AtomNet: Designing Tiny Models from Operators Under Extreme MCU Constraints

Zhiwei Dong[1,3*], Mingzhu Shen[3*], Shihao Bai[3], Xiuying Wei[3], Jinyang Guo[2], Ruihao Gong[2, 3], Song-Lu Chen[1], Xianglong Liu[2], Xu-Cheng Yin[1]

[1]University of Science and Technology Beijing
[2]Beihang University
[3]SenseTime Research

dongz.cn@outlook.com, m.shen23@imperial.ac.uk, baishihao@sensetime.com, w2295297125@163.com
jinyangguo@buaa.edu.cn, xhplus@163.com, songluchen@ustb.edu.cn, xlliu@nlsde.buaa.edu.cn, xuchengyin@ustb.edu.cn

Abstract

Tiny machine learning (TinyML) has attracted heightened attention for its ability to provide low-cost and instantaneous performance on edge devices. Particularly, the commonly used microcontroller unit (MCU) imposes extreme constraints on peak memory (SRAM) and storage (Flash). Existing TinyML methods often rely on customized and hard-to-obtain inference libraries, as well as necessitate a time-consuming search for a deployable architecture using advanced Neural Architecture Search (NAS) algorithms. To solve these problems, we fully exploit the resources in the MCU and deduce hardware-oriented guidelines for designing models under extreme MCU constraints. In detail, we delve into thorough information about the atom operators by collecting the runtime data of Flash, SRAM, and latency to build a dataset named AtomDB. Based on AtomDB, several critical operator guidelines are established to fully utilize limited Flash and SRAM, while minimizing latency. By transferring the guidelines to analyze blocks, we propose a hybrid pattern that organizes appropriate blocks at different network stages to form the AtomNet, a more hardware-oriented architecture, to handle the former SRAM bottleneck and the latter Flash bottleneck. Extensive experiments demonstrate the effectiveness of the exploitation of the hardware characteristics. Remarkably, AtomNet pioneeringly achieves **3.5%** accuracy enhancement and more than **15%** latency reduction on 320KB MCU using readily available official inference libraries for ImageNet tasks, surpassing the current state-of-the-art method.

Introduction

IoT devices are widely used in various applications such as personal health care and smart home. One core component of these devices is the microcontroller units (MCU), which have limited computation resources including peak memory (SRAM) and storage (Flash). For instance, the STM32F412 offers a maximum support of 256KB SRAM and 1MB Flash. Therefore, popular efficient models (Tan and Le 2019; Howard et al. 2017; Sandler et al. 2018; Howard et al. 2019; Wu et al. 2019; Han et al. 2020; Mehta and Rastegari 2021; Dosovitskiy et al. 2020; Liu et al. 2021) suffer difficulties to be deployed on IoT devices, and pursuing desired models with high accuracy on MCU still remains a big challenge.

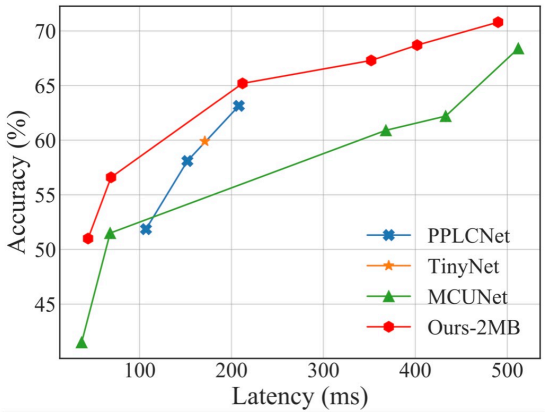


Figure 1: Latency and Top-1 accuracy comparisons among different models on ImageNet dataset. Latency is tested on STM32H743 (512KB SRAM/2MB Flash).

Recently, researchers attend more attention to model design under special constraints on MCU. MCUNet (Lin et al. 2020) advises a system-algorithm co-design framework and an optimized resolution-width search space to search the network architecture. MCUNetV2 (Lin et al. 2021) proposes to solve the peak memory limitation of MCU with a patch-based inference framework. While their works already achieve practical performance, their models are derived from a coarse-grained net-level analysis with high-cost NAS methods, which can not fully utilize the resources like the SRAM or Flash on MCU, hindering more efficient and effective architecture design.

To this end, we propose to fully utilize hardware potential and introduce a hardware-oriented model design framework. Unlike existing methods from the network perspective, we aim to dive into each individual operator such as convolution and relu, by collecting massive runtime data and making in-depth analyses. We have considered all operators that are suitable for MCUs, ensuring that the space from which

*These authors contributed equally to this work.

