Boosting DNN Cold Inference on Edge Devices

Rongjie Yi[1], Ting Cao[2], Ao Zhou[1], Xiao Ma[1], Shangguang Wang[1], Mengwei Xu[1]

[1] State Key Laboratory of Networking and Switching Technology
[2]Microsoft Research

DNN is indispensable for mobile apps

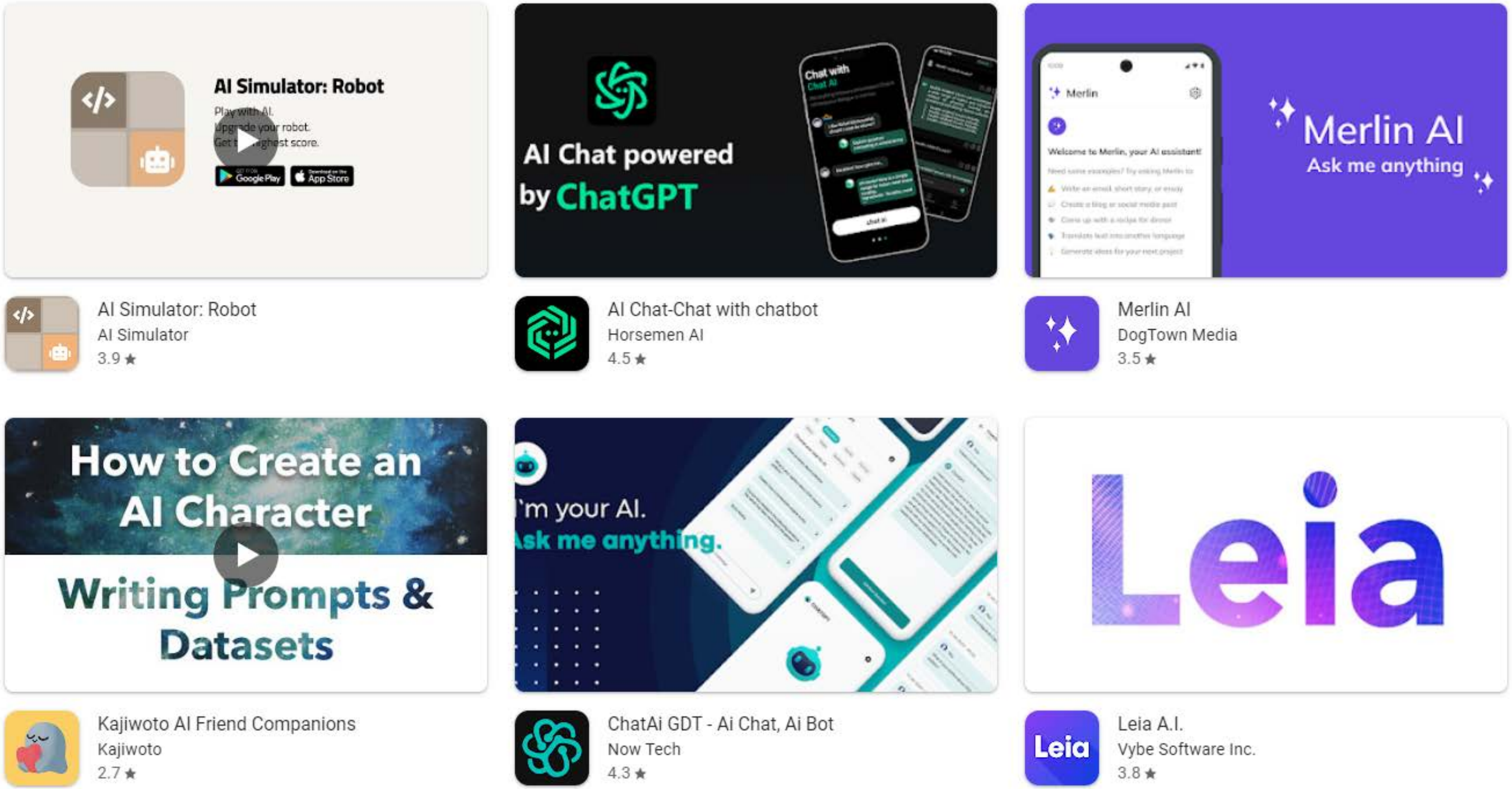
Face recognition

Voice recognition

Augmented Reality

DNN is indispensable for mobile apps

F ality



DNN-embedded apps on Google Play

DNN is indispensable for mobile apps

Face recognition

Voice recognition

Augmented Reality

Deploy

smartphones

IoTs

wearables

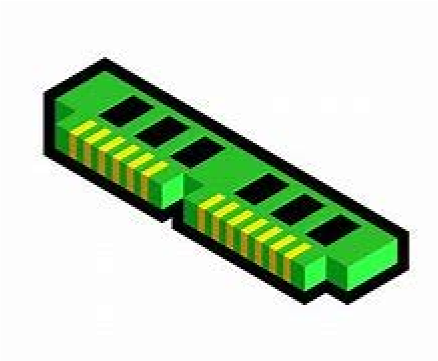
Challenges of deploying DNNs on devices

Challenges of deploying DNNs on devices

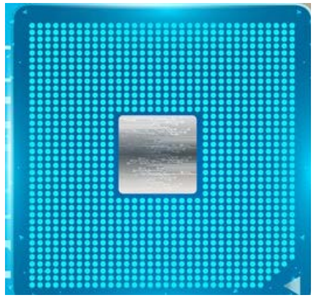
1. The tightly constrained hardware resources

Challenges of deploying DNNs on devices

1. The tightly constrained hardware resources



memory



compute

energy

To obtain more accurate DNN inference results on devices with limited memory

Challenges of deploying DNNs on devices

1. The tightly constrained hardware resources
2. The volatile, multi-tenant(app) runtime environment

Challenges of deploying DNNs on devices

1. The tightly constrained hardware resources
2. The volatile, multi-tenant(app) runtime environment

For devices with limited memory,

- It's inevitable to switch between multiple DNN inference.
- DNNs cannot always reside in device memory.

Challenges of deploying DNNs on devices

1. The tightly constrained hardware resources
2. The volatile, multi-tenant(app) runtime environment

For devices with limited memory,

- It’s inevitable to switch between multiple DNN inference.
- DNNs cannot always reside in device memory.

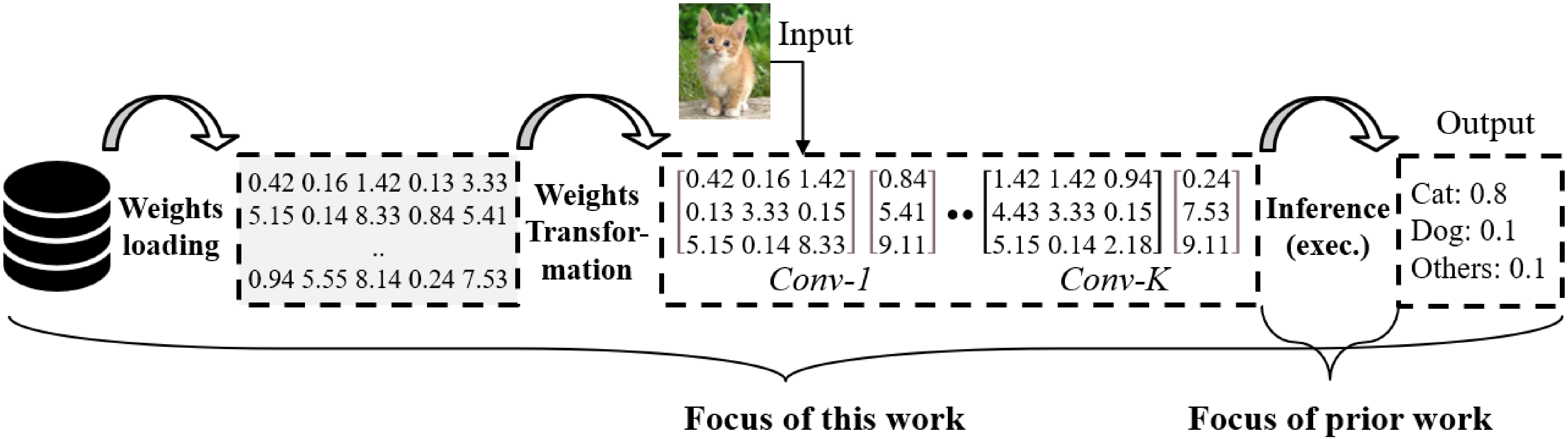
DNN inference often occurs in cold inference manner

What is cold inference?

load and initialize the model weights into memory before execution.

