M.S. THESIS

Splash: Thermal-aware Mobile-Cloud Collaborative Inference

열 인지 기반 모바일-클라우드 협동 추론

2023 년 2 월

서울대학교 대학원

협동과정 인공지능전공

정 창 진

M.S. THESIS

Splash: Thermal-aware Mobile-Cloud Collaborative Inference

열 인지 기반 모바일-클라우드 협동 추론

2023 년 2 월

서울대학교 대학원

협동과정 인공지능전공

정 창 진

Splash: Thermal-aware Mobile-Cloud Collaborative Inference

열 인지 기반 모바일-클라우드 협동 추론

지도교수 전 병 곤

이 논문을 공학석사 학위논문으로 제출함

2022 년 12 월

서울대학교 대학원

협동과정 인공지능전공

정 창 진

정창진의 석사 학위논문을 인준함

2023 년 2 월

위 원 장	유 승 주	(인)
부위원장	전 병 곤	(인)
위 원	이 영 기	(인)

Abstract

Continuous vision AI applications request multiple Deep Neural Network (DNN) model inferences on mobile devices for a long time which incurs the overheating problem. While running a DNN inference on mobile devices is growing in prominence, heat generated by utilizing multiple processors concurrently makes the device temperature exceed the predefined threshold easily. Alternative approaches of mobile-cloud collaborative systems have the limitation of only focusing on minimizing the makespan of inference requests without considering the thermal conditions of mobile devices.

This paper introduces Splash, a thermal-aware mobile-cloud collaborative DNN inference system, exploiting observations that performance per temperature among heterogeneous processors varies with the type of workload and dynamic condition changes. With the lightweight and efficient prediction models of future temperature and the latency of inference requests, Splash can leverage augmented heat budgets through thermal-aware scheduling. We evaluate our system using realistic workloads on two commodity smartphones for 10 minutes and show that Splash achieves up to 1.93× higher FPS without throttling than the state-of-the-art multi-DNN inference framework.

Keywords: Mobile Deep Learning, Thermal Throttling, Cloud Offloading, Multi-DNN Inference

Student Number: 2021-21645

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

The rapid advance of deep neural networks (DNN) stimulates the emergence of continuous vision AI applications (Figure 1.1). They provide highly immersive user experiences in diverse application domains such as education, remote work, entertainment, and health care [9, 4, 10, 11]. Especially, many vision AI applications execute multiple DNN models concurrently to analyze the scenes, generate virtual contents and seamlessly render them [12, 13]. They impose multiple challenging requirements such as tight end-to-end latency (e.g., >50 ms for an extended reality application), high analysis accuracy, and stable long-time execution [14].

The thermal problem remains the central challenge for the long-term and stable execution of continuous vision AI applications. Recently, there has been a rich body of work to support the on-device execution of neural networks including the development of neural accelerators [15, 16, 17, 18, 19, 20], lightweight models [21, 5, 22], and inference systems [12, 13]. However, their primary focus lies in accelerating the inference speed. Such works indirectly

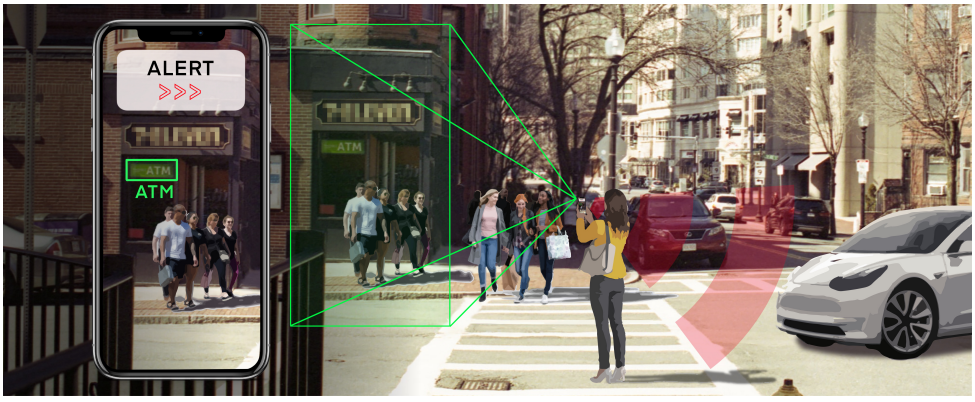


Figure 1.1: An illustration of continuous vision AI application [1]. It executes both a text detection model [2] and an object detection model [3] concurrently for a long time.

help to reduce the heat for a short-term execution, but still do not prevent excessive heat generation in long-term operation.

Mobile devices adopt a dynamic voltage and frequency scaling system (DVFS) [23] to cool down hardware components. For heterogeneous processors, manufacturer-specific thermal management systems are typically used such as Qualcomm SDM [24] for Qualcomm Snapdragon-powered devices and Exynos TMU [25] for Samsung devices. However, they result in a sudden performance drop, which degrades the quality of experience. Various works extend the existing DVFS to minimize performance loss based on application-aware optimization with heterogeneous processors [26, 27, 28]. Yet, the current approaches have limitations on practicality in that they build prediction models in advance or require an external power monitor [29] for learning. Offloaded or split inference approaches can be considered as an alternative solution [30, 31, 32]. However, it is also non-trivial to adopt them due to the heat generated by network transmission along with dynamic DVFS configurations affected by the heat.

In this paper, we present SPLASH, a thermal-aware DNN inference system effectively utilizing heterogeneity of thermal efficiency across mobile processors and cloud offloading. To quantify the thermal efficiency, we define *Performance Per Temperature (PPT)* as a ratio of the FPS and the temperature increase for the execution of a task. Based on the definition, we present two key observations: (1) the rank of the PPT across processors can significantly change due to the difference in workload characteristics. For example, in the object detection task, the PPT of the GPU is the highest among processors, but in the segmentation task, it becomes the lowest. (2) the PPTs vary by 67% with the dynamics of network bandwidth, so the rank of PPT of the cloud offloading can also fluctuate depending on network bandwidth.

By leveraging such observations, SPLASH selects the optimal worker predicted to be helpful for alleviating thermal throttling according to workloads and network conditions. In addition to the utilization of PPT, SPLASH is designed under three considerations that are required for practicality. First, the system needs to be deployed without any additional measurement instrument (C1). Second, the system requires to be independent of the device (C2). Third, the scheduler has to achieve satisfactory accuracy without the significant overhead (C3). So, SPLASH only uses obtainable values from sensors on the device to monitor the device status to address C1. For C2, we design an online prediction model for temperature and latency that does not require the offline profiling stage, which is often used to model the target device. We also engineer the models to be lightweight and accurate in dealing with the dynamic nature of temperature and network bandwidth for C3.

We evaluate SPLASH in two realistic multi-DNN workload applications on two mobile commodity devices. From the experiments, SPLASH increases the frame rate without throttling up to 1.93× more compared to the state-of-

the-art baseline and 1.63× to the best-extended baseline. Through the case study, we show that SPLASH runs up to 2.20× more time to throttling by exploiting augmented thermal budgets attributed to assigning more tasks to thermally efficient workers.

The key contributions of this paper are as follows:

- We demonstrate the prevalence of thermal throttling with multi-DNN workloads and the severity of the performance degradation from thermal throttling through motivational studies.
- We define a metric, *Performance Per Temperature*, to quantify thermal efficiency with both the performance of each DNN model and the temperature increase corresponding to a processor.
- We propose SPLASH, a mobile-cloud collaborative DNN inference system utilizing the heterogeneity of thermal efficiency among processors. SPLASH is the first mobile-cloud collaborative DNN inference system to take the thermal properties of mobile devices into account.
- We evaluate with two real workloads and extend the FPS without throttling up to 1.63× on Google Pixel 4, and up to 1.93× on Samsung Galaxy S20 without prior knowledge of the devices.

Chapter 2

Motivation

In this chapter, we delve into the effect of thermal throttling on performance when running multi-DNN workloads through the following experiments. Then, we show existing DNN inference engines do not take into account the thermal efficiency of the processor, revealing that only minimizing the latency results in poor performance for long-term execution.

Performance drop by thermal throttling. Simultaneously running multiple processors on a mobile platform can occur thermal throttling in a short time, incurring severe degradation of user experiences. Unlike desktops and servers, mobile devices are equipped with system-on-chips (SoC), and heterogeneous processors are in the same SoC die, which leads to tight thermal coupling that the processors affect the temperature of one another, restricting heat transfer [33]. In addition, because of the small form factor, mobile devices cannot equip active cooling methods such as external fans and liquid cooling.

We experimented with a synthetic workload running MobileNet [21] on

Platform	Mobile (Google Pixel 4)	Cloud
CPU	Kryo 485 64-bit 2.84 GHz Octa-Core	AMD Ryzen 7 2700X 3.7 GHz Octa-Core
GPU	Adreno 640	NVIDIA Titan RTX
Accelerators	Hexagon 690, Google Edge TPU	-
Memory	6 GB LPDDR4	32 GB DDR4
OS	Android 11	Ubuntu 20.04

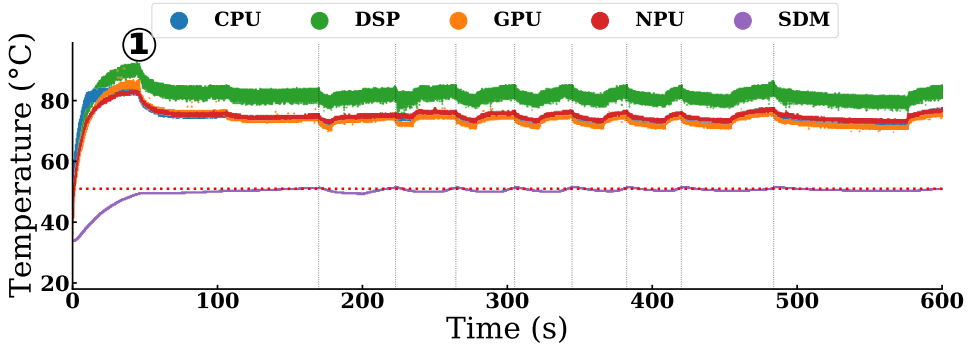
Table 2.1: Specification of the mobile and cloud platform.