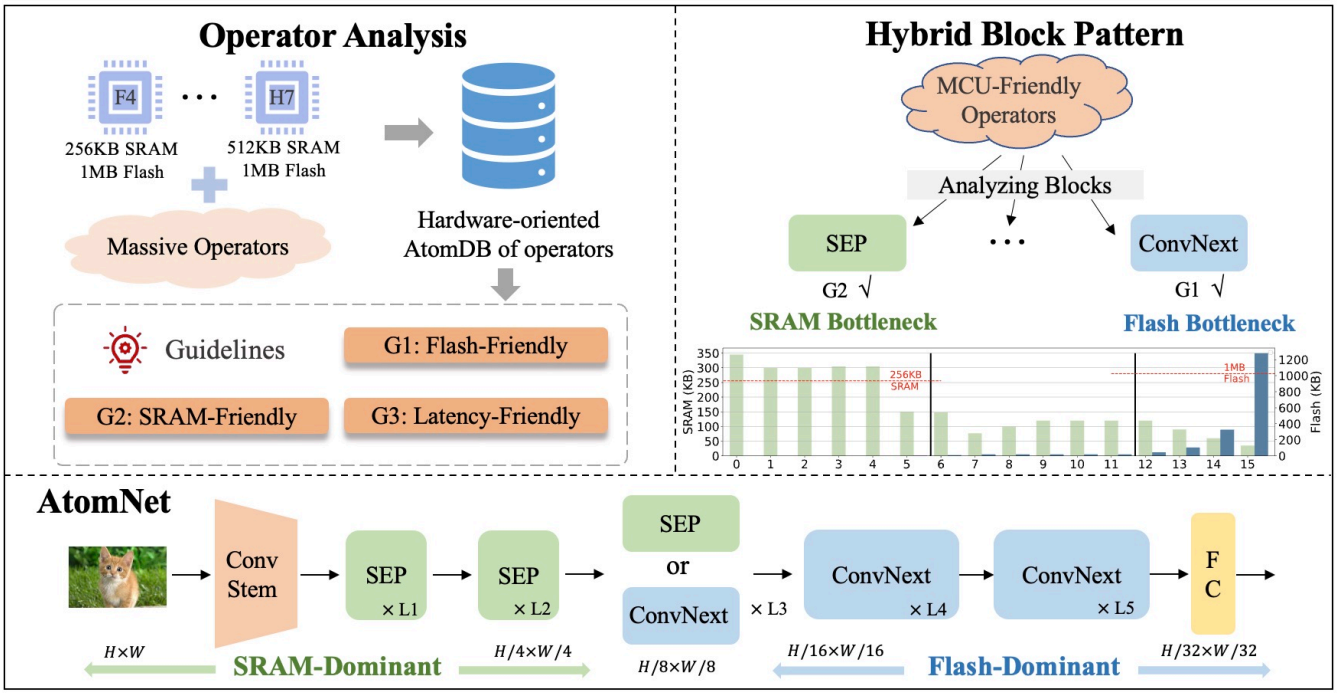


Figure 2: The overall framework of our method. The first part involves a database named AtomDB, collecting hardware information of massive operators on MCU. Guidelines about the Flash, latency, and SRAM are concluded from the database to instruct the choice of blocks. Based on these findings, the second part of the framework illustrates an AtomNet with a hybrid block pattern, which arranges SEP (Howard et al. 2017) and ConxNext (Liu et al. 2022) at different stages of a network.

we collected data includes the maximum set of operators we confirmed to be runnable. This proposal is also supported by the serial working mode of MCU, ensuring the accurate expansion of information from operators to higher block level, then to stages, and finally to the network. For example, at the block level, we have operators such as convolution and fully connected, including blocks like Separable (SEP, DepthWise convolution and PointWise convolution). Moving to the stage level, we consider the collection of blocks with the same resolution, e.g., $\frac{input\_H}{4} \times \frac{input\_W}{4}$ and at the network level, we encompass the final model.

In detail, a database dubbed AtomDB with rich and accurate information is established, an example is shown in Table 1. To efficiently and precisely accomplish this construction process, we meticulously formulate several strategies and make great efforts to accelerating measurements. Leveraging the operator-level insights offered by AtomDB, we introduce three carefully crafted guidelines for optimizing MCU-friendly neural networks in terms of Flash, SRAM, and latency, in order to fully exploit hardware resources in extreme MCU constraints.

Furthermore, we apply the operators insights to analyze blocks, a surprising finding is that no block behaves the best on all the metrics, motivating us to mingle different blocks to wisely exploit the resources of MCU. To tackle SRAM and Flash bottlenecks in different stages of a model, we recommend a hybrid block pattern to maximize the utilization of hardware resources. Finally, we employ a simple zero-cost NAS method to search for AtomNets in a hardware-oriented search space. Our proposed AtomNets can be applied to diverse tasks and datasets, and suit well for various MCU like 256KB, 320KB, and 512KB SRAM limits and diverse inference libraries like the TinyEngine (Lin et al. 2020) and STM32 X-CUBE-AI (STMicroelectronics 2023). More essentially, the overall framework from operators to networks shall shed new light on the research of efficient network design, especially for resource-limited devices like MCU, as we have achieved a **3.5%** increase in accuracy while reducing latency by **15%** on ImageNet tasks. Our contributions can be summarized as follows:

- We propose a hardware-oriented model design framework based on the construction of a massive database, AtomDB. Leveraging the operator-level insights from AtomDB, we propose three guidelines for optimizing operators in terms of Flash, SRAM, and latency, fully leveraging the limited resources of MCUs.
- By transferring the guidelines to analyze blocks, a hybrid block pattern is deduced to construct the final searched network AtomNet. With this pattern, the SRAM bottleneck in the former stages of a network as well as the Flash bottleneck in the later stages can be dealt with more effectively.
- Experiments on various datasets (ImageNet (Deng et al. 2009), VWW (Chowdhery et al. 2019), and Pascal VOC (Everingham et al. 2010)) and tasks (classifica-

tion and object detection) prove the superiority of our AtomDB and AtomNet. For the first time, we push the ImageNet performance to **73.0%** and VWW performance to **95.0%** on MCU with 512KB SRAM.

