A Multi-task Supervised Compression Model for Split Computing - Supplementary Material -

Yoshitomo Matsubara *
University of California, Irvine
yoshitom@uci.edu

Matteo Mendula
University of Bologna
matteo.mendula@unibo.it

Marco Levorato
University of California, Irvine
levorato@uci.edu

A. Hyperparameters for Ladon model training

Step 1: Pre-training encoder-decoder

We pre-train the encoder-decoder modules of our Ladon models with the Adam optimizer [4], minimizing the loss function defined in Eq. (2) in the main paper. We use outputs of the first, second, third, and fourth residual / ResNeSt blocks from the pre-trained ResNet-50 as $\mathbf{h}_i^t(\mathbf{x})$ and extract those of the corresponding residual blocks in our ResNet-50-based Ladon models as $\mathbf{h}_i^s(\mathbf{x})$. β is a hyperparameter to control the rate-distortion tradeoff. Given a Ladon architecture, we train five individual models with $\beta = 0.32, 0.64, 1.28, 2.56$, and 5.12. The same procedure is applied when teacher and student models are based on ResNeSt-269e [10]. We use the trainind set of the ILSVRC 2012 dataset [8] for 10 epochs, and training batch size is 32. The initial learning rate is 0.001 and exponentially decayed by a factor of 0.1 after the first 5 and 8 epochs.

Step 2: Fine-tuning decoder and subsequent modules

Following Step 1, we freeze parameters of the encoder and entropy bottleneck. We then fine-tune the remaining modules including a classification head as illustrated in Fig. 1 (bottom). Specifically, we fine-tune the modules for 10 epochs using the pretrained ResNet-50 (ResNeSt-269e) as a teacher model for a standard knowledge distillation (KD) loss function [3]

$$\mathcal{L} = \alpha \cdot \mathrm{CE}(\hat{\mathbf{y}}, \mathbf{y}) + (1-\alpha) \cdot \tau^2 \cdot \mathrm{KL}\left(\mathbf{o}^{\mathrm{S}}, \mathbf{o}^{\mathrm{T}}\right), \quad \text{(S1)}$$

where CE and KL are cross-entropy and Kullbuck-Leibler divergence, respectively. $\hat{\mathbf{y}}$ and $\mathbf{y}$ are true and predicted class labels. $\alpha \in [0, 1]$ and τ are hyperparameters. We use $\alpha = 0.5$ and $\tau = 1$ in this study. $\mathbf{o}^{\mathrm{T}}$ and $\mathbf{o}^{\mathrm{S}}$ indicate*softened* output distributions produced by teacher and student models, respectively. $\mathbf{o}^{\mathrm{T}} = [o_1^{\mathrm{T}}, o_2^{\mathrm{T}}, \ldots, o_{|\mathcal{C}|}^{\mathrm{T}}]$ where $\mathcal{C}$ is a set of object categories in the target task, which is an image classification for the ILSVRC 2012 dataset [8]. o_i^{T} is the teacher model's softened output value (scalar) for the i-th object category:

$$o_i^{\mathrm{T}} = \frac{\exp\left(\frac{v_i^{\mathrm{T}}}{\tau}\right)}{\sum_{k \in \mathcal{C}} \exp\left(\frac{v_k^{\mathrm{T}}}{\tau}\right)}, \quad \text{(S2)}$$

where v_i^{T} is the teacher model's logit value for the i-th object category. The same rule is applied to the student model.

Here, we use a stochastic gradient descent (SGD) optimizer with the initial learning rate of 0.001, momentum of 0.9, and weight decay of 0.0005. The learning rate is exponentially decayed by a factor of 0.1 after the first 5 epochs. At the end of this step, all the modules in our Ladon model required for the image classification task are ready to serve.

Step 3: Fine-tuning other task-specific modules

Following Step 2, we freeze all the learnt parameters and then introduce other task-specific modules.

Object Detection We introduce to the fine-tuned Ladon model, an object detection head that consists of Faster R-CNN [7] and Feature Pyramid Network (FPN) [5] modules. We train the object detection head on COCO 2017 [6] for 26 epochs (ResNet-based Ladon) and 28 epochs (ResNeSt-based Ladon), minimizing a linear combination of bounding box regression, objectness, and object classification losses [7]. The SGD optimizer uses the initial learning rate of 0.02, momentum of 0.9, weight decay of 0.0001, and batch size of 8. Its learning rate is exponentially decayed by a factor of 0.1 after the first 16 and 22 epochs. For ResNeSt-based Ladon, we use the weight decay of 0.0005 and exponentially decay the learning rate by a factor of 0.1 after the first 10, 18 and 24 epochs.

Semantic Segmentation For a semantic segmentation head, we employ DeepLabv3 [1]. With the SGD optimizer, we train the semantic segmentation head by minimizing the standard cross-entropy for 90 epochs, using momentum of 0.9, weight decay of 0.0001, and batch size of 16. For the

*This work was done prior to joining Spiffy AI.

