warmup
$\mathcal{L}_{train} = \mathcal{L}(x_{tr})$
Update w
Laplace Distributions
For every candidate operation o_i^k
Bounds Estimation B_i^k
$Lap\left(0, \frac{2B_i^k}{\varepsilon_i^k}\right)$
Extract final DNN sub-model given w, α, γ
n>N?
Neural Architecture/Injection Layer Search
For each epoch n out of a total of N epochs
$\mathcal{L}_{train} = \lambda\mathcal{L}(x_{tr}) + (1-\lambda)\mathcal{L}(\hat{x}_{tr})$
Update w
Update Laplace?
$Acc(x_{val}, \hat{x}_{val})$
Reward $\mathcal{R}$
Update α
Update γ

Fig. 6. Basic workflow for the privacy-aware NAS for split computing.

where the search is for the optimal parameters α^* and γ^* which minimize a validation loss $\mathcal{L}_{val}$ given optimal parameter weights w^* that minimize the training loss $\mathcal{L}_{train}$. Here, $\mathcal{L}_{train}$ is defined based on the hybrid loss function in 2 except that it is jointly training all the weight parameters w of the supernet as discussed in Section V-B. In terms of $\mathcal{L}_{valid}$, ProxylessNAS leveraged REINFORCE [43] to update the architectural parameters α so as to find optimal gates g which maximize a reward $\mathcal{R}$. We reuse their update rule for our α parameters, and adopt a similar approach for injection parameters γ to identify the optimal injection gates μ. Specifically, we also use REINFORCE and define $\mathcal{L}_{val}(\gamma) = -J(\gamma)$ to specify an update rule for γ parameters as follows:

$$J(\gamma) = \mathbb{E}_{\mu\sim\gamma}[\mathcal{R}(\mathcal{M}_{g,\mu})] = \sum_k q_k \mathcal{R}(\mathcal{M}_g(\rho = \rho_k)) \qquad (6)$$

$$\nabla_\gamma J(\gamma) \approx \frac{1}{S}\sum_{s=1}^{S} \mathcal{R}(\mathcal{M}_{g^s,\mu^s})\nabla_\gamma log(q(\mu^s)) \qquad (7)$$

wheres S is the total number of samples, μ^s is the selected injection layer gate for sample s, $q(\mu^s)$ is the probability of sampling μ^s, and $\mathcal{M}_{g^s,\mu^s}$ is the sampled sub-model given the active gates g and μ in sample s.

E. Privacy-Aware NAS Workflow

From here, we conceptualize a privacy-aware NAS workflow in Figure 6 for designing DNN models suited for split-computing operation under a desired privacy budget ϵ. As shown, the warmup phase first trains the supernet on clean data representations to obtain initial estimates of B_i^k for each candidate operation o_i^k at the k^{th} layer within the supernet. Subsequently, their corresponding Laplacian distributions $Lap(0, \frac{2B_i^k}{\epsilon_i^k})$ are constructed to maintain the ϵ guarantee across all possible operations and injection layers. Next, the main search procedure can be invoked to train the shared parameter weights, w, each iteration. As stated in Section V-B, training is based on a mix of clean/noisy public data representations. For validation, sampled models' accuracy on a clean/noisy validation data mix are used to estimate the reward $\mathcal{R}$, and update the α and γ parameters accordingly. In the background, the successive bounds refinement is invoked periodically for each o_i^k to update their Laplacian distributions. Finally, the best performing submodel architecture and its optimal noise injection layer are identified for the latter post-design stages. We provide an analysis of the added search overhead complexity is provided in the experiments section.

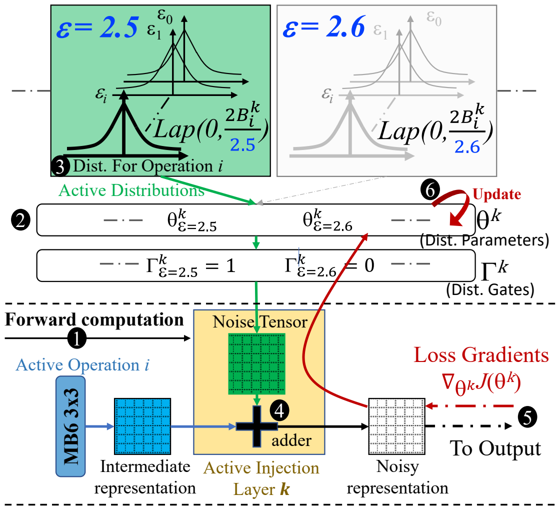

Fig. 7. Numbered step-by-step privacy-accuracy co-search (detailed in text)

F. Extension to Privacy-Accuracy co-search

We further include support for a *privacy-accuracy* co-search feature within PrivyNAS to facilitate the minimization of ϵ as an optimization objective. As such, rather than defining a single Laplacian distribution per each o_i^k given a predetermined ϵ, we associate a *multitude of distributions* with each operation satisfying different ϵ guarantees. Consequently, a dictionary of Z Laplacian distributions for each operation would be continuously updated as part of the successive bounds' refinement. To restrain selecting trivial distributions that could lead to a complete loss of utility, we further define *distribution parameters* θ_ϵ^k at each k layer to be associated with each prospective ϵ value that can be sampled from Z at layer k. That way, θ_ϵ^k parameters would be able to learn which ϵ values, and in turn distributions, to select for layer k when k is sampled as the noise injection layer.

