Play It Cool: Dynamic Shifting Prevents Thermal Throttling

Yang Zhou [1] Feng Liang [1] Ting-wu Chin [2] Diana Marculescu [1 2]

Abstract

Machine learning (ML) has entered the mobile era where an enormous number of ML models are deployed on edge devices. However, running common ML models on edge devices continuously may generate excessive heat from the computation, forcing the device to "slow down" to prevent overheating, a phenomenon called *thermal throttling*. This paper studies the impact of thermal throttling on mobile phones: when it occurs, the CPU clock frequency is reduced, and the model inference latency may increase dramatically. This unpleasant inconsistent behavior has a substantial negative effect on user experience, but it has been overlooked for a long time. To counter thermal throttling, we propose to utilize *dynamic networks* with shared weights and dynamically shift between large and small ML models seamlessly according to their thermal profile, *i.e.*, shifting to a small model when the system is about to throttle. With the proposed dynamic shifting, the application runs consistently without experiencing CPU clock frequency degradation and latency increase. In addition, we also study the resulting accuracy when dynamic shifting is deployed and show that our approach provides a reasonable trade-off between model latency and model accuracy.

1. Introduction

ML models have become the *de-facto* building blocks for virtually all computer vision (CV) and natural language processing (NLP) applications. Additionally, because of the expected lower inference latency and higher user data security compared to cloud servers, it has become increasingly desirable to deploy ML models on edge devices. However, the demanding hardware requirements of ML models has become a barrier preventing the deployment of large and powerful models on devices with limited thermal and computational resources. This paper focuses on an overlooked thermal issue of deploying ML models on edge devices.

When running heavy workload on edge devices, heat is continuously generated, but due to the limited thermal capacitance of the hardware platform, heat may not be adequately dissipated. When temperature rises above the hardware thermal threshold, thermal throttling is triggered. The CPU clock frequency is thereby forced to drop, thus reducing the temperature to protect hardware functionality. However, the latency increases as a result. To understand the impact of thermal throttling on mobile phones, we first conduct experiments on two intensive tasks: continuous image classification and continuous question answering (See Fig. 1(a)). We find that running full models continuously results in thermal throttling (Fig. 1(b)). Moreover, as shown in Fig. 1(d), due to thermal throttling, the CPU clock frequency is decreased and therefore the latency of the model running on the edge device will drastically go up.

We argue that thermal throttling poses a serious threat to mobile ML applications that are latency-critical. For example, during real-time visual rendering for video streaming or gaming, a sudden surge of processing latency per frame will have substantial negative effect on user experience. Also, modern mobile operating systems often provide special services and applications for vision impaired individuals, such as VoiceOver on iOS and TalkBack on Android. The user typically interacts with mobile phones by relying completely on speech, so the quality of these services is highly dependent on the responsiveness or the latency of the application.

We propose to use weight sharing *dynamic networks* (Han et al., 2021; Yu et al., 2018; Hou et al., 2020) to prevent thermal throttling. The proposed dynamic shifting is depicted in Fig. 1(c). Initially, the large model in the dynamic neural network suite runs. When the CPU temperature is about to exceed a set threshold, the small model in the same suite runs instead. Then, when the CPU temperature drops and becomes stable, the large model runs again. With the proposed dynamic shifting, the application will run consistently at the same latency and without experiencing CPU clock frequency degradation (see Fig. 1(d)). Moreover, since the weights are shared, no additional memory loading is needed

[1]The University of Texas at Austin [2]Carnegie Mellon University. Correspondence to: Yang Zhou <yangzhou25672@utexas.edu>.

DyNN workshop at the 39th International Conference on Machine Learning, Baltimore, Maryland, USA, 2022. Copyright 2022 by the author(s).