What is cold inference?

load and initialize the model weights into memory before execution.



Cold inference

Warm inference

DNN cold inference’s classification

DNN cold inference’s classification

I. Active Cold Inference

- Developers avoids a model residing in memory for a long time.

DNN cold inference's classification

I. Active Cold Inference

- Developers avoids a model residing in memory for a long time.
- Mobile apps re-launch models that are infrequently used.

DNN cold inference's classification

I. Active Cold Inference

- Developers avoids a model residing in memory for a long time.
- Mobile apps re-launch models that are infrequently used.

PDF Scanner

Beauty Camera

Voice Assistant

DNN cold inference's classification

I. Active Cold Inference

- Developers avoids a model residing in memory for a long time.
- Mobile apps re-launch models that are infrequently used.

⇩

Pack all DNNs into device memory through weights sharing

DNN cold inference's classification

I. Active Cold Inference

- Developers avoids a model residing in memory for a long time.
- Mobile apps re-launch models that are infrequently used.

⇩

Pack all DNNs into device memory through weights sharing

⇩

Unscalable

DNN cold inference’s classification

I. Active Cold Inference
II. Passive Cold Inference

DNN cold inference's classification

I. Active Cold Inference

II. Passive Cold Inference

- Mobile Phone OS kills background apps to reduce memory footprint

Mobile Phone OS

DNN cold inference's classification

I. Active Cold Inference

II. Passive Cold Inference

- Mobile Phone OS kills background apps to reduce memory footprint
- Mobile Browsers relaunch a model whenever web pages are opened

Mobile Browsers

DNN cold inference's classification

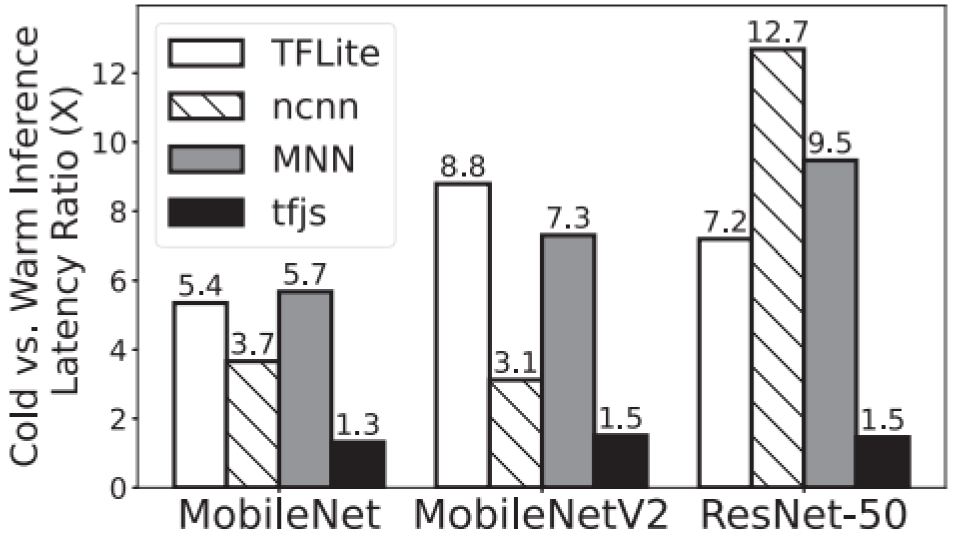
I. Active Cold Inference

II. Passive Cold Inference

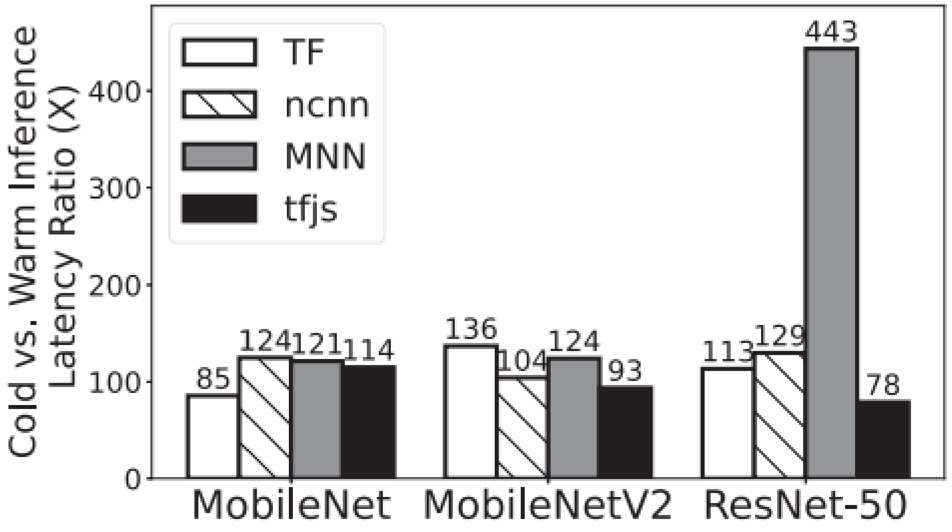
- Mobile Phone OS kills background apps to reduce memory footprint
- Mobile Browsers relaunch a model whenever web pages are opened
- DNN-based software could crash and needs to fast re-launch.

Autonomous Driving

Cold inference is poorly supported



Google Pixel 5 CPU

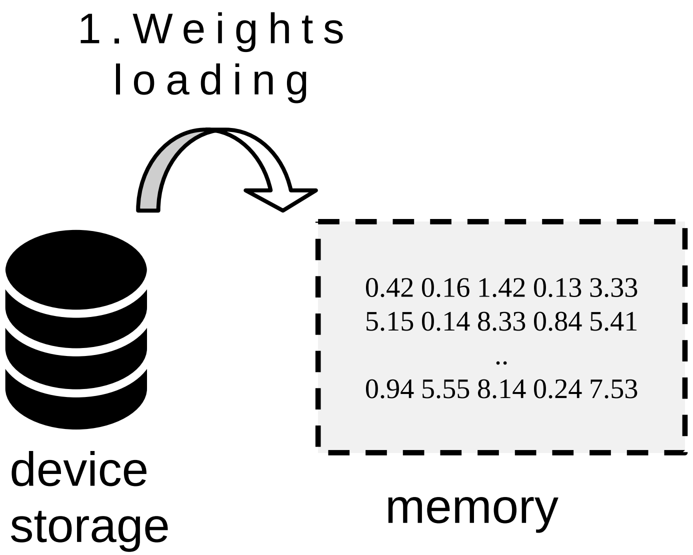


Jetson TX2 GPU

Huge gap between cold/warm Inference latency hurt the user experience.