We provide an illustrative example with numbered steps in Figure 7 on how this co-search progresses during a single epoch: **(1)** In the forward pass, a single operation o_i^k is activated at layer k outputting an intermediate data representation. **(2)** At active noise injection layer k, parameters θ_ϵ^k are used to select privacy budget ϵ, and its corresponding set of distributions through setting its corresponding gate Γ_ϵ^k to 1. **(3)** Based on the sampled ϵ and active operation o_i^k, the corresponding $Lap(0, \frac{2B_i^k}{\epsilon})$ is retreived. **(4)** Samples are drawn from the selected distribution to populate a noise tensor which is to be added to the intermediate representation. **(5)** The output noisy tensor is used to compute the output. **(6)** Gradients from the loss function are backpropagated to update θ^k parameters. The distribution parameters are updated in a manner similar to that for the architectural α parameters using $\mathcal{R}$, except that only the subset of θ^k parameters belonging to the current active k layer are updated at a time.

G. Reward definition and privacy-accuracy co-search support

For remote inference models, there exists an inherent trade-off between accuracy, performance, and privacy guarantees. Hence, we define the search's reward function, $\mathcal{R}$, as:

$$\mathcal{R} = acc(m) \times (\frac{L_{target}}{L(m)})^{\omega_L} \times (\frac{\epsilon_{target}}{\epsilon(m)})^{\omega_\epsilon} \quad (8)$$

where $acc(m)$, $\epsilon(m)$[1] , and $L(m)$ are the respective clean/noisy test accuracy, privacy budget, and latency achieved by model m. ϵ_{target} and L_{target} resemble the desired target privacy budget and latency by the designer. ω_ϵ and ω_L are configurable design trade-off parameters. As we are only concerned with resource efficiency from the edge devices' perspective, we breakdown $L(m)$ to its dominant components as follows:

$$L_m = L_{exec} + L_{Tx}; \quad L_{Tx} = \frac{Data\_Size}{Throughput} \quad (9)$$

where L_{exec} is the execution latency on the edge device for the local submodel $\mathcal{M}_l$, while L_{Tx} is the data transmission latency from the edge to the cloud dependent on the experienced wireless throughput at the edge and the transmissible data size.

VI. Experiments

Experimental Settings. We implement our PrivyNAS on top of ProxylessNAS [21] where we keep their default architectural search hyperparameter settings with the MobileNetV2 backbone except for the stride of the first stage blocks which is reduced from 2 to 1. We keep the same set of candidate operations $\{o_i\}$: mobile inverted bottleneck convolutions (MBConv) of kernel sizes $\in \{3, 5, 7\}$ and expansion ratios $\in \{3, 6\}$ in addition to the $skip$ operation to control network depth. The backbone comprises 6 search blocks with a maximum number of 4 MBConv layers per block, followed by one last block with a single MBConv layer. This arrangement enables having 21 potential candidate positions for the noise injection layer following each searchable residual block. We use initial learning rates of 0.15, 1×10^{-3}, 1×10^{-3}, and 5×10^{-4} to train w, α, θ, and γ, respectively. For the main procedure, we set λ= 0.5 and #epochs=200. For the privacy-accuracy co-search experiment, we assign 30 Laplacian distributions for each injection layer uniformly covering the ϵ choices within the range of [0.1,3].

Performance Characterization. We use the Nvidia Jetson TX2 (TX2) as our target edge hardware platform. The TX2 is capable of 1.33 TFLOPs and a power budget of 15 W. In order to provide accurate estimation of the models' performance during the PrivyNAS search process. We benchmark every operation from the supernet on the actual hardware device using the standard caffe framework. Using these measurements, we construct a lookup table that can be instantiated during the search in $\mathcal{O}(1)$ complexity. Thus, a performance characterization of the candidate client model designs can be determined conditioned on the underlying architecture and the chosen split layer.For the communication model, we set $Throughput$=10 Mbps and use *int8* format for transmissible data size.

TABLE IV
Comparing models' accuracy (%) at ϵ=2.8. For the NAS models, we set ω_e=0, and $L_{target} = 50$ms when $\omega_l \neq 0$. *(c)* indicates training using only clean data while *(ret.)* stands for retrained.

Model	Total Train	Clean Test	Noisy Test	Total Test
MobileNetv2	98.9 *(c)*	92.5	27.9	60.24
MobileNetv2 *(ret.)*	85.72	88.7	65.1	76.92
ProxylessNAS ($\omega_l = 0$)	91.1 *(c)*	80.35	65.58	72.23
ProxylessNAS$_{ret}$ ($\omega_l = 0$)	85.86	80.05	**78.36**	**79.16**
Privacy-aware ($\omega_l = 0$)	**90.76**	**80.44**	**79.22**	**79.90**
Privacy-aware ($\omega_l = 0.05$)	**88.99**	**79.36**	**76.99**	**78.11**

A. *Privacy-aware Search Analysis*

We first assess the privacy-aware NAS in comparison to a conventional privacy-agnostic NAS decoupling the architectural design from privacy considerations. Here, the conventional approach is emulated through a regular search from ProxylessNAS [21], followed by a separate independent process to integrate ϵ-DP inference privacy onto the final model. We also compare against the MobileNetV2 from section IV since it shares the same backbone architecture as ProxylessNAS [21]. Note that performance evaluations for MobileNetv2 and ProxylessNAS are estimated based on their best accuracy scores by choosing their optimal splitting layer (the 5th residual block and layer 13, respectively). A privacy budget of $\epsilon = 2.8$ is used for all implementations. The Reward function $\mathcal{R}$ used to update α and γ is determined here through the validation accuracy of a model *acc(m)* (i.e., $\omega_L = 0, \omega_\epsilon = 0$).