Related Work

Efficient models. In recent years, several model compression techniques including pruning (Han, Mao, and Dally 2016; He et al. 2018; Liu et al. 2019), quantization (Esser et al. 2019; Gong et al. 2019; Jacob et al. 2018; Li et al. 2021), network architecture search (Tan et al. 2019; Cai, Zhu, and Han 2019; Cai et al. 2020; Shen et al. 2021) were proposed to optimize efficient models (Howard et al. 2017; Sandler et al. 2018; Tan and Le 2019; Howard et al. 2019). For example, MobileNets (Howard et al. 2017; Sandler et al. 2018; Howard et al. 2019) utilize separable convolution (SEP) to reduce the computation of the normal convolution. TinyNet is (Han et al. 2020) devoted to downsizing neural architectures by twisting three dimensions of depth, resolution, and width. PPLCNet (Cui et al. 2021) and MobileOne (Vasu et al. 2022) avoid using the shortcut (He et al. 2016) design inside the block to improve model efficiency on CPU. However, these methods either exceed the SRAM/Flash constraint (Han et al. 2020) or achieve poor practical acceleration on MCU due to the neglect of its hardware characteristics. In contrast, our AtomDB and AtomNet are designed to achieve practical acceleration by analyzing the property of each operator when deploying on MCU.

MCU-based network. To deploy deep learning models on MCU, many approaches were proposed to satisfy extreme peak memory (SRAM) and storage (Flash) limits for TinyML. MCUNet (Lin et al. 2020) first proposed to design network architectures with customized hardware inference libraries and also proposed a system-algorithm co-design approach under extreme constraints. MCUNetV2 (Lin et al. 2021) customized a patch-based inference framework to solve the problem of memory bottleneck in the early stages while resulting in slower inference speed. Besides, they both employ very costly NAS algorithms and rely on latency feedback at the network level. In contrast, our method, with the assistance of the accurate and fine-grained information in AtomDB, produces models more quickly and at a lower cost while maintaining excellent accuracy.

Methods

In this section, we present our proposed network design framework by boiling down the problem of a network to the fine-grained operator analysis regarding Flash, SRAM, and latency with a systematic database AtomDB. Equipped with this database, several MCU-friendly guidelines are concluded to guide the mixture of hybrid block patterns and finally inspire the design of our proposed AtomNet.

Problem Definition

Objective. To design deployable models on MCU, SRAM, and Flash also needed to be considered along with latency due to the special constraints on MCU. We formulate the overall objective as follows:

$$\begin{aligned} &\max\ ACC(m) \\ &\text{s.t.}\quad \text{latency} < t, \quad \text{SRAM} < s, \text{Flash} < f. \end{aligned} \tag{1}$$

Given the upper limit of latency, SRAM, and Flash as t, s, and f, we aim to obtain a model m with maximum ACC.

Actual measurements. The rough estimation might lead to a certain disparity with the real measurement that hinders the practical deployment of selected architectures. In theory, SRAM and Flash are largely affected by the amount of peak activation and total parameters respectively. While in reality, the real deployment can bring extra SRAM consumption due to the memory scheduling mechanism and can introduce extra consumption of SRAM known as the scratchpad. Besides, extra Flash costs are non-neglectable due to the need for storing the operating system and inference library. Thus, compared to inaccurate approximation, the SRAM and Flash are still required to be evaluated on MCU to rigorously reach the runnable requirement.

For latency, apart from obtaining the real data of each operator on MCU, we argue that analyzing the ratio of MACs and latency called the **computation efficiency** defined in Equation (2) can reflect more intrinsic information of operators, which is also discussed in the Roofline (Williams, Waterman, and Patterson 2009) model.

$$\text{Computation Efficiency} = \frac{\text{MACs}}{\text{latency}}. \tag{2}$$

Investigating Equation (2), we know that with the same latency requirement, operators which have larger computation efficiency can be allocated more MACs. And MACs (FLOPs) are often adopted to indicate the potential for accuracy (Lin et al. 2020; Krishnakumar et al. 2022). Namely, by finding operators with high computation efficiency, we can enjoy large MACs with low latency and thus also provide higher accuracy potential.

Framework Overview

Most existing works only employ hardware information as a black-box reward to solve the problem defined in Equation (1). However, we endeavor to measure massive operators on MCU and provide white-box insights from the latency, Flash, and SRAM aspects instead of focusing on searching with complicated NAS methods. To achieve this objective, we begin with a careful investigation of single operators which are actually executed on the inference engine after fusion and optimization (e.g. ConvBNRelu), then collect their latency, SRAM, and Flash on real hardware. Guided by this knowledge, the whole network can be better constructed to maximize resource utilization of MCU and thus enjoy preferable accuracy.

To fully exploit hardware characteristics and deduce hardware principles, we propose to design tiny models from operators with our proposed framework as shown in Figure 2. **Opeator Analysis**: Through massive operator analysis based on the construction of AtomDB, guidelines can be deduced from Flash, SRAM, and latency aspects. **Hybrid Block Pattern**: With these guidelines, we are motivated to conduct block analysis to tackle resource-limited

type	groups	c_in	h_in	w_in	k1	k2	c_out	h_out	w_out	Flash	MACs	query latency	test latency
Pad	-	32	64	64	-	-	32	66	66	0B	0	0.255ms	0.254ms
Conv2d	32	32	66	66	3	3	32	64	64	416B	1179680	12.852ms	12.828ms
Conv2d	1	32	64	64	1	1	32	64	64	1152B	4194336	12.134ms	12.132ms

Table 1: Comparison of profiling the entire block directly versus querying each operator from AtomDB. The c_in, h_in, w_in, c_out, h_out, w_out, and k1, k2 represent the input, output and kernel channels, height, width respectively.

