EFFICIENT EDGE INFERENCE BY SELECTIVE QUERY

Anonymous authors
Paper under double-blind review

ABSTRACT

Edge devices provide inference on predictive tasks to many end-users. However, deploying deep neural networks that achieve state-of-the-art accuracy on these devices is infeasible due to edge resource constraints. Nevertheless, cloud-only processing, the de-facto standard, is also problematic, since uploading large amounts of data imposes severe communication bottlenecks. We propose a novel end-to-end hybrid learning framework that allows the edge to selectively query only those hard examples that the cloud can classify correctly. Our framework optimizes over neural architectures and trains edge predictors and routing models so that the overall accuracy remains high while minimizing the overall latency. Training a hybrid learner is difficult since we lack annotations of hard edge-examples. We introduce a novel proxy supervision in this context and show that our method adapts seamlessly and near optimally across different latency regimes. On the ImageNet dataset, our proposed method deployed on a micro-controller unit exhibits 25% reduction in latency compared to cloud-only processing while suffering no excess loss.

1 INTRODUCTION

Deep Neural Networks (DNNs) achieve state-of-the-art (SOTA) performance on challenging tasks such as image recognition (Tan & Le, 2019), language modelling (Devlin et al., 2018b), and machine translation (Wu et al., 2016). However, SOTA performance is only achievable on cloud devices since edge devices impose severe constraints on model memory usage, energy consumption, and inference latency. Thus, on-device solutions lead to significant performance degradation. Nevertheless, the current status quo of cloud-only processing is also problematic. Uploading large amounts of data through wireless services imposes bottlenecks due to communication latency and significant battery utilization arising from the transmission.

Figure 1: HYBRID MODEL. Cheap base (b) & routing models (r) run on a micro-controller; Expensive global model (g) runs on a cloud. r uses x and features of b to decide if g is evaluated or not.

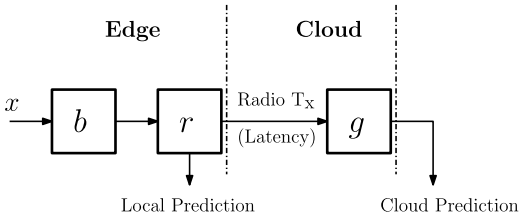


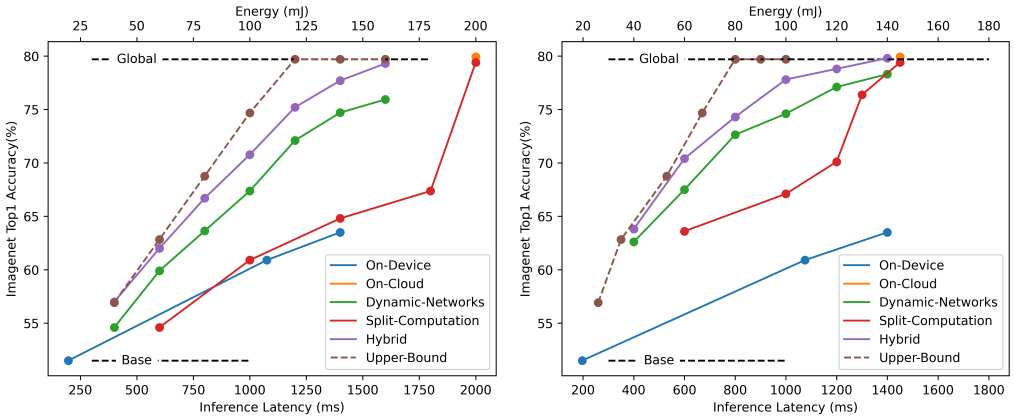
Table 1: Device & Model Characteristics: Edge (STM32F746 MCU), Cloud (V100 GPU). It takes 2000ms to communicate an ImageNet image from the edge to the cloud (see Appendix §A.2)

Device	Device Characteristics		Model Performance		
	Memory	Storage	Accuracy	Size	Latency
MCU	320KB	1MB	51.1%	0.6MB	200ms
GPU	16GB	1TB	79.9%	9.1MB	25ms

Let us examine a large-scale classification task through the lens of a tiny edge device since this will serve to highlight novel aspects of the paper. We emphasize that the methods proposed in this paper generalize to any hardware and latencies (see Section 3).

ImageNet Classification on an MCU. Table 1 displays accuracy and processing latencies on a typical edge (MCU) model and a cloud model (see Appendix §A.2). The cloud has GPUs, and its processing speed is 10x relative to MCU processing. In addition, the cloud has a much larger working memory and model storage space compared to the MCU.

Impact of Communication Latency. If communication latency were negligible, we would simply transfer all the data to the cloud. In a typical system with NB-IoT communication (see Sec. A.2), data transfer to the cloud can take about $2s$—about $100\times$ the processing time on a GPU—and thus,



(a) (b)

Figure 2: ImagetNet Classification: Accuracy vs Energy and Latency plot: (a) Constant Communication Latency (2000ms), and (b) Dynamic Communication latency $[200, 2000]$ms. It clearly shows that the proposed hybrid model pareto dominates on-device as well as other baselines while getting significantly closer to the upper-bound in hybrid setup.

to reduce communication, one should maximize MCU utilization for classification. The argument from an energy utilization viewpoint is similar; it costs $20\times$ more for transmitting than processing. In general, communication latency is dynamically changing over time. For instance, it could be as large as 10x the typical rate. In this case, we would want the MCU unit to dynamically adapt its processing so that at fast rates, much of the processing happens on the cloud, and for slow rates, the MCU must still decide what examples to process locally.