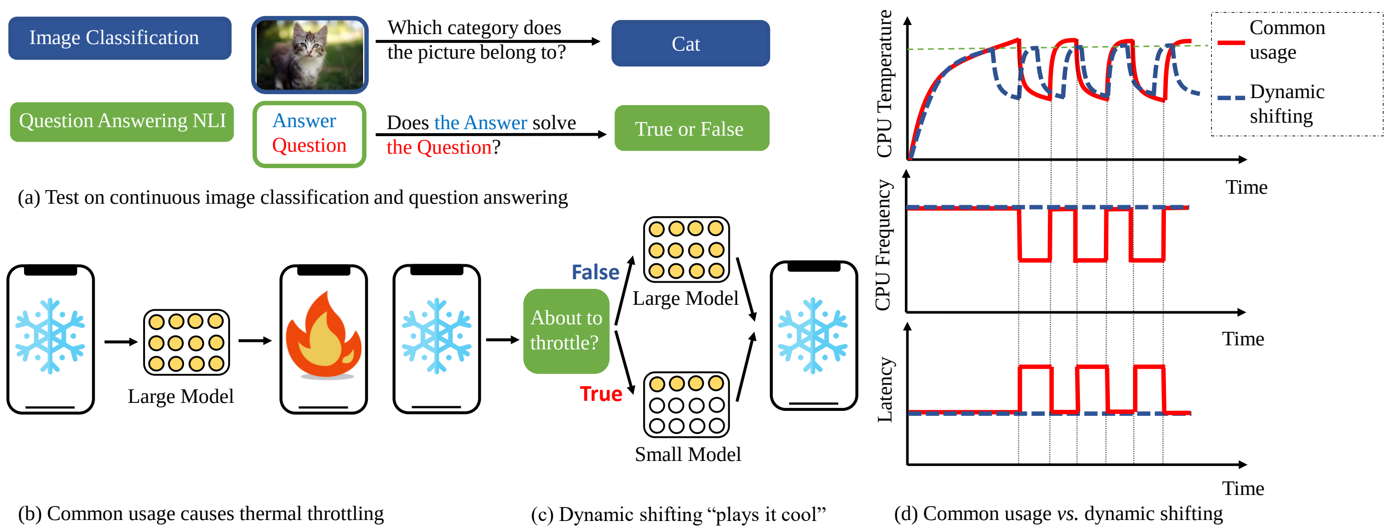


Image Classification
Which category does the picture belong to?
Cat
Question Answering NLI
Answer
Question
Does the Answer solve the Question?
True or False
(a) Test on continuous image classification and question answering
Large Model
About to throttle?
False
True
Large Model
Small Model
(b) Common usage causes thermal throttling
(c) Dynamic shifting "plays it cool"
CPU Temperature
CPU Frequency
Latency
Time
Common usage
Dynamic shifting
(d) Common usage vs. dynamic shifting

Figure 1. Comparison between common usage and our dynamic shifting. (a). We adopt two widely user applications: image classification in CV and question answering natural language inference (QNLI) in NLP. (b). Common usage leads to thermal throttling, resulting in an unpleasant 'hot and slow' experience. (c). Our method can 'play it cool' via dynamically shifting models according to the thermal profile. (d). Thermal throttling results in CPU frequency drop and latency increase. Our method is able to run consistently without these issues.

during model shifting. We provide details of our dynamic shifting algorithm in section 3.

Our dynamic shifting is suitable for different types of models: convolution based slimmable networks (Yu et al., 2018) and transformer based DynaBERT (Hou et al., 2020). Also, we demonstrate that dynamic shifting can generalize to other mobile CPU platforms by conducting the same experiments on a Raspberry Pi 4B platform. In addition, we study the resulting accuracy when dynamic shifting is deployed and show that our approach provides a reasonable trade-off between model latency and model accuracy.