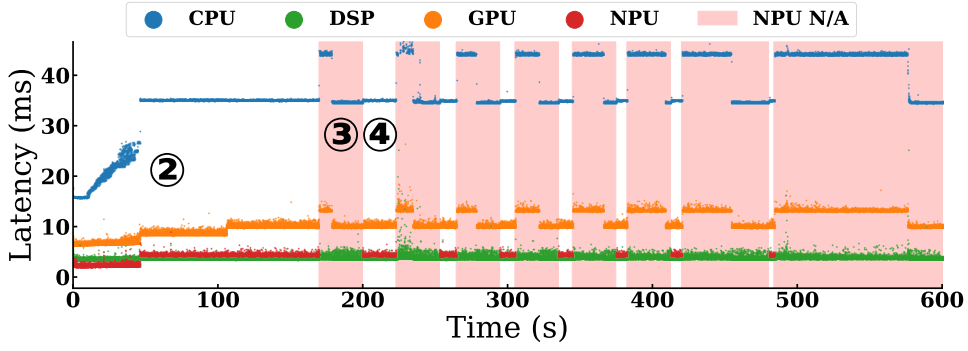
the state-of-the-art heterogeneous DNN inference engine, Band [12], concurrently and continuously for 10 minutes.[1] The specification of the mobile device, Google Pixel 4, used in the experiment is shown in Table 2.1.

The experimental results of the timeline of temperature and latency of each processor are described in Figure 2.1. In just 40 seconds, the temperatures of all processors increase rapidly over 80°C (①). When the temperature measured from the SDM (i.e., Qualcomm thermal management system) exceeds the first pre-defined thermal threshold, 49°C, the system incrementally drops the frequency of the CPU and GPU. Accordingly, the CPU and GPU latency increases step-wise, as shown in Figure 2.1b ②. Although the thermal throttling temporarily lowers each temperature of the CPU and GPU, the temperature of SDM keeps increasing because of continuous heat generation from the processors. After 170 seconds from the start, the temperature of SDM reaches the second thermal threshold depicted as the red dotted horizontal line on Figure 2.1a, 51°C. On that critical level of thermal status, the system turns off NPU and lowers the frequency of CPU and GPU further (③). As a consequence, the temperature of the device decreases and makes the NPU turn back on to continue to rerun inferences (④). This cycle

[1]We conducted all measurements in Chapter 2 and 3 with Google Pixel 4 on the same ambient temperature, 25°C, using the thermal incubator [7]

(a) The timeline of temperature changes on processors. The temperature reaches its peak in 40 seconds.





(b) The timeline of latency changes of inference requests on processors. The latency of MobileNet [21] of most processors increases step-wise because of thermal throttling.

Figure 2.1: Performance degradation from thermal throttling. The NPU shutdown and latency increase of processors deteriorate the user experience.

happens repeatedly but with a shorter activation period each time.

Limitations of existing DNN inference systems. Even with the help of a cloud server, we observe that the state-of-the-art inference engine cannot avoid thermal throttling since it does not take network costs on the thermal conditions of mobile devices. We conducted an experiment with a person finder application [4] specified in Table 6.1. As shown in Figure 2.2, thermal throttling occurs with the workload on the state-of-the-art engine (Figure 2.2 Mobile) at 200 seconds. When we augment the system with a cloud, the

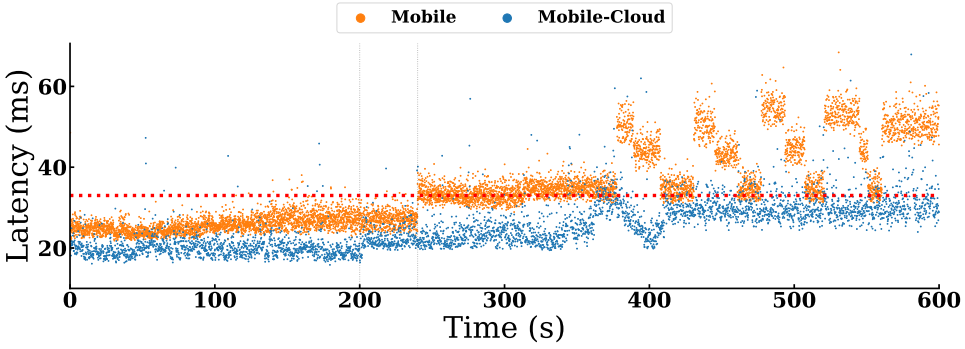


Figure 2.2: The timeline of latency changes during the execution of the person finder workload [4] on the existing DNN inference system. The Mobile-Cloud system is throttled 40 seconds earlier than Mobile system.

Processor	**CPU**	**GPU**	**DSP**	**NPU**	**CLOUD**
Latency (ms)	10.99	7.65	8.62	9.39	8.80
Performance (FPS)	90.92	130.66	115.93	106.43	113.51
Temperature increase (°C)	6.9	1.9	1.2	0.4	0.4
Performance per temperature (FPS/°C)	13.18	68.77	72.46	266.09	283.79

Table 2.2: Thermal efficiency among processors and cloud offloading varies in the same model, RetinaFace [8]. The measured network bandwidth on cloud offloading is 249 Mbps.

latency of each frame gets faster overall, thanks to offloading inferences to the cloud server, resulting in a rising SLO satisfaction rate of 43.7%. However, cloud offloading also costs the thermal budgets of mobile devices since the extra heat source (i.e., the modem for data transfer) comes in. As a result, the extended engine (Figure 2.2 Mobile+Cloud) faces earlier thermal throttling at 240 seconds, resulting in faster degradation of user experience.

It is noteworthy that even if a processor has the highest performance, it can be undesirable for the long-term execution to select the fastest processor if it generates a large amount of heat. For example, we measured the latency and the amount of temperature increase while executing the face detection task with RetinaFace [8]. In Table 2.2, it is shown that the FPS of the GPU

is 1.15× higher than that of the cloud. However, the temperature increases 4.75× more. Thus, the cloud can be a better choice for long-term execution when the execution on the cloud fits SLO, and the mobile device temperature is about to reach the throttling threshold.

Summary. In mobile devices, thermal management systems can cause severe performance degradation when utilizing heterogeneous processors, as they do not take the performance cost incurred by heat generation into account. Naïvely offloading the inferences to the cloud cannot strike a balance between the gain in performance and cost in performance due to the heat generation from extra sources. To this end, we present a mobile-cloud collaborative DNN inference engine that proactively mitigates the thermal throttling problem by adaptively choosing cloud offloading when the SLO permits and maximizing the utilization of the on-device heterogeneous processors while keeping them from reaching the throttling threshold.

Chapter 3

Performance Per Temperature (PPT)

To identify the thermal impact of the execution, we define an efficiency metric regarding thermal information, which a thermal-aware scheduling policy can utilize.

In order to consider both temperature and performance, and compare them quantitatively, we introduce *Performance Per Temperature (PPT)* as a measure for pursuing the minimal temperature increase and the maximal throughput as below.

$$\texttt{PPT} = \frac{\texttt{FPS}}{\Delta\texttt{temp}} \tag{3.1}$$

By defining PPT as a ratio of the FPS and the temperature increase for the execution of a task, it effectively specifies the thermal efficiency of the execution, allowing us to find the following key observations.

Workload affects the rank of performance per temperature across

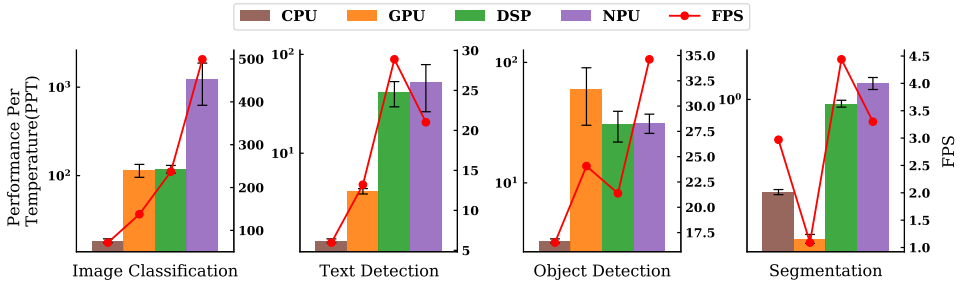


Figure 3.1: The processors' performance per temperature across workloads. MobileNetV2 [5], EAST [2], EfficientDet-Lite [3], and DeepLabV3 [6] are used for the image classification, the text detection, the object detection, and the segmentation task respectively. The range of the error depicts the instrument error.

processors. The rank of the PPT is not fixed for processors, but it is dependent on the characteristics of a workload. Figure 3.1 shows the PPT of each processor corresponding to the workload. For the image classification task and the text classification task, the rank is CPU $<$ GPU $<$ DSP $<$ NPU. However, for the object detection task, the order of GPU, DSP, and NPU is changed. Especially for the segmentation task, the GPU becomes the worst.