Proposed Hybrid Method. In order to handle latency requirements and a pretty small footprint of edge devices, we need a hybrid method. As illustrated schematically in Fig. 1, our proposal consists of an edge device equipped with a base predictor, b, and a router, r, which selectively queries the cloud-based global model, g, to meet the target specifications. Specifically, the scenarios described earlier lead us to pose two constrained learning problems.

Target Latency. For a user interested in overall latency, the router takes, at each time, the base, communication, and server latency as input and solves for the base coverage level (minimum number of examples required to be processed on the edge device) to meet the overall latency target.

Target Error. When communication latency is significant, as described above, we are motivated to minimize the overall latency required to achieve the target error.

We propose a novel end-to-end framework for learning hybrid models that takes latency and edge hardware specifications as inputs and outputs a hybrid model to meet target latency or target error constraints. To handle dynamic latency, our router tunes a parameter to meet target latency requirements. Table 2 compares salient attributes of our method relative to prior works.

In this context, a fundamentally challenging hybrid learning problem that arises is learning routing policies.

Table 2: Comparing features of our proposal against baseline. E.-to-E. stands for 'End-to-End', and Arch. for 'Architecture'.

Method	Low Latency	Deploy. on Edge	E.-to-E. Training	Arch. Search	High Accuracy
On-Device	✓	✓	-	✓	✗
On-Cloud	✗	✗	-	✓	✓
Split-Comp.	✗	✗	✓	✗	✗
Dynamic.	✓	✓	✗	✗	✗
Hybrid	✓	✓	✓	✓	✓

Novel Proxy Supervision. Training routing models is difficult because we do not a priori know what examples are hard to classify on edge. More importantly, we only benefit from transmitting those hard-to-learn examples that the cloud model correctly predicts. In this context, we encounter three situations, and depending on the operational regime of the hybrid system, different strategies may be required. To expose these issues, consider an instance-label pair (x, y), and the three typical possibilities that arise are:

- Edge and Cloud are both accurate, $b(x) = g(x) = y$;
- Edge and Cloud are both inaccurate, $b(x) \neq y$ and $g(x) \neq y$;
- Edge is inaccurate but Cloud is accurate $b(x) \neq g(x) = y$.

Our objective is to transmit only those examples satisfying the last condition. *It reduces latency by ceasing data transfers when cloud predictions are bound to be incorrect*. However, the limited capacity of the router (deployed on the MCU) limits how well one can discern which of these cases are true, and generalization to the test dataset is difficult. As such, the routing would benefit from supervision, and we introduce a novel *proxy supervision* (see Sec. 2.1) to learn routing models while accounting for the base and global predictions.

Contributions. In summary, we list our contributions below.

- *Novel End-to-End Objective.* We are the first to propose a novel global objective for hybrid learning that systematically learns all of the components, base, router, global model, and architectures under overall target error or dynamic target latency constraints.
- *Proxy Supervision.* We are the first to provably reduce router learning to binary classification and exploit it for end-to-end training based on novel proxy supervision of routing models. Our method adapts seamlessly and near optimally across different latency regimes (see knee in Fig. 2).
- *Hardware Agnostic.* Our method is hardware agnostic and generalizes to any edge device (ranging from micro-controllers to mobile phones), any server/cloud, and any communication scenarios. Our experiments include (a) MCU and GPU (see Sec. 3.1), (b) Mobile Devices and GPUs (see Sec. 3.1), (c) on the same device (see Sec. 3.2, Appendix A.11.1).
- *SOTA Performance on Benchmark Datasets.* We run extensive experiments on benchmark datasets to show that the hybrid design reduces inference latency as well as energy consumption per inference. We will release our PyTorch implementation in the final version.

1.1 Prior Works with Empirical Comparisons on MCU

We use Table 1 as a running example to survey prior works, but our inferences here, as seen in Sec. 3 hold generally. Our goal is to illustrate key differences through this practical scenario. We refer the reader to Appendix A.2 for more experimental details. We consider two communication contexts: (a) Fixed but large latency (2000ms); (b) Dynamic Latency: A random variable that is uniformly distributed in the interval $[200, 2000]$ms.

All-Cloud Solution. Here, the edge device transmits all examples to the cloud. It incurs a communication cost in addition to the cloud inference latency. This solution is sub-optimal as seen by the orange circle with 80% accuracy and inference latency of 2025ms in Figure 2(a). Our goals are:
(i) To understand the trade-off between latency and accuracy;
(ii) To compare prior methods by the latency level at which cloud-accuracy is achieved.

All-Edge Solution. While we can leverage various methods (Howard et al., 2019; Hinton et al., 2015; Cai et al., 2020), MCUnet Lin et al. (2020) is the most suitable deployable model for this micro-controller. [1] All-edge baseline is blue curve in Figure 2(a). The smallest MCUNet model achieves 51.1% accuracy with 200ms inference latency. In contrast, the largest MCUNet model achieves 63.5% accuracy with 1400ms inference latency; models any larger cannot be deployed due to the MCU hardware limitations listed in Table 1. Thus, *cloud-accuracy is unachievable with an all-edge solution for any latency.*

Split-Computation methods. Kang et al. (2017) splits the model between the edge and cloud device. More specifically, the initial part of the DNN is stored and executed on the edge device. Then, the edge transmits the intermediate features to the cloud, where the remaining computation of the DNN is performed. This split takes into account the latency budget and the edge transmission cost. More recent variants (Li et al., 2021; Odema et al., 2021) add a branch-out stage that allows for classification on the edge device whenever the decision is deemed correct. We employ this method in our comparisons. A fundamental issue is that requiring copies of the same model on server/edge severely constrains practical use cases. For instance, in Table 1 server model is 9X of MCU model. In this case, we could either use the maximum-size edge model and copy it to the server or store a partial copy of the server model locally. The former leads to topping accuracy at 63.5% (blue-line) at high latency, while the latter is plotted as the red curves in Figure 2. While results improve in the dynamic setting,