As shown in Table IV, a pretrained MobileNetV2 on clean data samples achieves the best accuracy on clean test data by 92.5%. However, it fails to do the same for the noisy test data, degrading the average accuracy to 60.24%. After noisy retraining, the MobileNetV2 average classification accuracy increases by $\sim$16% as it generalizes better to noisy samples, where noisy classification accuracy is improved by 37.2% at the expense of 3.8% degradation in clean test accuracy. The model from the conventional ProxylessNAS achieves an overall test accuracy of 72.23%, with a 14.8% variance in accuracy between clean and noisy representations. After noisy retraining for 150 epochs, the variance in accuracy is reduced to 1.69%, reaching an average accuracy of 79.16%. Our privacy-aware search renders a model which outperforms others in terms of average overall accuracy with 79.9%, indicating the value of the joint training during the search itself (Section V-B). Also, the variance between the clean and noisy test accuracies for the privacy-aware search reached 1.22%, indicating how the supernet is trained during the search to generalize its classification performance to samples perturbed in proportion to the desired ϵ budget.

B. *Injection Parameters γ Analysis*

We further analyze the accuracy of both our privacy-aware and the retrained ProxylessNAS models for each potential injection layer position to assess the merit of the γ parameters. In Figure 8, the two DNNs are compared over the 21 potential injection layer positions from the search space. Note that not only does the Figure compare optimal noise injection layer choices, but all potential choices including suboptimal ones.

[1] We slightly abuse the 'ϵ' notation and reuse it for the privacy objective function in addition to the privacy budget. Purpose can be inferred from context.

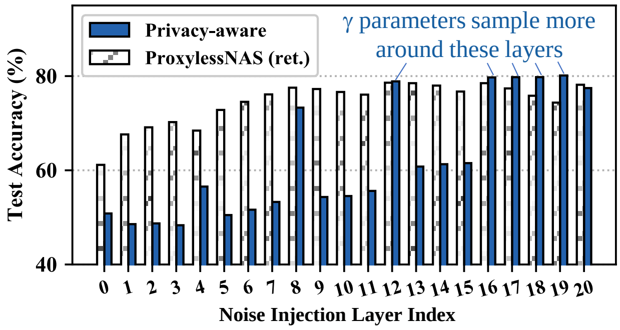

Fig. 8. Accuracy estimates of ProxylessNAS$_{ret}$ and our Privacy-aware ($\omega_l = 0$) across every potential injection layer at $\epsilon = 2.8$.

This is to demonstrate how the γ parameters of PrivyNAS learn to sample more frequently the most promising splitting layer candidates, and subsequently focus the supernet's training around them. As such, we find that the ProxylessNAS model outperforms PrivyNAS at subopitmal noise injection layer choices as the former randomly samples candidate injection layers with the same probability. However, ProxylessNAS does not outperform PrivyNAS at the latter's best injection positions (layers $12, 16, 17, 18$, and 19) which provide the highest accuracy overall under ϵ guarantees. As mentioned, this is attributed to the γ parameters that learn to optimize the model architectural design around these most promising injection layer positions as a result of their higher rewards compared to sub-optimal candidates.

C. Optimizing for performance and inference privacy

Since one motivation of split computing is to reduce computational overheads on user edge devices, we conduct another experiment in which both our privacy-aware model ($\omega_l = 0.05$, $L_{target} = 50ms$ and the *ProxylessNAS* architectures from Table IV) are first trained from scratch on clean data samples. Afterwards, their cloud-side parameters are trained on noisy samples at $\epsilon = 2.8$. Our analysis of accuracy and latency is performed at layers 8 and 12, which were the respective optimal noise injection layers for our privacy-aware model and the conventional ProxylessNAS one, respectively. As illustrated in Figure 9, the latency-agnostic *ProxylessNAS* model achieved the highest accuracy scores at its best injection layer, 12. This is because its architecture was only optimized for accuracy without any consideration of performance overheads, and thus it incurs a high execution latency for its local DNN components reaching 101.09 ms. Whereas at injection layer 8 (our model's best), we find that our model, designed with an L_{target}=50 ms, takes 53 ms latency to execute its local DNN components ($\mathcal{M}_l$) – a 35.2% reduction from that of ProxylessNAS model. Though such performance improvement comes at the expense of 2.2% accuracy drop from the layer 12 accuracy of ProxylessNAS model, our model improves accuracy by 3.4% in the scope of layer 8 only. This analysis showcases the inherent trade-offs existing between various conflicting design objectives.