2. Related Work

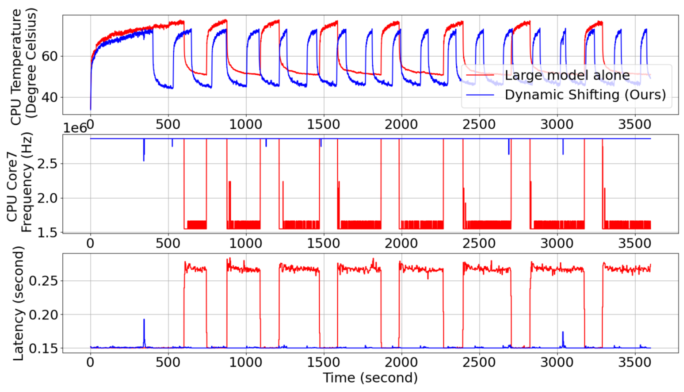
Dynamic Thermal Management (DTM) has mainly focused on finding the optimal trade-off between controlling the temperature and maintaining performance through hardware-based methods such as Temperature-Tracking Dynamic Frequency Scaling (Skadron et al., 2004), Dynamic Voltage Scaling (Brooks & Martonosi, 2001; Huang et al., 2000; Donald & Martonosi, 2006), Fetch and Clock gating (Brooks & Martonosi, 2001; Manne et al., 1998; Sanchez et al., 1997; Skadron et al., 2002), mitigating computation (Skadron et al., 2004), hybrid DTM method (Skadron et al., 2004), *etc*. These methods mainly target CPUs when running general tasks. Recently, (Benoit-Cattin et al., 2020) have looked specifically at ML inference workloads and have proposed to use peripheral active cooling hardware on edge devices to control the temperature of the device. In comparison, we propose a software-only method to prevent thermal throttling by dynamically shifting between models that are part of the same dynamic network (Han et al., 2021; Yu et al., 2018; Hou et al., 2020).

Algorithm 1 Dynamic Shifting Algorithm

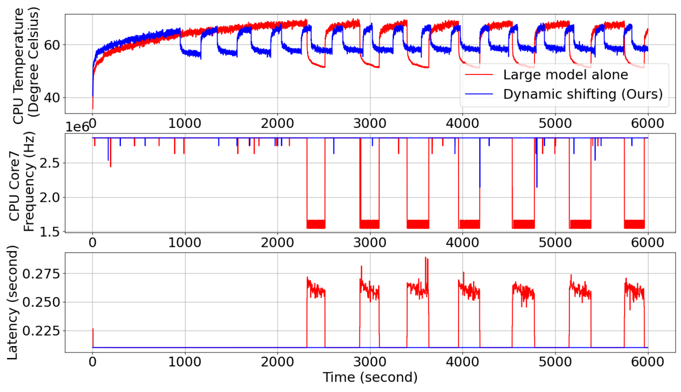
```
Require: Dynamic Network: mModule, Temperature Threshold:
    tLim, Derivative Threshold: gLim
Require: setModel() sets model to small or large for mModule;
    findGrad() computes the derivative;
 1: Initialize avgTemp, grad to 0
 2: mModule.setModel(LARGE)
 3: while true do
 4:   Get cpuTemp
 5:   avgTemp ← $\alpha\times$ avgTemp $+(1-\alpha)\times$ cpuTemp
 6:   grad ← $\beta\times$ grad $+ (1-\beta)\times$ findGrad(avgTemp)
 7:   if mModule.inLarge() AND cpuTemp $>$ tLim then
 8:     mModule.setModel(SMALL)
 9:     Clear prevAvgTemp, avgTemp, grad
10:   else if mModule.inSmall() AND grad $>$ gLim then
11:     mModule.setModel(LARGE)
12:     Clear prevAvgTemp, avgTemp, grad
13:   end if
14: end while
```

3. Dynamic Shifting

The main goal of the proposed approach is to run inference continuously within the latency constraint while not throttling. Once the large and small models are picked, we follow Algorithm 1 to implement dynamic shifting. We start from the large model. When the CPU temperature exceeds a given temperature threshold, we shift to the small model. After shifting to the small model, the CPU temperature starts dropping. Since the small model is less computationally intensive, running the small model meets the throughput requirement. Later, when the temperature drops and becomes stable, the smart phone CPU reaches a thermal equilibrium,



(a) Running ResNet50 alone *vs.* Dynamic Shifting between Slimmable ResNet50 1.0x and Slimmable ResNet50 0.25x on continuous Image Classification task (60 minutes)