[1] While we are aware that Lin et al. (2021) provides an updated version of the MCUNet model, their pre-trained models are not available publicly. Hence, we were unable to use the same in this study. Besides, any improvement in the edge model would yield improvements in all the hybrid baselines.

they are not comparable to the other baselines. This fact is not surprising since the amount of pre-processing gained by having a shared model is more than offset by allowing models optimized to the edge. Since the MCUs are too small, only a tiny part of the large DNN can be stored/executed on the edge device. The resulting intermediate feature size is larger than the input size. Thus, cloud accuracy can only be achieved by transmitting all examples to the cloud at a latency of an all-cloud solution.

Dynamic Neural Network methods are surveyed in (Han et al., 2021). These methods budget more compute for hard instances. It includes (a) cascade-based early exit networks (Park et al., 2015; Bolukbasi et al., 2017; Wang et al., 2018), where the constituents are independently designed without feature sharing; and (b) early exit networks (Teerapittayanon et al., 2017; Dai et al., 2020; Li et al., 2019) where classifiers are introduced at intermediate layers. These works generally disregard the severely resource-constrained setting such as in Table 1. Within this framework, we experimented with different approaches and found that (Bolukbasi et al., 2017; Nan & Saligrama, 2017; Li et al., 2021) serves as a good baseline. These allow for different models optimized for the base and global, but the routing scheme is designed post-hoc based on classifying locally depending on prediction uncertainty. This is depicted as green-line in Figure 2.

Proposed hybrid method, in contrast to Dynamic Networks, globally optimizes all of the parts (base, router, and cloud) and is the purple line in Figure 2. It is evidently close to the upper bound, which is derived in Appendix A.2 based on the practical assumption that the base model is agnostic to what examples are classified by cloud correctly. Prior works including dynamic networks and split-computation methods general miss key details, namely, (a) no explicit accounting number of examples required to be covered locally, (b) no supervision for the router, which makes them myopic, and (c) evaluations only on small datasets (see Appendix §A.7 for details), etc.

Selective Classification. Recently, (Liu et al., 2019; Gangrade et al., 2021; Geifman & El-Yaniv, 2019) proposed learning with a reject option, wherein a model can abstain from predicting uncertain instances. Although we obtain a selective classifier by ignoring the global model (see §3.2), we focus on improving the hybrid system, including the global model, which is a different objective compared to this line of research.

2 METHOD

Notation. Let $\mathcal{X}$ be a feature space and $\mathcal{Y}$ a set of labels. Hybrid design consists of the following:

- A *base model* $b : \mathcal{X} \to \mathcal{Y}$, deployed on an edge device.
- A *global model* $g : \mathcal{X} \to \mathcal{Y}$ deployed on the cloud.
- A *routing model* $r : \mathcal{X} \to \{0, 1\}$ deployed on the edge.

We will treat these models as soft classifiers, outputting $|\mathcal{Y}|$-dimensional scores $\{b_y\}$ and $\{g_y\}$, and two scores r_0 and r_1 for the routing model. The hard output for the base is the top entry $b(x) = \arg\max_y b_y(x)$, and similarly for g. In this paper, r is realized by the a 2-layer DNN with input $b_y(x)$ - this is intentionally chosen to have minimal implementation complexity. r further admits a scalar tuning parameter t, and assigns x to the global model if $r_1 > r_0 + t$, i.e.

$$r(x; t) = \mathbb{1}\{r_1(x) > t + r_0(x)\}.$$

Varying t (Alg. 4) trades-off accuracy and resource usage, and allows us to avoid retraining r at each resource level by locally tuning a router. By default $t = 0$. The hybrid prediction for an instance x is

$$\hat{y}(x) := (1 - r(x))b(x) + r(x)g(x). \tag{1}$$

Evaluation Metrics. *Hybrid accuracy* is the accuracy of the hybrid predictions, defined as

$$\mathcal{A}(r, b, g) = \mathbb{P}(\hat{y}(X) = Y) = \mathbb{P}(r(X) = 0, b(X) = Y) + \mathbb{P}(r(X) = 1, g(X) = Y).$$

This accuracy is fundamentally traded-off with the *coverage* of the hybrid models, which is the fraction of instances that are processed by the cheap base model only, i.e.

$$\mathcal{C}(r, b, g) := \mathbb{P}(r(X) = 0).$$

Modeling Resource Usage. Coverage offers a generic way to model the resource usage of hybrid inference as follows. The resource cost of most models is mainly a function of the architecture. We let α denote a generic architecture and say that $f \in \alpha$ if the function f is realizable by the architecture. Then the resource cost of f is denoted $\mathcal{R}(\alpha)$. Our hybrid design always executes the base and the

router, and so the mean resource usage of the hybrid model (r, b, g) with $b \in \alpha_b$ and $g \in \alpha_g$ is

$$\mathcal{R}(r, b, g) := \mathcal{R}_r + \mathcal{R}(\alpha_b) + (1 - \mathcal{C}(r, b, g))\mathcal{R}(\alpha_g), \tag{2}$$

where $\mathcal{R}_r$ is a small fixed cost of executing r.