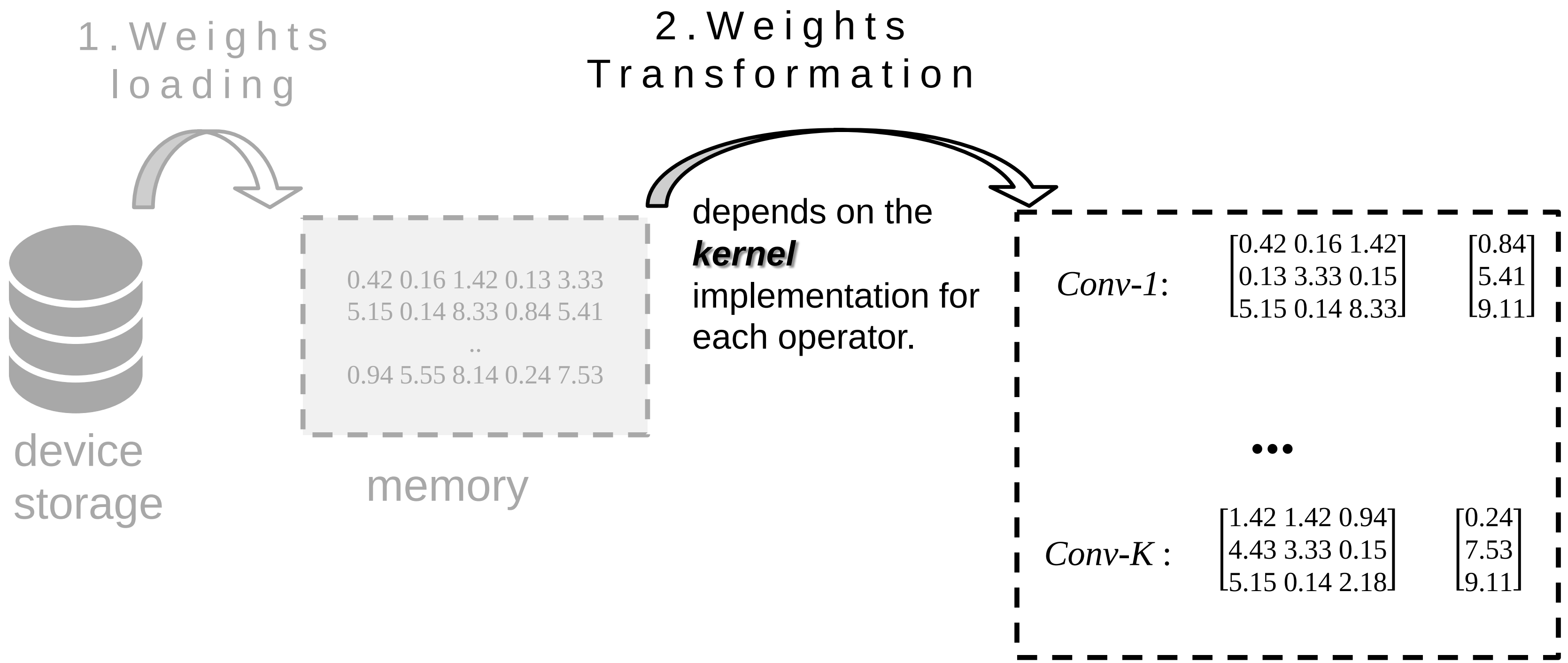
Cold inference breakdown

Cold inference breakdown



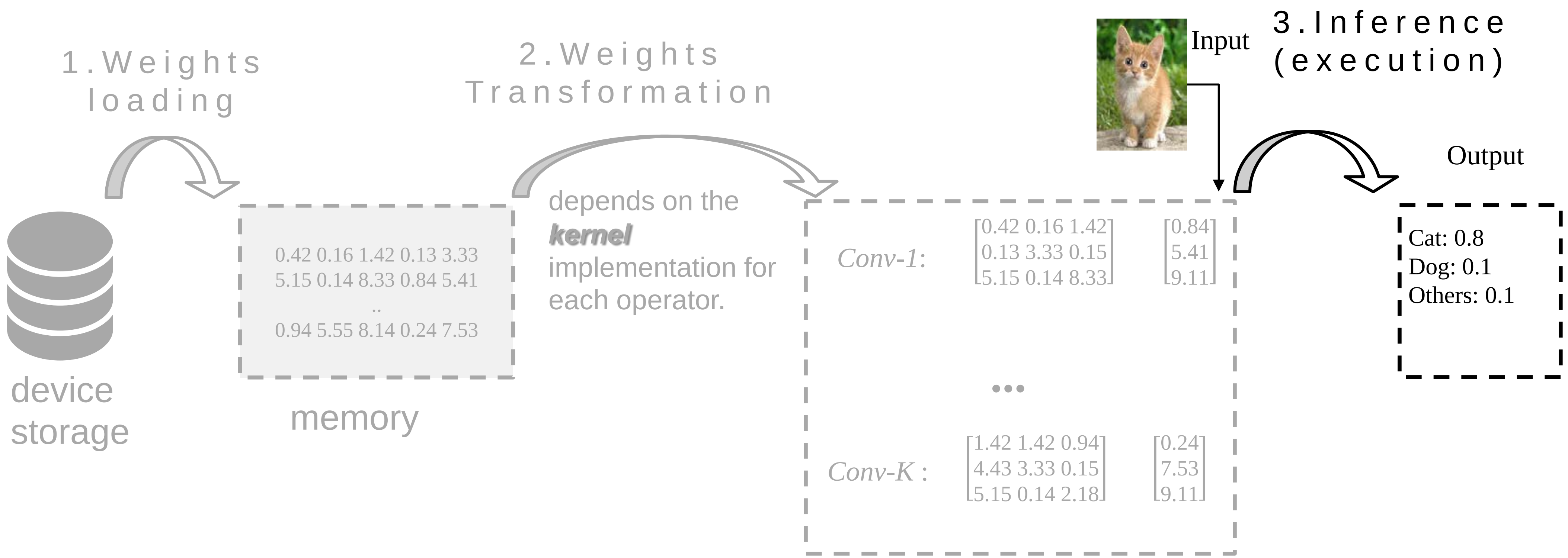
Reading the model weights from device storage into memory

Cold inference breakdown



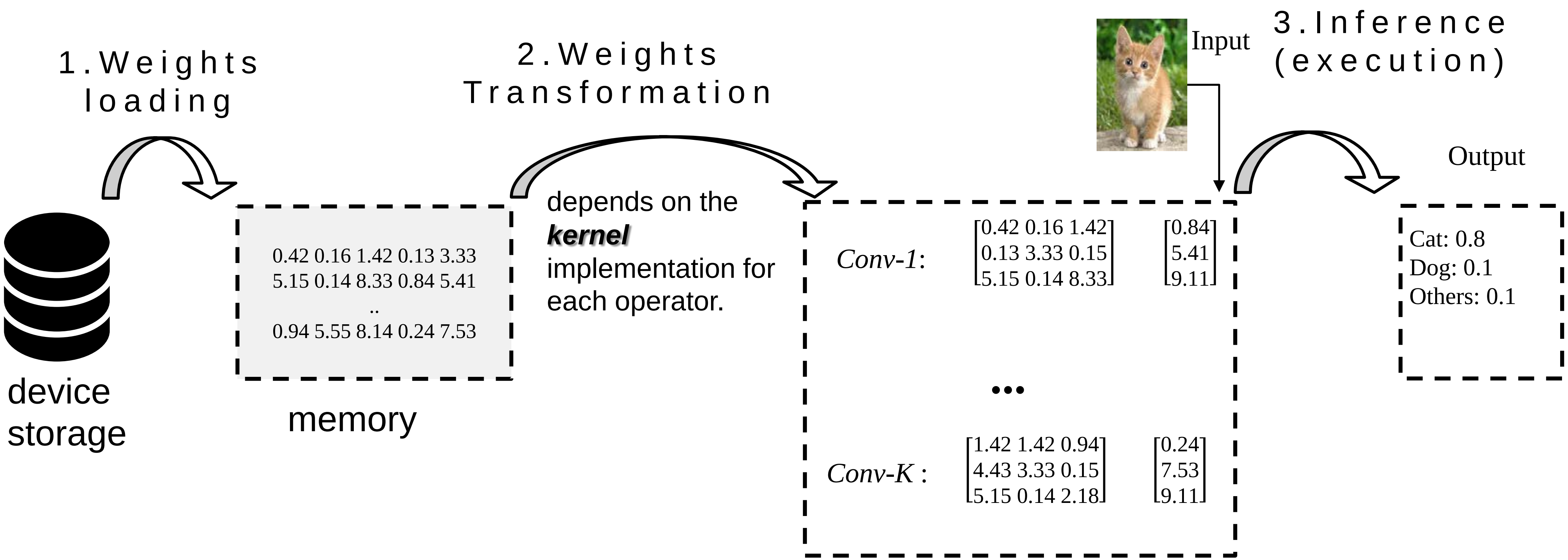
Converting raw weights into the proper format to facilitate the inference

Cold inference breakdown



The actual inference process by invoking each operator of the model

Cold inference breakdown



Cold inference breakdown

A breakdown of ResNet-50 cold inference latency on edge CPU and GPU.

Device Platform	Google Pixel 5	Jetson TX2
Processor	CPU	GPU
Weights loading	37.86ms	46.72ms
Weights Transformation	1135.28ms	4620.85ms
Execution	190.12ms	802.77ms
Total cold inference	1363.23ms	5467.48ms
Warm inference	185.82ms	137.02ms

How to speedup cold inference?