(b) Running BERT w50 d50 alone *vs.* Dynamic Shifting between BERT w50 d50 and BERT w25 d50 on continuous Question Answering task (100 minutes)

Figure 2. The common usage *v.s.* the proposed dynamic shifting on CV and NLP tasks on Honor V30 Pro (Best viewed in color).

and the excessive heat is dissipated. Therefore, if the temperature derivative calculated when running the small model is close enough to zero or exceeds a negative derivative threshold, we shift to the large model again for better inference accuracy. To compute the derivative, the CPU temperature is first smoothed out using the Exponential Moving Average (EMA). The derivative is thereafter approximated by the rate of change of the temperature readings. In practice, we use EMA again to smooth out the computed derivatives to reduce noise.

4. Experiments and Discussion

Setup, dataset, model, platform, settings. We use slimmable networks (Yu et al., 2018) and DynaBERT (Hou et al., 2020) for CV and NLP tasks, respectively. Mobile phone experiments are conducted on the Honor V30 Pro[1]. To run model inferences on the phone, we encapsulate PyTorch models into TorchScript files and build testing programs upon the published Android Studio code[2]. To test the generability of our dynamic shifting, we also conduct experiments on Raspberry Pi 4B 4GB using onnxruntime.

We find that the PyTorch torchscript can only trace a static computation graph of the model and is not suitable for shared weight dynamic networks. We have to trace the large and the small models in two separate files. Once encapsulated in the torchscript, the large and the small model no longer share weights, and because of this, extra memory overhead is noted when shifting between models. We report the time taken to load in models during shifting in Appendix A. Moreover, we want to emphasize that if true weight sharing[3] is used, the extra memory overhead will not occur.

Table 1. Summary of the latency and accuracy of running Dynamic Shifting on Slimmable ResNet50 and DynaBERT on Honor V30 Pro (DS refers to our method Dynamic Shifting)

	Latency (s)	Accuracy
ResNet50 1.0x (alone)	0.205	0.761
Slimmable ResNet50 (DS)	0.150 (-26.8%)	0.695 (-0.066)
BERT d0.5 w0.5 (alone + 1.4x latency)	0.217	0.900
DynaBERT (DS + 1.4x latency)	0.211 (-2.88%)	0.893 (-0.008)

4.1. Mobile Phone Experiments

To make the comparison to running the large model alone easier, we fix the inference latency of the small model to have the same inference latency as the large model by injecting idle time after every small model inference. We find that the mobile phone throttling point is determined by multiple factors not only by the temperature. For this work, we simplify the problem and only consider using CPU temperature when designing the shifting threshold. Time taken for logging the data is measured and reported in Appendix A. We document the experiment details in Appendix B.

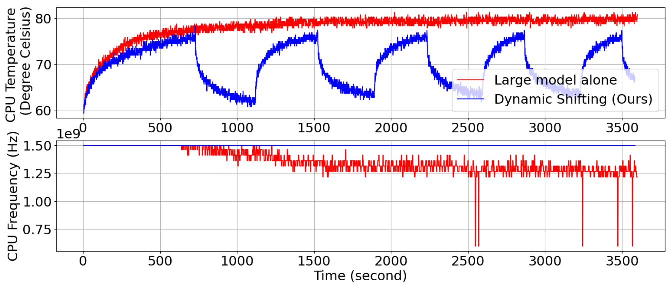
4.1.1. Slimmable ResNet50 Experiments

Running ResNet50 alone throttles the device roughly at CPU temperature $77°C$ (see the red line in Figure 2 (a)). Starting from this, we set the temperature threshold to $73°C$. The derivative threshold after EMA is set to -0.07. We shift between the largest and the smallest model in slimmable