This generic structure can model many resources such as energy usage or total FLOPs used. Our paper is mainly concerned with *inference latency*. To model this, we take $\mathcal{R}(\alpha_b)$ to be the mean time required to execute a base model on the edge, and $\mathcal{R}(\alpha_g)$ to be the *sum of* the mean computational latency of executing g on the cloud and the mean communication latency of sending examples to the cloud. In the previous example of Table 1, these numbers would be 200ms and 2025ms respectively.

Overall Formulation. Let $\mathscr{A}_b$ and $\mathscr{A}_g$ be sets of base and global architectures that incorporate implementation restrictions, and ϱ a target resource usage level. Our objective is

$$\max_{\alpha_b \in \mathscr{A}_b, \alpha_g \in \mathscr{A}_g} \max_{r, b \in \alpha_b, g \in \alpha_g} \mathcal{A}(r, b, g) \text{ s.t. } \mathcal{R}(r, b, g) \leq \varrho. \tag{3}$$

The outer $\max$ over (α_b, α_g) in (3) amounts to an architecture search, while the inner $\max$ over (r, b, g) with a fixed architecture corresponds to learning a hybrid model.

Below, we describe our method for solving (3). Briefly, we propose to decouple the inner and outer optimization problems in (3) for efficiency. We train hybrid models by an empirical risk minimisation (ERM) strategy. Furthermore, in Sec. 3.3, we perform architecture search using fast proxies for the accuracy attainable by a given pair of architectures without directly training hybrid models.

2.1 Learning Hybrid Models

This section trains hybrid models for fixed architectures α_b, α_g, i.e., the inner minimisation problem

$$\max_{r, b \in \alpha_b, g \in \alpha_g} \mathcal{A}(r, b, g) \quad \text{s.t.} \quad \mathcal{R}(r, b, g) \leq \varrho. \tag{4}$$

Since architectures are fixed in (4), the resource constraint amounts to a constraint on the hybrid coverage $\mathcal{C}(r, b, g)$. As is standard, we will approach (4) via an ERM over a Lagrangian of relaxed losses. However, several design considerations and issues need to be addressed before such an approach is viable. These include: (a) cyclical non-convexity in (4), (b) supervision for the router, and (c) relaxed loss functions. We discuss these below and summarise the overall scheme in Algorithm 1.

Alternating optimisation. Problem (4) has a *cyclical non-convexity*. A given r affects the optimal b and g (since these must adapt to the regions assigned by r), and vice-versa. We approach this issue by alternating optimisation. First, we train global and base models with standard methods. Then, we learn a router r under a coverage penalty. The resulting r feeds back into the loss functions of b and g, and these get retrained. This cycle may be repeated.

Algorithm 1 Training Hybrid Models

1: **Input:** Training data $\mathcal{D} = \{(x^i, y^i)\}_{i=1}^N$
2: **Hyper-parameters:** λ_r, Number of epochs E
3: **Initialize:** random r^0, pre-trained b^0, g^0.
4: **for** $e = 1$ **to** E **do**
5: Randomly Shuffle $\mathcal{D}$
6: Generate oracle dataset $\mathcal{D}_{o;(b,g)}$ using Eq. 6
7: $r^e = \arg\min_r \mathcal{L}_{\text{routing}}(r, b^{e-1}, g^{e-1})$
8: $g^e = \arg\min_g \mathcal{L}_{\text{global}}(r^e, b^{e-1}, g)$
9: $b^e = \arg\min_b \mathcal{L}_{\text{base}}(r^e, b, g^e)$
10: **Return :** (r^E, b^E, g^E)

Modularity of training. The scheme allows modularisation by varying which, and to what extent, steps 7,8,9 in Alg.1 are executed. It helps train a cheap routing model to hybridise a given pair of pre-trained b and g; or in using an off-the-shelf global model that is too expensive to train. Finally, we can learn each component to different degrees, e.g., we may take many more gradient steps on g than r or b in any training cycle.

Learning Routers via Proxy Supervision. Given a base and global pair (b, g), Eqn.(4) reduces to

$$\max_r \mathbb{E}[(1 - r(X))\mathbb{1}\{b(X) = Y\} + r(X)\mathbb{1}\{g(X) = Y\}] \qquad \text{s.t.} \quad \mathbb{E}[r(X)] \leq C_\varrho, \tag{5}$$

where C_ϱ is the coverage needed to ensure $\mathcal{R} \leq \varrho$. While a naïve approach is to relax r and pursue ERM, we instead reformulate the problem. Observe that (5) demands that

$$r(X) = \begin{cases} 0 & \text{if } b(X) = Y, g(X) \neq Y \\ 1 & \text{if } b(X) \neq Y, g(X) = Y. \end{cases}$$

Further, while $b(X) = g(X)$ is not differentiated, the coverage constraint promotes $r(X) = 0$. Thus, the program can be viewed as a supervised learning problem of fitting the *routing oracle*, i.e.

$$o(x; b, g) = \mathbb{1}\{b(x) \neq g(x) = y\}. \tag{6}$$

Indeed, o is the ideal routing without the coverage constraint. For any given (b, g) and dataset $\mathcal{D} = \{(x^i, y^i)\}$, we produce the oracle dataset $\mathcal{D}_{o;(b,g)} := \{(x^i, o(x^i; b, g))\}$. It serves as supervision for the routing model r. This allows us to utilise the standard machine learning tools for practically learning good binary functions, thus gaining over approaches that directly relax the objective of (5).

The oracle o does not respect the coverage constraint, and we would need to assign some points from the global to base to satisfy the same. From a learnability perspective, we would like the points flipped to the base to promote regularity in the dataset. However, this goal is ill-specified, and unlikely to be captured well by simple rules such as ordering points by a soft loss of g. We handle this issue indirectly by imposing a coverage penalty while training the routing model and leave it to the penalised optimisation to discover the appropriate regularity.