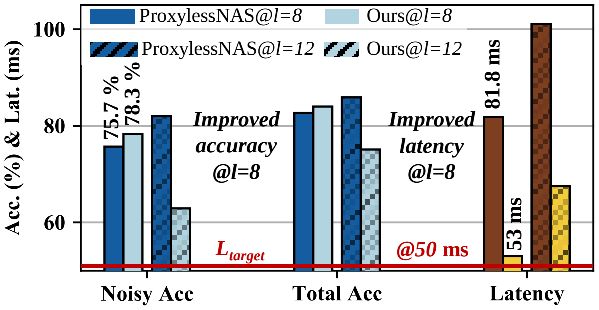

Fig. 9. Comparing noisy accuracy, total accuracy, and latency between our Privacy-aware ($\omega_l = 0.05$) and ProxylessNAS$_{ret}$ after cloud-side DNN retraining, where injection layers, 8 and 12, are the respective bests for each architecture. (*Note legend patterns hold for both accuracy and latency bars colored in blue and brown, respectively*)

TABLE V
COMPARING CO-SEARCH MODELS OVER ϵ AND UTILITY. THE CONVNET [10] DID NEITHER SPECIFY THE POSITION OF THE NOISE INJECTION LAYER (INJ.) NOR THE BREAKDOWN OF TOTAL ACCURACY.

	ϵ	Inj.	Clean	Noisy	Total
ConvNet [10]	3.5	-	-	-	79.52
Privacy-aware ($\omega_l = 0$)	2.8	19	80.44	79.22	**79.90**
Co-search($\omega_\epsilon = .5$)	0.1	12	78.68	22.97	46.27
Co-search($\omega_\epsilon = .1$)	**1.5**	**12**	**78.78**	**75.17**	**76.96**
Co-search($\omega_\epsilon = .1$)	**1.6**	**13**	**80.94**	**77.26**	**79.08**

D. Privacy-Accuracy Co-search

Next, we analyze the privacy-accuracy co-search from Section V-F. We set $\omega_l = 0$ and ϵ_{target} to 2.8 from the reward function in (8). We compare the final model from this search against the previously obtained privacy-aware model. We also compare against the model from [10] where *ϵ-DP* was adopted for inference privacy using a ConvNet architecture which is less dense than the MobileNetV2 variants. As shown in Table V, the ConvNet achieves 79.52% accuracy with an ϵ guarantee of 3.5, which is on the same accuracy level as the privacy-aware DNN of a stricter ϵ guarantee of 2.8 from the previous experiment. We conducted the co-search experiment initially for $\omega = 0.5$ which was biased towards minimizing ϵ leading to a complete loss of the model's utility. We re-ran the experiment for an $\omega = 0.1$ which provided a reasonable trade-off between the accuracy and privacy. The final model achieved 76.96% overall accuracy, 2.94% less than that of the privacy-aware approach but providing a more tighter privacy guarantee of $\epsilon = 1.5$.

To better visualize the accuracy-privacy trade-off, we contrast the test accuracy for both the privacy-aware and co-search models across different ϵ values in Figure 10. It can be observed that their accuracies remain close even at $\epsilon = 1.5$ with 1.16% difference in accuracy in favor of the co-search. Below $\epsilon = 1.5$, we can see that the accuracy degradation is steeper for the privacy-aware model compared to its counterpart, which is reasonable given how the privacy-aware model was not trained on privacy guarantees $\epsilon < 2.8$. However, this serves the privacy-aware model at the looser ϵ budgets where we observe that it outperforms the co-search model for values greater than the turnover point at $\epsilon = 1.7$.

The co-search model was superior at tighter ϵ budgets considering how its search process performed a fair amount

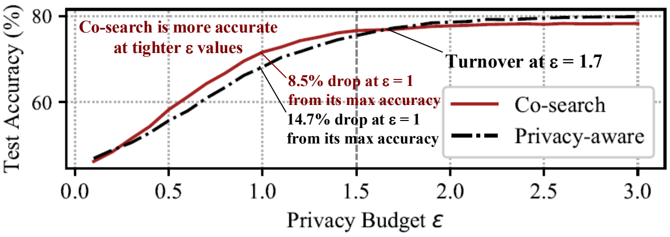

Fig. 10. Models' test Accuracy across different privacy budgets.

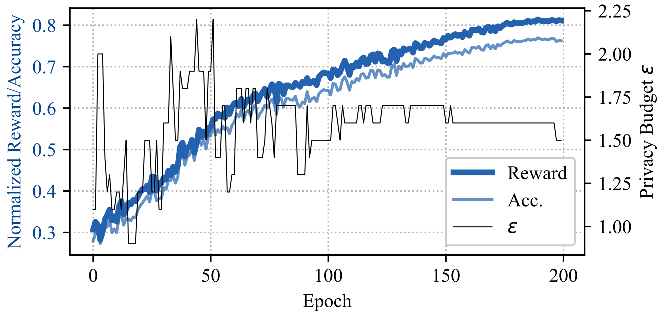

Fig. 11. Reward progress and privacy budget ϵ convergence for our co-search ($\omega_\epsilon = 0.1$).

of exploration to determine the suitable ϵ values around which the design would be fine-tuned. For a better visualization, we demonstrate the progression of the reward for the co-search process in Figure 11 with its respective accuracy and ϵ components. In the first 50 epochs of the experiment, the swing of ϵ values is bigger than that in the latter stages, implying how the search spends time initially learning which values of ϵ to operate around. Furthermore, the selection of ϵ values < 1.5 highlights the attempts made by the co-search process to optimize the design to operate under such tight constraints, and hence the reason it performed better at tighter privacy budgets.

