Achieving High TinyML Accuracy through Selective Cloud Interactions

Anil Kag [1] **Igor Fedorov** [2] **Aditya Gangrade** [3] **Paul Whatmough** [2] **Venkatesh Saligrama** [1]

Abstract

Edge devices provide inference on predictive tasks to many end-users. However, deploying neural networks that achieve state-of-the-art accuracy on edge is infeasible due to resource constraints. Nevertheless, cloud-only processing 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 classifies correctly. It trains edge, cloud predictors, and routing to maximize accuracy while minimizing the 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 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), and machine translation (Wu et al., 2016). However, SOTA performance is only achievable on the cloud since edge devices impose severe constraints on model storage, energy consumption, and inference latency. Thus, on-device solutions lead to significant performance degradation. Nevertheless, cloud-only processing is also problematic. Uploading large amounts of data through wireless services imposes bottlenecks due to communication latency and significant battery utilization.

Our proposed hybrid method, illustrated schematically in Fig. 1, consists of an edge device equipped with a base predictor, b, and a router, r, which selectively queries the cloud-based global model, g, on examples that are hard to predict with the edge-devices capabilities. Our objective is to *maximize base (edge) utilization while achieving global (cloud) performance.* To build intuition on the novel aspects of our approach, let us examine the problem of large-scale classification on a Micro-Controller Unit (MCU).

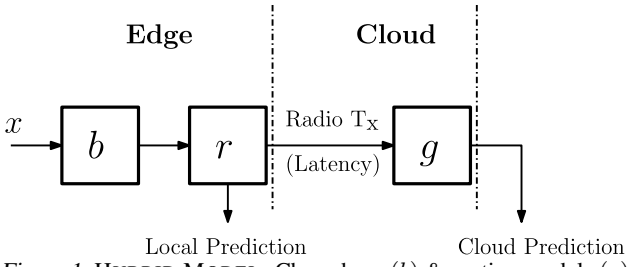


Figure 1. HYBRID MODEL. Cheap base (b) & routing models (r) run on a micro-controller; Expensive global model (g) runs on a cloud. r uses x and features of b to decide if g is evaluated or not.

ImageNet-Scale 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 is 10x faster than an MCU. As a result, to reduce processing latency, one should transfer all the data to the cloud. However, data transfer to the cloud takes about $2s$—about $100\times$ the processing time on a GPU—and thus, 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.

Hybrid Learning Objective. To minimize latency at a target accuracy, *a low-complexity edge must learn not only to predict hard-to-locally-classify examples, but also to predict whether the cloud model would misclassify the example*. Equivalently stated as a coverage (fraction of local predictions) maximization problem under accuracy constraint.

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

[1]Boston University, USA [2]ARM Inc. [3]Carnegie Mellon University, USA. Correspondence to: Anil Kag <anilkag@bu.edu>.

DyNN workshop at the 39^{th} International Conference on Machine Learning, Baltimore, Maryland, USA, 2022.

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	✓	✓	✓	✓	✓

Efficient Edge Solutions. Prior works on efficient architectures under latency constraints (Cai et al., 2020; Lin et al., 2020; Howard et al., 2019) seek edge-only solutions that fit into an MCU. While these methods are complementary to our work, we point out that the accuracy of our model on the MCU is about 50%. This is so low that regardless of which of these methods is used, it is unlikely to bridge the large accuracy gap. Thus hybrid solutions are required.

Split-Computation and Early Exit Methods. Systems researchers (Kang et al., 2017; Odema et al., 2021) proposed storing and executing a part of the large cloud-DNN on edge, and when possible predict locally (early exit), thereby reducing latency. Since MCUs are too small, only a tiny part of the large DNN can be stored/executed locally. Besides, initial DNN features are large, thus providing little communication gains over image transmission. In addition, the low-early-exit accuracy requires a large number of cloud queries (see §A.6 for details). The resulting trade-off between accuracy and latency is too weak to serve as a baseline (see Fig. 2).

Dynamic Networks. (Park et al., 2015; Bolukbasi et al., 2017; Teerapittayanon et al., 2017) designed dynamic DNNs with more budget for hard instances, disregarding the severely resource-constrained setting. In contrast, we overwhelmingly reduce resource usage to deploy models on edge with cloud accuracy. It necessitates an end-to-end optimization of routing, base, and global. Prior works optimize these in a decoupled manner. For instance, dynamic architectures use entropy thresholding for routing. Such a router is locally myopic as it does not account for cloud predictions. Fig. 2 shows that a carefully designed router outperforms dynamic networks. See Appendix §A.5 for detailed comparison.

1.1. Proposed Hybrid Solution

We propose a novel end-to-end framework for learning hybrid models that incorporate edge-device specifications, cloud, and latency constraints. Table 2 compares hybrid design with prior works along various dimensions.

Novel Proxy Supervision. Training routing models is difficult because we do not apriori know hard-to-classify edge-examples. More importantly, we only benefit from sending hard examples that the cloud correctly predicts. Consider an instance-label pair (x, y) with three typical possibilities:

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

Our goal 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. As such, the routing would benefit from supervision, and we introduce a novel *proxy supervision* to learn routing models while accounting for the base and global predictions.