Focusing competency and loss functions. To improve accuracy while controlling coverage, we focus the capacity of each of the models on the regions relevant to it - so, b is biased towards being more accurate on the region $r^{-1}(\{0\})$, and similarly g on $r^{-1}(\{1\})$. Similarly, for the routing network r, it is more important to match $o(x)$ on the regions where it is 1, since these regions are not captured accurately by the base and thus need the global capacity. We realise this inductive bias by introducing model-dependent weights in each loss function to emphasise the appropriate regions.

Routing Loss consists of two terms, traded off by a hyperparameter λ_r. The first penalises deviation of coverage from a given target (cov), and the second promotes alignment with o and is biased by the weight $W_r(x) = 1 + 2o(x)$ to preferentially fit $o^{-1}(\{1\})$. Below, ℓ denotes a surrogate loss (such as cross entropy), while $(\cdot)_+$ is the ReLU function, i.e., $(z)_+ = \max(z, 0)$. Sums are over training data of size N. Empirically, we find that $\lambda_r = 1$ yields effective results.

$$\mathcal{L}_{\text{routing}}(r; o) = \lambda_r \left(\text{cov} - \left(1 - \frac{1}{N} \sum_x (r_1(x) - r_0(x))_+\right)\right)_+ + \sum_x W_r(x)\ell(o(x), r(x)). \tag{7}$$

Base Loss and *Global Loss* are weighted variants of the standard classification loss, biased by the appropriate weights to emphasise the regions assigned to either model by the routing network - $W_b(x) = 2 - r(x)$ and $W_g(x) = 1 + r(x)$.

$$\mathcal{L}_{\text{base}}(b; r, g) = \sum W_b(x)\ell(y, b(x)); \qquad \mathcal{L}_{\text{global}}(b; r, g) = \sum W_g(x)\ell(y, g(x)).$$

3 EXPERIMENTS

In this section, first, we train hybrid models for resource-constrained MCU devices, thus demonstrating the effectiveness of hybrid training. Next, we show that hybrid models can be adapted to resource-rich edge devices such as mobile phones. Next, we probe various aspects of our framework through ablations, including (A) validation on other datasets, (B) sensitivity of the solution to small communication latencies, and (C) effectiveness as an abstaining classifier for situations when a global model may be unavailable. Finally, we discuss a simple joint architecture search method for finding hybrid architectures with better performance than off-the-shelf architectures (see Appendix A.3, A.4 for details).

Experimental Setup. We focus on the large-scale ImageNet dataset(Russakovsky et al., 2015), consisting of 1.28M train and 50K validation images. We follow standard data augmentation (mirroring, resize and crop) for training and single crop for testing. We borrow the pre-trained baselines from their public implementations (see Sec. A.6.2). Appendix A.6.1 lists our hyperparameters settings.

Throughout, we use the largest model in the OFA space (Cai et al., 2020) as our global model and henceforth refer to it as MASS-600M (which needs 600M MACs for evaluation). This model has an accuracy of 80%, and a computational latency of about 25ms on a V100 GPU. There are three main reasons for using this model. Firstly, this model is SOTA at its required computational budget. Secondly, this model is small enough for training to be tractable under our computational limitations (unlike larger models such as EfficientNet-B7, which needs 37B MACs). Finally, since it is the largest model in our architecture search space, it yields a consistent point of comparison across experiments.

To expose gains from the different hybrid components we consider: (a) 'Hybrid-(r)' - training routing models with a pre-trained base and global model, (b) 'Hybrid-(rb)' - training routing and base models with a pre-trained global model, and (c) 'Hybrid-(rbg)' - training all three components. Unless explicitly stated, we create models by training routing and base, i.e., 'Hybrid-rb' since training the global model is computationally expensive. We train hybrid models to achieve a similar coverage as the oracle. Post-training, we tune the threshold t to generate r with varying coverage (Alg. 4).

Baseline Algorithms. We investigated prior methods (Kang et al., 2017; Nan & Saligrama, 2017; Bolukbasi et al., 2017; Li et al., 2021; Teerapittayanon et al., 2017), and the entropy thresholding, which we report here, dominates these methods across all datasets. This fact has also been observed in other works (Gangrade et al., 2021; Geifman & El-Yaniv, 2019). It is not surprising since they lack proxy supervision (2nd term in Eq. 7), and the routing model does not get instance-level supervision.

Coverage as Proxy. As discussed in Sec. 2, for simplicity, we use coverage as the proxy for communication latency, the major contributor to the inference latency. We explicitly model latency in Sec. 3.1, where we define the latency of the hybrid system as the sum of three latencies: (a) on-device inference, (b) communication for the examples sent to the cloud, and (c) on-cloud inference.

3.1 Hybrid Models for Resource-Limited Edge Devices

In this section, we train hybrid models with off-the-shelf architectures using Algorithm 1. First, we delve into the illustrative experiment in Figure 2. Next, we study a similar problem for a resource-rich edge device, specifically mobile phones, and train hybrid models at various base complexities.

Resource Constrained MCU. *Proposed hybrid method with MCUs for ImageNet task is near optimal (close to upper bound); realizes 25% latency/energy reduction while maintaining cloud performance, and at accuracy loss of less than 2% realizes 30% latency/energy reduction.*
Recall from Sec. 1 that we deployed an ImageNet classifier on a tiny STM32F746 MCU (320KB SRAM and 1MB Flash), the same setting as MCUNets (Lin et al., 2020). Using their TFLite model (12.79M MACs, 197ms latency, 51.1% accuracy) as the base, we create a hybrid model by adding the MASS-600 as the global model. We provide additional setup details in the Appendix §A.2.