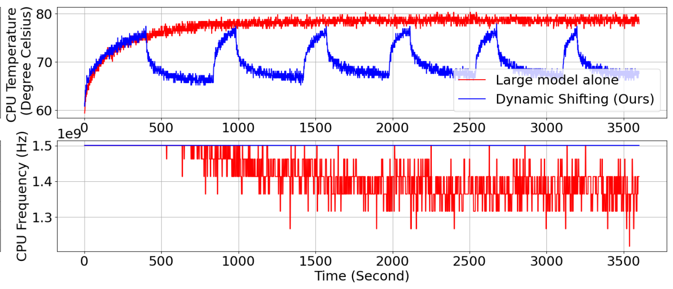
[1]Phone released in 2019, CPU: Kirin 990 5G 2x2.86 GHz Cortex-A76 & 2x2.36 GHz Cortex-A76 & 4x1.95 GHz Cortex-A55, CPU frequency is reported on core 7.

[2]https://github.com/pytorch/android-demo-app

[3]We found that now there is hardly a mobile framework that supports shared weights dynamic networks. Currently, to implement a true weight sharing dynamic network on mobile devices, one must directly write a model-specific C++ or Java program.



(a) Running ResNet50 alone *vs.* Dynamic Shifting between Slimmable ResNet50 x1.0 and Slimmable ResNet50 x0.25 on continuous Image Classification task (60 minutes)



(b) Running BERT w50 d50 alone *vs.* Dynamic Shifting between BERT w50 d50 and BERT w25 d50 on continuous Question Answering task (100 minutes)

Figure 3. The common usage *v.s.* the proposed dynamic shifting on CV and NLP tasks on Raspberry Pi 4B (Best viewed in color).

ResNet50. From Figure 2 (a), dynamic shifting is shown to prevent thermal throttling. The CPU temperature and Latency remain stable throughout the entire hour of the experiment.

We also report the accuracy and latency comparison in Table 1. The latency is calculated by averaging all inference latency during the experiments. The accuracy is estimated by the ratio of large and small model. Although it may sacrifice some accuracy, the proposed dynamic shifting has a faster inference speed. Most importantly, our dynamic shifting approach enjoys a consistent inference. We also provide ablation studies on the choice of different combinations of temperature and derivative thresholds in Appendix C.

4.1.2. DynaBERT Experiments

We find that BERT models (Devlin et al., 2018) in general are too computationally intensive for mobile phones. (see Appendix B). To test our approach, we choose to dynamically shift between BERT d0.5 w0.5 and BERT d0.25 w0.5, the two smallest models in the DynaBERT suite. To make the smallest model capable to reaching a suitable operating temperature, we increase latency to 1.4X the original BERT d0.5 w0.5 latency. With 1.4X latency, BERT d0.5 w0.5 throttles at $70°C$. Accordingly, the temperature threshold is set to $65°C$. The derivative threshold is set to -0.008. We compare dynamic shifting with running BERT d0.5 w0.5 alone at the same latency is in Figure 2 (b). The CPU frequency and the CPU latency also remain stable for the whole 100 minutes in the experiment. The latency and accuracy trade-off is also shown in Table 1. Dynamic Shifting, in general, cannot prevent BERT models from thermal throttling because of the model's enormous computational intensity. However, under some limitations, dynamic shifting can still be helpful when deploying BERT models on mobile phones.

4.2. Raspberry Pi 4B Experiments

We conduct the same experiments on Raspberry Pi 4B, aiming to show that dynamic shifting can generalize well to other mobile CPU platforms. We switch between ResNet50 1.0x and ResNet50 0.25x and between BERT d0.5 w1.0 and BERT d0.5 w0.25. Experiment setup details are in Appendix B. Unlike mobile phones, when Raspberry Pi 4B throttles, it maintains the CPU temperature at the throttling temperature by slightly lowering the frequency as shown in Figure 3 (a). Therefore, after throttling, the latency for ResNet50 rises only by 4.5% and 5.7% for BERT w1.0 d0.5. We argue that Raspberry Pi 4B's thermal throttling strategy is not applicable to mobile phones since the high temperature is not suitable to wearable devices. Nevertheless, we present the temperature and the frequency results in Figure 4 and show that dynamic shifting can stop Raspberry Pi 4B from throttling.