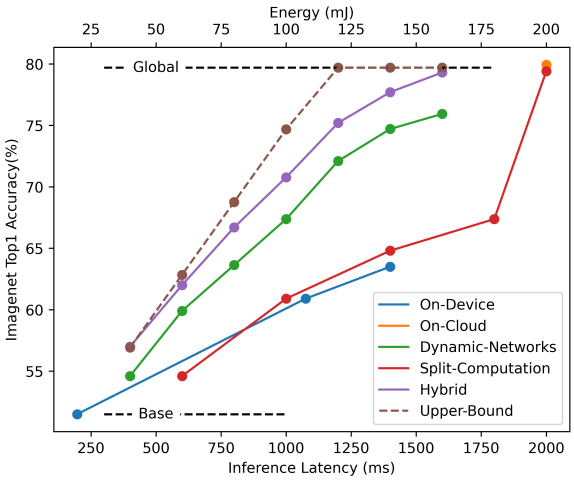


Figure 2. Image recognition on the ImagetNet dataset: Accuracy vs Energy and Latency plot. This clearly shows that the hybrid design pareto dominates on-device as well as other baselines while getting significantly closer to the upper-bound in hybrid design.

Highlights of Proposed Hybrid Approach. Fig. 2 displays the performance of various strategies for the situation in Table 1. Here the purple 'Hybrid' line depicts the accuracies achievable by our proposal. The green line shows hybrid accuracies achieved by dynamic neural networks (Bolukbasi et al., 2017; Nan & Saligrama, 2017; Li et al., 2021), including base-model entropy or margin-based thresholding techniques. We attribute our gains to router proxy supervision and end-to-end hybrid system training. Additionally, improvements in on-device accuracy are possible (blue), but with relatively small gains, and naïve split computation (red) have poor performance due to edge-device limitations. Finally, we derive an upper bound (Appendix §A.2) on attainable accuracy, as indicated by the dashed brown line.

Fig.2 shows that hybrid model mirrors the upper bound and achieves global accuracy with a 25% reduction in latency.

Contributions

- We propose a hybrid framework wherein an edge, to optimize accuracy at user-specified latency, selectively queries only hard examples the cloud can correctly classify.
- We propose novel proxy supervision to train router. Our method adapts near optimally across different latencies.
- Preliminary experiments show that the hybrid design reduces inference latency and energy consumption.
- We provide pseudo-code for training hybrid models.

2. Method

Notation. Let $\mathcal{X}$ be a feature space and $\mathcal{Y}$ a set of labels. A hybrid design is composed of the following three models:

- 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.

These are soft classifiers, outputting $|\mathcal{Y}|$-dimensional scores $\{b_y\}$ and $\{g_y\}$, and two scores r_0 and r_1 for router. Hard output for the base is $b(x) = \arg\max_y b_y(x)$, and similarly for g. r is a 2-layer DNN with input $b_y(x)$ and has a scalar tuning parameter t, and assigns x to global if $r_1 > r_0 + t$, i.e.

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

Varying t (Alg. 2) 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 $\hat{y}(x) := (1-r(x))b(x)+r(x)g(x)$.

Evaluation Metrics. We refer to the overall accuracy of the hybrid predictions as *hybrid accuracy*, defined as

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

This accuracy is traded-off with the *coverage* of hybrid models, which is the fraction of instances processed by the cheap base model only, i.e. $\mathcal{C}(r, b, g) := \mathbb{P}(r(X) = 0)$.

Modeling Resource Usage. Let α denote an architecture, and say that $f \in \alpha$ if the function f is realizable by α. Resource cost of f is denoted $\mathcal{R}(\alpha)$. In our hybrid design, the base and router are always executed. So the mean resource usage of the hybrid model (r, b, g) with $b \in \alpha_b$ and $g \in \alpha_g$ is

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

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

Eq. 1 can model resources such as energy or total FLOPs. To model *inference latency*, we take $\mathcal{R}(\alpha_b)$ to be the base inference latency on edge, and $\mathcal{R}(\alpha_g)$ to be the *sum of* the computational latency of g on the cloud and the communication latency of sending examples to the cloud. In Table 1, these numbers would be 200ms and 2025ms respectively. Thus, resource constraint reduces to a coverage constraint.

Algorithm 1 Training Hybrid Models

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

2.1. Learning Hybrid Models

Learning objective to train hybrid models with base and global architectures α_b, α_g, with C_ϱ coverage level is

$$\max_{r, b\in\alpha_b, g\in\alpha_g} \mathcal{A}(r, b, g) \quad \text{s.t.} \quad \mathcal{C}(r, b, g) \le C_\varrho. \quad (2)$$

We solve (2) via an ERM. We discuss several design issues below and summarise the overall scheme in Algorithm 1.

Alternating optimisation. Problem (2) 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 g and b 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.

Learning Routers via Proxy Supervision. Given a fixed pair of base and global (b, g), problem (2) reduces to

$$\begin{aligned}\max_r \mathbb{E}[(1 - r(X))\mathbb{1}\{b(X) = Y\} + r(X)\mathbb{1}\{g(X) = Y\}] \\ \text{s.t.} \quad \mathbb{E}[r(X)] \le C_\varrho, \end{aligned} \quad (3)$$

While a naïve idea is to relax r and pursue ERM, we instead reformulate the problem. Observe that (3) 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, coverage constraint promotes $r(X) = 0$. Thus, the program can be viewed as a supervised learning to fit the *routing oracle*, i.e.

$$o(x; b, g) = \mathbb{1}\{b(x) \neq g(x) = y\}. \quad (4)$$

Indeed, o is the ideal routing without the coverage constraint. Given (b, g) and dataset $\mathcal{D} = \{(x^i, y^i)\}$, we produce the oracle dataset $\mathcal{D}_{o;(b,g)} := \{(x^i, o(x^i; b, g))\}$ to supervise the router r. This allows us to use the standard learning tools for practically learning good binary functions, thus gaining over approaches that directly relax the objective of (3). Note that the oracle o does not respect the coverage constraint. We handle this issue indirectly by imposing a coverage penalty.

Focusing competency and loss functions. We bias b to be more accurate on the region $r^{-1}(\{0\})$, and g on $r^{-1}(\{1\})$. Similarly, for the router r, it is more important to match $o(x) = 1$ region, not captured accurately by the base.