Block	Width	Depth	Kernel size	Expansion Ratio	SE	Stride
Conv	{16, 24, 32, 48, 64}	1	3	-	N	2
SEP	{16, 24, 32, 48, 64}	1	3	1	N	1
SEP	{32, 48, 64, 80, 96, 112}	1, 2	3	1	N	2
SEP or ConvNext	{48, 64, 80, 96, 112, 128}	1, 2	3	1,3,6	N	2
ConvNext	{48, 64, 80, 96, 112, 128, 144}	2, 4, 6	3	6	Y	2
ConvNext	{64, 80, 96, 112, 128, 144, 160}	1, 2	3	6	Y	2

Table 2: The micro and macro search space are defined as follows. The block pattern is fixed with hybrid block pattern. N/Y refers to not have/have SE in the block. The choice of the input resolution is {256, 224, 192, 160, 128, 92, 80, 64, 48}.

bottlenecks occurring in different stages of a model and thus propose the hybrid block pattern. **AtomNet**: Therefore, a hardware-oriented search space is proposed, and we employ simple-yet-effective MACs metric as the accuracy indicator to search for a series of AtomNets that maximize the utilization of hardware resources.

Opeator Analysis

AtomDB Construction Unlike LUT (Dai et al. 2019; Wu et al. 2019), which is composed by recording limited operators in the search space, AtomDB measures massive operators on the MCU and plays a foundation role in the framework. We build a database to collect the necessary information about every single operator and adopt several techniques to optimize the construction process efficiently. An **Atom operator** is defined to be a fused kernel in the inference engine to be compatible with the compiler optimization, such as Convolution fused with an activation function. As the atom operator is the smallest part of a network and contributes to the latency independently, the database is called AtomDB.

Kernel Additivity. AtomDB highly relies on kernel additivity and we verify that the deviation between the sum of all operators and directly testing a model as a whole is less than 0.1% for MCU. Here, we also provide a specific example to validate this point. Taking a SEP block as an example, the results of using these attributes to query in AtomDB and directly test are shown in Table 1. It can be noticed that the test latency of these three atoms alone is nearly the same as that in the block query with a negligible difference. This proves that the latency of a single atom is additive.

Guidelines from AtomDB The rich and accurate information in AtomDB sheds insights into architecture design. Here, we mainly conclude three useful guidelines for operators on MCU from the Flash, SRAM, and latency perspectives. All the ">" symbols represent "better than".

- **G1** *(Flash aspect): Choose Flash-friendly operators (DW > PW) and ignore NC.*
  The Flash of NC shows linearity to the square of channels, which can be easily observed.. Thus, taking NC is not wise on Flash-restricted devices as it can easily occupy a large amount of Flash.
- **G2** *(SRAM aspect): Reduce or avoid using Mul or Add operator behind features with large resolution due to their exponential increase in SRAM usage.*
  Based on the principles of SRAM consumption, the characteristics of different operators can be analyzed. Specifically, operators like Mul and Add inherently cause SRAM costs that are double the size of the input feature map.
- **G3** *(latency aspect): Allocate more MACs to better computation efficiency operators.*
  To achieve token-mixing and channel-mixing (Yu et al. 2022), DW and PW or NC shall be chosen. However, **G1** has mentioned that it is not wise to adopt NC due to Flash. Since PW has better computation efficiency than DW (256k MACs / ms > 91k MACs / ms), allocating more MACs to PW appears to be a promising way to achieve better latency.

Hybrid Block Pattern

Block analysis. Following many works like (He et al. 2016) that consider the block unit when constructing the model, we intend to expand our thorough analysis of atom operators from AtomDB to blocks. By reviewing several common schemes, we first exclude blocks with normal convolution under the guideline **G1** and then consider 3 types of blocks with and without SE (Hu, Shen, and Sun 2018). The SE module is often employed as an additional module to blocks, particularly in tiny models (Howard et al. 2019; Cui et al. 2021) because of its lightweight and superiority in accuracy.

For Flash, different designs bring extra distinct non-negligible Flash for the last classification head and the SE module. The Flash of the last classification head is the least when adopting ConvNext and MBConv. Besides, ConvNext

Model / Library	Quant	MACs	SRAM	Flash	Top-1	STM F4	TE F7	STM F7	STM H7
STM32F412 (256KB SRAM, 1MB Flash)									
MbV1 0.5× (r=192)	mixed	110M	<256KB	<1MB	60.2%	-	-	-	-
MCUNet-256KB	int8	68M	238KB	1007KB	60.3%	OOM	681ms	803ms	368ms
MCUNet	int4	134M	233KB	1008KB	62.0%	-	-	-	-
Ours-256KB	int8	80M	232KB	981KB	**64.0%**	2124ms	-	**626ms**	275ms
MCUNetV2-M4	int8	119M	196KB*	1010KB	64.9%	-	-	-	-
Ours-256KB-Max	int8	110M	252KB	1019KB	**66.7%**	3585ms	-	1030ms	405ms
STM32F746 (320KB SRAM, 1MB Flash)									
MbV2 0.35× (r=144)	int8	24M	308KB	862KB	49.0%	OOM	-	276ms	123ms
Proxyless 0.3× (r=176)	int8	38M	292KB	892KB	56.2%	OOM	-	468ms	209ms
MCUNet-320KB	int8	82M	293KB	897KB	61.8%	OOM	819ms	980ms	433ms
Ours-320KB	int8	84M	240KB	978KB	**64.6%**	-	-	813ms	**367ms**
STM32H743 (512KB SRAM, 2MB Flash)									
MbV1 0.75× (r=224)	mixed	317M	<512KB	<2MB	68.0%	OOM	-	-	-
MCUNet	int8	126M	452KB	2014KB	68.5%	OOM	-	OOM	521ms
MCUNet	int4	474M	498KB	2000KB	70.7%	OOM	-	-	-
Ours-512KB	int8	163M	370KB	2017KB	**70.8%**	OOM	-	OOM	**490ms**
MCUNetV2-H7	int8	256M	465KB*	2032KB	71.8%	OOM	-	-	-
Ours-512KB-Large	int8	233M	469KB	2027KB	**72.2%**	OOM	-	OOM	836ms
Ours-512KB-Max	int8	362M	469KB	1925KB	**73.0%**	OOM	-	OOM	1188ms

