Proceedings of the 3rd International Conference on Software Engineering and Machine Learning
DOI: 10.54254/2755-2721/145/2025.21894

Review of Tiny Machine Learning Model Pruning Techniques on Resource-constrained Devices

Shenyi Zhang

Zhejiang University - University of Illinois at Urbana-Champaign Institute, Jiaxing, China
shenyi.23@intl.zju.edu.cn

Abstract: Lightweight pruning facilitates the deployment of machine learning models on resource-constrained devices. This review systematically examines pruning techniques across different technical paths, along with lightweight strategies that incorporate pruning. Regarding pruning techniques, the review successively delves into the principles, implementation methods, and applicable scenarios of structured pruning, unstructured pruning, and automated pruning. Structured pruning holds significant advantages in hardware implementation; unstructured pruning, on the other hand, demonstrates unique potential in fine-grained optimization, while automatic pruning methods achieve more precise model compression through intelligent search strategies. Subsequently, it centers on the synergy among pruning and other lightweight approaches, presenting the integration of pruning with quantization, the combination of pruning and distillation, as well as the pruning concepts incorporated in lightweight neural network architectures. The review concludes by highlighting some current challenges facing pruning technologies and offering insights into potential future research directions. These integration strategies not only enhance the model's operational efficiency in resource-constrained environments, but also offer innovative ideas for further compression while maintaining high accuracy.

Keywords: TinyML, pruning, quantization, distillation, lightweight neural network architecture

1. Introduction

Tiny Machine Learning(TinyML) is primarily applied in embedded edge devices, where low power consumption, real-time performance, and minimal latency are critical[1]. This research can define Tiny Machine Learning Model as “machine learning aware architectures, frameworks, techniques, tools, and approaches which are capable of performing on-device analytics for a variety of sensing modalities (vision, audio, speech, motion, chemical, physical, textual, cognitive) at mW (or below) power range setting, while targeting predominately battery-operated embedded edge devices suitable for implementation at large scale use cases preferable in the IoT or wireless sensor network domain”[2]. Motivated by the urgent demand in highly resource-limited environments, this study explores how to optimize and integrate advanced pruning techniques with other lightweight strategies to achieve efficient model performance without compromising accuracy. The lightweight strategies mainly concentrate on two aspects: model compression and the design of lightweight neural networks. Model compression includes methods like quantization and knowledge distillation, and is essentially enhanced by efficient pruning techniques. Efficient pruning techniques form the backbone of model

DOI: 10.54254/2755-2721/145/2025.21894

compression in TinyML by systematically eliminating redundant parameters and reducing computational complexity with slightly sacrificing performance. Lightweight neural network design seeks to minimize resource consumption starting from the design of convolutional structures, an approach that, when combined with targeted pruning, yields models that are well-suited for deployment on resource-constrained devices. By enhancing model compression efficiency, the proposed approach promises notable improvements in latency and energy efficiency for resource-constrained IoT and embedded applications.

2. Pruning Techniques

The neural network pruning framework proposed by Han et al. is widely adopted in current research[4]. It begins with thoroughly training the initial network, followed by parameter importance evaluation through weight magnitude or gradient statistics, which informs the subsequent selective pruning. After pruning, the model undergoes fine-tuning to restore performance. Multiple iterations progressively optimize the network structure, forming a closed-loop "pruning–recovery–verification" process as shown in Fig.1.

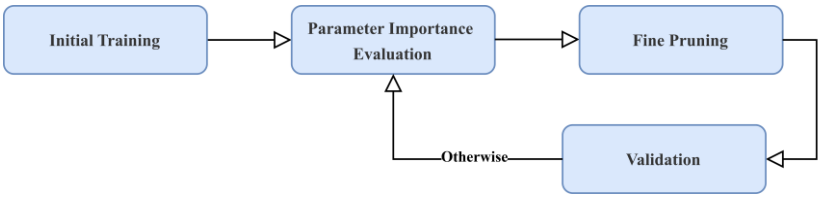

Figure 1: Neural network pruning framework

2.1. Structured Pruning

In structured pruning, the L-norm serves as a convenient metric for gauging the overall importance of convolutional kernels or channels in structured pruning, effectively identifying redundant components that can be removed[3]. Its application spans the entire pruning pipeline: ranking and pruning based on L-norm values, then continually updating these evaluations in subsequent training iterations. This review primarily synthesizes optimization avenues for L-norm based pruning methodologies.