Routing Loss consists of two terms, traded off by a hyper-parameter λ_r. The first penalises coverage deviation 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 $o^{-1}(\{1\})$. Let ℓ be cross entropy loss and $(\cdot)_+$ be the ReLU function.

$$\begin{aligned}\mathcal{L}_{\text{routing}}(r; o) = \lambda_r \left(\text{cov} - \left(1 - \frac{1}{N}\sum_x (r_1(x) - r_0(x))_+\right)\right)_+ \\ + \sum_x W_r(x)\ell(o(x), r(x)). \end{aligned} \quad (5)$$

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 $W_g(x) = 1 + r(x)$. $\mathcal{L}_{\text{base}}(x) = \sum W_b(x)\ell(y, b(x))$; $\mathcal{L}_{\text{global}}(x) = \sum W_g(x)\ell(y, g(x))$.

3. Hybrid Models for Resource-Limited Edge

In this section, we train hybrid models using the Algorithm 1. Empirical evaluations demonstrate that the hybrid models achieve near SOTA accuracy with $30 - 70\%$ less communication. In appendix, we describe ablations.

Experimental Setup. We focus on the ImageNet (Russakovsky et al., 2015) dataset. We use publicly available pre-trained baselines. See §A.1, §A.4.2, §A.4.1 for details.

We use the largest model of the (Cai et al., 2020) as the global, and refer 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.

Unless explicitly stated, we create hybrid models by training routing and base as training the global model is a computationally expensive exercise. We train hybrid models to achieve oracle labeling coverage. Post-training, we tune the threshold t to generate r with varying coverage (Alg. 2).

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

	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	-	62.3	71.2	77.9	79.5	-
Upper-Bound	-	62.8	74.7	79.9	79.9	-

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 these prior works do not include proxy supervision (2nd term in Eq. 5), and the routing model does not get instance-level supervision.

Resource constrained MCU. Recall from Sec. 1 that we deployed an ImageNet classifier on a tiny STM32F746 MCU (320KB SRAM, 1MB Flash), the same setting as MCUNets (Lin et al., 2020). Using their TFLite model (12.79M MACs, 197ms latency, 51.1% accuracy) as base, we create a hybrid model by adding MASS-600 as the global model. We provide additional details in the Appendix §A.2.

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

- *Pareto Dominance over on-device.* Hybrid model gains 10% accuracy over the 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.
- *MCU Implementation.* We deployed base and router on the MCU with negligible ($\sim 2\%$) slowdown (see §A.7).

Table 4. Results for hybrid models with base at various coverage levels. MASS-600 model achieving $\approx 80\%$ Top1 accuracy is used as global model. Base model belongs to MBV3 space.

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
143M	73.3	Entropy	75.1	76.8	77.6
		Hybrid	75.9	77.8	79.0
215M	75.7	Entropy	77.1	78.3	78.9
		Hybrid	77.6	79.0	79.6

Resource Constrained Mobile Device. To show that hybrid model benefits extend beyond the MCUs, we trained hybrid models for various mobile device constraints. 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 and operate the base at a fixed coverage level.

Table 4 shows the hybrid accuracy at three coverage levels: 90%, 80% and 70%. Hybrid models result in:

- *Up to 70% latency reduction to achieve SOTA accuracy.* Hybrid model with MBV3-215M base achieves 79.59% (near SOTA) accuracy with 70% coverage.
- *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 achieves performance of larger models at various coverages.* 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.

References

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

Cai, H., Gan, C., Wang, T., Zhang, Z., and Han, S. 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.

Chen, M., Miao, Y., Hao, Y., and Hwang, K. Narrow band internet of things. *IEEE Access*, 5:20557–20577, 2017. doi: 10.1109/ACCESS.2017.2751586.

Devlin, J., Chang, M., Lee, K., and Toutanova, K. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018. URL http://arxiv.org/abs/1810.04805.

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

Gangrade, A., Kag, A., and Saligrama, V. Selective classification via one-sided prediction. In Banerjee, A. and Fukumizu, K. (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.

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

He, K., Zhang, X., Ren, S., and Sun, J. 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.

Howard, A., Sandler, M., Chu, G., Chen, L., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., Le, Q. V., and Adam, H. Searching for mobilenetv3. *CoRR*, abs/1905.02244, 2019. URL http://arxiv.org/abs/1905.02244.

Huang, G., Liu, Z., van der Maaten, L., and Weinberger, K. Q. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

Kang, Y., Hauswald, J., Gao, C., Rovinski, A., Mudge, T., Mars, J., and Tang, L. 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.

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

Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., and Soricut, R. ALBERT: A lite BERT for self-supervised learning of language representations. *CoRR*, abs/1909.11942, 2019. URL http://arxiv.org/abs/1909.11942.

Li, M., Li, Y., Tian, Y., Jiang, L., and Xu, Q. Appealnet: An efficient and highly-accurate edge/cloud collaborative architecture for dnn inference, 2021.

Lin, J., Chen, W.-M., Lin, Y., Cohn, J., Gan, C., and Han, S. Mcunet: Tiny deep learning on iot devices. *arXiv preprint arXiv:2007.10319*, 2020.

Maas, A. L., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., and Potts, C. 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.

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

Odema, M., Rashid, N., Demirel, B. U., and Faruque, M. A. A. 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.

Park, E., Kim, D., Kim, S., Kim, Y.-D., Kim, G., Yoon, S., and Yoo, S. 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.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

Tan, M. and Le, Q. EfficientNet: Rethinking model scaling for convolutional neural networks. In Chaudhuri, K. and Salakhutdinov, R. (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.