Table 3: Performance comparison of different methods such as MBV1 (Howard et al. 2017), MBV2 (Sandler et al. 2018), Proxyless (Cai, Zhu, and Han 2019), MCUNet (Lin et al. 2020) and MCUNetV2 (Lin et al. 2021) when deploying on different hardware. OOM means out of memory. The latency information of some methods is not reported because these models are not open-sourced or need a customized inference library (the inference library is not open-sourced) or require int4 quantization (not supported by the hardware used in our experiment). "STM" represents STM32 X-CUBE-AI inference library (STMicroelectronics 2023). "TE" represents TinyEngine (Lin et al. 2020, 2021). We only use the first 2 characters to represent a hardware model. For example, "STM F4" represents STM32F412. "*" means the SRAM is optimized using patch-based inference.

with the SE module still behaves the best with the smallest Flash. Therefore, ConvNext is the most Flash-friendly block on MCU given the rank bellow:

$$\text{Flash} : ConvNext > MBConv > SEP \tag{3}$$

For SRAM, according to **G2**, SE is not suggested when the resolution is large, and also validates quantitatively that adopting SE modules inside the block hurts SRAM. Without the SE module, it can be easily found that SEP requires less SRAM than ConvNext and MBConv. So, the following rank is suggested:

$$\text{SRAM} : SEP > ConvNext > MBConv \tag{4}$$

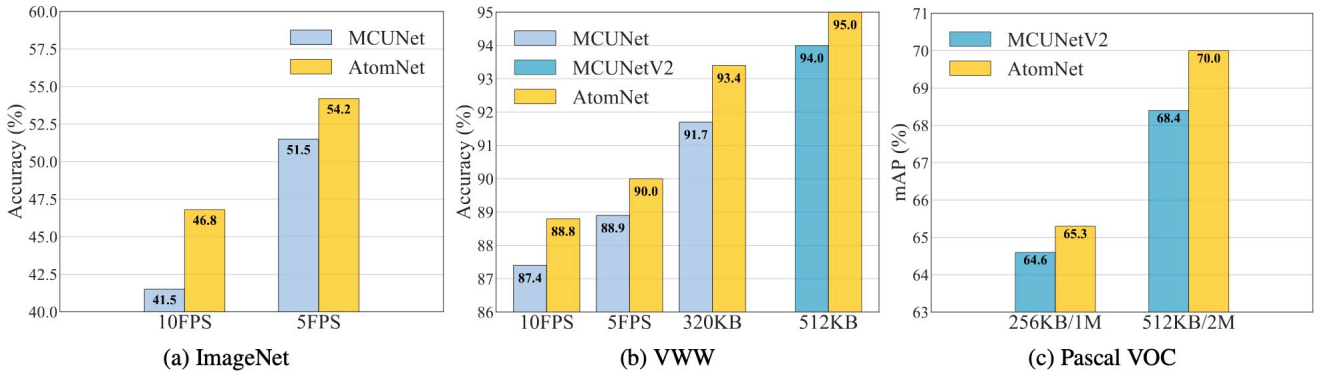
For latency, **G3** advises putting more MACs to PW and decreasing the MACs ratio of DW and PW. When these blocks are not combined with the SE module, ConvNext behaves the best armed with a smaller ratio of DW/PW and better exploitation of computation efficiency. And the SE modules will scale the latency of these blocks equivalently. Therefore, we derive the following rank:

$$\text{latency} : ConvNext > MBConv > SEP \tag{5}$$

Bottleneck analysis. Existing methods have recognized that efficient models encounter different bottlenecks that hinder the deployment of these models. Diving into the features of a network on MCU, we discover the Flash bottleneck in the later stages apart from the imbalanced memory bottleneck (Lin et al. 2021) as shown in Figure 2. As the SRAM and Flash distribution of a typical efficient model (Cui et al. 2021) indicate, the early stages are greatly limited by SRAM because of the large resolution and large feature map, whereas the later stages are mainly restricted by Flash with the increasing number of channels in later several blocks.

Hybrid block pattern. The block analysis indicates that no block holds the most ideal situation on latency, SRAM, and Flash (e.g., ConvNext presents the best performance on latency but does not rank first on SRAM). Considering the special properties of different MCU hardware bottlenecks, adopting hybrid blocks appears to be a more sensible way which can take advantage of different blocks to fulfill MCU resources as much as possible.

With different key constraints at different stages, we shall arrange distinct blocks. For the SRAM-dominant stages, it is more advisable to select the SEP block without the SE module since it requires the least SRAM. For the Flash-dominant stages, we recommend taking ConvNext as it is the most Flash-friendly block. In this way, a hybrid block pattern can be confirmed by taking SEP in the first stage and ConvNext in the last two stages to better handle the bottleneck.