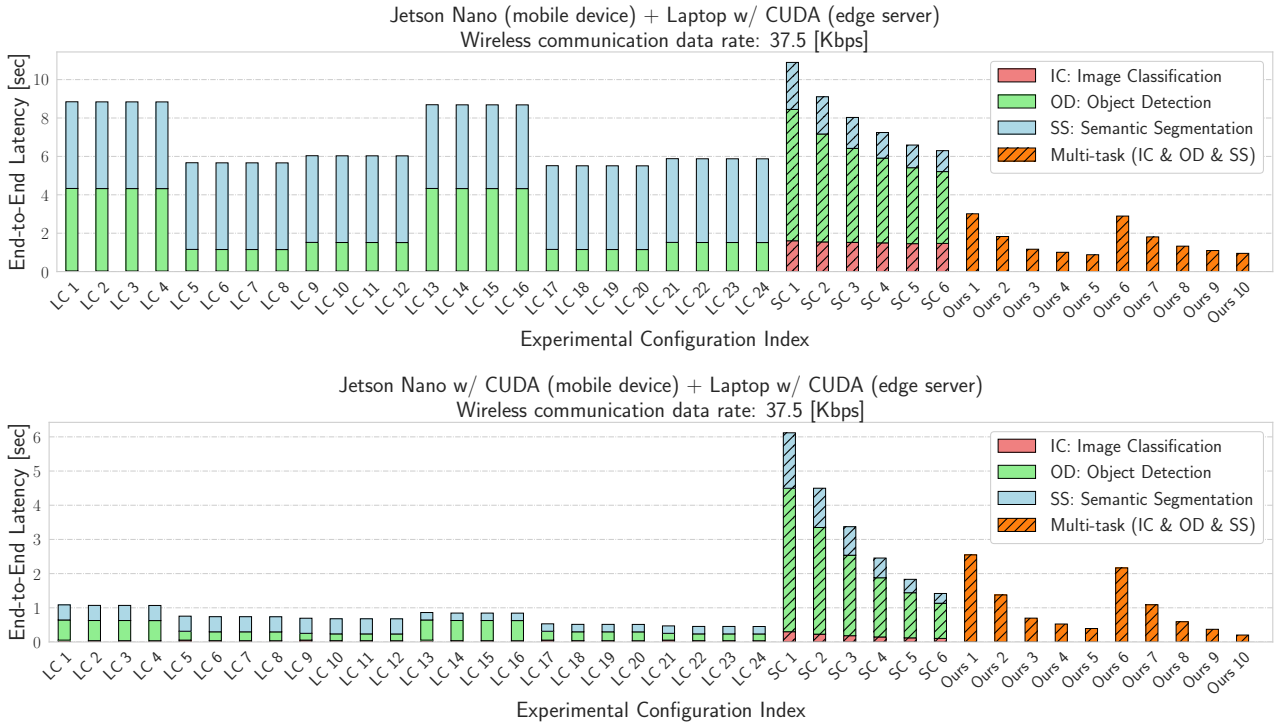

Figure S1. End-to-end latency for Jetson Nano (mobile device), laptop with CUDA (edge server), and wireless communication data rate of 37.5 Kbps. Top/bottom: local computing without/with CUDA.