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

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

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

A. Appendix

A.1. Dataset Details

Imagenet (Russakovsky et al., 2015). This is the popular image classification dataset consisting of 1000 classes. It has nearly 1.28M training and 150K validation images. A typical image in this dataset has $224 \times 224 \times 3$ pixels. We follow standard data augmentation (mirroring, resize and crop to shape 224×224) for training and single crop for testing. Following earlier works (Howard et al., 2019; Cai et al., 2020), we report the results on the validation set.

CIFAR-100 (Krizhevsky, 2009). This is a 100 way image classification dataset consisting of images with $32 \times 32 \times 3$ pixels. It has 50K train and 10K test images. We follow standard data augmentation (standard gaussian normalization/mirroring/shifting) used in earlier works (He et al., 2016; Huang et al., 2017). In addition, we use the cut-out (DeVries & Taylor, 2017) with the standard window size of 8.

IMDb (Maas et al., 2011). This is a sentiment classification dataset with two classes (positive and negative). It consists of raw text of movie reviews. There are 25K train and 25K test data points. We borrow text parsers and tokenizers from the models available in the HuggingFace[1] library.

A.2. Illustrative Example Details

Edge and Cloud devices. We use a V100 GPU as the cloud device. It has a 16GB VRAM and the server associated with this GPU has 1TB storage. We use STM32F746 [2], an ARM Cortex-M7 MCU as the edge device. It has 320KB SRAM and 1MB Flash storage.

On-Cloud Baseline. We use the MASS-600 (Cai et al., 2020) as the global model in this experiment. It achieves 79.9% Top1 accuracy on the Imagenet dataset. This model has a computational footprint of 595M MACs. As per our benchmarking, the inference latency of this model on a V100 GPU is 25ms.

On-Device Baselines. (Lin et al., 2020) have explored deploying tiny MCUNet models on the STM32F746 MCU. We borrowed their pre-trained MCUNet models from their github repository [3] [4]. There are three different models with varying inference latencies and accuracies, namely: (a) model-A (12M MACs, 0.6M parameters, 200ms latency, 51.1% accuracy), (b) model-B (81M MACs, 0.74M parameters, 1075ms latency, 60.9% accuracy), and (c) model-C (170M MACs, 1.4M parameters, 1400ms latency, 63.5% accuracy). We use the smalled model, i.e., model-A as the base model in training the hybrid model. This model has the least latency as well as the least accuracy among the three models. Note that this model has the input resolution of $96 \times 96 \times 3$ and as a result the input size becomes nearly 28KB.

Energy Profile Base Model Execution and Communication Cost. We assume that the MCU device operates at 200MHz clock speed and is connected to a 3.6V power supply. It has an active mode current characteristics of 72μA per MHz. Thus, it consumes $= 3.6 \times 72 \times 200 = 51.84$ mW (i.e. mili Joules per second) energy in active mode. As a result, the energy consumed by the base model (200ms latency) on this edge device is $= 51.84 \times 0.2 = 10.368$ mili Joules. Similar calculation yield the energy consumption the on-device models: model-B (60mJ) and model-C (80mJ).

We use the NB-IoT (Chen et al., 2017) communication protocol with 110kbps transmission rate and a transmission current characteristics of 30mA. Thus, it consumes $= 30 \times 3.6 = 108$ mili Joules per second in transmission mode. Let us calculate the time taken to transfer the input image from the base model on the edge device to the global model on cloud. The base model input has size 28KB and the transmission rate is 110kbps. Thus, it takes $= \frac{28*8}{110} \approx 2$ seconds to transfer this image from the edge device to the cloud. As a result, the energy consumed by the edge device in transmitting this image is $= 108 * 2 = 216$mJ.

Split-Computation Baseline. We use this term to refer methods (Kang et al., 2017; Teerapittayanon et al., 2017) wherein the initial part of the global model executes on-device while the remaining executes on-cloud. The initial network allows for an early exit classifier for easy examples. Split into early layers results in high communication cost but split in later layers results in higher base computation latency, making it ineffective in the edge-cloud setup.

We trained the BranchyNet (Teerapittayanon et al., 2017) method to represent this baseline. We create an exit classifier after

[1] https://huggingface.co

[2] https://www.st.com/en/microcontrollers-microprocessors/stm32f746ng.html

[3] https://github.com/mit-han-lab/tinyml/tree/master/mcunet

[4] https://github.com/rouyunpan/mcunet

the first MBConv layer in the global model and train such an exit classifier. This split results in spending 48M MACs in the base computation and then features are transferred to the global model in case the exit classifier sends the example to further layers. Note that in this case the feature cost is more than twice the input size and thus, this baseline suffers much worse from the communication delay than others. This method can always ditch the processing on the edge and send all examples to the cloud, thus achieving global accuracy with same latency and energy consumption as the on-cloud solution.

Dynamic Neural Networks. This baseline refers to various recent methods (Nan & Saligrama, 2017; Bolukbasi et al., 2017; Li et al., 2021) that propose dynamic computation and includes base-model based entropy or margin thresholding. For a fair comparison to the proposed hybrid method, we use entropy thresholding on the base model to represent this baseline.

Hybrid Model training. We use the MCUNet model-A as the base model and the MASS-600 as the global model. We train the hybrid models using these base and global architecture pairs. Let c be the coverage of the base model, i.e. c fraction of examples are inferred on the base and $1-c$ fraction of examples are sent to the global model. At $c = 100\%$, we obtain the on-device performance. Although hybrid models trivially achieve on-cloud performance by sending all examples to the global model, i.e. at $c = 0\%$, our hybrid training procedure obtains the on-cloud solution at with less communication. Let ℓ_b denote the inference latency on the edge (in this case it is 200ms), while ℓ_g denote the inference latency on the cloud (in this case it is the sum of communication cost and inference latency of the global model, i.e. $2000 + 25 = 2025$ms). Thus, we can write the inference latency of the hybrid model at a coverage c as follows:

$$\text{Hybrid Latency@c} = \ell_b + (1-c)\ell_g$$

Similarly, using we can compute the energy consumption for the hybrid inference as follows:

$$\text{Hybrid Energy@c} = \mu_b + (1-c)\mu_g$$

where μ_b and μ_g are the edge device energy consumption for on-device and on-cloud inference. In this case, $\mu_b = 10.37$mJ and $\mu_g = 216$mJ.