(a) ImageNet (b) VWW (c) Pascal VOC

Figure 3: Performance comparison of our AtomNet and MCUNet on different datasets under different constraints.

AtomNet

Search space. When designing a model from operators and blocks, the search space can be divided into macro search space with a choice of resolution, depth, and width as suggested in (Cai, Zhu, and Han 2019; Radosavovic et al. 2020; Cai et al. 2020), and micro search space with the choice of the hybrid block pattern as shown in Table 2. With AtomDB, we can search for a series of models with detailed information on SRAM, Flash, and latency on MCU coupled with a zero-cost NAS method.

Optimization Algorithms. Unlike existing NAS methods that require extensive training to estimate accuracy, we aim to search for models using a zero-cost metric to streamline the process. As discussed in (Krishnakumar et al. 2022), MACs serve as an effective zero-cost proxy for accuracy. Moreover, it generalizes well across various search spaces, enabling us to select models with the highest MACs within our search scope while adhering to latency constraints.

Experiments

Experimental Settings

Training configurations. For the ImageNet dataset, we use the SGD optimizer for optimization. The initial learning rate, the weight decay, and the batch size are set as 0.4, $3e^{-4}$, and 1024, respectively. Cosine learning rate decay is adopted in the training process. For the VWW dataset, we finetune the network trained on the ImageNet for 30 epochs as the evaluation model. For the PASCAL VOC dataset, we train the network for 300 epochs with the AdamW optimizer (Loshchilov and Hutter 2017).

Deployment. Following (Lin et al. 2020), we quantize our AtomNet to INT8 and use STM32 X-CUBE-AI inference library when deploying them on different hardware: STM32F412 (256KB SRAM/1MB Flash), STM32F746 (320KB SRAM/1MB Flash) and STM32H743 (512KB SRAM/2MB Flash).

Comparison on Image Classification

AtomNet pushes the limits of ImageNet performance under the MCU constraints. As shown in Table 3, AtomNet achieved 66.7% Top-1 accuracy on STM32F412 (256KB SRAM/1MB Flash) and 73.0% Top-1 accuracy on STM32H743 (512KB SRAM/2MB Flash), which outperforms the current state-of-the-art method MCUNetV2 by 1.5% and 1.2% under the corresponding constraint.

AtomNet achieves better latency-accuracy trade-off. As shown in Table 3, compared with the MCUNet series, MobileNetV2 0.35×, and ProxylessNAS 0.3×, we have achieved significant accuracy improvements in all these three hardware for 256KB, 320KB, and 512KB. For the 10FPS and 5FPS settings, **??** demonstrates that we improve by a large margin with 5.3% and 2.8%. Compared with the time budget of MCUNet 256KB (368ms, 60.3%), our model not only takes less time (275ms) but also improves by 3.7% in accuracy. And we also witness the same trend in comparison with MCUNet-512KB (521ms, 68.5%), our result has 70.8% with only 490ms latency (Table 3). Besides, in Fig. 1, we fairly compared with several recently proposed models including PPLCNet (Cui et al. 2021) series and TinyNet-E (Han et al. 2020) under the 2MB Flash setting. It can be seen that ours-2MB results achieve a better Latency-Accuracy tradeoff which further verifies the effectiveness of our proposed method.

Comparison on VWW and Object Detection

Visual Wake Words (VWW) is an essential task for the MCU platform, which aims to detect whether a person appears in an image. We mainly compare our method with the baseline method MCUNet (Lin et al. 2020) since MCUNetV2 (Lin et al. 2021) is not open-sourced. As illustrated in **??**, AtomNet outperforms MCUNet at 10FPS, 5FPS, and 320KB SRAM settings by 1.4%, 1.1% and 1.7%, respectively. It is worth noting that our largest model achieves the highest accuracy of 95.0% for the VWW task, compared with MCUNetV2 94.0%.

To further verify the generalization ability of our AtomNet for different tasks, we evaluate our AtomNet under the object detection task on Pascal VOC. As shown in **??**, we outperform MCUNetV2 with an accuracy improvement of 0.7% and 1.6% mAP under the SRAM 256KB and 512KB limits, respectively.

	Constraints	Guidelines	MACs	Flash	SRAM	Latency	Top-1
ResNet (He et al. 2016)	Flash (G1)	☹	483M	3048KB	-	-	61.6%
Ours-256KB	Flash (G1)	☺	80M	**981KB**	232KB	275ms	**64.0%**
TinyNet-E (Han et al. 2020)	SRAM (G2)	☹	25M	1950KB	270KB	171ms	59.9%
TinyNet-Modify	SRAM (G2)	☺	30M	1990KB	**217KB**	174ms	**60.7%**
MCUNet-256KB (Lin et al. 2020)	latency (G3)	☹	68M	1007KB	238KB	368ms	60.3%
Ours-256KB	latency (G3)	☺	80M	981KB	232KB	**275ms**	**64.0%**

Table 4: Guidelines studies with constraints on Flash, SRAM, and latency. ResNet represents the modified version of ResNet-18, which has 1/2 number of channels and all the other settings are the same as the original ResNet-18 (He et al. 2016). TinyNet-Modify is redesigned by us according to the guidelines in this paper.

Model	SRAM	Flash	Latency	Top-1	Cost
MBConv	238KB	1007KB	368ms	60.8%	-
ConvNext	233KB	1005KB	374ms	63.8%	∼1h
SEP	240KB	1002KB	403ms	65.6%	∼1h
Hybrid	252KB	1019KB	405ms	**66.7%**	∼1h
Hybrid-EA	245KB	1001KB	**385ms**	66.6%	∼100h

Table 5: Comparison of models using single block pattern and hybrid block pattern. For MBConv, we directly use the model MCUNet-256KB. Other models are searched with constraints. EA (Real et al. 2019) are used.