Li et al. employ a predefined per-layer pruning ratio to remove filters with smaller L1 norms, thereby achieving structured pruning by discarding these low-norm filters in their entirety[5]. Wu et al. introduce an L1 and an L2 regularization term into the error function. As a result, the weights gradually decrease during training and can finally be removed when the training is completed[6]. Luo et al. introduce an L1-norm based and L2-norm regularization based Extreme learning machine(ELM)[7]. It leverages the grouping effect of the L2 penalty and the sparsity characteristic of the L1 penalty.

In order to achieve a deeper reduction in model redundancy and enhance the robustness of the pruned model, L-norm-based optimization strategies have been introduced. Cheng et al. propose an improved similarity metric by extending the concept of cross-covariance among feature maps into a Hilbert space and computing its Hilbert–Schmidt norm[8].

Wang et al. show the possibility of linear representational redundancy among the feature maps produced by convolutional kernels[9]. They introduce an additional loss term, termed CCM-loss, into the training objective. By optimizing this term during back propagation, the convolution kernels' outputs can be made partially linearly representable by one another, which in turn allows for merging or removing redundant channels during subsequent pruning. CCM-loss directly acts on the

DOI: 10.54254/2755-2721/145/2025.21894

convolutional kernel outputs, concentrating on the similarity or correlation of the data at the feature-mapping level.

Wang et al. proposed a structured pruning algorithm for CNN models based on GraSP algorithm[10]. In each training iteration, the algorithm first selects the filter index set G_k containing those with the highest absolute gradient values and the set A_k containing the largest parameter magnitudes. It then takes the union $S_k = G_k \cup A_k$ as a candidate set of filters to retain, while all others are set to zero or flagged for pruning, allowing those with high gradient contributions or large magnitudes to receive more training opportunities in subsequent iterations.

Improvement methods based on different information sources (data correlation, gradient information) can target various aspects of optimization for L-norm pruning, ultimately addressing the issues of high cost and high complexity under the over-parameterization of neural networks. By implementing effective redundancy identification, the original model's accuracy can be nearly unchanged after pruning.

2.2. Automatic Pruning

Building on structured strategies, automated pruning incorporates automated search and iterative optimization, leading to more diverse strategies and a more flexible, refined pruning process.

A major optimization direction for imposing constraints on the original weights involves combining structured pruning with ADMM to achieve deeper sparsification, by splitting the constrained optimization problem—featuring complex regularization—into two parts: one updates the network weights, and the other updates the variables that represent the sparse structure:

$$\min_{W,Z} \mathscr{L}(W) \text{ subject to } W = Z \in \Omega \tag{1}$$

where $\mathscr{L}(W)$ denotes the training loss of the network, Ω denotes the feasible set for the sparse structure, and Z denotes the pruned (or sparsified) network weights[11]. Building on this framework, Liu et al. introduced an additional "purification" step in the ADMM process: after completing the primary ADMM iterations, they conduct a local readjustment or reactivation check for pruned weights to remove redundant residual weights or correct over-pruning. Lin et al. incorporated ADMM into the Artificial Bee Colony (ABC) algorithm to automatically discover the optimal number of retained channels for each layer, thereby significantly reducing manual hyperparameter tuning while enhancing network compression efficacy[12].

Another optimization direction ains at achieving lower hyperparameter sensitivity and stronger recoverability. Xiao et al. introduce a pruning method in which auxiliary parameters, instead of directly constraining or modifying the original network weights, determine which weights or neurons are pruned[13]. Through a specialized gradient update scheme and a recoverable pruning mechanism, any vitally important weights erroneously pruned can be automatically restored upon discovery. Liu et al. treat network pruning as a process of structure search combined with meta-learning[14]. By making use of a trained Pruning - net to directly generate the weights for a given sub - network, they are able to efficiently adapt to various sub - structures and significantly reduce the necessity for further fine - tuning.