Optimization Knobs#1: Kernel selection

- A DNN model can be represented as a **directed data graph** consisting of many **operators**.
- **Kernels** represent the different implementation of an operator

Optimization Knobs#1: Kernel selection

- A DNN model can be represented as a **directed data graph** consisting of many **operators**.
- **Kernels** represent the different implementation of an operator

One operator, many kernels:

e.g. Convolution Operator's Kernels:

Kernels	Cold Inference Time (ms)		
	Load Weights	Weights Trans.	Execution
3x3s1-winograd-pack4	0.70	38.23	2.98
sgemm-pack4	0.70	2.21	8.14
pack4	0.70	2.22	18.63
3x3s1-winograd	0.70	65.67	3.37
3x3s1	0.70	0.00	8.01
general	0.70	0.00	87.12

Optimization Knobs#1: Kernel selection

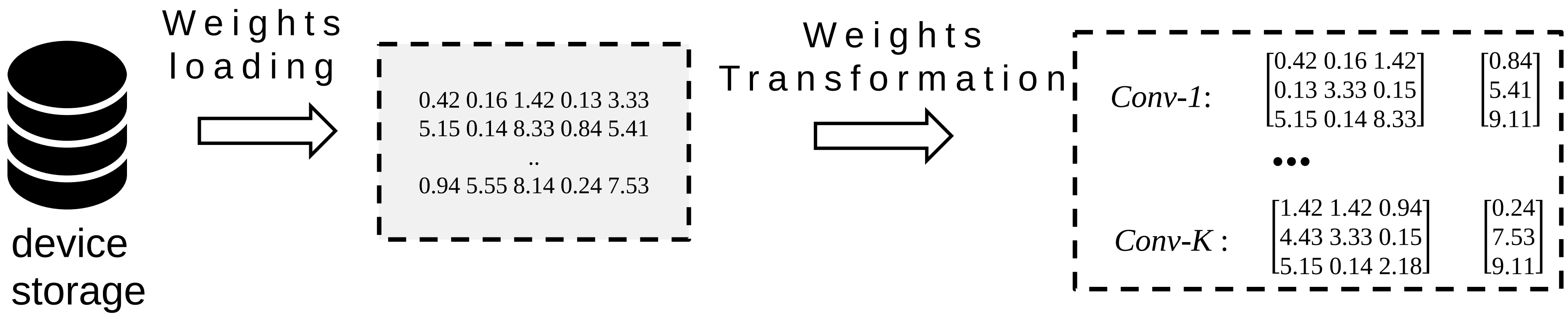
The current kernel selection policy:

- hard-coded
- only considers the warm inference speed

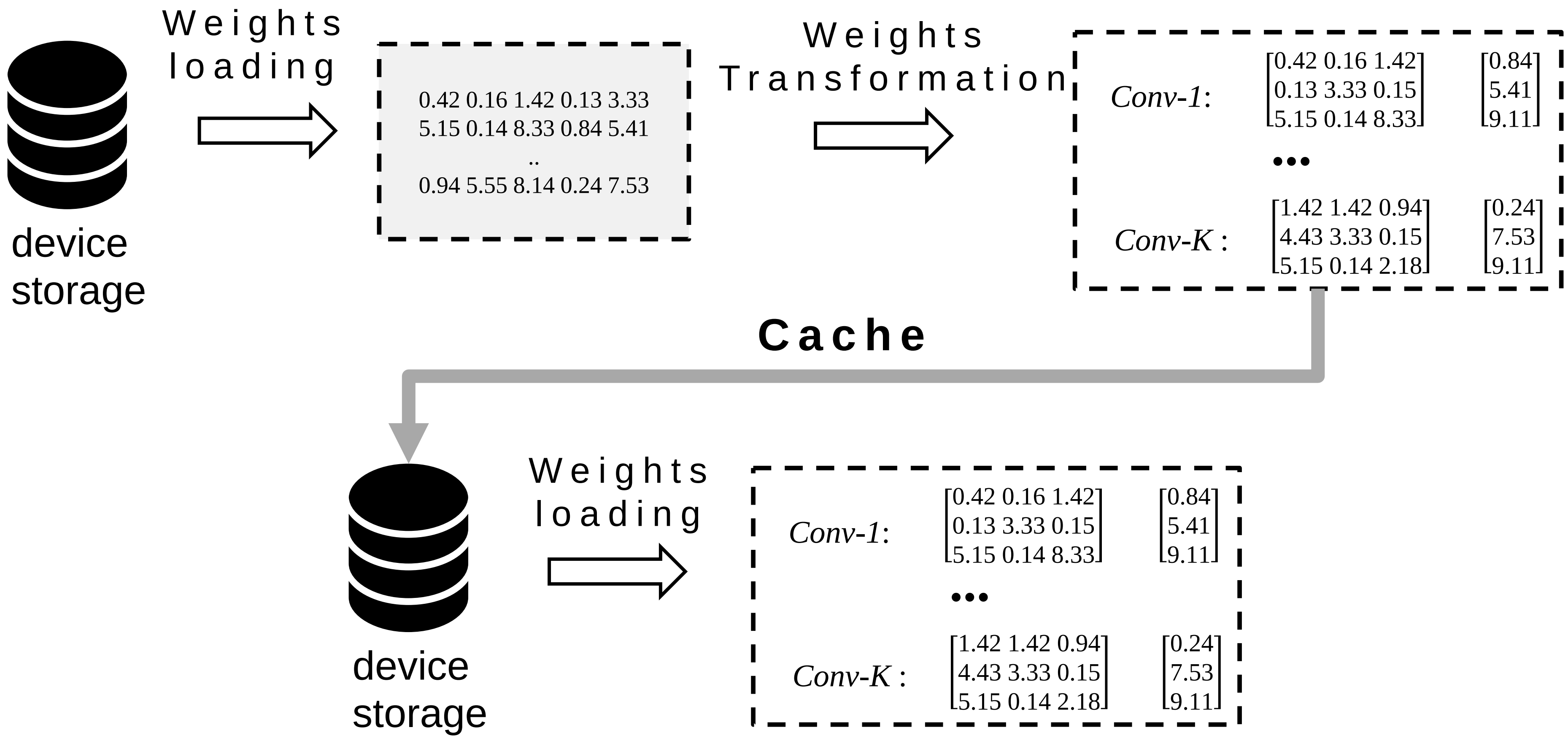
Our new kernel selection policy:

- Automatic
- be optimal for cold inference

Optimization Knobs#2: Bypassed weights transformation



Optimization Knobs#2: Bypassed weights transformation



Optimization Knobs#3: Pipelined inference

Edge devices are typically equipped with multi-process architecture

e.g. Arm big.LITTLE architecture:

- " LITTLE" processors are designed for maximum power efficiency.
- "big" processors are designed to provide efficient compute performance.

Optimization Knobs#3: Pipelined inference

Edge devices are typically equipped with multi-process architecture

e.g. Arm big.LITTLE architecture:

- ″ LITTLE″ processors are designed for maximum power efficiency.
- “big” processors are designed to provide efficient compute performance.

Multi-Thread the kernel preparation and execution

Optimization Knobs#3: Pipelined inference

Weights loading: bounded by disk I/O.

Weights Transformation: bounded by memory I/O.

Execution: bounded by computation.

Optimization Knobs#3: Pipelined inference.

