We thank all reviewers for their comments and thank R2 and R3 for suggesting the literature. We will revise our paper accordingly. The presentation will be polished and more discussions will be included in the broader impact section.

R1: Novelty compared to Once-for-All. This paper targets at solving an **entirely new** challenge: on-device transfer learning on memory-constrained edge devices, which is **fundamentally different** from existing NAS for efficient inference problem in Once-for-All. Whether and how can Once-for-All help towards addressing this new problem is an open research question and **never** explored before. Only updating biases along with the memory-saving insights behind it (Sec. 3.1) are also **entirely new**. To our best knowledge, we are the **first** to introduce this finding. Moreover, based on the insights from only updating biases, we further designed a new technique 'Lite Residual Learning', which **efficiently** recovers the lost expressivity from not updating the weights. The effectiveness of our method has been **thoroughly verified** (**9.5-12.5**× memory saving on multiple datasets in Fig.4, up to **13.3**× memory saving in Tab.1). We believe our findings will open up new opportunities for on-device learning.

R1: Scalability of the computation efficiency and results on more datasets. Our approach might be misunderstood by the reviewer. First, we **never** train sub-nets when collecting the training data for the accuracy predictor; **instead**, we directly inherit the weights from the super-net to initialize the sub-nets, thus scalable to large datasets. Second, this work targets at on-device transfer learning (much less data/memory), **not** conventional transfer learning. Therefore, we focus on datasets with fewer images (e.g., Flowers) that are **much closer** to real-world on-device scenarios than large datasets. Certainly, our method **generalizes** to large datasets. In Table A, we justify the effectiveness of TinyTL on Food101 (the largest dataset in [7, 27]). TinyTL **consistently** achieves **significant** memory saving (**7.3**×) with little accuracy loss.

	Mem.	Food101
Full [27]	802MB	87.7%
TinyTL	109MB	87.2%

Table A: Results on Food101.

R2: Details of feature extractor adaptation. We will add more details to the main paper in the final version. **(L53-54)** The discrete optimization space includes depth ('Repeat': 1,2,3), width ('Expand Ratio': 3,4,6) and kernel size ('Kernel Size': 3,5,7) [Appendix E]. Each architecture configuration corresponds to a sub-net. The objective is to find the best sub-net that maximizes transfer accuracy. **(L171-175)** The super-net is a normal neural network with the maximum depth, width, and kernel size. Sub-nets are derived from the super-net by sparsely activating parts of the model according to the architecture configuration. Specifically, consider a 7x7 conv layer denoted as $W_{0:c1,0:c2,0:7,0:7}$, an example of the candidate weight operation set (in Eq.5) is $\{W_{0:c1,0:c2,0:7,0:7}, W_{0:c1,0:c2,1:6,1:6}, W_{0:c1,0:c2,2:5,2:5}\}$, which corresponds to kernel size = 7/5/3. **(L186-189)** In the process of fine-tuning the super-net, we only **update the memory-efficient modules** (bias, lite residual, classifier head), while **freezing the memory-heavy modules**. Since sub-nets inherit weights from the super-net, all sub-nets are adapted to the target dataset while keeping the memory footprint small. **Random sampling** can ensure each sub-net is evenly trained, while accuracy-based sampling biases towards *early* good performers and keeps sampling them more frequently without exploring others. A sub-net that performs well early does not guarantee to be the best in the end. Therefore we chose random sampling. **(L190)** The accuracy predictor can predict the transfer accuracy given a sub-net architecture. Conventionally, we need to evaluate many sub-nets on the target dataset to find the best one, which is expensive. Instead, we exploit a highly efficient accuracy predictor [Appendix C] to reduce the cost. '450 [sub-net, accuracy]' is the collected dataset for training the accuracy predictor.

R2, R5: Cost of feature extractor adaptation. We have **strong** reasons to believe that the whole feature extractor adaptation process (including fine-tuning the super-net) is feasible on edge devices [Appendix B]. First, as we freeze the weights of the feature extractor, the peak memory cost of fine-tuning the super-net is **only** 64MB under batch size 8, which is 4x smaller than the DRAM size of RPi-1. Moreover, combined with group normalization (refer to '**R3:** Streaming Training'), TinyTL can support training with batch size 1, where the peak memory cost is **only** 26MB. It allows fitting the whole process into the on-chip SRAM of TPU, which is **128x** energy-efficient than DRAM (Fig.1). Second, our total computational cost is **18x** smaller than fine-tuning the full network [27] while preserving accuracy.

R2, R5: Effects of freezing biases. Adapting biases is necessary. Without it, the accuracy drops by 1.7% on Cars, 0.5% on Flowers, and 4.1% on Aircraft (Table B).

	Cars	Flowers	Aircraft
w/ bias	91.6%	97.5%	84.0%
w/o bias	89.9%	97.0%	79.9%

Table B: Effects of freezing biases.

R5: Results without feature extractor adaptation. If disabling the feature extractor adaptation, the accuracy drops by 2.2% on Cars, 0.6% on Flowers, and 2.5% on Aircraft (shown in Tab.1, page7). Feature extractor adaptation is critical.

R5: Apply to conventional transfer learning. 'Fine-tuning the full network' can also benefit from feature extractor adaptation (FA). Compared to InceptionV3+Full, FA+Full improves the accuracy from 91.3% to 93.2% on Cars, from 96.3% to 98.3% on Flowers, from 85.5% to 88.9% on Aircraft. We will include this feature in code release.

R3: Streaming Training. TinyTL supports streaming training by replacing batch normalization (BN) with group normalization (GN), which supports batch 1 training. We observe little loss of accuracy from BN to GN: 89.4%->89.0% on Cars, 96.9%->96.7% on Flowers, 81.5%->81.1% on Aircraft. We will include the new results in the revision.

R3: Hardware deployment. Fig.4 used theoretical values as Pytorch does not support fine-grained memory management. We target co-designing the on-device training framework to fully exploit the theoretical benefits, which is beyond the scope of this paper. We will make this clear in the revision.

R3: Fig 3, Fig 6. In Fig.3, 'ours' refers to TinyTL FA from Tab.1. In Fig.6, the parameter size consists of two parts: i) frozen parameters (2.3MB,8bits); ii) trained parameters (11.3MB,32bits). We will make it more clear in the revision.