One cause for this is the difference in software- and hardware-level implementation across DNN architectures which influences the power consumption and the latency. If the power consumption and the execution time are linearly correlated to the computation size, the rank of the PPT would barely change across processors. However, as [34, 35] point out, the amount of computation does not represent the inference time due to the difference in memory access in actual implementation such as firmware, libraries, and algorithms. Also, [36] reports that the correlation between the power consumption and the computation size is reduced underlying different DNN models. Thus, based on the implementation, the latency and the temperature increase can change, making a difference in the rank of PPT according to the workload.

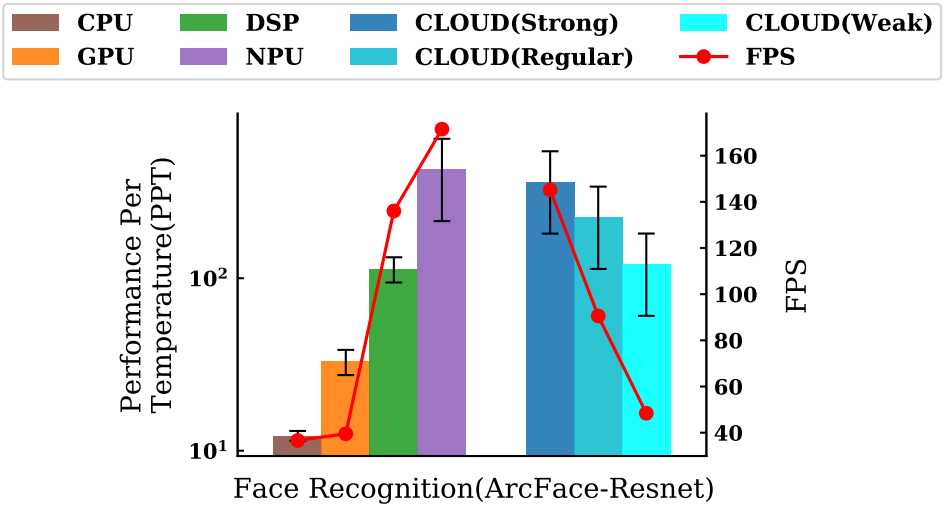


Figure 3.2: The performance per temperature with the variation of network bandwidth. The measured bandwidth of `Strong`, `Regular`, and `Weak` is 249 Mbps, 128 Mbps, and 13 Mbps, respectively.

The other cause for this phenomenon is fallback operators. When the processor cannot run some operators in the requested model, it is inevitable to execute it on the CPU processor, partially accelerating the supported operations [12]. In the case of the segmentation task, 17.9% of operators are unsupported by the GPU, thereby significantly reducing the PPT. Note that the PPT of the GPU is even less than that of the CPU due to the copy overhead.

The performance per temperature of a cloud changes due to dynamic network conditions. Because of the dynamics of the wireless network, the PPT of the cloud differs. Factors that affect signal strength, such as the distance between a device and a router, and the structure of a place, make it difficult to predict the end-to-end latency. In addition, uncertainty is added since the network routing between the server and the end device changes.

Figure 3.2 illustrates the PPT variation depending on the signal strength, only adjusting the RSSI controlling the distance. In the case of the weak signal, the latency of the cloud processor increases by 2.99×, and the PPT decreases by 67% compared to the strong signal case, so the PPT becomes similar to the DSP processor.

This implies that the thermal-aware scheduling system should also be aware of the dynamic network condition as well to utilize the cloud processor intelligently.

Chapter 4

SPLASH

In this chapter, we propose Splash, a thermal-aware DNN inference system with mobile-cloud collaborative execution to maximize long-term performance. To the best of our knowledge. Splash is the first system that considers both temperature and latency and maximizes the long-term performance for DNN execution on heterogeneous processors. Splash assigns the requested inference of the DNN model to one of the processors by calculating the real-time ever-changing thermal efficiency of the processors. To do this, we build system components that predict the thermal efficiency of the given workload and the network condition. As continuous vision AI applications demand the stream execution of DNN models, our system continuously selects the processor per each request.

4.1 System Overview

Figure 4.1 illustrates the system architecture of Splash. User applications request an inference (①) of a DNN model registered beforehand. Then, a cen-

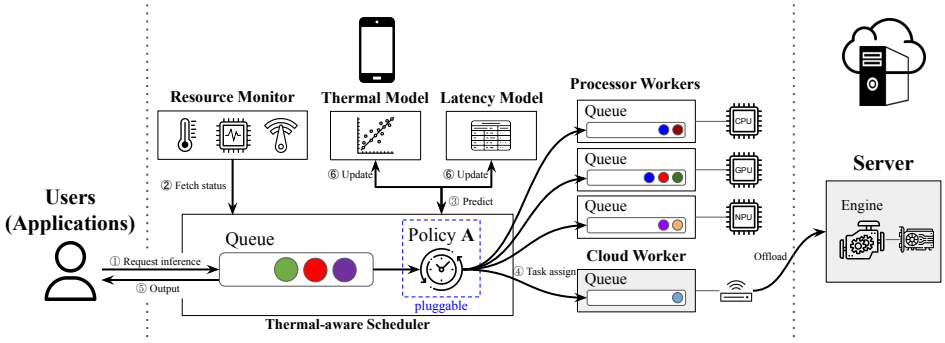


Figure 4.1: System architecture of SPLASH.

tral `Thermal-aware Scheduler` creates a `task` corresponding to the request and enqueues it to the task queue. Before assigning the task, the scheduler fetches device information (②) spanning temperatures of components, frequencies of processors, and signal strength of network connection from `Resource Monitor` so that `Thermal Model` can estimate the future temperature of each worker after a worker executes the task while `Latency Model` predicts the expected latency (③).

Given the information predicted, the scheduler assigns the task (④) to one of the available `Workers`, following its specific policy, which is pluggable according to the users' needs. The workers execute the inference with input data on its processor and then give the output to the scheduler, which passes it to the user (⑤). When the scheduler gives a task to `Cloud Worker`, the worker offloads the task to the cloud, which can utilize a powerful server GPU with a DNN serving engine. After finishing the user request, the scheduler feeds back to the prediction models to update their model parameter (⑥) with the result for a more accurate next prediction.

4.2 Considerations for Practicality

We focus on building a system that can be deployed practically on mobile devices by identifying the following considerations for the prediction models of Splash.

C1: A practical system requires to use of only internal sensors on the device. Various research [26, 37] has proposed thermal management systems that use additional instruments such as power monitors. However, in practical usage, service developers cannot benefit from the systems because they typically cannot utilize such instruments. In order to be deployed to the users, the system has to avoid the use of the additional device. Therefore, we design our system to fetch the values only from the available sensors on the mobile device.

C2: A deployable system needs to be independent of the device. Some work [38, 39] utilizes offline profiling or device-specific information such as floorplan to precisely model certain devices. The models so created are accurate since they are specialized to the target device, but sacrifice portability. As the modern smartphone market is hugely fragmented [40], such an approach cannot be used for a large portion of users. Thus, to be independent of the target device model, Splash profiles the thermal status and the latency information on runtime only.

C3: The scheduling module should provide sufficient accuracy without substantial overhead. Online models are known to be prone to error with coarse-grained sampling [41], because of the dynamic nature of thermal and network behavior. So, we adopt fine-grained sampling to retrieve the thermal status and the latency per request. However, with such granularity, the scheduling overhead can significantly hinder performance.

So, we engineer a lightweight model for temperature and latency to yield satisfactory accuracy with negligible overhead.

4.3 Lightweight Thermal Model

We estimate the future temperature with a simple-yet-effective linear model, which jointly considers the thermal coupling of the hardware components, latency, and network communication. Predicting the temperature of a system can be seen as a typical system identification problem that requires selecting a model structure based on an understanding of real system dynamics. Subsequently, the model parameters are optimized via parameter updates from the data collected in the actual environment. Prior works [42, 41, 43] have addressed modeling of the future temperature as a linear combination of current temperature and power consumption based on the previous study [44] as below:

$$\mathtt{T}_{t+1} = A\mathtt{T}_t + B\mathtt{P}_t \tag{4.1}$$

where $\mathtt{T}_t$ and $\mathtt{P}_t$ are the temperature and the power consumption of a timestamp t, respectively.

As processors are thermally coupled in SoC [33], all temperatures of processors should be packed into the current temperature. So, we denote the current temperature of the processors as a column vector $\mathtt{T}$ as below:

$$\mathtt{T} = [\mathtt{T}_{w_1},\ \mathtt{T}_{w_2},\ ...,\ \mathtt{T}_{w_{|W|}}] \tag{4.2}$$

where W is a set of available workers.

Unfortunately, most commodity smartphones do not have available sensors for measuring the power consumption of each hardware component [41].

While previous work [26] utilizes a power monitor device [29] to overcome the limitation, building prediction models offline would not be acceptable in actual use cases (**C1**). Instead, we estimate the power usage only using observable features in mobile devices.

As we aim to model for DNNs that are relatively not complex in control flow and mainly consist of data flow, we found a linear model can effectively forecast the temperature increase. As it will be shown in Chapter 6.5, it predicts the future temperature with insignificant overhead because we choose a lightweight model structure with a restricted number of model parameters (**C1, C2**).