E. Hyperparameter Analysis

We conduct multiple searches using different values of ω_L for the the reward function $\mathcal{R}$ in equation (8) to analyze the different performance trade-offs offered by the resulting models. All the searches are performed at fixed values of ω_ϵ=0.1, $\epsilon_{target} = 2.8$, and L_{target}=50 ms. The results are shown in Figure 12. The search at ω_l=0.1 provided a model with an early splitting layer at the 5th splitting layer candidate. As such, the model incurs a low local execution latency below the the intended target value of L_{target} at 43 ms. Not without demerits, the early noise injection layer leads the model to sustain a considerable drop in its accuracy at 67%. Given the limited accuracy score, the search attempted to maximize the reward by opting for a more aggressive privacy budget at ϵ=1.6. On the other extreme, an $\omega = 0$ does not consider any performance speedups, leading the search to realize a latency-intensive model through prioritizing maximizing accuracy and privacy objectives (79.1% and 2.3, respectively). The larger weight assigned to the accuracy objective has led the search in this case to prioritize at the expense of a slightly more relaxed ϵ=2.3 compared to the previous ϵ=1.6. The search at ω_L=0.01 strikes a reasonable balance across all objectives, realizing a model design with layer 12 as its splitting layer, providing an acc=78.7%, ϵ=1.6, and L=109 ms. Based on the values, it can be inferred that the search maximized $\mathcal{R}$ by prioritizing acc and ϵ over L due to their larger weights in the formula. This analysis and the one in VI-D has shown that values between 0 and 0.1 for the ω_L and ω_ϵ are sufficient to skew the search towards one objective without drastically affecting the other objectives.

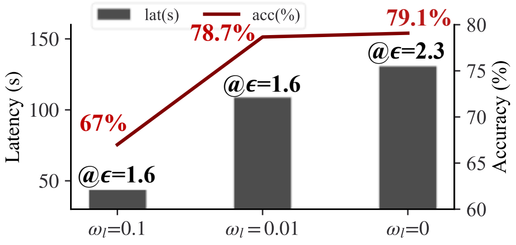

Fig. 12. A comparison of PrivyNAS co-searches conducted under different values ω_l (0, 0.01, 0.1) given a fixed $\omega_\epsilon = 0.1$. The ϵ values satisfied by the model in each search are displayed on top of each bar chart.

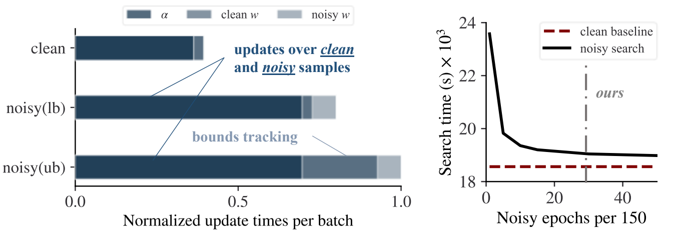

Fig. 13. Analyzing search time overheads of PrivyNAS. **Left:** a breakdown of the components for a reference clean NAS strategy (i.e., ProxylessNAS) compared to the lower (lb) and upper (ub) bounds of noisy updates experienced within PrivyNAS per batch. **Right:** Comparing the overall search times for the noisy PrivyNAS search when varying the number of noisy epochs. The *clean baseline* is scaled to the same number of samples as that of the *noisy search.*

F. Search Time Analysis

In this experiment, we compare the timing overheads incurred by the noisy search of PrivyNAS against that of the clean baseline of ProxylessNAS. We run each search strategy for 12 epochs, and record the average per-batch processing time across all epochs. Given the preiodic application of successive bounds refinement, we specify lower and upper bounds on the PrivyNAS search times to reflect when B updates were being performed, denoted by *noisy (lb)* and *noisy (ub)*, respectively. Figure 13 (*left*) depicts the comparison results normalized to noisy (ub) times. From this Figure, two key observations can be made: (*i*) the noisy search operates on both clean and perturbed data representations, leading the search to incur double the number of architectural (α) and w parameter updates. (*ii*) the average time per batch for clean w update (i.e., updating weights using unperturbed samples) in a noisy (ub) epoch is 7.15× more than that of its counterpart in a noisy (lb). This is due to the added sensitivity bounds computation overhead based on the forward passes of the clean data samples on a moving average basis, contributing an additional non-linear overhead to the overall inference cost of the clean samples.

To further analyze this non-linear increase in timing overheads resulting from periodic bounds update, we conduct an additional experiment in the right Figure 13 in which the update period,

TABLE VI
COMPARISON WITH RELEVANT PRIVACY PRESERVING METHODS. '○' INDICATES PARTIAL CONSIDERATION AS AN ANALYTICAL EXPERIMENT.