5. Conclusion And Future Work

In this work, we show that thermal throttling is a serious problem in deploying common neural networks on mobile phone. To counter thermal throttling, we propose to use dynamic shifting between models in dynamic networks with shared weights based on the device CPU temperature. Through the experiments, we demonstrate the effectiveness of dynamic shifting in Computer Vision and Natural Language Processing tasks under some conditions.

In this work, we only consider shifting between large and small models. Further attempts can incorporate models with sizes in between into the shifting iteration. Also, future work that takes multiple factors into account when designing threshold for shifting between models can potentially perform more efficiently than ours. Besides, dynamic shifting by its own is not sufficient for models that are compute intensive such as BERT models, because even running only the smallest model would not make the phone operate under suitable temperature. We leave the problem for future work.

6. Acknowledge

The research is supported in part by NSF CCF Grant No. 2107085 and NSF CSR Grand No. 1815780.

References

Benoit-Cattin, T., Velasco-Montero, D., and Fernández-Berni, J. Impact of thermal throttling on long-term visual inference in a cpu-based edge device. *Electronics*, 9(12): 2106, 2020.

Brooks, D. and Martonosi, M. Dynamic thermal management for high-performance microprocessors. In *Proceedings HPCA Seventh International Symposium on High-Performance Computer Architecture*, pp. 171–182. IEEE, 2001.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Donald, J. and Martonosi, M. Techniques for multicore thermal management: Classification and new exploration. *ACM SIGARCH Computer Architecture News*, 34(2):78–88, 2006.

Han, Y., Huang, G., Song, S., Yang, L., Wang, H., and Wang, Y. Dynamic neural networks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

Hou, L., Huang, Z., Shang, L., Jiang, X., Chen, X., and Liu, Q. Dynabert: Dynamic bert with adaptive width and depth. *Advances in Neural Information Processing Systems*, 33:9782–9793, 2020.

Huang, M., Renau, J., Yoo, S.-M., and Torrellas, J. A framework for dynamic energy efficiency and temperature management. In *Proceedings of the 33rd annual ACM/IEEE international symposium on Microarchitecture*, pp. 202–213, 2000.

Manne, S., Klauser, A., and Grunwald, D. Pipeline gating: Speculation control for energy reduction. In *Proceedings. 25th Annual International Symposium on Computer Architecture (Cat. No. 98CB36235)*, pp. 132–141. IEEE, 1998.

Sanchez, H., Kuttanna, B., Olson, T., Alexander, M., Gerosa, G., Philip, R., and Alvarez, J. Thermal management system for high performance powerpc/sup tm/microprocessors. In *Proceedings IEEE COMPCON 97. Digest of Papers*, pp. 325–330. IEEE, 1997.

Skadron, K., Abdelzaher, T., and Stan, M. R. Control-theoretic techniques and thermal-rc modeling for accurate and localized dynamic thermal management. In *Proceedings Eighth International Symposium on High Performance Computer Architecture*, pp. 17–28. IEEE, 2002.

Skadron, K., Stan, M. R., Sankaranarayanan, K., Huang, W., Velusamy, S., and Tarjan, D. Temperature-aware microarchitecture: Modeling and implementation. *ACM Transactions on Architecture and Code Optimization (TACO)*, 1 (1):94–125, 2004.

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. R. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

Yu, J., Yang, L., Xu, N., Yang, J., and Huang, T. Slimmable neural networks. *arXiv preprint arXiv:1812.08928*, 2018.

A. Time taken to shift between models and logging

Since TorchScript cannot trace dynamic networks, weights are not shared between the large and the small networks. We report the extra load latency from both the phone and Raspberry Pi 4B in Table 2. The phone load models from PyTorch Torchscripts, while the Raspberry Pi 4B uses Onnxruntime files, which internally uses PyTorch Torchscript to encapsulate PyTorch models. We give both the mean and the standard deviation of the measurements. In each entry, the left value stands for the latency overhead for loading the large model, while the right value stands for the latency overhead for loading the small model.