Figure 2 shows the accuracy vs latency and energy. Table 3 shows the latency and energy of the hybrid approach against baselines. Deploying hybrid model on an MCU results in following benefits:

- *Pareto Dominance over on-device model.* Hybrid model provides 10% accuracy gains over the best on-device model with similar latency. It achieves the best on-device accuracy with half the latency.
- *Pareto Dominance over other baselines.* Hybrid model achieves 5% higher accuracy than the dynamic networks that use the entropy thresholding baseline (see Figure 2). In passing we recall from our earlier discussion that entropy thresholding dominates prior methods in this context.
- *Significant latency reduction to achieve SOTA accuracy.* The hybrid model achieves near SoTA accuracy with 25% reduction in latency and energy consumption.
- *Micro-controller Implementation.* We deployed base and router on the MCU with negligible ($\sim 2\%$) slowdown (see Appendix A.10 for details).

Although our setup in Figure 2(a) assumes a constant communication latency of 2000ms, we can easily modify it to incorporate dynamic communication latency shown in Figure 2(b). Due to lack of space, we refer the reader to Appendix A.13 for the exact formulation.

Resource Constrained Mobile Device. *Gains of hybrid method are consistent, near-optimal, and dominate prior methods, across diverse device specifications, including large footprint mobile devices.*
To show this, we choose the popular MBV3 (Howard et al., 2019) architectures as the base since they have been heavily deployed on mobile devices with resource constraints such as storage and compute. We create hybrid models using the following base models in the MBV3 family: MBV3-48 (2.54M params, 48M MACs), MBV3-143 (4M params, 143M MACs), and MBV3-215 (5.48M params, 215M MACs). We use MASS-600 as the global model and operate the base model at a fixed coverage level.

Table 3: Hybrid models on STM32F746 MCU: Accuracy achieved by different methods at various latency.

	Accuracy (%) v/s Resource Usage					
Latency (ms)	200	600	1000	1400	1600	2000
Energy (mJ)	11	53	96	140	161	216
On-Cloud	-	-	-	-	-	79.9
On-Device	51.1	-	60.9	63.5	-	-
Entropy	-	59.9	67.4	74.7	76.93	-
Hybrid-r	-	61.1	69.5	76.3	78.68	-
Hybrid-rb	-	62.0	70.8	77.7	79.3	-
Hybrid-rbg	-	62.3	71.2	77.9	79.5	-
Upper-Bound	-	62.8	74.7	79.9	79.9	-

Table 4 shows the hybrid accuracy at three coverage levels: 90%, 80% and 70%. Hybrid models operating at a fixed coverage provide the following benefits:

Table 4: Results for hybrid models with base at various coverages. MASS-600 model achieving $\approx 80\%$ Top1 accuracy is used as global model. Base model belongs to MBV3 space. Upper bounds (Appendix §A.2) are also reported and nearly match hybrid.

Base MACs	Base Acc.(%)	Method	Acc. (%) at Cov.		
			90%	80%	70%
48M	67.6	Entropy	70.7	73.3	74.9
		Hybrid	71.6	74.6	76.8
		Upper Bnd	72.0	75.2	78.9
143M	73.3	Entropy	75.1	76.8	77.6
		Hybrid	75.9	77.8	79.0
		Upper Bnd	75.9	78.2	79.9
215M	75.7	Entropy	77.1	78.3	78.9
		Hybrid	77.6	79.0	79.6
		Upper Bnd	77.6	79.1	79.9

Table 5: Joint evolutionary search for hybrid models base constraints: 75M, 150M, & 225M. Table shows hybrid and base accuracies at different coverages. Upper bounds are reported in Appendix §A.2. Excess gains represent improved neural-network architecture.

Base MACs	Base Acc.(%)	Method	Acc. (%) at Cov.		
			90%	80%	70%
74M	70.8	Entropy	73.2	75.3	76.8
		Hybrid	74.0	76.4	78.2
		Upper Bnd	73.9	77	79.9
149M	74.5	Entropy	76.1	77.6	78.2
		Hybrid	76.9	78.5	79.4
		Upper Bnd	76.6	78.7	79.9
225M	76.5	Entropy	77.4	78.3	79.1
		Hybrid	78.3	79.6	80.2
		Upper Bnd	77.8	79.4	79.9

- *Up to* 70% *latency reduction to achieve SOTA.* Hybrid model with MBV3-215M base achieves 79.59% (near SOTA) with 70% coverage. It communicates with global model for only 30% inputs.
- *Hybrid models outperform the entropy thresholding.* Hybrid model with MBV3-48M achieves nearly 2% higher accuracy than entropy thresholding baseline.
- *Hybrid model with a smaller base model achieves performance of larger models at various coverage levels.* Using MBV3-48M base, the hybrid model exceeds the accuracy of MBV3-143M at 80% coverage. Similarly, it exceeds the accuracy of MBV3-215M at 70% coverage.

3.2 Ablative Experiments.

We refer the reader to Appendix A.11 for setup and results. Below we summarize our findings.

(A) **Other Datasets.** We train hybrid models on CIFAR-100 (Krizhevsky, 2009) and IMDb (Maas et al., 2011) (reported in Appendix), and observe similar gains.