Upper Bound. Let us develop an upper-bound on achievable accuracy as a function of latency. Recall from Table 1 that communication latency dominates processing time. Also recall the notion of coverage, c, which denotes the fraction of examples locally processed. Note that there is a one-to-one correspondence between coverage and latency of the hybrid system. Suppose α_b, α_g denotes base accuracy and cloud accuracy. The base predictor predicts $(1-\alpha_b)$ fraction incorrectly, and among these suppose we make the reasonable assumption that the router is agnostic to which of those are correctly classified at the cloud. Then an upper bound on the target hybrid accuracy is given by the expression:

$$\text{Hybrid Acc@c} \leq \min\left\{\frac{\alpha_g - \alpha_b}{1-\alpha_b} * (1-c) + \alpha_b, \alpha_g\right\}$$

Notice that $c = 0$, $c = 1$, we recover global and base accuracies, and $c = \alpha_b$, is the cut-off point, namely cloud accuracy is achieved at the latency associated with the coverage c.

A.3. Algorithms

In addition, we frequently tune a given router r and base and global models to locally trade-off resource usage levels and accuracy (which saves on retraining on each different value of ϱ that one may be interested in. This is realised by finding a value t adjusted to the constraint, and using the routing function $r(x;t) = \mathbb{1}\{r_o(x) \geq r_1(x) + t\}$. Such a t may be found as in Algorithm 2.

Algorithm 2 Tuning Routing Model

1: **Input:** Validation data $V = \{(x^j, y^j)\}_{j=1}^M$, target resource level ϱ, Hybrid model (r, b, g).
2: $\mathcal{T} \leftarrow \{r_0(x) - r_1(x) : x \in V\}$.
3: $c^* \leftarrow \min c : \mathcal{R}_r + \mathcal{R}(\alpha_b) + (1-c)\mathcal{R}_g \leq \varrho$.
4: $t^* \leftarrow c^*$th quantile of $\mathcal{T}$.
5: **Return:** t^*.

A.4. Implementation Details

A.4.1. Hyper-parameter Settings.

We use SGD with momentum as the default optimizer in all our experiments. We initialize our hybrid models from the corresponding pre-trained models and use a learning rate of $1e-4$ for learning base and global models. We use a learning

rate of $1e-2$ for learning the routing network. We decay the learning rate using a cosine learning rate scheduler. As recommended in the earlier works, we use a weight decay of $1e-5$. We set the number of epochs to be 50. We use a batch size of 256 in our experiments.

A.4.2. MODEL DETAILS

Entropy Thresholding Baseline. As per recommendation in the literature (Teerapittayanon et al., 2017; Gangrade et al., 2021) we compute the entropy H of the base prediction probability distribution $b_y(x)$. This baseline allows access to a tunable threshold t. Predictions with entropy below this threshold are kept with the base model while the predictions with entropy above this threshold are sent to the cloud model. We use similar tuning as Algorithm 2 to trade-off resource usage.

Routing Model. Our routing model uses predictions from the base model and creates a 2-layer neural network from these predictions. We create meta features from these predictions to reduce the complexity of the network, by (a) adding entropy as a feature, (b) and adding correlations between top 10 predictions, resulting in a 101 dimensional input feature vector. The feed-forward network has 256 neurons in the first and 64 neurons in the second layer. The final layer outputs a two dimensional score leading to binary prediction for the routing r. Note that the routing network described in this manner contributes to less than 2% compute budget of the base model and hence its compute cost is negligible in comparison to the base and global models.

MBV3. We have used the MobileNetV3 (Howard et al., 2019) models as base in the hybrid models designed for mobile devices (see Sec. 3). We borrowed pre-trained models from publicly available implementation [5]. Table 5 lists the performance and compute characteristics of these borrowed models.

Table 5. MBV3 models in our setup.

	Top1 Accuracy	#Params	#MACs
MBV3-48	67.613	2.54M	48.3M
MBV3-143	73.3	3.99M	143.4M
MBV3-112	71.7	-	112M
MBV3-91	70.4	-	91M
MBV3-215	75.721	5.48M	215.3M

MASS. We borrowed the pre-trained (Cai et al., 2020) models from the official public repository [6]. Table 6 lists the accuracy, number of parameters and MACs for these models. We note that these models have been specialized by the authors with fine-tuning to achieve the reported performance.

Table 6. Once-for-All Pre-trained models in our setup.

	Top1 Accuracy	#Params	#MACs
MASS-600 ('flops@595M_top1@80.0_finetune@75')	79.9	9.1M	595M
MASS-482 ('flops@482M_top1@79.6_finetune@75')	79.6	9.1M	482M
MASS-389 ('flops@389M_top1@79.1_finetune@75')	79.1	8.4M	389M
MASS-240 ('LG-G8_lat@24ms_top1@76.4_finetune@25')	76.4	5.8M	230M
MASS-151 ('LG-G8_lat@16ms_top1@74.7_finetune@25')	74.6	5.8M	151M
MASS-101 ('note8_lat@31ms_top1@72.8_finetune@25')	72.8	4.6M	101M
MASS-67 ('note8_lat@22ms_top1@70.4_finetune@25')	70.4	4.3M	67M

A.5. Difference between AppealNet and our Hybrid design.