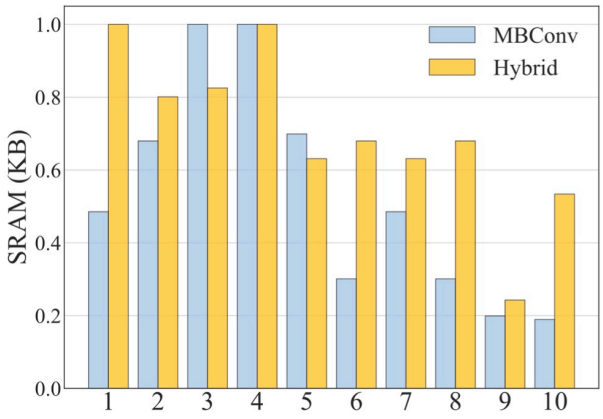


Figure 4: SRAM usage benefit brought by hybrid block pattern. SRAM usage becomes more balance cross the layers.

Ablation Study

Effectiveness of guidelines. To demonstrate the effectiveness of our guideline, in Table 4, we compare the models designed with and without using our guideline. **For Flash (G1)**, we can conclude that Normal Convolution (NC) based networks (e.g., ResNet in Table 4) easily exceed the limits of Flash. By utilizing Depth-Wise Convolution (DW), we have a significant reduction in Flash and also achieve higher accuracy. **For SRAM (G2)**, TinyNet-E (Han et al. 2020) is built based on the MBConv block with SE in the SRAM-dominant stages. Therefore, by replacing the MBConv block with SEP without SE in the SRAM-dominant stage, the SRAM can be reduced. **For Latency (G3)**, the main difference between MCUNet-256KB and our-256KB model is that we tend to use more ConvNext blocks when satisfying the SRAM and Flash constraints. On the other hand, MCUNet-256KB only uses MBConv blocks. Therefore, we show superiority over MCUNet-256KB by a large margin in terms of latency.

Effectiveness of hybrid block pattern. To validate the effect of our hybrid block pattern, in Table 5, we compare the performance of our AtomNet with hybrid block pattern with the models only consisting of MBConv, SEP, ConvNext. Under comparable constraints, our AtomNet with the hybrid block pattern achieves higher accuracy. However, the model with only MBConv or ConvNext needs to reduce the channel in the SRAM-dominant stages, which degrades the performance. In Figure 4, we visualize the improvement of our proposed hybrid block in SRAM-dominant stages compared with a single block pattern.

Discussion with NAS methods. Our work focuses on the MCU with limited SRAM and Flash and uses hardware-oriented principles for better model design which is orthogonal to common NAS algorithms. With a simple zero-cost metric, we can even outperform the SOTA MCUNets by a large margin. Equipped with an advanced NAS method (Real et al. 2019) involving training, we can further enjoy the merits and boost the performance with a 5% reduction in latency (Tab. 5). However, we do not apply this as a default setting due to its high cost. The results verify that the white-box knowledge mined from hardware analysis and MCU-friendly design patterns enable us to search for models with less requirement on complex NAS methods.

Conclusion

With the help of AtomDB and operator-level guidelines to analyze blocks, we propose a hybrid block pattern under extreme MCU resource constrain. Further boosted by a simple hardware-oriented search strategy, AtomNet improves hardware utilization and computing efficiency. Our method results in state-of-the-art performance, pushing the limits of accuracy performance for classification tasks and object detection tasks under TinyML setting.

References

Cai, H.; Gan, C.; Wang, T.; Zhang, Z.; and Han, S. 2020. Once for All: Train One Network and Specialize it for Efficient Deployment. In *ICLR*.

Cai, H.; Zhu, L.; and Han, S. 2019. ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware. In *ICLR*.

Chowdhery, A.; Warden, P.; Shlens, J.; Howard, A.; and Rhodes, R. 2019. Visual wake words dataset. *arXiv preprint arXiv:1906.05721*.

Cui, C.; Gao, T.; Wei, S.; Du, Y.; Guo, R.; Dong, S.; Lu, B.; Zhou, Y.; Lv, X.; Liu, Q.; et al. 2021. PP-LCNet: A Lightweight CPU Convolutional Neural Network. *arXiv preprint arXiv:2109.15099*.

Dai, X.; Zhang, P.; Wu, B.; Yin, H.; Sun, F.; Wang, Y.; Dukhan, M.; Hu, Y.; Wu, Y.; Jia, Y.; et al. 2019. Chamnet: Towards efficient network design through platform-aware model adaptation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 11398–11407.

Deng, J.; Dong, W.; Socher, R.; Li, L.-J.; Li, K.; and Fei-Fei, L. 2009. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR*.

Dosovitskiy, A.; Beyer, L.; Kolesnikov, A.; Weissenborn, D.; Zhai, X.; Unterthiner, T.; Dehghani, M.; Minderer, M.; Heigold, G.; Gelly, S.; et al. 2020. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*.

Esser, S. K.; McKinstry, J. L.; Bablani, D.; Appuswamy, R.; and Modha, D. S. 2019. Learned step size quantization. *arXiv preprint arXiv:1902.08153*.

Everingham, M.; Van Gool, L.; Williams, C. K.; Winn, J.; and Zisserman, A. 2010. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2): 303–338.