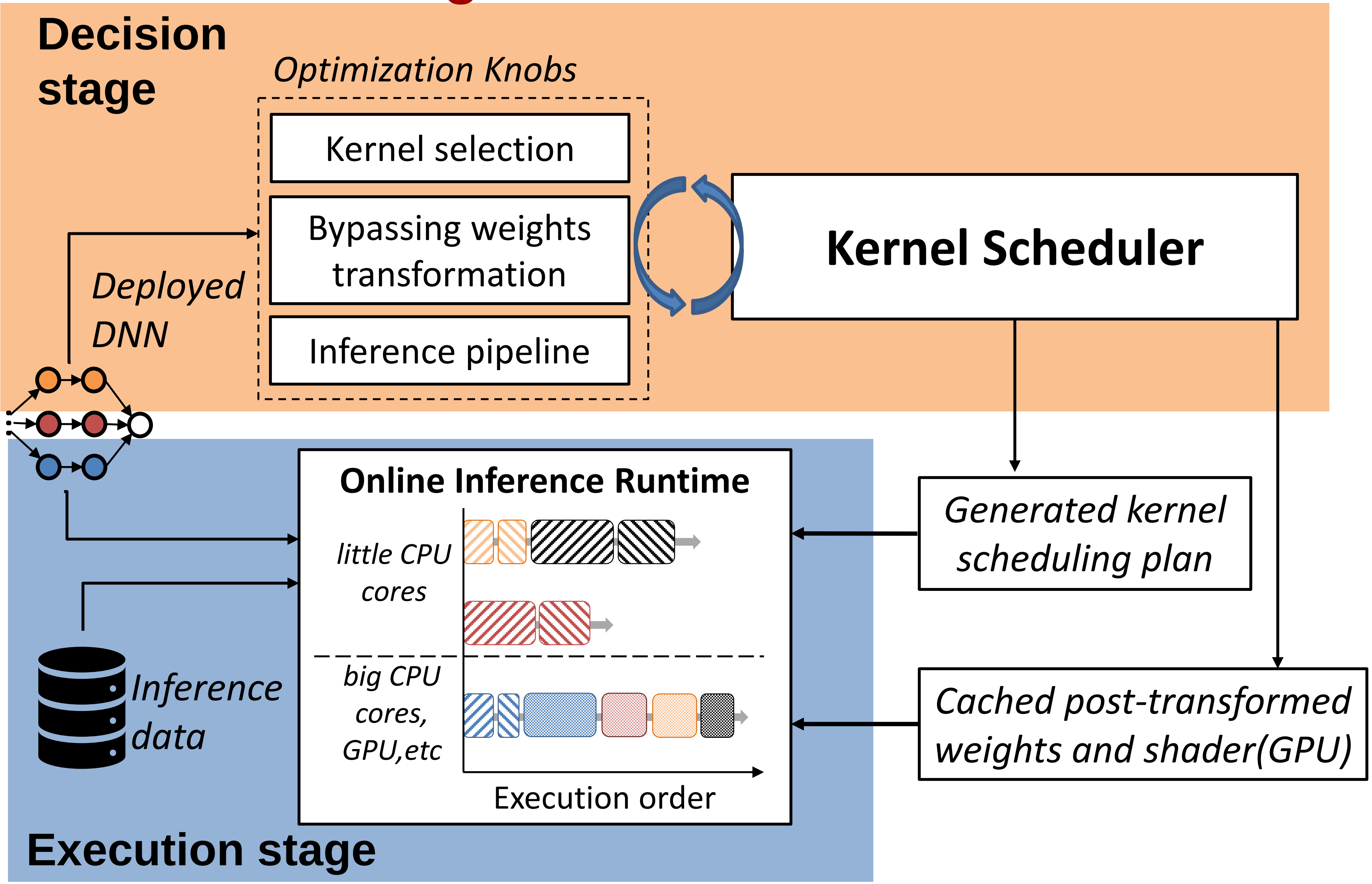
Weights loading: bounded by disk I/O.

Weights Transformation: bounded by memory I/O.

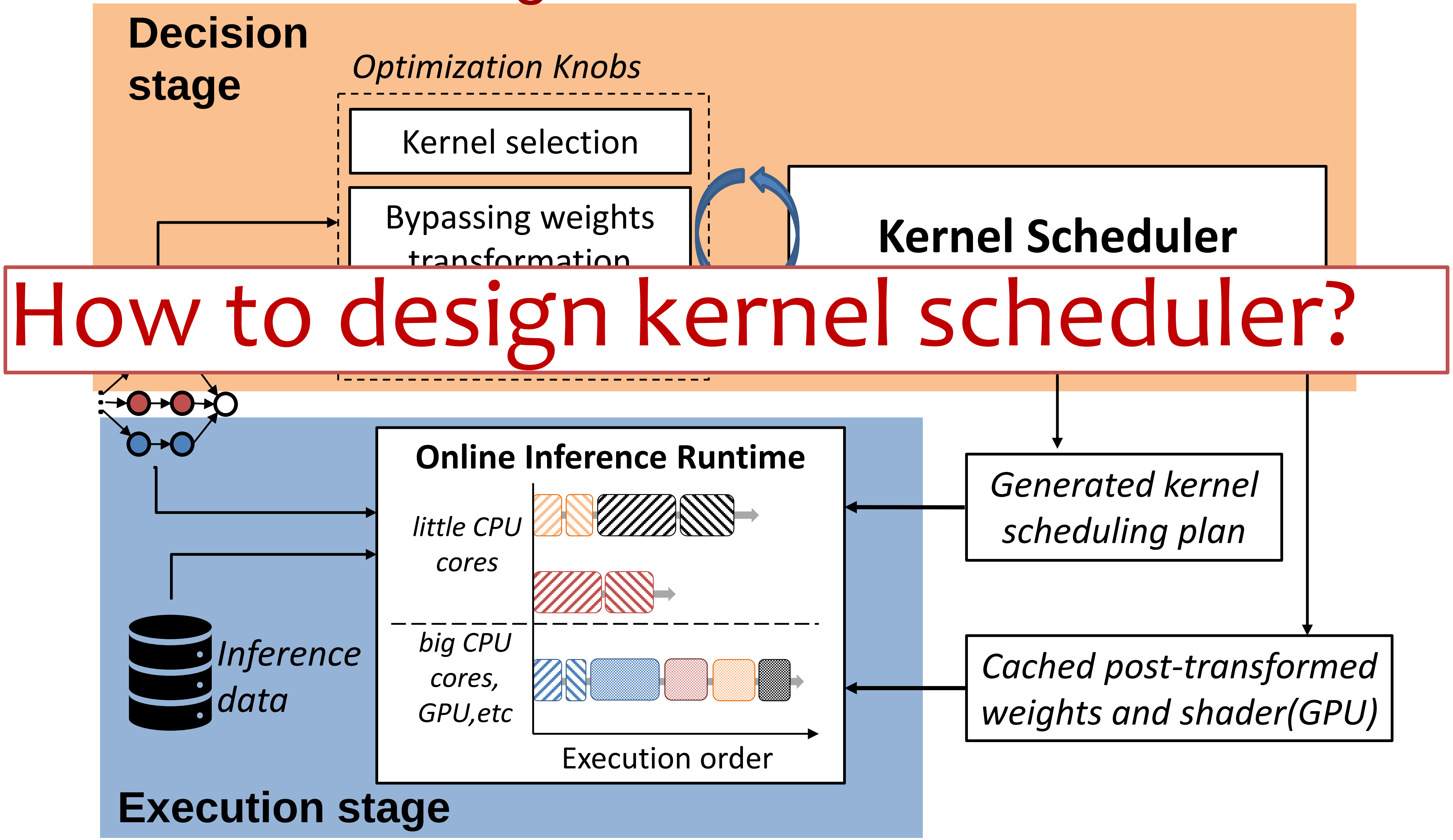
Execution: bounded by computation.

Pipelined inference

NNV12: design overview



NNV12: design overview



Problem Formulation

The need for a kernel scheduler:

I. Which kernel to use for each operator;

II. Whether to load the raw weights or the cached post-transformed weights for each kernel;

III. When and where to execute each operation.

Problem Formulation

Use term *operation* to indicate each stage of operators kernel:

I. weights loading *operation*;
II. weights transformation *operation*;
III. execution *operation*.

e.g. operator 1 can be indicate:

1. Load *operation* 1;
2. Transformation *operation* 1;
3. Execution *operation* 1.

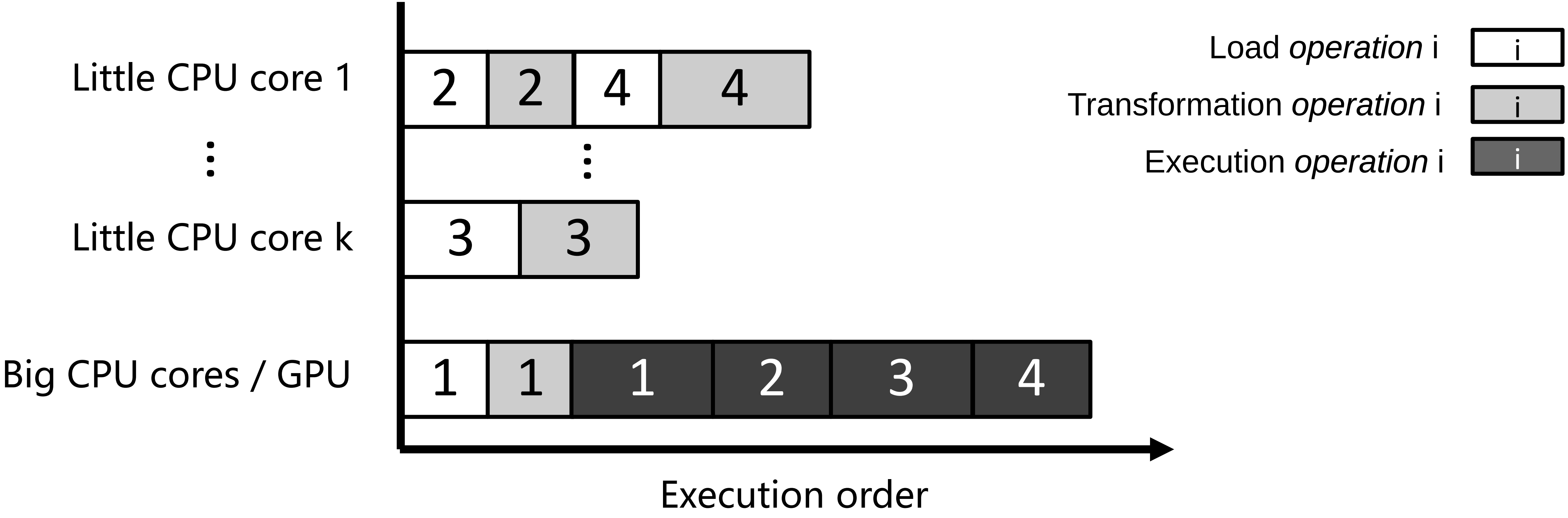
Problem Formulation

Kernel Scheduler Output:

Problem Formulation

Kernel Scheduler Output:

1. Generated kernel scheduling plan



Problem Formulation

Kernel Scheduler Output:

1. Generated kernel scheduling plan
2. Cached post-transformed weights and shader(GPU)

Cached weights

Conv-1: $\begin{bmatrix} 0.42 & 0.16 & 1.42 \\ 0.13 & 3.33 & 0.15 \\ 5.15 & 0.14 & 8.33 \end{bmatrix}$ $\begin{bmatrix} 0.84 \\ 5.41 \\ 9.11 \end{bmatrix}$

•••

Conv-K : $\begin{bmatrix} 1.42 & 1.42 & 0.94 \\ 4.43 & 3.33 & 0.15 \\ 5.15 & 0.14 & 2.18 \end{bmatrix}$ $\begin{bmatrix} 0.24 \\ 7.53 \\ 9.11 \end{bmatrix}$

Cached shaders
(only for GPU)

Conv-1: shader-1

•••

Conv-K : shader-K

Heuristic-Based Kernel Scheduler

Heuristic and Assumptions

Heuristic-Based Kernel Scheduler

Heuristic and Assumptions

1. Each kernel's execution *operation* executed sequentially.

- Load *operation* i: no precursor operations
- Transformation *operation* i's precursor operation:
  Load *operation* i
- Execution *operation* i's precursor operation:
  Transformation *operation* i & Execution *operation* i-1

Heuristic-Based Kernel Scheduler

Heuristic and Assumptions

1. Each kernel's execution *operation* executed sequentially.
2. Each kernel's execution *operation* always occupies all big cores/GPU .

Executing execution *operation* on LITTLE cores could easily bottleneck the whole inference, leaving the big cores under-utilized.

Heuristic-Based Kernel Scheduler

Heuristic and Assumptions

1. Each kernel's execution *operation* executed sequentially.
2. Each kernel's execution *operation* always occupies all big cores/GPU .
3. Load and transformation *operations* of the same kernel are always bundled together.

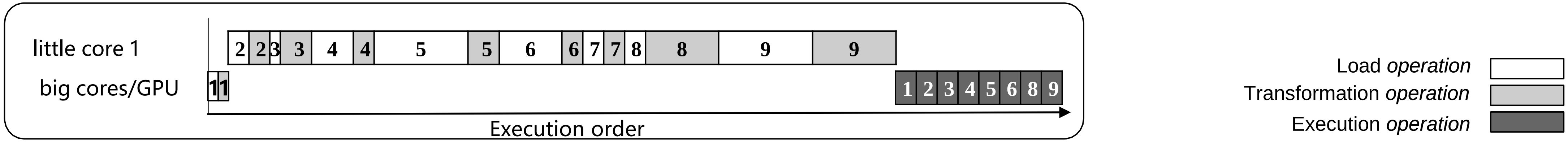
Transformation *operation* i's precursor operation: Load *operation* i