Modeling for processor workers. For processor workers, power consumption can be estimated from core frequency, $\mathtt{F}$, which is known to have a strong linear relationship with power consumption [43].

$$\mathtt{F} = [\mathtt{F}_{w_1}, \mathtt{F}_{w_2}, ..., \mathtt{F}_{w_{|W|}}] \tag{4.3}$$

Plus, the expected latency $\mathtt{L}$ is included in the feature to reflect the amount of computation in the task and the idle energy consumption.

Considering the two features $\mathtt{F}$ and $\mathtt{L}$, the thermal structure for processors is modeled with X_P as follow:

$$\mathtt{T}_{t+1} = \theta_0 \mathtt{T}_t + \underbrace{\theta_1 \mathtt{F}_t + \theta_2 \mathtt{L}_t + \theta_3}_{\text{Power estimation}} \tag{4.4}$$

where θ_i is the model parameter for each feature.

Modeling for a cloud worker. Although a cloud worker does not compute the inference directly, it consumes power while transmitting input and receiving an output, generating heat in the process.

Feature		Description
X_P	$\mathtt{F}$	Frequency of all processors
	$\mathtt{L}$	Expected latency
X_C	$\mathtt{I}$	Size of the input to transmit
	$\mathtt{O}$	Size of output to receive from the server
	$\mathtt{R}$	Received Signal Strength Indicator (RSSI)
	$\mathtt{L}^{cloud}$	Expected latency to offloading.

Table 4.1: Features for the estimation of power consumption for the thermal model.

For the cloud worker, we select four features, denoted as X_C, that have a strong correlation with the power consumption of the network usage as shown in Table 4.1. The sizes of the input and output are directly connected to the power consumed by Network Interface Card (NIC) for both transmitting and receiving data. Note that since the power to transmit and receive data differs, the size of input and output should have separate model parameters. The signal strength of a wireless network influences the power usage in that the mobile platform adjusts the transmitting power depending on the strength [45] and, in the case it is weak, triggers the retransmission, thus increasing transmission time [46]. So, it is included in the model parameter. The expected latency should also be considered as NIC goes into the idle power usage state while it waits for the output, affecting the power consumption.

After all, the thermal model structure for the cloud worker is formulated as follows:

$$\mathtt{T}_{t+1} = \theta_0 \mathtt{T}_t + \underbrace{\theta_1 \mathtt{I}_t + \theta_2 \mathtt{O}_t + \theta_3 \mathtt{R}_t + \theta_4 \mathtt{L}_t^{cloud} + \theta_5}_{\text{Power estimation}} \tag{4.5}$$

where θ_i is the model parameter for each feature. According to the type

of network (e.g., Wi-Fi, LTE, and 5G), power consumption must be differentiated so that SPLASH prepares a thermal model per each network type.

Parameter updates. The thermal model updates its parameter by solving a least square problem with a normal equation, defined like below:

$$\theta = (X^T X)^{-1} X^T Y \tag{4.6}$$

where X and Y are the features and the measured temperature, respectively. The thermal-aware scheduler logs the features input to the previous prediction and the measured target, i.e., the true temperature and latency. Then, using the history, it updates the model at each inference. In order to avoid excessive memory use and to follow the recent trend, SPLASH does not store all the inference history, but only the recent part of them with the window size.

4.4 Adaptive Latency Model

While an inference time of the DNN model is considered to be predictable since it has few control flows [47], in the mobile device, it does not hold since thermal management systems like DVFS keep modifying the frequency of the processor. For example, the executions of the same DNN model show 3× difference according to the device temperature, as already depicted in Figure 2.1b. Additionally, when it comes to cloud augmentation, the latency of the cloud differs depending on the network condition, as shown in Figure 3.2. Thus, we design our latency model to be adaptive to such latency variety. As the inference request should wait until the earlier tasks are exhausted in a queue to respect the potential dependence, the latency is obtained by adding the inference time and the expected queuing delay.

Latency estimation for processor workers. For processors, the latency variability is originated from the core frequency change by thermal management systems. So, we model the inference time for processors as a map from the device temperature to the expected duration with a granularity of 1°C. Each value is updated by applying an exponential moving average with a factor of 0.1 after completing the task.

Latency estimation for a cloud worker. An inference time for a cloud worker consists of communication time and computation time, so it is formulated as below:

$$\mathtt{L}_t^{cloud} = \mathtt{Comp}_t + \mathtt{Comm}_t \tag{4.7}$$

Compared to mobile devices, the computation time $\mathtt{Comp}_t$ of the server shows fewer variability [47]. So, for computation time, the inference server profiles the execution time for the request, and reply it to the cloud worker. Then, the latency model of the cloud worker updates the computation time with a momentum similar to the processor workers without temperature mapping.

For the estimation of communication time $\mathtt{Comm}_t$, we model the communication time with bandwidth and latency for both downlink and uplink:

$$\mathtt{Comm}_t = \alpha_t^{\uparrow} + \frac{\mathtt{I}_t}{\beta_t^{\uparrow}} + \alpha_t^{\downarrow} + \frac{\mathtt{O}_t}{\beta_t^{\downarrow}} \tag{4.8}$$

where $\mathtt{I}$ and $\mathtt{O}$ are the size of inputs and outputs respectively. The latency and bandwidth for uplink and downlink are denoted as α and β with $\uparrow$ and $\downarrow$, respectively.

We design a linear regression for estimating communication time $\mathtt{Comm}_t$ with a short period of data to swiftly adapt to drastically changing network

Algorithm 1 Splash scheduler

```
1: function SCHEDULE
2:     workers, waiting_time ← [ ], [ ]
3:     for task in task_queue do
4:         workers ← GETAVAILABLEWORKERS()
5:         waiting_time ← UPDATEWAITINGTIME(workers)
6:         w_o ← GETOPTIMALWORKER(task, waiting_time)
7:         ENQUEUETASK(w_o, task)
```

conditions. The model structure of the communication time estimation is as follows:

$$\texttt{Comm}_t = \theta_0 \texttt{R}_t + \theta_1 \texttt{I}_t + \theta_2 \texttt{O}_t + \theta_3 \tag{4.9}$$

where R is the signal strength characterized by RSSI on mobile devices, which reflects the quality of the wireless connection. The model parameter tracks the up- and down-stream bandwidth (θ_1, θ_2), latency (θ_3), and RSSI-latency relationship (θ_0, θ_3) based on history. The model parameters are updated with the same logic as the temperature model.

4.5 Thermal-aware Scheduling and Policies

The scheduler of SPLASH schedules an inference request to one of the workers by making use of the real-time PPT value. When a user request inference to SPLASH, the scheduler crafts a task with a model identifier and a service-level objective designated by the user, and enqueues the task in its task queue. After fetching the status of the mobile device from the resource monitor and it schedules a task in a first-in-first-out manner, which is described in Algorithm 1 in detail. As a central scheduling component, it checks the health of all workers first and calculates waiting times for executing the current task on each worker (lines 4-5). Based on the information, the scheduler selects

Algorithm 2 Minimum Heat within SLO policy

```
1: function GetOptimalWorker(job, waiting_time)
2:     opt_worker, min_Δtemp ← null, ∞
3:     for worker in workers do
4:         exec_time ← PredictExecutionTime(worker, job)
5:         latency ← exec_time + waiting_time[worker]
6:         if latency > job.slo then
7:             continue
8:         Δtemp ← PredictTemperature(worker, job)
9:         if min_Δtemp > Δtemp then
10:            opt_worker, min_Δtemp ← worker, Δtemp
11:    return opt_worker
```

an optimal worker according to its specific policy and enqueues the task into the device queue of the worker (lines 6-7).

A scheduling policy used by the scheduler can be switched to any others, as the policy of the scheduler is designed to be pluggable to meet the application demand. We present several exemplary scheduling policies that can balance the latency and the heat generation with configurable parameters: (i) Minimum temperature increase within an SLO constraint and (ii) Maximum weighted PPT.

Minimum Heat within SLO. By the monotonous property of the logarithm, maximizing the PPT is the same as maximizing the difference between the logarithmic FPS and the logarithmic temperature increase as well.

$$\begin{aligned} w^* &=_{w \in W} \texttt{PPT}_w \\ &=_{w \in W} \log \texttt{FPS}_w - \log \Delta\texttt{temp}_w \end{aligned} \tag{4.10}$$

For the ideal case, the optimal solution would expect the maximal FPS and the minimal temperature increase. However, in a realistic setting, the latency and the amount of heat generation do not hold in the linear relationship due to multiple factors introduced in Chapter 3, so finding the ideal

Algorithm 3 Max Weighted PPT policy

```
1: function GetOptimalWorker(job, waiting_time)
2:     opt_worker, max_ppt ← null, −∞
3:     for worker in workers do
4:         Δtemp ← PredictTemperature(worker, job)
5:         exec_time ← PredictExecutionTime(worker, job)
6:         latency ← exec_time + waiting_time[worker]
7:         ppt ← (η − 1) ∗ Δtemp − η ∗ latency
8:         if max_ppt < ppt then
9:             opt_worker, max_ppt ← worker, ppt
10:    return opt_worker
```

solution becomes infeasible. Thus, to solve the multi-objective problem, we adopt the ϵ-constraint method to transform it into a single objective problem with SLO constraints.

$$\begin{aligned} w^* =_{w \in W} \quad & -\log \Delta\texttt{temp}_w \\ \text{s.t.} \quad & \log \texttt{FPS}_w > \log \texttt{SLO} \\ =_{w \in W} \quad & \Delta\texttt{temp}_w \\ \text{s.t.} \quad & \texttt{FPS}_w > \texttt{SLO} \end{aligned} \tag{4.11}$$

By doing so, the scheduler can switch between to pursue performance maximization or temperature increase minimization based on the given SLO constraint. In addition, by converting constraints into the penalty term, we formulate an unconstrained optimization problem.

$$w^* =_{w \in W} \Delta\texttt{temp}_w + \eta \max(0, \texttt{SLO} - \texttt{FPS}_w) \tag{4.12}$$

This formulation enables the scheduling policy to include explicit information about the SLO. So, the scheduler can make more precise decisions under the consideration of SLO violation.