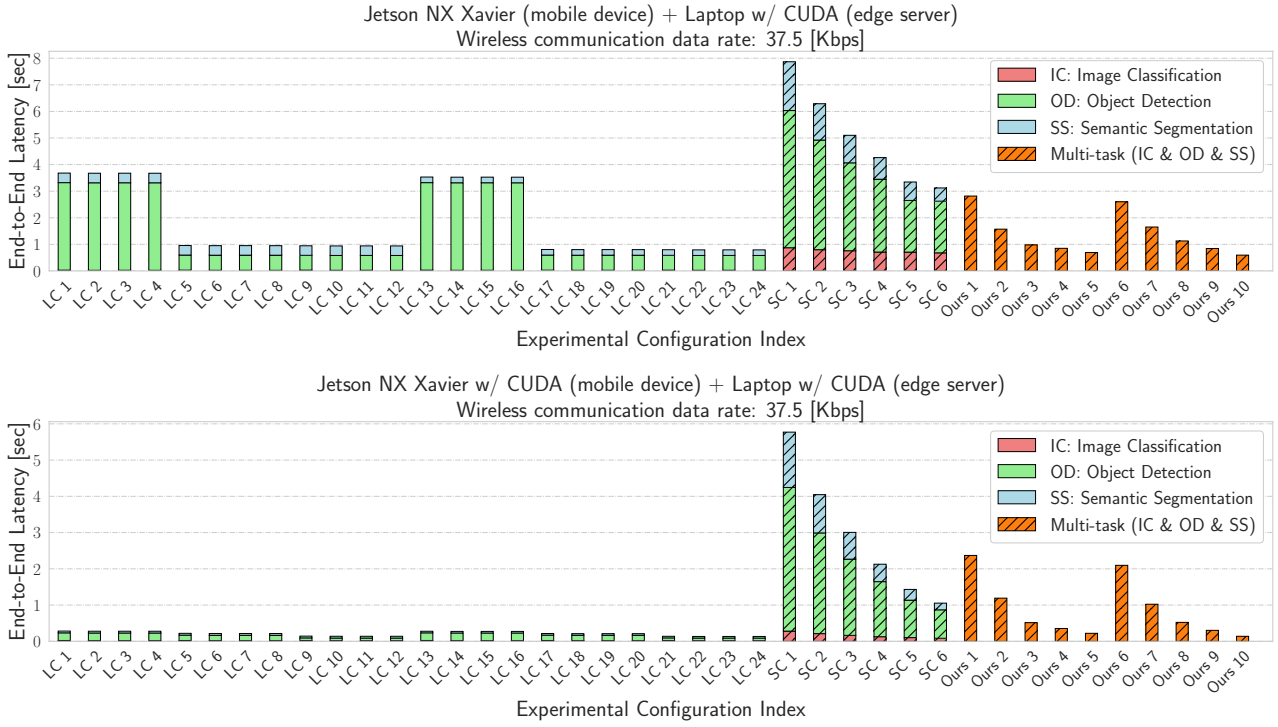

Figure S2. End-to-end latency for Jetson NX Xavier (mobile device), laptop with CUDA (edge server), and wireless communication data rate of 37.5 Kbps. Top/bottom: local computing without/with CUDA.

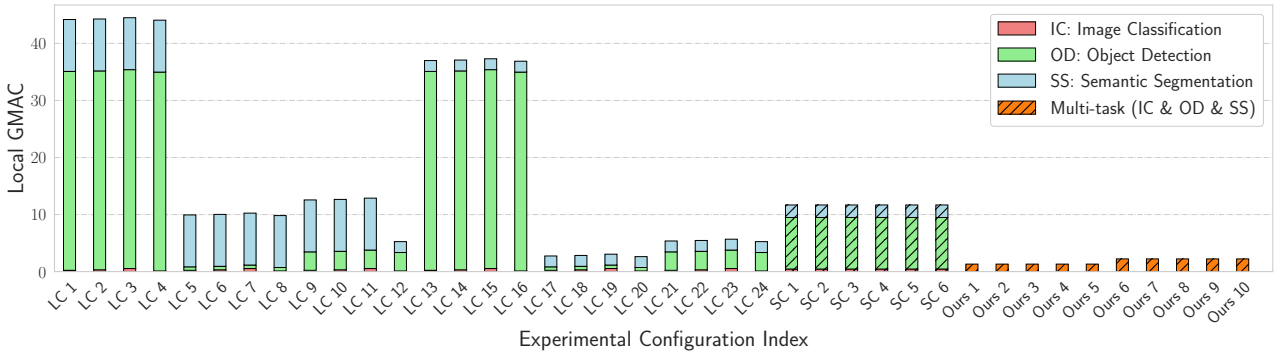

Figure S3. Local Giga Multiply-Accumulate Operation (GMAC).

first 30 epochs, we use COCO 2017 training dataset [6] with the initial learning rates are 0.02 and 0.01 for the semantic segmentation head and its auxiliary classifier, respectively. We follow [1] and reduce the learning rates at every iteration

$$\eta = \eta_0 \times \left(1 - \frac{t}{N_{\text{iter}}}\right)^{0.9}, \tag{S3}$$

where η_0 is the initial learning rate. t and N_{iter} are the current iteration count and the total number of iterations, respectively.

For the last 60 epochs, we use the PASCAL VOC 2012 training dataset [2] and fine-tune the semantic segmentation head. Other hyperparameters are the same as those for the first 30 epochs.

B. End-to-end Latency with LoRa

In the main paper (Figs. 5) and 6), we show end-to-end latency evaluations using a challenged wireless network, where we assume the data rate is only 100 Kbps. Here, we consider LoRa [9], a further challenged network condition whose maximum data rate is 37.5 Kbps, and perform another end-to-end latency evaluation using the same experimental configurations (see Table 3).

Figures S1 and S2 show the end-to-end latency evaluation results for Jetson Nano and Jetson NX Xavier (top: CUDA OFF, bottom: CUDA ON) as mobile devices, respectively. Note that the new experimental configuration does not affect the performance of the local computing (LC) baselines in Figs. 5 and 6 since the LC baselines do not offload computation to the edge server. In other words, the configuration makes it more difficult for the SC baselines and our proposed method to outperform the LC baselines as the lower communication data rate will further delay communications between mobile devices and edge (cloud) servers.