	PrivyNAS	PrivyNet [15]	Arden [10]	mSieve [44]	LATENT [23]	Shredder [11]	Cloak [12]	DPFE [36]	C2PI [34]
Inference Privacy	✓		✓	✓		✓	✓	✓	✓
ϵ-Differential Privacy	✓		✓	✓	✓				
Context-aware				✓		✓	✓	✓	
Encryption-based									✓
Noise layer co-optimize	✓	✓	○			○		○	✓
Architecture co-design	✓	✓							
Performance optimize	✓	○	○			○	○	○	✓

k, is varied to compare the overall search time overheads in seconds over the course of the 150 epochs (which are preceded by 50 clean warmup epochs). For instance, A value of 30 on the x-axis indicates that we perform one noisy (ub) every 30 epochs, and the rest incur the overhead of noisy (lb). The reference time is that of a clean baseline with double the number of samples to match that of the noisy search. Visibly, as the period of updates increases, the noisy search timing overhead reverts back at $k \geq 20$ epochs to values close to that of the reference baseline. In numbers, Performing an update at $k = 1$ takes 23.58×10^3 seconds compared to 19.04×10^3 seconds at $k = 30$, whereas clean reference time takes 18.56×10^3 seconds.

For the most part, this timing analysis holds for both the privacy-aware and co-search classes of search supported by PrivyNAS. The only difference lies in the added time due to the distribution parameters ϑ updates in a noisy(ub) epoch. In our experiments, the cost per epoch in the constraint-aware was ≈0.04 seconds compared to ≈0.16 seconds in the co-search one. Still, these update costs are relatively negligible compared to other components since they are only incurred in noisy (ub) epochs. Ultimately, the extent of the effect of these non-linear costs on the search times would vary depending on the supernet structure, the noisy injection layer candidates, and the number of Laplacian distributions associated with each layer.

G. Comparing Relevant Privacy Works

In Table VI, we compare PrivyNAS against its most relevant privacy preserving works in terms of the supported features of interest. Apart from PrivyNet [15] and LATENT [23], all other works targeted the inference privacy problem setting. Nonetheless, we include these works as the former considered co-optimizing privacy and certain architectural parameters, whereas the latter is another privacy work instance adopting *local* differential privacy. As shown in the Table, context-awareness and encryption are two features that are not provided natively through PrivyNAS. Still, the integration of additional privacy-preserving techniques on top of the ϵ-DP guarantee from PrivyNAS is feasible if stronger measures are needed for the application. For instance, privacy measures can be maximized through encrypting transmissible data from a model with base ϵ-DP guarantees using a Secure Multi-party Computation (SMC) approach [34]. PrivyNet [15] represents one work that has considered architectural parameter optimization when incorporating privacy measures. Their approach is different from PrivyNAS in that their use-case targets DNN model *training* in an edge-cloud setting, and in that they characterize privacy loss through peak-signal-to-noise ratio of reconstructed images. Regarding performance optimization, PrivyNAS and C2PI [34] are two works that actively consider performance optimization costs when integrating their privacy measures.

VII. DISCUSSION

Key Findings. In our experiments, we found that model splitting around the middle layers 8-13 achieved the best trade-offs amongst the accuracy, privacy, and latency objectives. This means that performance efficiency can roughly be doubled without drastically influencing other target objectives. Even so, PrivyNAS practitioners are still able to tune the weight hyperparameters, ω_L and ω_ϵ, to prioritize one objective over the other based on their needs. One research direction to soften the trade-off impacts is to devise new search spaces within PrivyNAS that are better suited to the edge-cloud deployment setting with the aim of providing shallower splitting candidates that improve on existing trade-offs.

Inference Privacy and ϵ-DP Composition. Through this ϵ-DP inference privacy model of computation leveraging local differential privacy, the composition property of DP becomes predominantly associated with the local DNN model. As such, an adversary with access to the cloud-side DNN outputs would find it extremely complicated to exploit the DP composition property. This is due to the low likelihood of encountering a 'linear-like' drop in the privacy budget as the number inference calls increases. One key reason being that the cloud operates and is trained directly on the already perturbed ϵ-DP data representations received from the client and not on the original inputs. Even if the unlikely scenario of an adversary having access to the local model is to be considered, ϵ privacy budget can be characterized according to the composition property from the early design stage. For instance in PrivyNAS, setting a tight privacy budget of $\frac{\epsilon}{k}$ would in theory lead to a model design with at least k inferences before privacy budget becomes ϵ,

ϵ-DP Inference and DL architecture: The consideration of inference privacy during the architectural design phase of DL models can be perceived as analogous to building a model that generalizes well to inputs from a different data distribution. For ϵ-DP inference, generalization is targeted towards samples experiencing randomized perturbations according to a formal measure of noise injection [45]. Subsequently, a privacy-aware design approach exhibits in essence a behavior similar to its conventional counterpart. That is, a more complex DNN architecture would generally lead to a better model utility under privacy considerations. Still, fine tuning the architectural design well-characterization of the desired privacy budget can lead to a good balance between generalization and overfitting.

Scalability: We have tested our hypothesis to consider ϵ-DP inference at design time using Convolutional Neural Networks (CNN) for image classification task. In practice, application providers in practice are more likely to reuse an existing state-of-the-art CNN architecture, and augment it with a privacy-preserving technique for remote inference if only to avoid going through the model design phase. Nonetheless, new classes of DNN solutions have been constantly materializing, and the application of AutoML techniques becomes more relevant in these early adoption phases. For instance, recent hardware-aware NAS targeted the design of novel transformer architectures for Natural Language Processing (NLP) in [46], [47]. It is for these emerging classes the impact of new concepts, like the inference privacy-aware design methodologies, can be maximized.