Max Weighted PPT. Assigning the inference request just to the maximal PPT processor may result in a sub-optimal decision since the objective

treats the temperature increase and latency in the same dimension.

$$w^* =_{w \in W} \log \texttt{PPT}_w =_{w \in W} \texttt{latency}_w + \Delta\texttt{temp}_w \tag{4.13}$$

To balance the impact between the temperature increase and the latency, the policy adds the weight parameter η to the Equation 4.13, which becomes:

$$w^* =_{w \in W} \eta\texttt{latency}_w + (1 - \eta)\Delta\texttt{temp}_w \tag{4.14}$$

Equation 4.14 can be simplified into line 7 in Algorithm 3 by taking the logarithm. When the weight parameter is $\eta = 1$, then the policy becomes the thermal-agnostic system considering only the minimal latency. In other words, when the weight parameter is $\eta = 0$, the policy only maximizes the thermal efficiency without considering the time the processor spends. According to the application requirements, users can configure the weight parameter so that the system can achieve a satisfactory QoE for long-term execution.

4.6 Summary

To the best of our knowledge, we design the first system that can directly balance heat generation and instantaneous throughput for maximizing long-term performance on DNN execution. With lightweight prediction models, the system is capable of predicting the temperature and the latency of the given task with decent accuracy. Also, since Splash is designed with considerations for practicality, it can be deployed to various mobile devices without requiring additional measurement instruments and device-specific information.

Chapter 5

Implementation

We implement the mobile runtime of Splash upon TensorFlow Lite 2.3.0 [48] with 1.8K lines of C++ code. `ResourceMonitor` of Splash samples the sensor values of thermal zones and current frequencies of processors from Android kernel `sysfs`, and the type of network and signal strength from Android system service, `NetworkManager`. `ResourceMonitor` runs on a separate thread of one of the Little CPU cores, affecting little impact on the main processes. It samples all the values every 1 ms and stores them in the memory to return instantly when the prediction models or the scheduler of Splash require.

Splash uses gRPC as the communication protocol between the mobile and server runtime. The cloud worker calls API of `RequestInference` with the model id, and the width, height, and byte array of the input image. Then, the server executes the corresponding DNN model with the input image and then responds to the client with the output of the inference and computation time.

The cloud server uses TensorRT engine 8.5.0 with the runtime implemented with 0.3K lines of Python 3.8 code. We use full-integer quantized models both on the cloud server and mobile devices to maintain the same accuracy.[1] We use DNN models listed in Table 6.1.

[1]In our experiment, we verified a negligible difference in accuracy between mobile and cloud on image classifications with ImageNet.

Chapter 6

Evaluation

In this chapter, we evaluate Splash extensively to answer the following questions below:

- Can Splash perform better on long-term workloads? (Chapter 6.2)
- How Splash performs better than the thermal-agnostic system? (Chapter 6.3)
- How accurate is the thermal prediction model of Splash? (Chapter 6.4)
- What overheads are imposed to support thermal-aware scheduling? (Chapter 6.5)

6.1 Evaluation Setup

Figure 6.1 shows our environmental setup. We deploy Splash on two representative mobile devices: Google Pixel 4 and Samsung Galaxy S20, and also deploy the server runtime of Splash to the machine depicted in Table 2.1.

Workload	Task	Model	Input	# of Requests
Person Finder	Face Detection	RetinaFace-MobileNet [8]	160×160×3	4
	Face Recognition	ArcFace-MobileNet [49]	112×112×3	8
Live Video Analytics	Object Detection	EfficientDet-Lite0 [3]	360x×360×3	2
	Image Classification	MobileNetV2 [5]	224×224×3	10

Table 6.1: Two continuous vision AI application workloads.

For Samsung Galaxy S20, Kryo 585 2.84GHz Octa-Core, Adreno 650, and Hexagon 698 are equipped. NPU for Samsung Galaxy S20 is Hexagon Tensor Accelerator, which is included in Hexagon 698 chipset.

For a fair comparison, we have the mobile devices wait enough time inside the thermal incubator to adapt to the ambient temperature, 25°C. Also, we set up an isolated wireless network where only one hop between the mobile and the server exists to maintain the same bandwidth. Except in cases where we adjust the RSSI, we maintain the exact distance between the mobile device and the router during the experiments.

Baselines. We evaluate the performance of Splash with two baselines. We use the state-of-the-art multi-DNN inference framework, Band [12], that runs the model-level Heterogeneous earliest finish time (HEFT) [50] scheduling policy as our baseline. As the second baseline, we extend Band to include an option for cloud offloading to have equal physical resources, Band+Cloud.

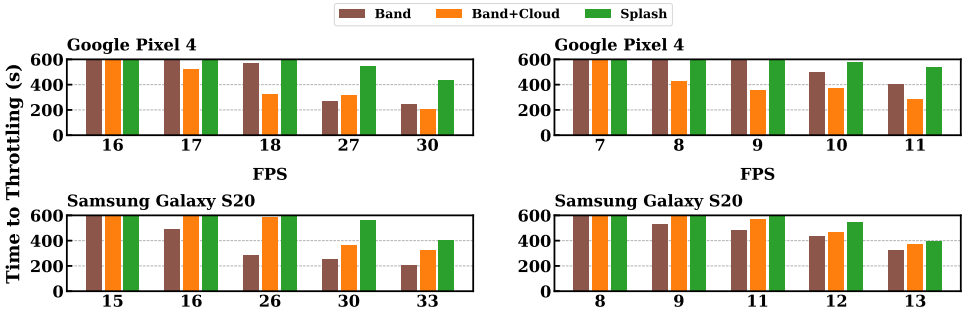
Metrics. We use *time to throttling* that measures how a system can last without throttling. Also, to assess the overall service quality, the SLO satisfaction rate is used. We denote a device is throttled when the `thermalStatus` from `PowerManager` in Android system service becomes *severe*, which is the level that user experience is largely impacted, according to [51].

Figure 6.1: The evaluation setup for SPLASH with two mobile devices, one cloud server with a network router, and the thermal incubator [7].

Workloads. We evaluate SPLASH with two workloads that have a camera frame sampling rate as a service-level objective as follows:

- Person Finder [4]: An application for finding a specific person such as a lost child or a criminal to chase.
- Live Video Analytics [52]: An analysis application of live video for detecting the object and classifying the object types in the large image.

The detail of the workloads is listed in Table 6.1. For the person finder workload, we assume a scenario in which the user application divides the camera frame into 4 subframes and issues requests for a face detection model, RetinaFace-MobileNet [8], on each frame. Then, the face detection requests return a total of 8 faces from all the subframes, so the user issues requests for a face recognition model, ArcFace-MobileNet [49] for each face. For the live video analytics, we assume a scenario in which the user application gives two



(a) Person Finder (b) Live Video Analytics

Figure 6.2: Experiment results over camera frame rate. SPLASH outperforms the baselines on both Google Pixel 4 and Samsung Galaxy S20 with two realistic workloads, reducing the amount of heat generation on the same performance. It displays the FPSes at which a system is first throttled.

camera frames to the object detection model, EfficientDet-Lite0 [3], where the model returns 10 bounding boxes. Then, the user requests image classification by executing MobileNet-V2 [5] for each box.

6.2 Performance for Long-term Workload

We evaluate our systems with the person finder and the live video analytics workloads in the long term, comparing two baselines on Google Pixel 4 and Samsung Galaxy S20 for 10 minutes.

For the mobile-only baseline in the person finder workload, our system extends the operational range of FPS without throttling by 1.53× and 1.93× on Google Pixel 4 and Samsung Galaxy S20 respectively, as shown in Figure 6.2. In the live video analytics workload on Samsung Galaxy S20, it achieves 1.5× FPS increase on operation without throttling. Note that the mobile-only baseline cannot preserve the thermal budget through offloading, so the time to throttling rapidly diminishes compared to SPLASH as the camera frame rate increases.