The overall trends in Fig. S1 are similar to those in Fig. 5 in the main paper. Our multi-task models (Ladon) saved up to 90.1% and 96.7% of the end-to-end latency with the LC and SC baselines, respectively. Using Jetson NX Xavier in this scenario (Fig. S2) made the LC baselines even stronger. While giving the LC baselines more advantage, the Ladon models reduced the end-to-end latency of the LC and SC baselines by up to 83.8% and 97.6% respectively.

C. Local GMAC and Peak Local Memory Usage

In this section, we briefly discuss computational loads on mobile device using local GMAC (Giga Multiply-Accumulate Operation) and peak local memory usage metrics. As shown in Figure S3, our approach consistently achieved lower GMAC on mobile devices than baseline methods considered in this study, saving up to 97.0% and 88.7% of local GMAC for the LC and SC baselines, respectively.

Table SI presents peak memory usage measured during our multi-task experiments. Note that the reported memory consumption may include those of background jobs running on the mobile devices. Overall, our approach improved peak memory usage on mobile devices over the baseline methods.

References

[1] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking Atrous Convolution for Semantic Image Segmentation. *arXiv preprint arXiv:1706.05587*, 2017.

[2] Mark Everingham, Luc Van Gool, CKI Williams, John Winn, and Andrew Zisserman. The PASCAL Visual Object Classes Challenge 2012 (VOC2012). 2012.

[3] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the Knowledge in a Neural Network. In *Deep Learning and Representation Learning Workshop: NIPS 2014*, 2014.

[4] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *Third International Conference on Learning Representations*, 2015.

[5] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature Pyramid Networks for Object Detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2117–2125, 2017.

[6] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence

Config.	Jetson Nano		Jetson NX Xavier	
	w/o CUDA	w/ CUDA	w/o CUDA	w/ CUDA
LC 1	2.17	0.785	4.49	2.93
LC 2	2.17	0.651	4.49	2.93
LC 3	2.17	0.632	4.49	2.93
LC 4	2.17	0.632	4.49	3.10
LC 5	2.17	0.785	4.49	2.93
LC 6	2.17	0.651	4.49	2.93
LC 7	2.17	0.632	4.49	2.93
LC 8	2.17	0.632	4.49	3.10
LC 9	2.17	0.785	4.49	2.93
LC 10	2.17	0.651	4.49	2.93
LC 11	2.17	0.632	4.49	2.93
LC 12	2.17	0.632	4.49	3.10
LC 13	2.16	1.11	4.36	3.65
LC 14	2.16	1.11	4.36	3.65
LC 15	2.16	1.11	4.36	3.65
LC 16	2.16	1.11	4.36	3.65
LC 17	2.16	1.11	4.36	3.65
LC 18	2.16	1.11	4.36	3.65
LC 19	2.16	1.11	4.36	3.65
LC 20	2.16	1.11	4.36	3.65
LC 21	2.16	1.11	4.36	3.65
LC 22	2.16	1.11	4.36	3.65
LC 23	2.16	1.11	4.36	3.65
LC 24	2.16	1.11	4.36	3.65
SC 1	1.16	0.678	3.21	3.14
SC 2	0.937	0.618	3.08	3.63
SC 3	0.798	0.712	3.16	3.54
SC 4	0.760	0.699	2.91	3.60
SC 5	0.774	0.738	2.85	3.31
SC 6	0.790	0.689	2.72	3.64
Ours 1	0.409	0.560	0.868	2.12
Ours 2	0.502	0.535	0.599	1.94
Ours 3	0.459	0.524	0.737	2.30
Ours 4	0.436	0.563	0.396	2.17
Ours 5	0.439	0.539	0.450	2.03
Ours 6	0.510	0.557	1.60	1.70
Ours 7	0.717	0.528	1.44	1.67
Ours 8	0.721	0.427	1.36	1.51
Ours 9	0.647	0.597	0.871	2.21
Ours 10	0.682	0.543	1.12	2.14

Table SI. Peak local memory usage [GB] during multi-task experiments. Reported numbers may include memory consumption by background jobs on mobile devices.

Zitnick. Microsoft COCO: Common Objects in Context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[7] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In *Advances in Neural Information Processing Systems*, pages 91–99, 2015.

[8] 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*, 115(3):211–252, 2015.

[9] Farzad Samie, Lars Bauer, and Jörg Henkel. IoT Technologies for Embedded Computing: A Survey. In *2016 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS)*, pages 1–10. IEEE, 2016.

[10] Hang Zhang, Chongruo Wu, Zhongyue Zhang, Yi Zhu, Haibin Lin, Zhi Zhang, Yue Sun, Tong He, Jonas Mueller, R Manmatha, et al. ResNeSt: Split-Attention Networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2736–2746, 2022.