2.3. Unstructured Pruning

Unstructured pruning attempts to remove redundant weight parameters from the tensor without modifying the network structure, making a large neural network sparse, which offers greater flexibility and achieves higher compression ratios. However, unstructured pruning can lead to a non-uniform distribution of zeros in the parameter tensors, and the compression degree of the model is generally not as good as that of structured pruning.

DOI: 10.54254/2755-2721/145/2025.21894

Wang et al. introduce a mask learning module that assigns a mask to each weight in the search space, allowing for the removal of redundant weights[15]. At the same time, layer-wise relevance propagation (LRP) is used as the pruning criterion, evaluating the importance of each neuron through backpropagation and updating the binary masks.

Zhang et al. proposed a joint dynamic pruning algorithm aimed at addressing the issues of model capacity loss and complexity reduction present in traditional static and dynamic pruning methods[16]. During the training process, convolution kernels with lower importance are not immediately permanently pruned but instead set to zero, allowing them to be updated during backpropagation. At the same time, the algorithm dynamically selects the convolution kernels involved in the computation based on the features of the input images.

3. Quantization

By integrating pruning techniques, quantization can be applied to a network that has already been sparsified, ensuring that only the most critical weights are quantized. This synergy allows the model to benefit simultaneously from the reduced computational load of pruning and the low-precision efficiency brought by quantization.

For TinyML applications, Quantization-Aware Training(QAT) and Quantization Guided Training(QGT) are widely applied. QAT improves models' adaptability to low-precision environments by considering the impact of quantization during training, while QGT further introduces optimization mechanisms to achieve lower bit-width quantization[2]. Dan et al. optimizes the quantization strategy by proposing Quantization-Aware Training (QAT)[17]. They introduce quantization operations during the training process to simulate quantized weights and activation values, allowing for the calculation of quantization errors. They optimize the loss function by combining classification loss with quantization loss, ensuring the model's performance in low-precision environments.

Ghamari et al. introduces an innovative method, Quantization Guided Training (QGT), building upon the standard Quantization-Aware Training (QAT) approaches[18]. Additionally, it incorporates customized regularization terms that guide the distribution of DNN weights towards optimizing for reduced quantization errors while maintaining high accuracy. Through QGT, it becomes possible to transparently identify model layers or modules that perform poorly during the quantization process. This feature aids developers in selectively optimizing specific components, thereby further enhancing the overall effectiveness of model compression. When combined with pruning, QGT facilitates a dual optimization where the pruning process removes redundant parameters, and quantization further refines the compressed architecture.

4. Knowledge Distillation

Suwannaphong et al. show that the accuracy of student models[19] still showed a significant decline when size are limited to extreme (64 kb), indicating the limitations of the traditional Knowledge Distillation method in minimal model compression[20]. Combining distillation with pruning can better preserve performance under tight resource constraints.

Deng et al. proposed a Feature-Enhanced Knowledge Distillation-based Object Detection Compression Algorithm(ODCA)[21]. ODCA initially incorporates a coordinate attention mechanism into the intermediate layers of the teacher network to enhance the teacher model's ability to represent foreground object features. Integrating pruning into this approach further streamlines the model by eliminating redundant channels. Another strategy is Relational Knowledge Distillation (RKD)[22], which employs distance-wise and angle-wise distillation losses. When paired with pruning, RKD

DOI: 10.54254/2755-2721/145/2025.21894

retains key relational structure while reducing overall network complexity, yielding a more robust, compact model.

5. Lightweight Neural Network Architectures in TinyML

Lightweight neural network architectures optimize from the perspectives of structural pruning concepts in lightweight architectures, including kernel size and convolutional techniques by employing smaller kernels and unconventional convolution methods. MobileNet[23] enhances the model by increasing the number of channels and enabling independent feature extraction, whereas ShuffleNet[24] improves performance by enhancing the flow of information between channels.