Please note that for DynaBERT, phone uses Bert w0.5 d0.5 as the large model and Bert w0.25 d0.5 as the small model, while the Raspberry Pi 4B uses Bert w1.0 d0.5 as the large model and Bert w0.25 d0.5 as the small model. Both phone and Raspberry Pi 4B uses Slimmable ResNet50 1.0x as the large model and Slimmable ResNet50 0.25x as the small model.

Table 2. Extra latency from shifting between models (large/small)

	Shifting Latency (s)	
	Phone	Raspberry Pi
Slimmable ResNet50	$1.000 \pm 0.252/0.997 \pm 0.321$	$0.887 \pm 0.070/0.143 \pm 0.006$
DynaBERT	$0.885 \pm 0.066/0.826 \pm 0.016$	$1.527 \pm 0.423/0.810 \pm 0.218$

While running the models, we log CPU temperature, CPU frequency, and inference latency after every inference. We report the extra gap time from logging (without logging the memory usage) from both the phone and from Raspberry Pi 4B in Table 2. We provide both the mean and the standard deviation of the measurements.

Table 3. Average time taken for logging

	Logging Latency (s)
Phone	0.023 ± 0.004
Raspberry Pi 4B	0.080 ± 0.014

B. Experiment Details

B.1. Mobile Phone Slimmable ResNet50

We select a random 224-by-224 image and run inferences of the static image continuously with no gap between consecutive inferences. The CPU temperature is read from the built-in sensor of the phone. The CPU temperature is read after every inference, together with CPU frequency and the model inference latency. We use the official PyTorch implementation of Slimmable ResNet50[4]. The EMA coefficients α and β in Algorithm 1 are chosen to be 0.995 and 0.99 respectively. When running all experiments on the phone, we keep the atmosphere in room temperature around $22°C$, and we make sure the phone is unplugged and has battery level over 70%. Besides, we kill all other tasks except the experiment app while running the experiments.

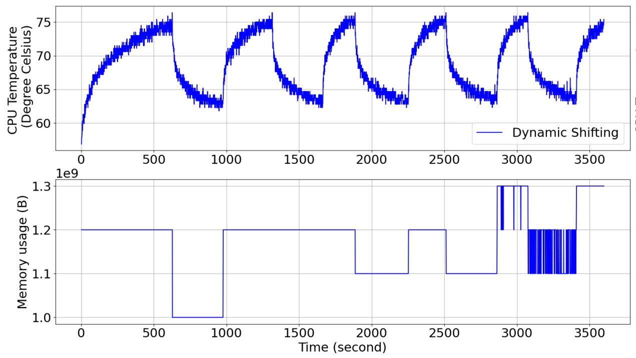
B.2. Mobile Phone DynaBERT

We randomly select one question answer pair from the QNLI dataset from GLUE (Wang et al., 2018) as input for every iteration of inferences. The total sum of input tokens of the question and answer pair is 128, the same as all other pairs in the QNLI dataset. Similar to the Slimmable ResNet50 experiments, we let the phone run inference continuously with no gap in between inputs. We use the official PyTorch implementation of DynaBERT[5]. The EMA coefficients we used during experiments are the same as in Slimmable ResNet50 experiments.

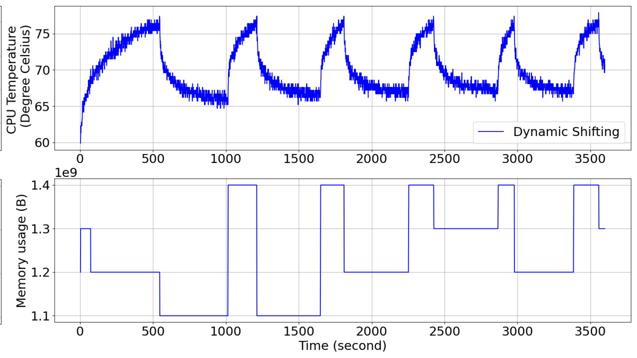
As we report in the paper, BERT models in general are computationally intensive to mobile phone CPUs. The Honor V30 Pro CPU temperature rises to $80°C$ in less than 32 seconds and experiences thermal throttling in less than 6 minutes when

[4]https://github.com/JiahuiYu/slimmable_networks
[5]https://github.com/huawei-noah/Pretrained-Language-Model/tree/master/DynaBERT

running the full BERT d1.0, w1.0 (depth 1.0x, width 1.0x). A similar trend is seen for other BERT models with either full depth or full width. Therefore, we choose to only use BERT models with half the width.