Below we highlight main difference between AppealNet ((Li et al., 2021)) and our proposal.

- AppealNet formulation does not explicitly model any coverage constraint that enables the base model to operate at a tunable coverage level. In contrast, we explicitly model a coverage penalty.

[5] https://github.com/rwightman/pytorch-image-models
[6] https://github.com/mit-han-lab/once-for-all

- Jointly learning the routing without any supervision is a hard problem. Instead, we relax this formulation by introducing the routing oracle that specializes in a routing network for a given base and global pair. With this oracle, the task of learning routing reduces to a binary classification problem with the routing labels obtained from the oracle. This also decouples the routing task from the base and global entanglement.
- AppealNet does not use supervision like us, and as such such strategies ultimately resemble thresholding on examples whose anticipated loss exceeds some threshold. To see this consider Algo. 1 line 7 (Li et al., 2021) for m examples. Taking the derivative wrt q yields $\beta\frac{1}{m}\sum_x \frac{1}{1-q(0|x)} = \frac{1}{m}\sum_x \ell(f_1(x), y) - \ell(f_0(x), y)$. The RHS is the *excess loss*. For small values of $q(0|x)$, we can approximate the LHS to yield: $\beta\frac{1}{m}\sum_x(1 + q(0|x)) \approx$ Excess-Loss. Simplifying we get: $\frac{1}{m}\sum_x q(0|x) \approx \frac{1}{\beta}$Excess-Loss $- 1$. This expression suggests a relaxed objective that we should enforce the fact that examples sent to the cloud is broadly proportional to excess loss, and as such represents very weak supervision.
- In addition, we propose a neural architecture search that finds a pair of base and global architectures that optimise the hybrid accuracy at any given combined resource usage.
- Empirically, AppealNet does not have any evaluations for the Imagenet scale dataset. The closest comparison we can find is with the Tiny-Imagenet dataset (one-tenth of the size of the Imagenet). While we cannot compare the two directly, since we solve a much harder problem than Tiny-Imagenet, we can make the following observations. At 70% coverage level, for AppealNet, the minimum performance difference between the hybrid model and the global model is $\approx 1.2\%$ (see AppealNet, Fig. 5(d)), while our closest to the global in case of the MobileNet baseline is 0.3% (see our paper Table 1, row 3). Note that AppealNet performance will go down on Imagenet in comparison to Tiny-Imagenet due to the hardness of the problem.

A.6. Difference between LENS and our Hybrid design.

Although below we highlight main difference between LENS ((Odema et al., 2021)) and our proposal, we emphasize that LENS studies the edge-cloud interactions purely from systems perspective and as such does not dwelve into the learning aspects and the trade-off required in routing the inputs in severely resource constrained edge devices as well as their limited communication capabilities.

- (A) **Our Objective:** On large-scale tasks (such as ImageNet), for the given WiFi rate, our goal is to realize the accuracy of an arbitrarily large DNN model (deployable on the cloud) by means of a hybrid method deployed on edge. We selectively route difficult inputs on the edge to the cloud, maintaining top-accuracy, while consuming minimal energy/latency.

  Our Edge. Our edge device only has CPU or MCU compute capabilities (see Sec. 3). In addition, these edge device are severely resouce constrained, namely they only allow low powered transmission as well as low transmission rates. These constraints limits the model that can be deployed on the edge to be very low footprint. For instance, our illustrative example only has 110Kbps transmission rate (see Sec. A.2).

  This together with our desired accuracy places stringent constraints on edge to crisply learn hard-to-predict examples, and characterizes **fundamental limits of hybrid ML**.

- (B) **Circuit/Systems Prior Works (ex: LENS or Neurosurgeon).** Their goal for a given dataset is to split/partition computation of a (suitably optimized) DNN to minimize latency/energy in response to changing wireless conditions.

  LENS Edge. (Odema et al., 2021) has GPU compute capabilities on the edge device. As a result, even large DNNs can be comfortably executed on this device without a significant delay as compared to the on-cloud solution. In addition, their edge device leverage high transmission rate (up to 25Mbps). As a result, LENS explores a different setting **agnostic** to data. They leverage low transmission latency to compensate the difference in edge and cloud GPU times, motivating partitioning.

- (C) **Objective (B) is suboptimal for objective (A).** (B) requires the same network model on the edge and the cloud, which artificially constrains DNNs to fit into edge's specifications, while hoping to realize cloud-server gains on the partitioned parts. For large-scale tasks (ImageNet), high-accuracy can only be achieved by large DNNs (even with NAS optimization (Cai et al., 2020)), which are not edge-deployable, and using different architectures on edge/cloud is fundamental in (A).

- (D) **LENS baselines are too weak.** Direct comparisons are difficult due to different system choices(see (B)). Still we can note that LENS:

  – reports results on small CIFAR-10 dataset, which is not representative

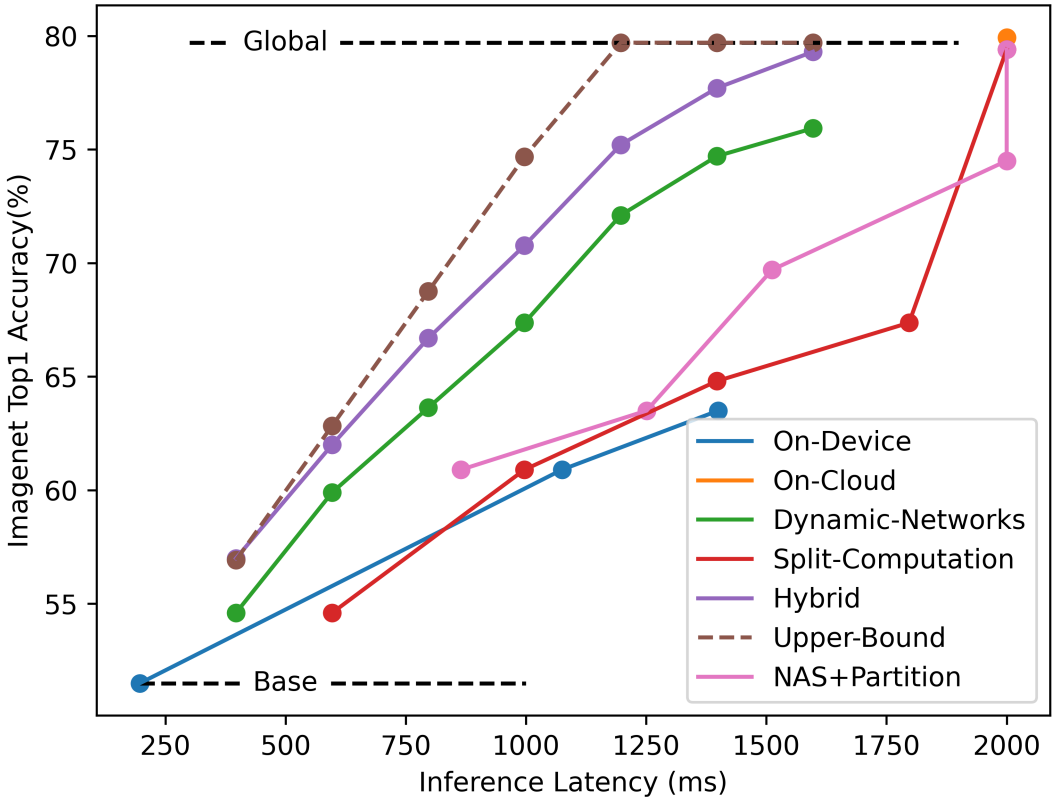


Figure 3. Image recognition on the ImagetNet dataset: Accuracy vs Energy and Latency plot. This clearly shows that the hybrid design pareto dominates on-device as well as other baselines while getting significantly closer to the upper-bound in hybrid design.

- uses VGG-16 based architecture(large DNN)—typically not edge-deployable
- with all-cloud processing, achieves 82%, significantly lower than VGG16 published results (93.56% see `https://github.com/geifmany/cifar-vgg`);
- with optimal NAS+partitioning gets 77% under 25% energy reduction (see Fig. 6)

In contrast, for CIFAR-10 we trained standard (tiny) models (see MCUNet model described as On-Device baseline in Sec. A.2) deployable on MCUs.

- MCUNet (Lin et al., 2020) is deployable on the resource constrained MCUs.
- VGG-16 on CPU has 280X worse latency w.r.t. our model
- With all-edge processing we get 91.7% accuracy consuming 11mJ, and 85% under 25% energy reduction.

This shows using same model on cloud/edge is suboptimal (see (C)).

- (E) **Large-scale Task: LENS code is unavailable; Direct Evaluation is difficult.** LENS employs GP-based bayesian optimization to NAS, which is known to produce poor results (see (White et al., 2021)), which is also evident from (D). Due to lack of publicly avaiable codebase, we created a similar baseline to see the performance gap between our proposal and LENS on the illustrative example in the introduction (see Figure 2). We optimized NAS+partitioning method by optimizing over OFA models/architectures (Cai et al., 2020). These architectures range from small to large models across diverse target accuracies. Hybrid methods overwhelmingly dominate NAS+partitioning methods (pink in Figure 3), again reinforcing our point (C).

A.7. MCUNet Router Deployment Overhead

We deploy both MCUNet and our base with routing model on the MCU using the TensorFlow Lite for Microcontrollers (TFLM) runtime. Due to lack of operator support for reductions and sorting in TFLM, we replace the relevant operators with

supported operations whose compute and memory complexity upperbounds the un-supported operations. Table 7 compares the performance energy profile of the hybrid model and the baseline when deployed on the micro-controller (STM32F746) with 320KB SRAM & 1MB Flash. It clearly shows that there is a negligible cost of deploying the proposed routing scheme and only results in $< 2\%$ slowdown.

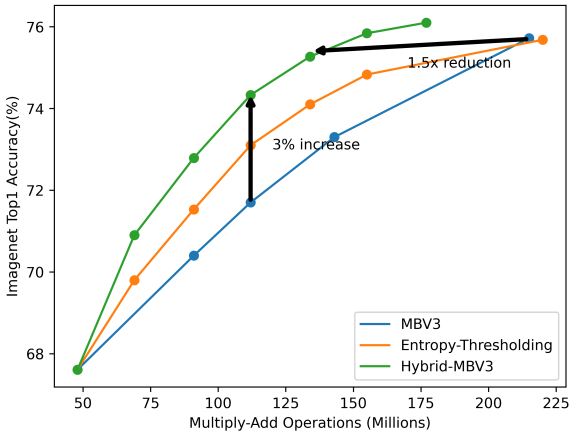
Table 7. Profiling the on device latency and energy overhead associated with deploying the Hybrid model (MCUNet + router) as compared to deploying the plain MCUNet model on the MCU.

Model	Latency	SRAM	Energy
MCUNet	0.25368s	156708 bytes	0.1112 joules
Hybrid-MCUNet	0.25951s	158036 bytes	0.1134 joules

A.8. Ablative Experiments

A.8.1. Base and Global on same device

So far we have focused on a setup where base and global models are deployed on separate hardware. In this experiment, we deploy the base and global models on the same device. As a result, there is no communication delay in the setup. In such a setup, we can 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 4 shows the hybrid model performance against the intermediate points in the MBV3 space.