Howard et al. propose MobileNets, an efficient deep learning model for mobile and embedded vision applications. The core of MobileNets lies in depthwise separable convolution, which splits the convolution operation into depthwise convolution and pointwise convolution. Depthwise convolution applies a single convolutional kernel to each input channel, while pointwise convolution applies a 1×1 convolution to combine the outputs of the depthwise convolution. A standard convolutional layer is parameterized by a convolution kernel K of size DK · DK · M · N, where DK represents the size of the convolution kernel, M is the number of input channels, and N is the number of output channels. The output feature map for standard convolution is computed as:

$$G_{k,l,n} = \sum_{i,j,m} K_{i,j,m,n} F_{k+i-1,l+j-1,m} \tag{2}$$

The computational cost is expressed as $D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F$. The output feature map of the depthwise convolution after separation is computed as

$$\widehat{G}_{k,l,n} = \sum_{i,j,m} \widehat{K}_{i,j,m,n} F_{k+i-1,l+j-1,m} \tag{3}$$

In the equation: $\widehat{K}$ is a small kernel of size $D_K \times D_K \times M$, where the M-th kernel in $\widehat{K}$ is applied to the F-th channel, generating the output feature map $\widehat{G}$ for the M-th channel. The amount computed at this time is $D_K \cdot D_K \cdot M \cdot D_F + M \cdot N \cdot D_F$, while the 1×1 kernel is computed as $D_K \cdot D_K \cdot M \cdot D_F \cdot D_F$. By representing the convolution kernel size as $D_K \times D_K$ and the process of merging and filtering, the reduced computation amount is given by

$$D_K \cdot D_K \cdot M \cdot D_F + M \cdot N \cdot D_F = \frac{1}{N} + \frac{1}{D_K^2} \tag{4}$$

MobileNets introduces two hyperparameters: the width multiplier α (which controls the number of input and output channels) and the resolution multiplier ρ (which reduces the resolution of the input image) to further optimize the model:

$$D_K \cdot D_K \cdot \alpha M \cdot D_F \cdot D_F + \alpha M \cdot \alpha N \cdot D_F \cdot D_F \tag{5}$$

The design of MobileNets concentrates 95% of the computation time on 1×1 convolutions, which can be implemented using highly optimized matrix multiplication (GEMM), further enhancing computational efficiency. The 1×1 convolutions do not require memory rearrangement (im2col) and can be directly implemented using GEMM, making them more efficient in practical applications.

Zhang et al. introduce an efficient convolutional neural network (CNN) architecture designed specifically for mobile devices, with the main goal of achieving efficient image classification and object detection under limited computational resources. ShuffleNet uses 1×1 convolutions to fuse information between channels:

$$Y_{i,j,k} = \sum_c X_{i,j,c} \cdot W_{c,k} \tag{6}$$

DOI: 10.54254/2755-2721/145/2025.21894

where Y is the output feature map, X is the input feature map, W is the convolution kernel, i, j are the spatial positions, c is the channel index, and k is the output channel index. Divide the input channels into several groups, and perform convolution operations independently on each group:

$$Y_{g,i,j,k} = \sum_{c \in G_g} X_{g,i,j,c} \cdot W_{g,c,k} \tag{7}$$

in which G_g denotes the channels of the g-th group. After grouped convolution, a channel shuffle operation is performed to disrupt the channel order, ensuring that information can flow between different groups and avoiding the phenomenon of information islands:

$$\text{Shuffle}(Y) = \text{Reshape}(\text{Transpose}(Y)) \tag{8}$$

These two models also represent the design strategies of emerging lightweight neural network architectures. Lightweight Deep Convolutional Neural Network (DCNN) technologies incorporate sparse hierarchical structures and modular designs in their architectural framework[25]. The sparse hierarchical structures reduce redundant parameters by designing networks with sparse connections and efficient hierarchical architectures. Based on MobileNet, MobileNetV2[26] was developed, introducing reverse residual structures and linear bottlenecks. The application of depthwise separable convolutions has also led to the creation of more lightweight models, such as EfficientNet[27], Xception[28], and GhostNet[29], among others. These models significantly reduce computational complexity while maintaining or improving accuracy, making them suitable for mobile devices and resource-constrained environments.

On the basis of ShuffleNet model, Li et al. optimized channel shuffling by enhancing the model's ability to extract features using a residual block structure based on the initial foundation images generated by the subnet reconstruction[30]. Gao et al. proposed a lightweight Multi-branch Aware Super-Resolution Network(MASRN) combining channel shuffling with attention mechanisms[31], where the basic building block is the Multi-branch Aware Module(MAM), which integrates the Channel Shuffle Pixel-wise Attention(CSPA) mechanism to effectively extract hierarchical contextual features.