Memory usage when running dynamic shifting between Slimmable ResNet50 1.0x and Slimmable ResNet50 0.25x



Memory usage when running dynamic shifting between DynaBERT d0.5 w1.0 and DynaBERT d0.5 w0.25

Figure 4. The memory overhead in Bytes throughout 3600 seconds of Dynamic Shifting on Raspberry Pi 4B (temperature plots might look slight different compared to plots in the body)

B.3. Raspberry Pi Slimmable ResNet50

We measure the CPU temperature by reading the built-in sensor inside Raspberry Pi 4B. Raspberry Pi 4B has a fixed thermal throttling point which can be modified to any value below 85°C. To match the mobile phone setting, we set the thermal throttling point to 80°C. We observe from the experiments that Raspberry Pi 4B operating system starts to throttle the frequency at around 78°C. Therefore, we set the temperature threshold to 77°C. For running dynamic shifting on Slimmable ResNet50, we set the derivative threshold at -0.02. We use the same EMA coefficients as in Mobile Phone experiments, namely, α and β to be 0.995 and 0.99 respectively.

B.4. Raspberry Pi DynaBERT

As in Slimmable ResNet50 experiments, we set the temperature threshold to be 77°C. We set the derivative threshold to be -0.012, higher than the derivative threshold in Slimmable ResNet50 to give more time to run the small model to cool down the CPU.

In the case of mobile phones, we find that using BERT d0.5 w1.0 model can throttle the CPU very quickly. In contrast, we find that it is not the case in Raspberry Pi 4B. Therefore, we shift between BERT d0.5 w1.0 and BERT d0.5 w0.25.

C. Ablation Study

Table 4. Accuracy comparison between different combinations of temperature and derivative thresholds (Best viewed in color)

		Temperature Threshold °C			
		75	73	70	65
	-0.005	0.693	0.684	0.675	0.668
Derivative	-0.01	0.700	0.688	0.678	0.667
Threshold	-0.07	0.710	0.701	0.684	0.671
	-0.10	0.716	0.698	0.692	0.677

The temperature threshold and the derivative threshold are two hyperparameters in dynamic shifting. Different choices of the temperature and derivative thresholds are explored. We run dynamic shifting using Slimmable ResNet50 1.0x and Slimmable ResNet50 0.25x for 30 minutes. To compare hyperparameters, we compute the average theoretical Top-1 accuracy of the first two stable shifting iterations. The models' accuracy is based on the Slimmable Networks paper (0.768 for model 1.0x and 0.638 for model 0.25x). The results are shown in Table 4.

The general trend in the Table 4 is that the accuracy is the highest in the lower left corner and the lowest in the upper right.

Towards the lower left, the temperature threshold is the highest and the derivative threshold is the lowest. The large model can potentially run longer, and the small model will reach the derivative threshold sooner. We choose $73°C$ rather than $75°C$ for the experiment in Figure 2 (a) to be more conservative towards thermal throttling. Also, we find that using $75°C$ makes the time running large model uneven between iterations, which can explain why in Table 1 $75°C$, -0.10 has average accuracy slightly lower than the accuracy of $73°C$, -0.07.

D. Memory Overhead

We include the memory usage plots from the Raspberry Pi 4B during the dynamic shifting in Figure 4. Please note that since logging memory usage increases the logging latency, experiments are run separately. The memory usage is not logged in the experiments presented in the body. The temperature graphs might look slightly different here than in Figure 4 as a result.