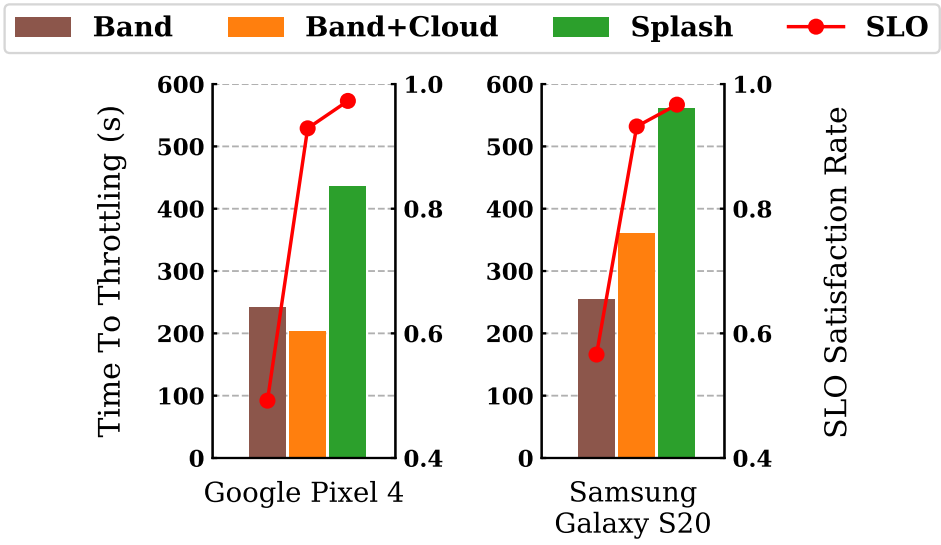


Figure 6.3: Experiment results of the person finder [4] workload with 30 FPS. SPLASH serves up to 2.20× longer to severe throttling status, resulting in a higher SLO satisfaction rate than baselines.

For the cloud-augmented baseline, our system increases the maximal FPS without throttling by up to 1.63×, effectively utilizing the saved thermal budget. The performance increase is more notable on Google Pixel 4 compared to that on Samsung Galaxy S20 (up to 1.16×) because the thermal cost of offloading is much larger in Google Pixel 4 than Samsung Galaxy S20 and the second baseline greedily assigns the task to the cloud worker to minimize the latency.

Figure 6.3 shows experimental results of the person finder workload on Google Pixel 4 and Samsung Galaxy S20. As Band does not leverage offloading and assigns all tasks to processors, the temperature of the device increases quickly, resulting in being throttled just in 242 seconds on Google Pixel 4 and 254 seconds on Samsung Galaxy S20. Thermal throttling degrades the performance in that it cannot satisfy the SLO of the user resulting in a lower

SLO satisfaction rate of 49.2%.

Although Band+Cloud is capable of offloading to the cloud, it still utilizes all the processors on the device at the same time to minimize latency. This thermal-agnostic task assignment makes the system be throttled in 203 seconds, which is earlier than the mobile-only baseline in the case of Google Pixel 4. However, after the processors are throttled, the system can offload tasks to the cloud alternatively, resulting in a higher SLO satisfaction ratio of 92.9%.

Splash with the *Minimum Heat within SLO* policy chooses the most thermally efficient worker within the SLO constraint. By thermally efficient task assignments, Splash preserves the temperature of hardware components on the mobile device, resulting in 2.20× more time to throttling in the long run.

6.3 Case study: Band+Cloud vs Splash

The scheduling difference at the specific frame between the second baseline and Splash is shown in Figure 6.4. Band+Cloud does not leverage the slack between the deadline and the finish time of the frame request as it is not designed to take care of thermal conditions, but only to minimize the makespan of one frame. Therefore, it assigns 4 RetinaFace inferences to three processors, GPU, DSP, and Cloud to work concurrently. This schedule can be optimal for minimizing latency but is not desirable for thermal aspects. In contrast, Splash assigns tasks considering both latency and temperature increase, so it utilizes the slack time to save the thermal budgets while respecting the deadline. It assigns all RetinaFace inferences to the Cloud of the highest thermal efficiency across workers as far as it does not make SLO violations. In the same context, it utilizes NPU at most for ArcFace-MobileNet requests.

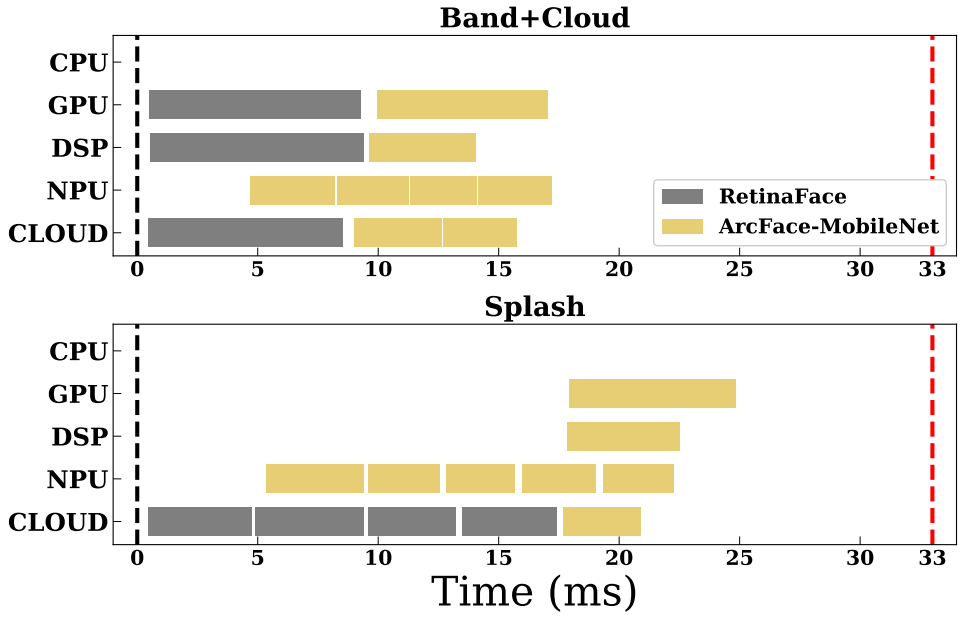


Figure 6.4: The difference of scheduling between Band+Cloud and our system within one frame on the person finder workload with 30 FPS. Splash assigns tasks to thermally efficient workers first.

Figure 6.5 shows the timelines of latency changes for the person finder 30 FPS experiment of Band+Cloud and Splash. As a result of the short-term perspective of scheduling, Band+Cloud shows a higher throughput at the start, but soon it gets throttled at 203 seconds, resulting in increased latency later. On the other side, our system saves thermal budgets proactively, lasting longer without being throttled until 436 seconds. Even after being throttled, Band+Cloud does assign a task without taking the temperature into account, so the thermal status gets worse, significantly violating the SLO. However, Splash is able to continuously select the thermal-efficient workers to manage the increase in temperature, which leads to a better SLO satisfaction ratio.

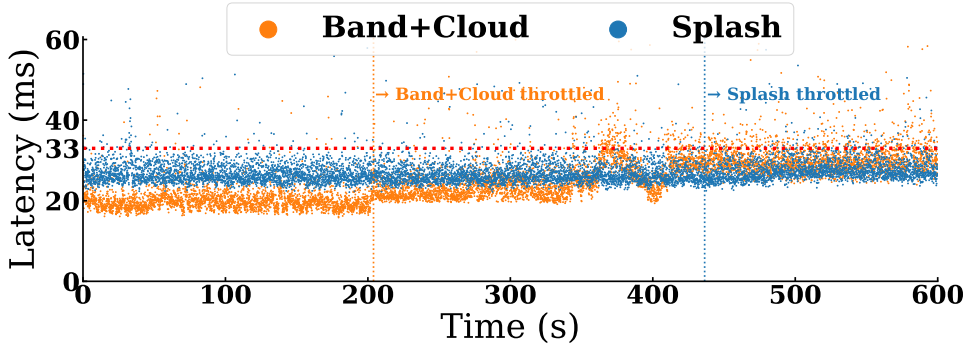


Figure 6.5: The timeline of latency changes over time of Band+Cloud and Splash in the person finder workload with FPS 30 on Google Pixel 4. Band+Cloud uses all available resources greedily for latency only, so it faces throttling sooner.

6.4 Thermal Model Accuracy

We assess the accuracy of thermal models in Splash. We run an application that issues requests for 4 models in Table 6.1 with a policy where the scheduler assigns the tasks randomly on any worker for 10 minutes with 15 ms intervals on Google Pixel 4 device. Then, we measure the root-mean-square deviation (RMSE) value for each prediction of thermal models per worker. As shown in Figure 6.6, we observe that our model predicts future temperature with acceptable accuracy in real-time. The thermal models give an error of 1.88°C, 0.40°C, 0.62°C, 0.34, and 1.00°C RMSE for each processor, respectively. For comparison, we measure the RMSE value between the current temperature and the future temperature as our baseline. The baseline gives an RMSE value of 4.8°C, 0.79°C, 1.53°C, 0.36, and 0.93°C, respectively. Note that due to frequency scaling by DVFS, the CPU temperature exhibits a larger error than those of other processors. Overall, our thermal model provides comparable accuracy to prior works [41, 53].