Similarly, terminal deployment and model optimization are complementary. Lai et al. proposed a deployment-driven model design approach[32]. For ML development and deployment optimized for edge devices, they utilize deployment tools instead of traditional ML frameworks to construct network models. These deployment tools integrate operator libraries of the target platform, ensuring that the designed models are compatible with the deployment environment.

6. Conclusion

This review introduces pruning techniques tailored for resource-constrained devices within the TinyML domain and discusses their synergy with other model compression approaches and their application in designing lightweight AI systems. The inherent model-simplification achieved through pruning, when integrated with complementary compression techniques, fosters the development of compact architectures that enable machine learning models to operate on significantly smaller hardware without compromising functionality. The paper cites a limited number of references, which may not comprehensively cover the latest developments and diverse methods in this field. Consequently, some innovative technologies are not discussed and compared in depth. Building on the existing research, future work could incorporate more empirical studies to validate the actual performance of integrating pruning techniques with other model compression methods through large-scale experiments. At the same time, by incorporating a wider range of literature, a unified and comprehensive evaluation framework for lightweight models could be developed, ensuring a fairer and more precise comparison among techniques such as pruning, quantization, and distillation across various tasks. Additionally, the co-design of algorithms with dedicated hardware will be an important

DOI: 10.54254/2755-2721/145/2025.21894

direction, and future research is expected to offer more customized optimization solutions for large-scale practical deployments.

References

[1] Dutta L & Bharali S (2021). TinyML meets IoT: A comprehensive survey. Internet of Things, 16, 100461.

[2] Ray, P. P. (2022). A review on TinyML: State-of-the-art and prospects. Journal of King Saud University-Computer and Information Sciences, 34(4), 1595–1623.

[3] Jiang, X., Li, Z., & Huang, L., et al. (2022). Review of neural network pruning techniques. Journal of Applied Science, 40(5), 838–849. https://doi.org/10.3969/j.issn.0255-8297.2022.05.013.

[4] Han, S., Pool, J., Tran, J., et al. (2015, December 7–12). Learning both weights and connections for efficient neural networks. In Advances in Neural Information Processing Systems 28 (Vol. 2, pp. 1135–1143). Montreal, Canada: Neural Information Processing Systems.

[5] Li, H., Kadav, A., Durdanovic, I., Samet, H., & Graf, H. P. (2016). Pruning filters for efficient convnets. arXiv preprint arXiv:1608.08710.

[6] Wu, W., Fan, Q., Zurada, J. M., et al. (2014). Batch gradient method with smoothing L1/2 regularization for training of feedforward neural networks. Neural Networks: The Official Journal of the International Neural Network Society, 5072–5078. https://doi.org/10.1016/j.neunet.2013.11.006

[7] Luo, X., Chang, X., & Ban, X. (2016). Regression and classification using extreme learning machine based on L1-norm and L2-norm. Neurocomputing, 174, 179–186.

[8] Cheng, D., Zheng, H., & Chen, J.(2024). Deep convolutional neural network pruning method based on similarity perception. Small Microcomputer System, 45(11), 2656–2662. https://doi.org/10.20009/j.cnki.21-1106/TP.2023-0302

[9] Wang, B. (2023). Convolutional neural network structured pruning with enhanced linear representation redundancy [Doctoral dissertation, Tianjin Normal University].

[10] Wang, J., Ji, F., & Yuan, X. (2022). Structured pruning algorithm based on gradient tracking. Computer simulation, 39(8), 347–355, 414. https://doi.org/10.3969/j.issn.1006-9348.2022.08.067

[11] Liu, N., Ma, X., Xu, Z., et al. (2020, February 7–12). AutoCompress: An automatic DNN structured pruning framework for ultra-high compression rates. In Proceedings of the 34th AAAI Conference on Artificial Intelligence (pp. 4876–4883). Association for the Advancement of Artificial Intelligence.

[12] Lin, M., Ji, R., Zhang, Y., et al. (2021, January 7–15). Channel pruning via automatic structure search. In Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence (Vol. 2 of 8, pp. 669–675). Curran Associates, Inc.

[13] Xiao, X., Wang, Z., & Rajasekaran, S. (2020, December 8–14). AutoPrune: Automatic network pruning by regularizing auxiliary parameters. In Advances in Neural Information Processing Systems 32 (Vol. 18 of 20, 32nd Conference on Neural Information Processing Systems [NeurIPS 2019], pp. 13636–13646). Curran Associates, Inc.