Study Limitations and Future Directions. Our focus in this paper was to empirically study how model architectural design can affect inference privacy given a standard dataset and task (i.e., image classification on CIFAR-10 dataset). The choice of CIFAR-10 was motivated by both the seminal works in [14] and PrivyNet [15] whose analysis focused on an image classification task with CIFAR-10 being their most challenging dataset. Still, further experimentation and analysis are needed beyond ours to assess the advantages of PrivyNAS in practical application settings. One way to achieve this is to scale the inference privacy problem setting to practical application domains entailing the edge-cloud architecture. An example is a mobile health application through which sensitive time-series data can be processed on the cloud [44]. Another direction is to study how the quality of a privacy leakage attack (e.g., membership inference or model inversion) degrades when applied to models provided by PrivyNAS compared to the baselines [34].

VIII. Conclusion

Accounting for inference privacy during the DNN design phase can be perceived as analogous to building a model that generalizes well to inputs from a different data distributions. Through a characterization of the formal ϵ-DP guarantees from the early design stages, a measure on the degree of indistinguishibility within the neural network itself can be promoted for the inference privacy use-case. In more realistic settings, knowledge of an architecture's inherent privacy-preserving capabilities can aid in making more informed decisions about which models to utilize in accordance with the desired privacy budget. Future research directions can investigate the applicability of privacy-aware AutoML for various other tasks, larger datasets, and evaluate the models resilience against known privacy attacks.

References

[1] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 1–9.

[2] O. Vinyals, L. u. Kaiser, T. Koo, S. Petrov, I. Sutskever, and G. Hinton, "Grammar as a foreign language," in *Advances in Neural Information Processing Systems*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, Eds., vol. 28, 2015.

[3] O. Faust, Y. Hagiwara, T. J. Hong, O. S. Lih, and U. R. Acharya, "Deep learning for healthcare applications based on physiological signals: A review," *Computer Methods and Programs in Biomedicine*, vol. 161, pp. 1–13, 2018.

[4] Y. Kang *et al.*, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," in *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '17, 2017, p. 615–629.

[5] Y. Matsubara *et al.*, "Head network distillation: Splitting distilled deep neural networks for resource-constrained edge computing systems," *IEEE Access*, vol. 8, pp. 212 177–212 193, 2020.

[6] M. Odema *et al.*, "LENS: Layer Distribution Enabled Neural Architecture Search in Edge-Cloud Hierarchies," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 403–408.

[7] A. Malawade, M. Odema, S. Lajeunesse-DeGroot, and M. A. Al Faruque, "Sage: A split-architecture methodology for efficient end-to-end autonomous vehicle control," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 20, no. 5s, pp. 1–22, 2021.

[8] A. Newcomb. (2018) Facebook data harvesting scandal widens to 87 million people. [Online]. Available: https://www.nbcnews.com/tech/tech-news/facebook-data-harvesting-scandal-widens-87-million-people-n862771

[9] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 1322–1333. [Online]. Available: https://doi.org/10.1145/2810103.2813677

[10] J. Wang *et al.*, "Not just privacy: Improving performance of private deep learning in mobile cloud," in *Proceedings of the 24th ACM SIGKDD Intl. Conf. on Knowledge Discovery & Data Mining*, 2018, pp. 2407–2416.

[11] F. Mireshghallah *et al.*, "Shredder: Learning noise distributions to protect inference privacy," in *Proceedings of the 25th Intl. Conf. on Arch. Support for Programming Languages and Operating Systems (ASPLOS'20)*, 2020.

[12] ——, "Not all features are equal: Discovering essential features for preserving prediction privacy," in *Proceedings of the Web Conference 2021*, ser. WWW '21, 2021, p. 669–680.

[13] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "GAZELLE: A low latency framework for secure neural network inference," in *27th USENIX Security Symposium (USENIX Security 18)*. USENIX Association, 2018, pp. 1651–1669.

[14] M. Abadi *et al.*, "Deep learning with differential privacy," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 2016, pp. 308–318.

[15] M. Li *et al.*, "Privynet: A flexible framework for privacy-preserving deep neural network training," *arXiv preprint arXiv:1709.06161*, 2017.

[16] C. Dwork, "A firm foundation for private data analysis," *Communications of the ACM*, vol. 54, no. 1, pp. 86–95, 2011.

[17] C. Dwork *et al.*, "The algorithmic foundations of differential privacy." *Found. Trends Theor. Comput. Sci.*, vol. 9, no. 3-4, pp. 211–407, 2014.

[18] F. Mirshghallah *et al.*, "Privacy in deep learning: A survey," *arXiv preprint arXiv:2004.12254*, 2020.

[19] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[20] M. Sandler *et al.*, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conf. on computer vision and pattern recognition*, 2018.

[21] H. Cai *et al.*, "Proxylessnas: Direct neural architecture search on target task and hardware," *arXiv preprint:1812.00332*, 2019.

[22] P. Kairouz, S. Oh, and P. Viswanath, "The composition theorem for differential privacy," in *International conference on machine learning*. PMLR, 2015, pp. 1376–1385.