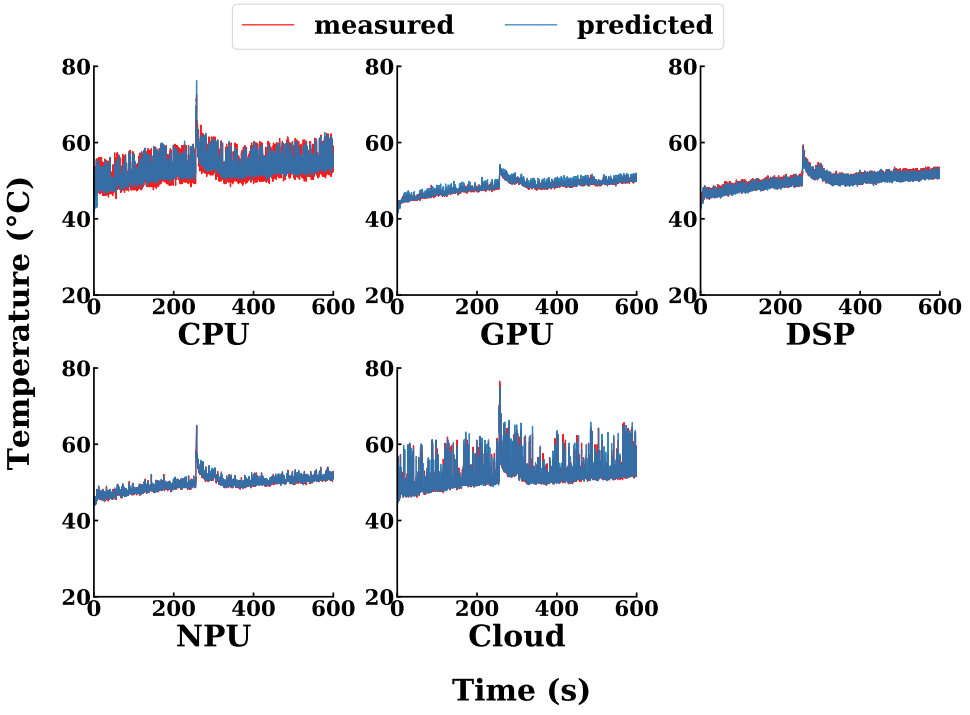


Figure 6.6: The performance of the thermal models of SPLASH. The predicted values follow the same trend as the measured values.

6.5 Scheduling Overhead

We evaluate the computational overhead for thermal-aware scheduling of SPLASH. We measure execution time on thermal prediction, scheduling with a policy, and parameter updates while running an application that issues an inference request of the EfficientDet-Lite0 [3] model repeatedly. Linear regression of future temperature prediction follows time complexity with $O(X \times \theta)$. As the number of features in X_P and X_C are 9 and 6, respectively, the overhead of temperature prediction is negligible, which is less than 2 microseconds. Scheduling with *Minimize heat generation with SLO constraint* policy

only takes less than 20 microseconds, including both the temperature and the latency model. Compared to the end-to-end latency (45 ms), the scheduling overhead of 20 microseconds is negligible, which takes only 0.04%.

For updating model parameters of the prediction models, SPLASH executes a normal equation whose time complexity follows with $O(X^2 \times W)$. We experimentally found that the window size of 2000 yields useful performance. The process takes up to 150 microseconds for the thermal models and 15 microseconds for the latency model of the cloud worker. However, it does not affect user experience as the update process can be progressed in a system idle state.

Chapter 7

Related Work

Mobile thermal management system. To resolve the overheating problem on mobile devices, prior works improve the existing vanilla DVFS. zTT [26] designed a DVFS that jointly scales CPU and GPU frequencies to achieve zero throttling by learning environmental changes based on deep reinforcement learning. Bhat et al. [27] proposed a technique to estimate power budgets based on the predicted temperature and utilize it on frequency scaling. There also have been efforts to mitigate thermal problems by efficient task scheduling. Lee et al. [33] suggested a thermally balanced assignment by exploiting processors' temperature imbalance.

Splash adapts to the existing thermal management systems and mitigates the thermal issues by scheduling tasks according to the heterogeneity of thermal efficiency across processors and collaboration with a cloud server.

On-device inference with heterogeneous processors. Considerable benefits of on-device inferences in terms of responsiveness have made active research on improvements. Band [12], the state-of-the-art multi-DNN

inference framework, used heterogeneous processors by partitioning a DNN model into subgraphs with considering fallback operators. CoDL [54] proposed an inference framework to co-utilize CPU and GPU by exploiting efficient data sharing and a lightweight latency predictor. Heimdall [13] proposed the pseudo-preemption technique for coordinating DNN and rendering for high GPU utilization. DeepMon [55] suggested a caching mechanism to reuse intermediate results of convolution layers to utilize mobile GPUs.

Compared to the prior works, Splash does consider thermal throttling due to heat generated while on-device inference by utilizing multiple processors with an understanding of the thermal coupling across processors on SoC.

Mobile-cloud collaborative execution. CloneCloud and Maui [56, 57] developed a system to decide partitioning points to divide an execution portion for mobile devices and the other for servers. To avoid immoderate dependence on network conditions and benefit from the performance increase of mobile devices, mobile-cloud collaborative DNN execution has been proposed. Neurosurgeon [58] extended a collaborative system with partitioning DNN models to maximize end-to-end throughput and energy efficiency. SPINN [30] suggested a progressive inference network for exploiting an early-exit in the collaboration process. Walle [31] and DynO [32] proposed a system to disperse loads of servers into mobile devices to leverage on-device inference performance.

However, Splash can be differentiated in that it takes the thermal conditions of mobile devices into account for deciding where to assign tasks across mobile processors and the cloud.

Chapter 8

Conclusion

Thermal throttling is a prevalent and severe problem for mobile devices, so thermal management systems have been of significant importance for continuous AI applications. In this paper, we presented Splash, the first thermal-aware DNN inference engine with mobile-cloud collaborative execution. Splash assigns user inference requests to thermally efficient workers based on the ever-changing PPT value of each worker according to the type of workload and network dynamism. Through extensive experiments on realistic multi-DNN workloads with representative hand-held devices, we have shown that Splash can save the thermal budgets of mobile devices by assigning tasks to more thermal efficient workers than the fastest-only workers. Splash can support 1.93× higher FPS without throttling than the state-of-the-art multi-DNN inference engine.

Bibliography

[1] Z. A. Pachodiwale, Y. Brahmankar, N. Parakh, D. Patel, and M. Eirinaki, "Viva: A virtual assistant for the visually impaired," in *Universal Access in Human-Computer Interaction. Design Methods and User Experience: 15th International Conference, UAHCI 2021, Held as Part of the 23rd HCI International Conference, HCII 2021, Virtual Event, July 24–29, 2021, Proceedings, Part I*, 2021.

[2] X. Zhou, C. Yao, H. Wen, Y. Wang, S. Zhou, W. He, and J. Liang, "East: An efficient and accurate scene text detector," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[3] M. Tan, R. Pang, and Q. V. Le, "Efficientdet: Scalable and efficient object detection," in *CVPR*, 2020.

[4] J. Yi, S. Choi, and Y. Lee, "Eagleeye: Wearable camera-based person identification in crowded urban spaces," in *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, MobiCom '20, 2020.

[5] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *CVPR*, 2018.

[6] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam, "Rethinking atrous convolution for semantic image segmentation," *arXiv preprint arXiv:1706.05587*, 2017.

[7] "Thermal Incubator C-IDN." `http://www.changshin-lab.com/eng/product/product.php`.

[8] J. Deng, J. Guo, E. Ververas, I. Kotsia, and S. Zafeiriou, "Retinaface: Single-shot multi-level face localisation in the wild," in *CVPR*, 2020.

[9] "Applications of AR glasses in industry." `https://www.queppelin.com/applications-of-ar-glasses-in-industry`.

[10] S. Huynh, R. K. Balan, J. Ko, and Y. Lee, "Vitamon: Measuring heart rate variability using smartphone front camera," in *Proceedings of the 17th Conference on Embedded Networked Sensor Systems*, SenSys '19, 2019.

[11] J. Bai, S. Lian, Z. Liu, K. Wang, and D. Liu, "Smart guiding glasses for visually impaired people in indoor environment," *IEEE Transactions on Consumer Electronics*, 2017.

[12] J. S. Jeong, J. Lee, D. Kim, C. Jeon, C. Jeong, Y. Lee, and B.-G. Chun, "Band: Coordinated multi-dnn inference on heterogeneous mobile processors," in *Proceedings of the 20th Annual International Conference on Mobile Systems, Applications and Services*, MobiSys '22, 2022.

[13] J. Yi and Y. Lee, "Heimdall: Mobile gpu coordination platform for augmented reality applications," in *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, MobiCom '20, 2020.

[14] T. Braud, F. H. Bijarbooneh, D. Chatzopoulos, and P. Hui, "Future networking challenges: The case of mobile augmented reality," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, 2017.

[15] "ARM Mali GPU." `https://developer.arm.com/ip-products/graphics-and-multimedia/mali-gpus`.

[16] "Qualcomm Adreno GPU." `https://www.qualcomm.com/products/features/adreno`.

[17] "Qualcomm Hexagon DSP." `https://developer.qualcomm.com/software/hexagon-dsp-sdk/dsp-processor`.

[18] J. Song, Y. Cho, J.-S. Park, J.-W. Jang, S. Lee, J.-H. Song, J.-G. Lee, and I. Kang, "7.1 an 11.5tops/w 1024-mac butterfly structure dual-core sparsity-aware neural processing unit in 8nm flagship mobile soc," in *2019 IEEE International Solid-State Circuits Conference - (ISSCC)*, 2019.

[19] "Mediatek NeuroPilot." `https://neuropilot.mediatek.com/`.

[20] "Google EdgeTPU." `https://cloud.google.com/edge-tpu`.

[21] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[22] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," *CoRR*, 2019.

[23] D. Brooks and M. Martonosi, "Dynamic thermal management for high-performance microprocessors," in *Proceedings HPCA Seventh International Symposium on High-Performance Computer Architecture*, 2001.

[24] "SDM Configuration." `https://gitlab.com/aosp1/device/google/coral/-/blob/master/thermal-engine-flame-normal_mode.conf`.