Gong, R.; Liu, X.; Jiang, S.; Li, T.; Hu, P.; Lin, J.; Yu, F.; and Yan, J. 2019. Differentiable soft quantization: Bridging full-precision and low-bit neural networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 4852–4861.

Han, K.; Wang, Y.; Zhang, Q.; Zhang, W.; Xu, C.; and Zhang, T. 2020. Model Rubik's Cube: Twisting Resolution, Depth and Width for TinyNets. *Advances in Neural Information Processing Systems*, 33.

Han, S.; Mao, H.; and Dally, W. J. 2016. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. In *ICLR*.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep Residual Learning for Image Recognition. In *CVPR*.

He, Y.; Lin, J.; Liu, Z.; Wang, H.; Li, L.-J.; and Han, S. 2018. AMC: AutoML for Model Compression and Acceleration on Mobile Devices. In *ECCV*.

Howard, A.; Sandler, M.; Chu, G.; Chen, L.-C.; Chen, B.; Tan, M.; Wang, W.; Zhu, Y.; Pang, R.; Vasudevan, V.; Le, Q. V.; and Adam, H. 2019. Searching for MobileNetV3. In *ICCV*.

Howard, A. G.; Zhu, M.; Chen, B.; Kalenichenko, D.; Wang, W.; Weyand, T.; Andreetto, M.; and Adam, H. 2017. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv*.

Hu, J.; Shen, L.; and Sun, G. 2018. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 7132–7141.

Jacob, B.; Kligys, S.; Chen, B.; Zhu, M.; Tang, M.; Howard, A.; Adam, H.; and Kalenichenko, D. 2018. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2704–2713.

Krishnakumar, A.; White, C.; Zela, A.; Tu, R.; Safari, M.; and Hutter, F. 2022. NAS-Bench-Suite-Zero: Accelerating Research on Zero Cost Proxies. *arXiv preprint arXiv:2210.03230*.

Li, Y.; Shen, M.; Ma, J.; Ren, Y.; Zhao, M.; Zhang, Q.; Gong, R.; Yu, F.; and Yan, J. 2021. MQBench: Towards reproducible and deployable model quantization benchmark. *arXiv preprint arXiv:2111.03759*.

Lin, J.; Chen, W.-M.; Cai, H.; Gan, C.; and Han, S. 2021. Memory-efficient Patch-based Inference for Tiny Deep Learning. *Advances in Neural Information Processing Systems*, 34: 2346–2358.

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

Liu, Z.; Lin, Y.; Cao, Y.; Hu, H.; Wei, Y.; Zhang, Z.; Lin, S.; and Guo, B. 2021. Swin transformer: Hierarchical vision transformer using shifted windows. *arXiv preprint arXiv:2103.14030*.

Liu, Z.; Mao, H.; Wu, C.-Y.; Feichtenhofer, C.; Darrell, T.; and Xie, S. 2022. A convnet for the 2020s. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 11976–11986.

Liu, Z.; Mu, H.; Zhang, X.; Guo, Z.; Yang, X.; Cheng, K.-T.; and Sun, J. 2019. MetaPruning: Meta Learning for Automatic Neural Network Channel Pruning. In *ICCV*.

Loshchilov, I.; and Hutter, F. 2017. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*.

Mehta, S.; and Rastegari, M. 2021. Mobilevit: light-weight, general-purpose, and mobile-friendly vision transformer. *arXiv preprint arXiv:2110.02178*.

Radosavovic, I.; Kosaraju, R. P.; Girshick, R.; He, K.; and Dollár, P. 2020. Designing Network Design Spaces. *arXiv preprint arXiv:2003.13678*.

Real, E.; Aggarwal, A.; Huang, Y.; and Le, Q. V. 2019. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, 4780–4789.

Sandler, M.; Howard, A.; Zhu, M.; Zhmoginov, A.; and Chen, L.-C. 2018. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In *CVPR*.

Shen, M.; Liang, F.; Gong, R.; Li, Y.; Li, C.; Lin, C.; Yu, F.; Yan, J.; and Ouyang, W. 2021. Once quantization-aware training: High performance extremely low-bit architecture

search. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 5340–5349.

STMicroelectronics. 2023. STM32 X-CUBE-AI. https://www.st.com/en/embedded-software/x-cube-ai.html. Accessed: 2023-08-11.

Tan, M.; Chen, B.; Pang, R.; Vasudevan, V.; Sandler, M.; Howard, A.; and Le, Q. V. 2019. MnasNet: Platform-Aware Neural Architecture Search for Mobile. In *CVPR*.

Tan, M.; and Le, Q. 2019. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, 6105–6114. PMLR.

Vasu, P. K. A.; Gabriel, J.; Zhu, J.; Tuzel, O.; and Ranjan, A. 2022. An Improved One millisecond Mobile Backbone. *arXiv preprint arXiv:2206.04040*.

Williams, S.; Waterman, A.; and Patterson, D. 2009. Roofline: an insightful visual performance model for multicore architectures. *Communications of the ACM*, 52(4): 65–76.

Wu, B.; Dai, X.; Zhang, P.; Wang, Y.; Sun, F.; Wu, Y.; Tian, Y.; Vajda, P.; Jia, Y.; and Keutzer, K. 2019. FBNet: Hardware-Aware Efficient ConvNet Design via Differentiable Neural Architecture Search. In *CVPR*.

Yu, W.; Luo, M.; Zhou, P.; Si, C.; Zhou, Y.; Wang, X.; Feng, J.; and Yan, S. 2022. Metaformer is actually what you need for vision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 10819–10829.