[23] P. C. M. Arachchige *et al.*, "Local differential privacy for deep learning," *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 5827–5842, 2019.

[24] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *arXiv e-prints*, 2016.

[25] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, "Once-for-all: Train one network and specialize it for efficient deployment," *arXiv preprint arXiv:1908.09791*, 2019.

[26] H. Bouzidi, M. Odema, H. Ouarnoughi, M. A. Al Faruque, and S. Niar, "Hadas: Hardware-aware dynamic neural architecture search for edge performance scaling," in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2023, pp. 1–6.

[27] U. Erlingsson, V. Pihur, and A. Korolova, "Rappor: Randomized aggregatable privacy-preserving ordinal response," in *Proceedings of*

the 2014 ACM SIGSAC Conference on Computer and Communications Security, ser. CCS '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 1054–1067. [Online]. Available: https://doi.org/10.1145/2660267.2660348

[28] Y. Mao, S. Yi, Q. Li, J. Feng, F. Xu, and S. Zhong, "Learning from differentially private neural activations with edge computing," in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 2018, pp. 90–102.

[29] N. Papernot, M. Abadi, Úlfar Erlingsson, I. Goodfellow, and K. Talwar, "Semi-supervised knowledge transfer for deep learning from private training data," 2017.

[30] B. Ghazi, N. Golowich, R. Kumar, P. Manurangsi, and C. Zhang, "On deep learning with label differential privacy," 2021.

[31] Q. Zhang, L. T. Yang, and Z. Chen, "Privacy preserving deep computation model on cloud for big data feature learning," *IEEE Transactions on Computers*, vol. 65, no. 5, pp. 1351–1362, 2015.

[32] J. Liu *et al.*, "Oblivious neural network predictions via minionn transformations," ser. CCS '17, 2017, p. 619–631.

[33] P. Mohassel and Y. Zhang, "SecureML: A System for Scalable Privacy-Preserving Machine Learning," in *2017 IEEE Symposium on Security and Privacy (SP)*, 2017, pp. 19–38.

[34] Y. Zhang *et al.*, "C2pi: An efficient crypto-clear two-party neural network private inference," in *Proceedings of the 60th ACM/IEEE Design Automation Conference (DAC).*, 2023.

[35] S. Kariyappa, O. Dia, and M. K. Qureshi, "Enabling inference privacy with adaptive noise injection," 2021.

[36] S. A. Osia *et al.*, "Deep private-feature extraction," *IEEE Transactions on Knowledge and Data Engineering*, vol. 32, no. 1, pp. 54–66, 2018.

[37] ——, "A hybrid deep learning architecture for privacy-preserving mobile analytics," *IEEE Internet of Things Journal*, vol. 7, no. 5, pp. 4505–4518, 2020.

[38] "pytorch-cifar," https://github.com/kuangliu/pytorch-cifar.

[39] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[40] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 2261–2269.

[41] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable architecture search," in *International Conference on Learning Representations ICLR 2019*, 2019.

[42] Z. Guo, X. Zhang, H. Mu, W. Heng, Z. Liu, Y. Wei, and J. Sun, "Single path one-shot neural architecture search with uniform sampling," in *European Conference on Computer Vision*. Springer, 2020, pp. 544–560.

[43] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3, pp. 229–256, 1992.

[44] N. Saleheen *et al.*, "mSieve: differential behavioral privacy in time series of mobile sensor data," in *Proceedings of the 2016 ACM Intl. Joint Conf. on Pervasive and Ubiquitous Computing*, 2016, pp. 706–717.

[45] P. Cuff and L. Yu, "Differential privacy as a mutual information constraint," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 43–54. [Online]. Available: https://doi.org/10.1145/2976749.2978308

[46] H. Wang, Z. Wu, Z. Liu, H. Cai, L. Zhu, C. Gan, and S. Han, "Hat: Hardware-aware transformers for efficient natural language processing," 2020.

[47] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2017.

Mohanad Odema received the B.Sc. degree in Communications and Electronics Engineering and the M.Sc. degree in Computer Engineering from Ain Shams University, Cairo, Egypt in 2014 and 2018, respectively. He is currently pursuing the Ph.D. degree in Computer Engineering with the University of California at Irvine (UCI). His current research interests are focused on design methodologies for robust and efficient edge computing using dynamic neural network architectures, especially for autonomous systems and mobile health applications.

Mohammad Abdullah Al Faruque (Senior Member, IEEE) received his B.Sc. degree in Computer Science and Engineering (CSE) from Bangladesh University of Engineering and Technology (BUET) in 2002, and M.Sc. and Ph.D. degrees in Computer Science from Aachen Technical University and Karlsruhe Institute of Technology, Germany in 2004 and 2009, respectively. Mohammad Al Faruque is currently with the University of California Irvine (UCI), a Full Professor and directing the Embedded and Cyber-Physical Systems Lab. Before he was with Siemens Corporate Research and Technology in Princeton, NJ. Prof. Besides 120+ IEEE/ACM publications in the premier journals and conferences, Prof. Al Faruque holds 11 US patents. Prof. Al Faruque is currently serving as the associate editors of the ACM Transactions on Design Automation on Electronics and Systems and the IEEE Design and Test. He is an IEEE senior member and an ACM senior member. He is also the IEEE CEDA Distinguished Lecturer for 2022-2023.