(a) MobileNetV3

Figure 4. MBV3: Plot for hybrid MACs vs accuracy.

Figure 4 shows the hybrid model performance against the intermediate points in the MBV3 space as well as entropy thresholding baseline. These experiments provide evidence for the following properties of hybrid models:

- *Hybrid achieves SOTA w.r.t a global model with up to* 40% *lower latency.* Global model in MBV3 achieves 75.7% accuracy by off-loading every example to the cloud while hybrid model achieves same accuracy by sending only 60% examples to the cloud. Thus, saving 40% communication cost.
- *Training a Hybrid model for intermediate latency is inexpensive.* To achieve a single model at any latency, we find an architecture with this constraint and train it to non-trivial performance. Hybrid model with extreme points trades off latency for accuracy and save compute for training models for any intermediate constraint.
- *Hybrid models dominate entropy thresholding baseline used in dynamic neural networks.* Hybrid models outperform entropy thresholding at every coverage level with up to 1.5% accuracy gains.

A.8.2. ABSTAINING CLASSIFIER

Hybrid scheme allows the system to operate without a global model. In this case. the result is an abstaining classifier operating on the device, i.e., it rejects few inputs and provides predictions on the rest. We create abstaining classifiers with the hybrid setup in the Sec. 3. We show the accuracy of the base model at various coverage levels in the Table 8.

Table 8. Abstaining Classifier with hybrid models from Sec. 3. Results for hybrid models with base at various coverage levels. MASS-600 model achieving $\approx 80\%$ Top1 accuracy is used as global model. Base model belongs to MBV3 space.

Base MACs	Base Acc. (%)	Cov.=90% Acc. (%)		Cov.=80% Acc. (%)		Cov.=70% Acc. (%)	
		Base	Hybrid	Base	Hybrid	Base	Hybrid
48M	67.6	73.3	71.6	78.6	74.6	83.4	76.8
143M	73.3	79.0	75.9	83.9	77.8	88.4	79.0
215M	75.7	81.3	77.6	86.1	79.0	90.1	79.6

Table 8 shows that the abstaining base achieves significantly better performance than the base at full coverage and outperforms the abstaining classifier from the entropy thresholding baseline.

A.8.3. ROUTER VALIDATION

We evaluate the performance of the router against the oracle supervision and show that the router learnt using the procedure described in Sec. 2.1 generalizes well. For instance, while training a hybrid model with pre-trained MBV3-small and MBV3-large models, on the oracle labels, the router achieves a training accuracy of $\approx 87\%$ and this translates into a validation accuracy of $\approx 84\%$. In contrast, entropy thresholding on the validation dataset achieves $\approx 77\%$ accuracy on the oracle labels.

A.8.4. IMDB EXPERIMENTS

We also learn hybrid model in the NLP domain. We train a sentiment classifier on the IMDb dataset (Maas et al., 2011). We pick-up off-the-shelf pre-trained classifiers, namely (a) albert-base-v1 (Lan et al., 2019) (11M params, 91% accuracy) as the base and (b) bert-large (Devlin et al., 2018) (340M params, 96% accuracy) as the global. We use the hidden state from the last timestep in the sequence along with the classifier logits and entropy as the feature for the routing model. In order to save computation, we learn the hybrid model by training only the router. Table 9 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 IMDb dataset.

Table 9. Hybrid models for IMDb at various coverages.

Base MACs	Base Acc.(%)	Cov.=97% Acc. (%)	Cov.=95% Acc. (%)	Cov.=93% Acc. (%)
Entropy	91	93.78	94.38	95.25
Hybrid	91	94.31	95.45	96.01

A.8.5. MCUNET EXPERIMENT WITH EFFICIENTNET-B7

In the main text, due to limited compute resources, we restricted our global model to be the MASS-600 (Cai et al., 2020) model. In this ablation, we explore the effect of deploying a significantly expensive model on the cloud. We choose the best performing model in the EfficientNet (Tan & Le, 2019) family, i.e., EfficientNet-B7. This model has 37B MACs and stores 66M network parameters. We borrow the implementation from the timm repository [7] that achieves an accuracy of 86.5% on the Imagenet classification task. In contrast, the MASS-600 model has ≈ 600M MACs, 9.1M parameters and achieves $\approx 80\%$ accuracy. For simplicity, we assume the cloud resources render the inference on EfficientNet-B7 to be similar to

[7] EfficientNet-B7-ns model from https://github.com/rwightman/pytorch-image-models

MASS-600. In this experiment, we train a hybrid-r model with MCUNet base used in the Sec. 3. Thus, we can compare the performance of the hybrid models across different global models. Table 10 compares the accuracies obtained by the hybrid models with two different global models (MASS-600 and EfficientNet-B7). It clearly shows the following benefits:

- Deploying a better global model improves the hybrid performance and with cloud resources such large models do not affect the energy consumption on the edge device.
- Assuming that the cloud has access to large compute pool, the inference latency on the edge device does not suffer as well.
- It shows that our algorithm procedure to train hybrid models works across global models in different architecture families.

Table 10. EfficientNet-B7 as Global model: Hybrid models on STM32F746 MCU: Accuracy achieved by different methods at various latency constraints. Base model is the MCUNet model with 12M MACs and 200ms latency.

Method	Accuracy (%) at Latency (ms)					
	200	600	1000	1400	1600	2000
On-Cloud (MASS-600)	-	-	-	-	-	79.9
On-Cloud (EfficientNet-B7)	-	-	-	-	-	86.5
On-Device	51.1	-	60.9	63.5	-	-
Hybrid (Global=MASS-600)	-	62.3	71.2	77.9	79.5	-
Hybrid (Global=EfficientNet-B7)	-	64.9	76.2	82.8	85.7	-