[14] Liu, Z., Mu, H., Zhang, X., et al. (2019, October 27–November 2). MetaPruning: Meta learning for automatic neural network channel pruning. In 2019 IEEE/CVF International Conference on Computer Vision (ICCV) (pp. 3295–3304). Institute of Electrical and Electronics Engineers.

[15] Wang, X., Liu, P., Xiang, S., et al. (2023). Unstructured pruning method based on neural architecture search. Pattern Recognition and Artificial Intelligence, 36(5), 448–458. https://doi.org/10.16451/j.cnki.issn1003-6059.202305005.

[16] Zhang, M., Lu, Q., Li, W., et al. (2021). Deep neural network compression algorithm based on joint dynamic pruning. Computer Application, 41(6), 1589–1596. https://doi.org/10.11772/j.issn.1001-9081.2020121914

[17] Dan, Y. (2023). Research and application of neural network quantitative perception training method [Doctoral dissertation, University of Electronic Science and Technology of China].

[18] Ghamari, S., Ozcan, K., Dinh, T., et al. (2021). Quantization-guided training for compact TinyML models [Preprint]. arXiv. https://arxiv.org/abs/2103.06231.

[19] Hinton, G. (2015). Distilling the knowledge in a neural network [Preprint]. arXiv. https://arxiv.org/abs/1503.02531.

[20] Suwannaphong, T., Jovan, F., Craddock, I., et al. (2024). Optimising TinyML with quantization and distillation of transformer and mamba models for indoor localisation on edge devices [Preprint]. arXiv. https://arxiv.org/abs/2412.09289

[21] Deng, X., Wang, Y., Luo, J., et al. (2024). Target compression algorithm based on feature enhanced knowledge distillation. Sensors and Microsystems, 43(5), 116–120. https://doi.org/10.13873/J.1000-9787(2024)05-0116-05.

[22] Park, W., Kim, D., Lu, Y., et al. (2019). Relational knowledge distillation. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 3967–3976). IEEE.

DOI: 10.54254/2755-2721/145/2025.21894

[23] *Howard, A. G. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications [Preprint]. arXiv. https://arxiv.org/abs/1704.04861*

[24] *Zhang, X., Zhou, X., Lin, M., et al. (2018). Shufflenet: An extremely efficient convolutional neural network for mobile devices. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 6848–6856). IEEE.*

[25] *Chen, F., Li, S., Han, J., Ren, F., & Yang, Z. (2024). Review of lightweight deep convolutional neural networks. Archives of Computational Methods in Engineering, 31(4), 1915–1937.*

[26] *Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L.-C. (2018). MobileNetV2: Inverted residuals and linear bottlenecks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 4510–4520).*

[27] *Tan, M., & Le, Q. (2019). EfficientNet: Rethinking model scaling for convolutional neural networks. In International Conference on Machine Learning (pp. 6105–6114). PMLR.*

[28] *Chollet, F. (2017). Xception: Deep learning with depthwise separable convolutions. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 1251–1258).*

[29] *Han, K., Wang, Y., Tian, Q., Guo, J., Xu, C., & Xu, C. (2020). GhostNet: More features from cheap operations. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 1580–1589).*

[30] *Li, W., Li, S., & Liu, R. (2020, October 25–28). Channel shuffle reconstruction network for image compressive sensing. In 2020 IEEE International Conference on Image Processing (ICIP) (pp. 2880–2884). Institute of Electrical and Electronics Engineers.*

[31] *Gao, X., Xu, L., Wang, F., et al. (2023). Multi-branch aware module with channel shuffle pixel-wise attention for lightweight image super-resolution. Multimedia Systems, 29(1), 289–303.*

[32] *Lai, L., & Suda, N. (2018). Rethinking machine learning development and deployment for edge devices [Preprint]. arXiv. https://arxiv.org/abs/1806.07846*

[33] *Han, L., Xiao, Z., & Li, Z. (2024). DTMM: Deploying TinyML models on extremely weak IoT devices with pruning [Preprint]. arXiv. https://arxiv.org/abs/2401.09068*