CIFAR-100. We use the efficient version of the Resnet-32 (He et al., 2016) from (Dong & Yang, 2019) as the global model (35M MACs, 0.5M params, 67.31% accuracy). Base is a tiny pruned Resnet model (2.59M MACs, 0.02M params, 52.56% accuracy). Table 6 shows the performance of the hybrid models and the entropy thresholding baseline at various coverage levels. It shows that hybrid models provide similar benefits on CIFAR-100 dataset.

(B) **Other communication regimes.** We train hybrid models in a setup where there is no communication delay. This can practically arise when both base and global model reside on same hardware. In such a setup, we use a simpler evaluation metric for inference latency, i.e., hybrid MACs, i.e., the amount of multiply-add operations required to execute the hybrid model. We pick up an architecture family and create a hybrid model using the smallest and largest architecture. For convenience, we perform this experiment for a known family, namely MobileNetV3 (MBV3) (Howard et al., 2019). From MBV3, we pick the smallest model (48M MACs, 67.6% accuracy) as the base and largest model (215M MACs, 75.7% accuracy) as global to create the Hybrid-MBV3 model. Figure 3 shows the hybrid model performance as well as entropy thresholding baseline. It shows that hybrid model achieves SOTA accuracy at 30% lower latency.

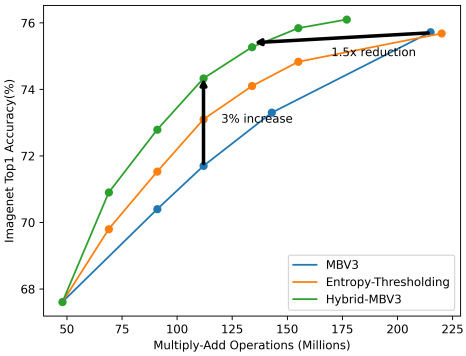


Figure 3: Plot for hybrid MACs vs accuracy. Base & Global models on same device.

(C) **Abstaining classifier.** When a global model is unavailable, the hybrid system can serve as an abstaining classifier. In this case, the result is an abstaining classifier on the device, i.e., it rejects a

few inputs and provides predictions on the rest. We create abstaining classifiers with the hybrid setup in the Sec. 3.1. We show the accuracy of the base model at various coverage levels in the Table 7. It shows that the abstaining base significantly outperforms the base at full coverage and the abstaining classifier from the entropy thresholding baseline, which demonstrates that our method is competitive against prior works on classification with abstention.

Table 6: Hybrid models for CIFAR-100 at various coverages.

Base MACs	Base Acc.(%)	Accuracy(%) at Cov.		
		80%	70%	40%
Entropy	52.56	57.77	59.95	65.1
Hybrid-r	52.56	58.43	61.18	66.9
Hybrid-rb	52.56	59.32	62.48	67.4

Table 7: Abstaining Classifier with hybrid models from Sec. 3.1. Results for hybrid models with base at various coverage levels.

Base MACs	Base Acc.(%)	Accuracy(%) at Cov.					
		90%		80%		70%	
		Base	Entropy	Base	Entropy	Base	Entropy
48M	67.6	73.3	71.7	78.6	75.6	83.4	80.5
143M	73.3	79.0	77.6	83.9	81.7	88.4	85.4
215M	75.7	81.3	78.6	86.1	82.0	90.1	86.6

(D) **Router validation.** We evaluate the performance of the router against the oracle supervision and show that the router learnt using the Algorithm 1 generalizes well on the test dataset.

3.3 Joint Neural Architecture Search for Hybrid Models.

This section proposes a joint neural architecture search (NAS) scheme to resolve the outer $\max$ problem of (3). NAS methods are strongly dependent on two aspects

Architecture Search Space. Our method is designed for implementation on a *Marked Architecture Search Space* (MASS). It is a set of architectures $\mathscr{A}$ such that each architecture $\alpha \in \mathscr{A}$ is associated with a known set of canonical parameters θ_α, which are known to be representative of the models under this architecture. That is, for most tasks, fine-tuning θ_α returns a good model with this architecture. Such search spaces have been proposed by, for instance, Cai et al. (2020) and Yu et al. (2019). We use Cai et al. (2020) as search space as it contains a range of architectures of varying capacities.

Proxy Score Function. Given a pair of base and global architectures (α_b, α_g), search process requires access to the value of the program (4). The score function estimates the accuracy of the hybrid model under a given resource constraint. Since training a hybrid model using these architectures would be slow, we use the proposed oracle supervision as proxy for the router. Thus, the routing oracle $o(\cdot, \alpha_b, \alpha_g)$ in conjunction with the canonical parameters $(\theta_{\alpha_b}, \theta_{\alpha_g})$ serves as an efficient proxy score function for evaluating the architecture pair (α_b, α_g).

Search Algorithm. Given these above two components, NAS problems can be reduced to a combinatorial optimization and approached by any standard heuristic. Due to its simplicity and prevalence, we propose using an simple evolutionary algorithm for this (e.g Elsken et al., 2019; Liu et al., 2021) to yield a concrete method. Algorithm 2 summarizes the search scheme. We describe these aspects in full detail in Appendix A.3.

Empirical Validation. We evaluate the proposed architecture search scheme against the off-the-shelf architectures in Table 5 as well as Appendix A.4. It shows that evolutionary search yields higher accuracy hybrid models than off-the-shelf classifiers.

4 Conclusion

We proposed a novel hybrid design where an edge-device selectively queries the cloud only on those hard instances that the cloud can classify correctly to optimize accuracy under latency and edge-device constraints. We propose an end-to-end method to train neural architectures, base predictors, and routing models. We use novel proxy supervision for training routing models. We show that our method adapts seamlessly and near optimally across different latency regimes. Empirically, on the large-scale ImageNet classification dataset, our proposed method, deployed on an MCU, exhibits a 25% reduction in latency compared to cloud-only processing while suffering no excess classification loss.