[25] "Exynos TMU." `https://www.kernel.org/doc/html/v5.4/driver-api/thermal/exynos_thermal.html`.

[26] S. Kim, K. Bin, S. Ha, K. Lee, and S. Chong, "Ztt: Learning-based dvfs with zero thermal throttling for mobile devices," in *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '21, 2021.

[27] G. Bhat, G. Singla, A. K. Unver, and U. Y. Ogras, "Algorithmic optimization of thermal and power management for heterogeneous mobile platforms," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2018.

[28] B. Egilmez, G. Memik, S. Ogrenci-Memik, and O. Ergin, "User-specific skin temperature-aware dvfs for smartphones," in *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2015.

[29] "Monsoon High Voltage Power Monitor." `https://www.msoon.com/high-voltage-power-monitor`.

[30] S. Laskaridis, S. I. Venieris, M. Almeida, I. Leontiadis, and N. D. Lane, "Spinn: Synergistic progressive inference of neural networks over device and cloud," in *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, MobiCom '20, 2020.

[31] C. Lv, C. Niu, R. Gu, X. Jiang, Z. Wang, B. Liu, Z. Wu, Q. Yao, C. Huang, P. Huang, T. Huang, H. Shu, J. Song, B. Zou, P. Lan, G. Xu, F. Wu, S. Tang, F. Wu, and G. Chen, "Walle: An End-to-End, General-Purpose, and Large-Scale production system for Device-Cloud collaborative machine learning," in *16th USENIX Symposium on Operating Systems Design and Implementation*, OSDI '22, 2022.

[32] M. Almeida, S. Laskaridis, S. I. Venieris, I. Leontiadis, and N. D. Lane, "Dyno: Dynamic onloading of deep neural networks from cloud to device," *ACM Trans. Embed. Comput. Syst.*, jan 2022.

[33] Y. Lee, K. G. Shin, and H. S. Chwa, "Thermal-aware scheduling for integrated cpus–gpu platforms," *ACM Trans. Embed. Comput. Syst.*, 2019.

[34] Y. Jeon and J. Kim, "Constructing fast network through deconstruction of convolution," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, (Red Hook, NY, USA), p. 5955–5965, Curran Associates Inc., 2018.

[35] Z. Qin, Z. Zhang, D. Li, Y. Zhang, and Y. Peng, "Diagonalwise refactorization: An efficient training method for depthwise convolutions," in *2018 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, 2018.

[36] P. Henderson, J. Hu, J. Romoff, E. Brunskill, D. Jurafsky, and J. Pineau, "Towards the systematic reporting of the energy and carbon footprints of machine learning," *Journal of Machine Learning Research*, vol. 21, no. 248, pp. 1–43, 2020.

[37] Q. Xie, J. Kim, Y. Wang, D. Shin, N. Chang, and M. Pedram, "Dynamic thermal management in mobile devices considering the thermal coupling between battery and application processor," in *2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 242–247, 2013.

[38] M. J. Dousti, M. Ghasemi-Gol, M. Nazemi, and M. Pedram, "Thermtap: An online power analyzer and thermal simulator for android devices," in *2015 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pp. 341–346, 2015.

[39] S. Murali, A. Mutapcic, D. Atienza, R. Gupta, S. Boyd, L. Benini, and G. De Micheli, "Temperature control of high-performance multi-core platforms using convex optimization," in *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '08, (New York, NY, USA), p. 110–115, Association for Computing Machinery, 2008.

[40] "Android Fragmentation Visualized (August 2015)." `https://cdn.opensignal.com/public/data/reports/global/data-2015-08/2015_08_fragmentation_report.pdf`, 2022.

[41] O. Kwon, W. Jang, G. Kim, and C. Lee, "Accurate thermal prediction for nans (n-app n-screen) services on a smart phone," in *2018 IEEE 13th International Symposium on Industrial Embedded Systems*, SIES '18, 2018.

[42] F. Paterna and T. Rosing, "Modeling and mitigation of extra-soc thermal coupling effects and heat transfer variations in mobile devices," in *2015 IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '15, 2015.

[43] P. Mercati, T. S. Rosing, V. Hanumaiah, J. Kulkarni, and S. Bloch, "User-centric joint power and thermal management for smartphones," in *6th International Conference on Mobile Computing, Applications and Services*, 2014.

[44] K. Skadron, M. R. Stan, K. Sankaranarayanan, W. Huang, S. Velusamy, and D. Tarjan, "Temperature-aware microarchitecture: Modeling and implementation," *ACM Trans. Archit. Code Optim.*, vol. 1, p. 94–125, mar 2004.

[45] N. Ding, D. Wagner, X. Chen, A. Pathak, Y. C. Hu, and A. Rice, "Characterizing and modeling the impact of wireless signal strength on smartphone battery drain," in *Proceedings of the ACM SIGMETRICS/International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '13, 2013.

[46] Y. G. Kim, Y. S. Lee, and S. W. Chung, "Signal strength-aware adaptive offloading with local image preprocessing for energy efficient mobile devices," *IEEE Transactions on Computers*, 2020.

[47] A. Gujarati, R. Karimi, S. Alzayat, W. Hao, A. Kaufmann, Y. Vigfusson, and J. Mace, "Serving DNNs like clockwork: Performance predictability from the bottom up," in *14th USENIX Symposium on Operating Systems Design and Implementation*, OSDI '20, 2020.

[48] "TensorFlow Lite." `https://www.tensorflow.org/lite`.

[49] J. Deng, J. Guo, N. Xue, and S. Zafeiriou, "Arcface: Additive angular margin loss for deep face recognition," in *CVPR*, 2019.

[50] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Transactions on Parallel and Distributed Systems*, 2002.

[51] "Android Developer Documentation: PowerManager." `https://developer.android.com/reference/android/os/PowerManager#THERMAL_STATUS_SEVERE`, 2022.

[52] X. Zeng, B. Fang, H. Shen, and M. Zhang, "Distream: Scaling live video analytics with workload-adaptive distributed edge intelligence," in *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*, SenSys '20, (New York, NY, USA), p. 409–421, Association for Computing Machinery, 2020.

[53] S. Kang, H. Choi, S. Park, C. Park, J. Lee, U. Lee, and S.-J. Lee, "Fire in your hands: Understanding thermal behavior of smartphones," in *The 25th Annual International Conference on Mobile Computing and Networking*, MobiCom '19, 2019.

[54] F. Jia, D. Zhang, T. Cao, S. Jiang, Y. Liu, J. Ren, and Y. Zhang, "Codl: Efficient cpu-gpu co-execution for deep learning inference on mobile devices," in *Proceedings of the 20th Annual International Conference on Mobile Systems, Applications and Services*, MobiSys '22, 2022.

[55] L. N. Huynh, Y. Lee, and R. K. Balan, "Deepmon: Mobile gpu-based deep learning framework for continuous vision applications," in *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '17, 2017.

[56] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: Elastic execution between mobile device and cloud," in *Proceedings of the Sixth Conference on Computer Systems*, EuroSys '11, 2011.

[57] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: Making smartphones last longer with code offload," in *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, MobiSys '10, 2010.

[58] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," in *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '17, 2017.

Acknowledgements

I am sincerely indebted to my advisor Byung-Gon Chun for constructive guidance and to the members of Software Platform Lab for thorough proofreading and valuable feedback. I am also deeply grateful to professor Young-Ki Lee and the members of SNU MR for their detailed suggestions and generous encouragement. It has been fortunate and pleased to co-work with to build such valuable systems.

Thanks should also go to Samsung for giving me this special opportunity to grow further and my teammates for understanding me to focus on research over the past 2 years. Lastly, I would like to mention that this work would not have been possible without emotional support from my family, especially Hyun-Ji, who has always relieved the pressure I felt whenever facing difficulty in achieving progress in research.

초록

모바일 기기에서 연속적인 시각 인공지능 서비스를 사용하는 경우, 오랜 시간 동안 연속적으로 발생하는 다수의 딥 뉴럴 네트워크 (DNN) 추론으로 인해 모바일 기기의 과열 문제가 발생하게 된다. 모바일 기기의 여러 프로세서를 동시에 활용하는 것이 DNN 추론 실행에 유망한 반면, 이로 인해 발생하는 열은 모바일 기기의 온도 임계치를 초과하게 만드는 원인이 된다. 대안으로 모바일 기기와 클라우드가 협동하여 추론하는 시스템이 제안되고 있지만, 기존 시스템은 단순히 추론의 지연 시간을 최소화하는 것에만 목적을 두고 모바일 기기의 열 상태를 고려하지 않는다는 한계가 있다.

본 논문에서는 이기종 프로세서 간의 온성비가 실행하는 워크로드와 동적인 네트워크 상태 변화에 따라 우선 순위가 변경하는 사실을 활용하는 열 인지 기반 모바일-클라우드 DNN 협동 추론 시스템인 Splash 를 제안한다. Splash 는 미래의 온도와 지연 시간을 효율적으로 추측하는 모델을 바탕으로 열 인지 스케줄링을 통해 모바일 기기의 열 여유를 확보한다. 본 논문에서는 두 개의 스마트폰에서 실제 워크로드를 10분간 실행하는 실험을 진행하여, Splash 가 기존 다중 DNN 추론 시스템에 비해 1.93 배 더 높은 FPS를 모바일 기기의 쓰로틀링 없이 지원할 수 있음을 보인다.

주요어: 모바일 딥러닝 추론, 쓰로틀링, 클라우드 오프로딩

학번: 2021-21645