REFERENCES

Tolga Bolukbasi, Joseph Wang, Ofer Dekel, and Venkatesh Saligrama. Adaptive neural networks for efficient inference. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 527–536. JMLR. org, 2017.

Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once for all: Train one network and specialize it for efficient deployment. In *International Conference on Learning Representations*, 2020. URL https://arxiv.org/pdf/1908.09791.pdf.

Min Chen, Yiming Miao, Yixue Hao, and Kai Hwang. Narrow band internet of things. *IEEE Access*, 5:20557–20577, 2017. doi: 10.1109/ACCESS.2017.2751586.

Xin Dai, Xiangnan Kong, and Tian Guo. *EPNet: Learning to Exit with Flexible Multi-Branch Network*, pp. 235–244. Association for Computing Machinery, New York, NY, USA, 2020. ISBN 9781450368599. URL https://doi.org/10.1145/3340531.3411973.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018a. URL http://arxiv.org/abs/1810.04805.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018b.

Terrance DeVries and Graham W Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017.

Xuanyi Dong and Yi Yang. Network pruning via transformable architecture search. In *Neural Information Processing Systems (NeurIPS)*, 2019.

Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *The Journal of Machine Learning Research*, 20(1):1997–2017, 2019.

Aditya Gangrade, Anil Kag, and Venkatesh Saligrama. Selective classification via one-sided prediction. In Arindam Banerjee and Kenji Fukumizu (eds.), *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, volume 130 of *Proceedings of Machine Learning Research*, pp. 2179–2187. PMLR, 13–15 Apr 2021. URL https://proceedings.mlr.press/v130/gangrade21a.html.

Yonatan Geifman and Ran El-Yaniv. Selectivenet: A deep neural network with an integrated reject option. In *International Conference on Machine Learning*, pp. 2151–2159, 2019.

Yizeng Han, Gao Huang, Shiji Song, Le Yang, Honghui Wang, and Yulin Wang. Dynamic neural networks: A survey, 2021.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016. doi: 10.1109/CVPR.2016.90.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. Searching for mobilenetv3. *CoRR*, abs/1905.02244, 2019. URL http://arxiv.org/abs/1905.02244.

Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor Mudge, Jason Mars, and Lingjia Tang. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. *SIGPLAN Not.*, 52(4):615–629, apr 2017. ISSN 0362-1340. doi: 10.1145/3093336.3037698. URL https://doi.org/10.1145/3093336.3037698.

Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, ., 2009.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. ALBERT: A lite BERT for self-supervised learning of language representations. *CoRR*, abs/1909.11942, 2019. URL http://arxiv.org/abs/1909.11942.

Hao Li, Hong Zhang, Xiaojuan Qi, Ruigang Yang, and Gao Huang. Improved techniques for training adaptive deep networks, 2019.

Min Li, Yu Li, Ye Tian, Li Jiang, and Qiang Xu. Appealnet: An efficient and highly-accurate edge/cloud collaborative architecture for dnn inference, 2021.

Ji Lin, Wei-Ming Chen, Yujun Lin, John Cohn, Chuang Gan, and Song Han. Mcunet: Tiny deep learning on iot devices. *arXiv preprint arXiv:2007.10319*, 2020.

Ji Lin, Wei-Ming Chen, Han Cai, Chuang Gan, and Song Han. Mcunetv2: Memory-efficient patch-based inference for tiny deep learning. In *Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

Yuqiao Liu, Yanan Sun, Bing Xue, Mengjie Zhang, Gary G Yen, and Kay Chen Tan. A survey on evolutionary neural architecture search. *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

Ziyin Liu, Zhikang Wang, Paul Pu Liang, Russ R Salakhutdinov, Louis-Philippe Morency, and Masahito Ueda. Deep gamblers: Learning to abstain with portfolio theory. In *Advances in Neural Information Processing Systems*, pp. 10622–10632, 2019.

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pp. 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P11-1015.

Feng Nan and Venkatesh Saligrama. Adaptive classification for prediction under a budget. In *Advances in Neural Information Processing Systems*, pp. 4727–4737, 2017.

Mohanad Odema, Nafiul Rashid, Berken Utku Demirel, and Mohammad Abdullah Al Faruque. Lens: Layer distribution enabled neural architecture search in edge-cloud hierarchies. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pp. 403–408, 2021. doi: 10.1109/DAC18074.2021.9586259.

Eunhyeok Park, Dongyoung Kim, Soobeom Kim, Yong-Deok Kim, Gunhee Kim, Sungroh Yoon, and Sungjoo Yoo. Big/little deep neural network for ultra low power inference. In *2015 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pp. 124–132, 2015. doi: 10.1109/CODESISSS.2015.7331375.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115 (3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

Mingxing Tan and Quoc Le. EfficientNet: Rethinking model scaling for convolutional neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 6105–6114, Long Beach, California, USA, 09–15 Jun 2019. PMLR. URL http://proceedings.mlr.press/v97/tan19a.html.

Surat Teerapittayanon, Bradley McDanel, and H. T. Kung. Branchynet: Fast inference via early exiting from deep neural networks, 2017.

Xin Wang, Yujia Luo, Daniel Crankshaw, Alexey Tumanov, Fisher Yu, and Joseph E. Gonzalez. Idk cascades: Fast deep learning by learning not to overthink, 2018.

Colin White, Willie Neiswanger, and Yash Savani. Bananas: Bayesian optimization with neural architectures for neural architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang. Slimmable neural networks. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=H1gMCsAqY7.