Heuristic-Based Kernel Scheduler

Algorithm of kernel scheduling

Heuristic-Based Kernel Scheduler

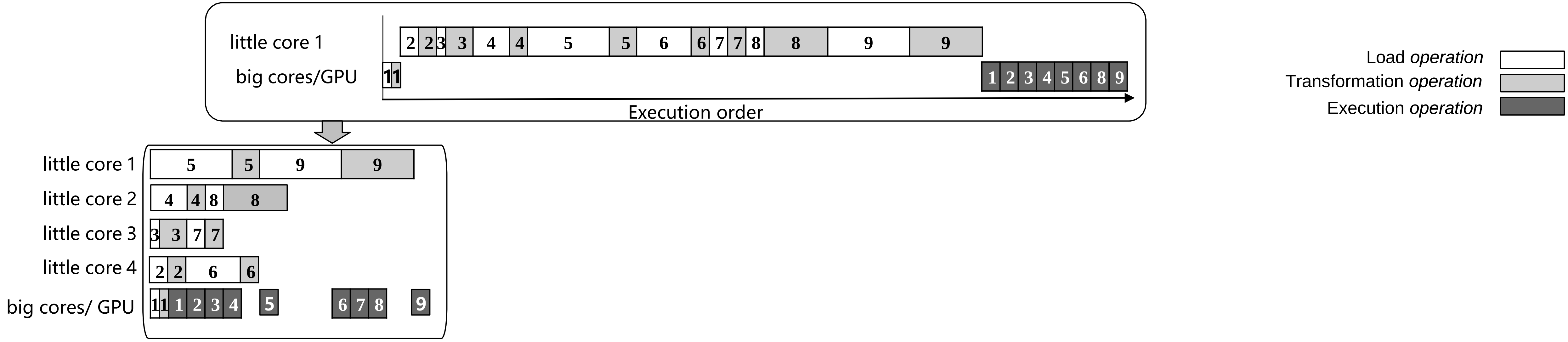
Algorithm of kernel scheduling



Measure the latency of operation

Heuristic-Based Kernel Scheduler

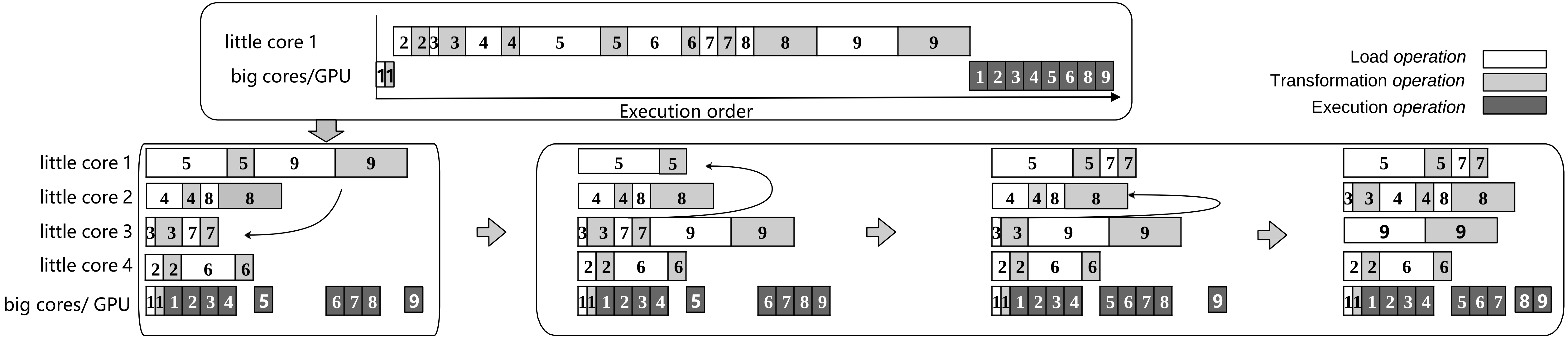
Algorithm of kernel scheduling



Init the kernel scheduling plan

Heuristic-Based Kernel Scheduler

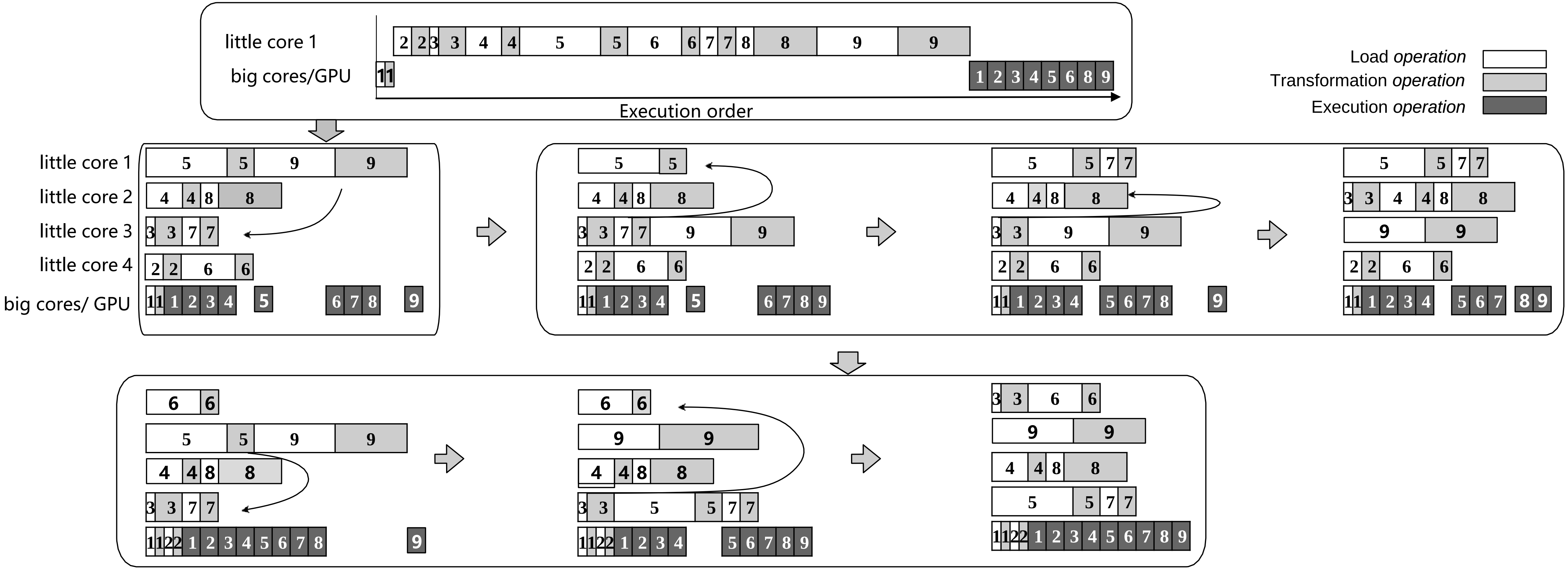
Algorithm of kernel scheduling



Adjusting the scheduling plan on the little CPU cores through loops

Heuristic-Based Kernel Scheduler

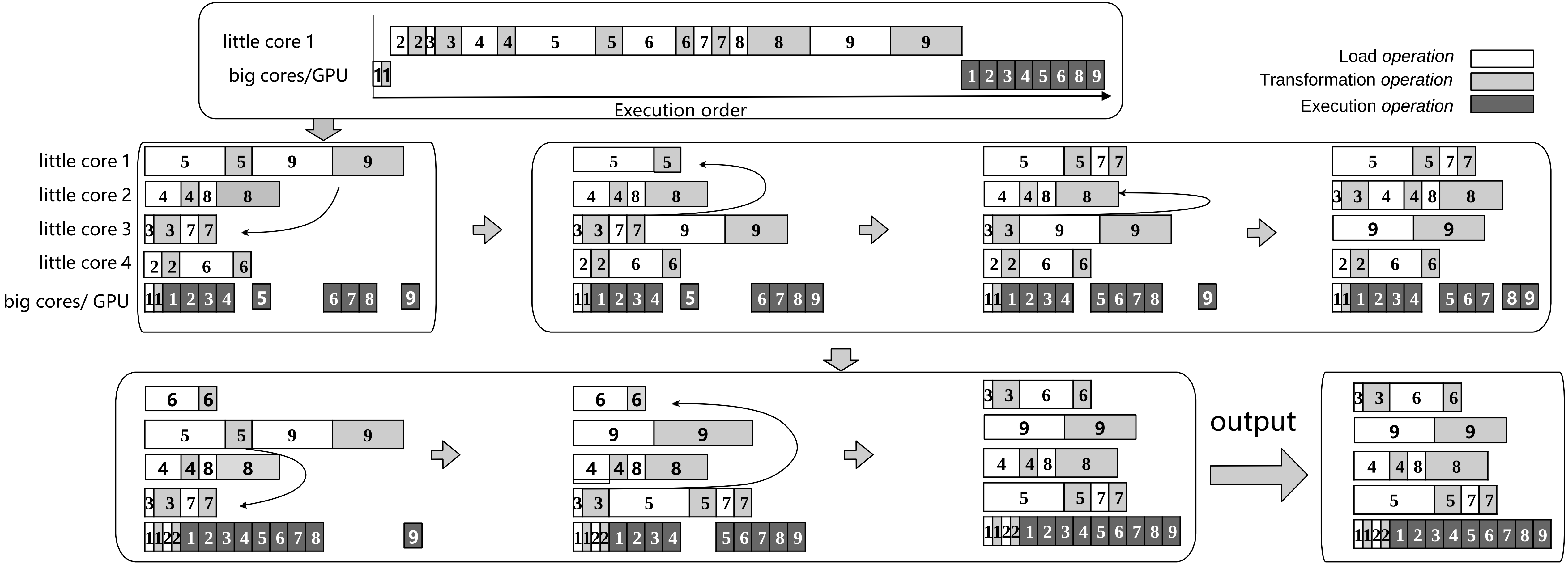
Algorithm of kernel scheduling



Adjust the scheduling plan through loops so that the total latency on the big CPU cores/GPU and small CPU cores is similar

Heuristic-Based Kernel Scheduler

Algorithm of kernel scheduling



Output obtained: kernel scheduling plan

Evaluation: setting

Model	Task	Parameters
AlexNet	classification	61.3M
GoogLeNet		7.1M
MobileNet		4.4M
MobileNetV2		3.7M
ResNet18		12.7M
ShuffleNet		3.6M
EfficientNetB0		5.4M
ResNet50		25.7M
SqueezeNet		1.4M
ShuffleNetV2		3.4M
MobileNetv2-YOLOv3	Object Detection	3.6M
MobileNet-YOLO		11.9M
CRNN-lite	OCR	2.4M

Evaluation: setting

Device	SoC	little cores
Meizu 16T	Snapdragon 855	4
Google Pixel 5	Snapdragon 765G	2
Redmi 9	MTK Helio G80	4
Meizu 18 Pro	Snapdragon 888	4

Device	GPU memory	little cores
Jetson TX2	8G	2
Jetson Nano	4G	2

Evaluation: setting

Device	SoC	little cores
Meizu 16T	Snapdragon 855	4
Google Pixel 5	Snapdragon 765G	2
Redmi 9	MTK Helio G80	4
Meizu 18 Pro	Snapdragon 888	4

Device	GPU memory	little cores
Jetson TX2	8G	2
Jetson Nano	4G	2

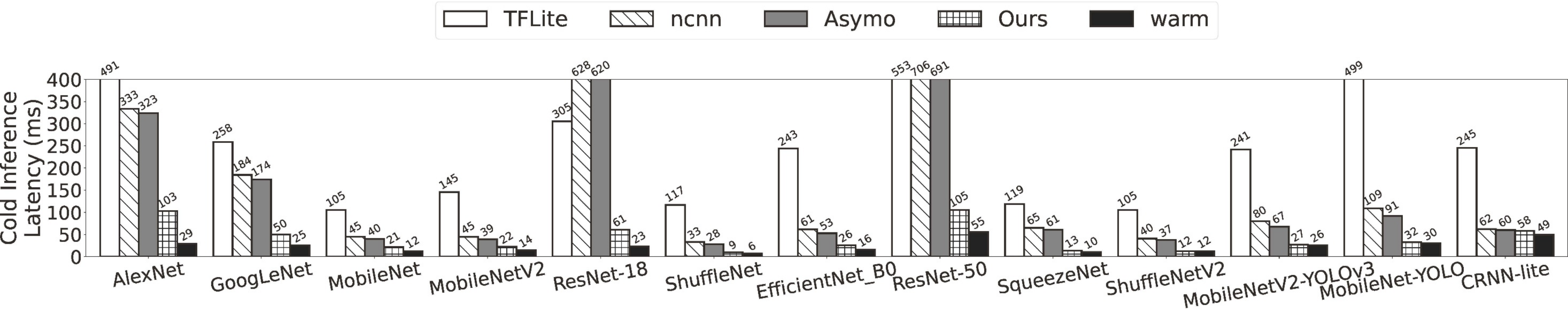
Baselines:

- ncnn
- Tensorflow/TFLite
- AsyMo[1]

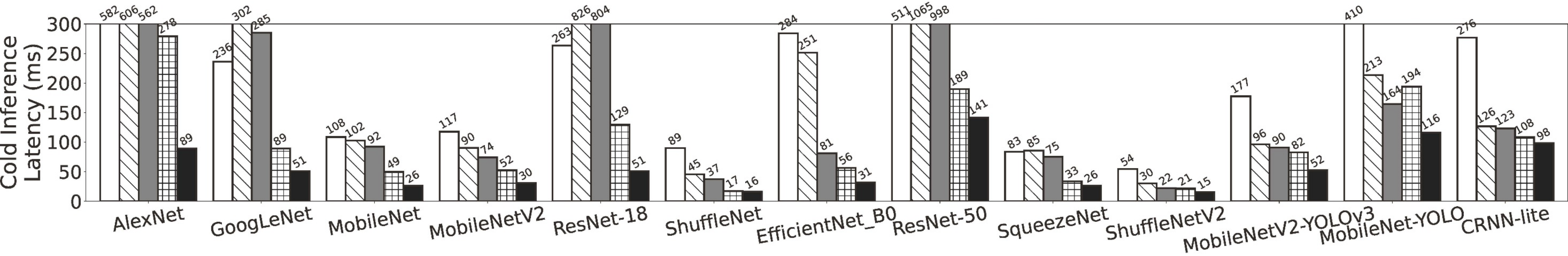
1. Wang et al. Asymo: scalable and efficient deep-learning inference on asymmetric mobile cpus. MobiCom' 21

Evaluation: end-to-end performance

The cold inference latency of NNV12 and baselines on edge CPUs



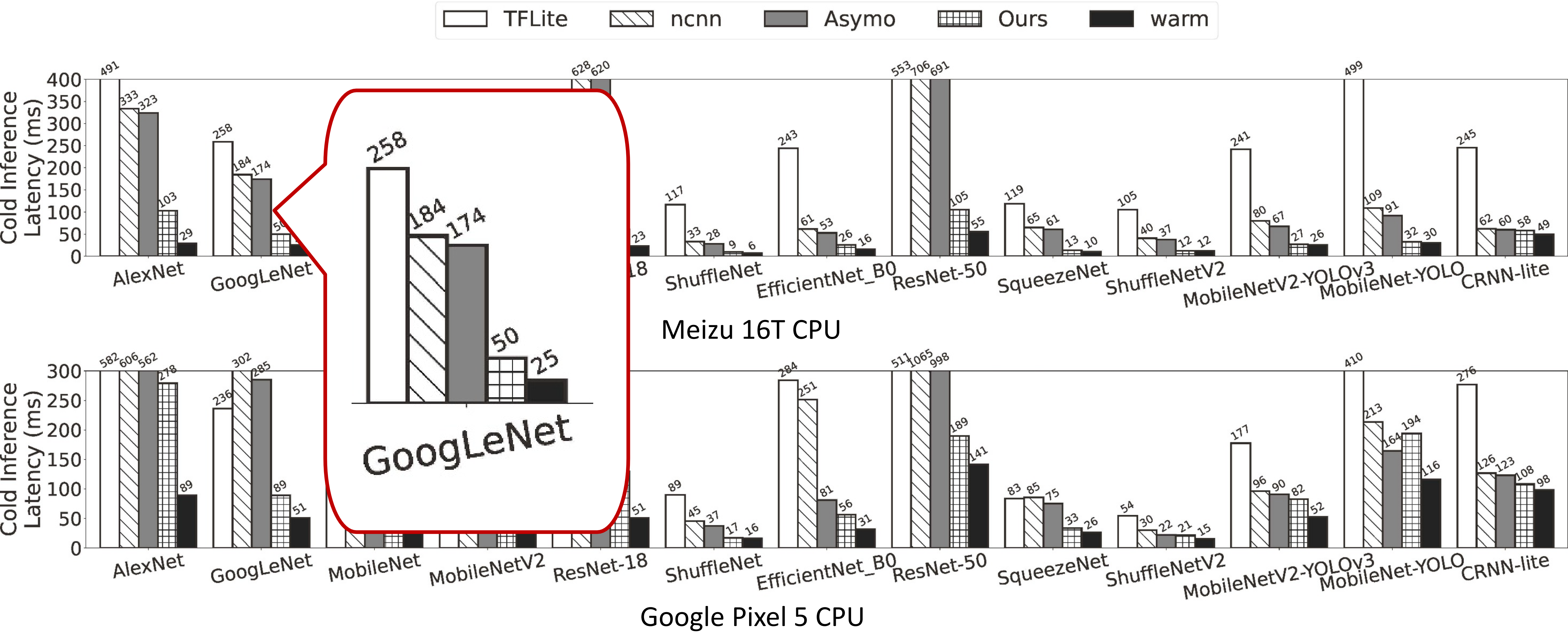
Meizu 16T CPU



Google Pixel 5 CPU

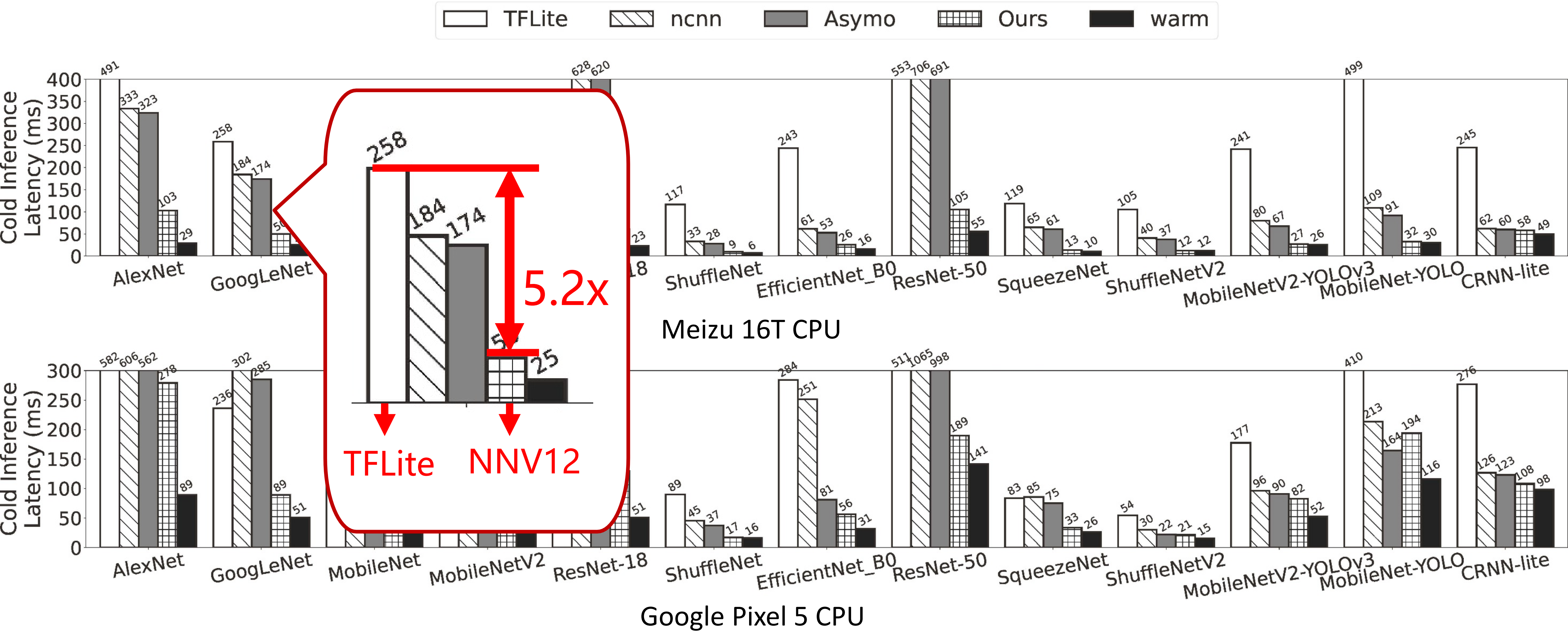
Evaluation: end-to-end performance

The cold inference latency on edge CPUs



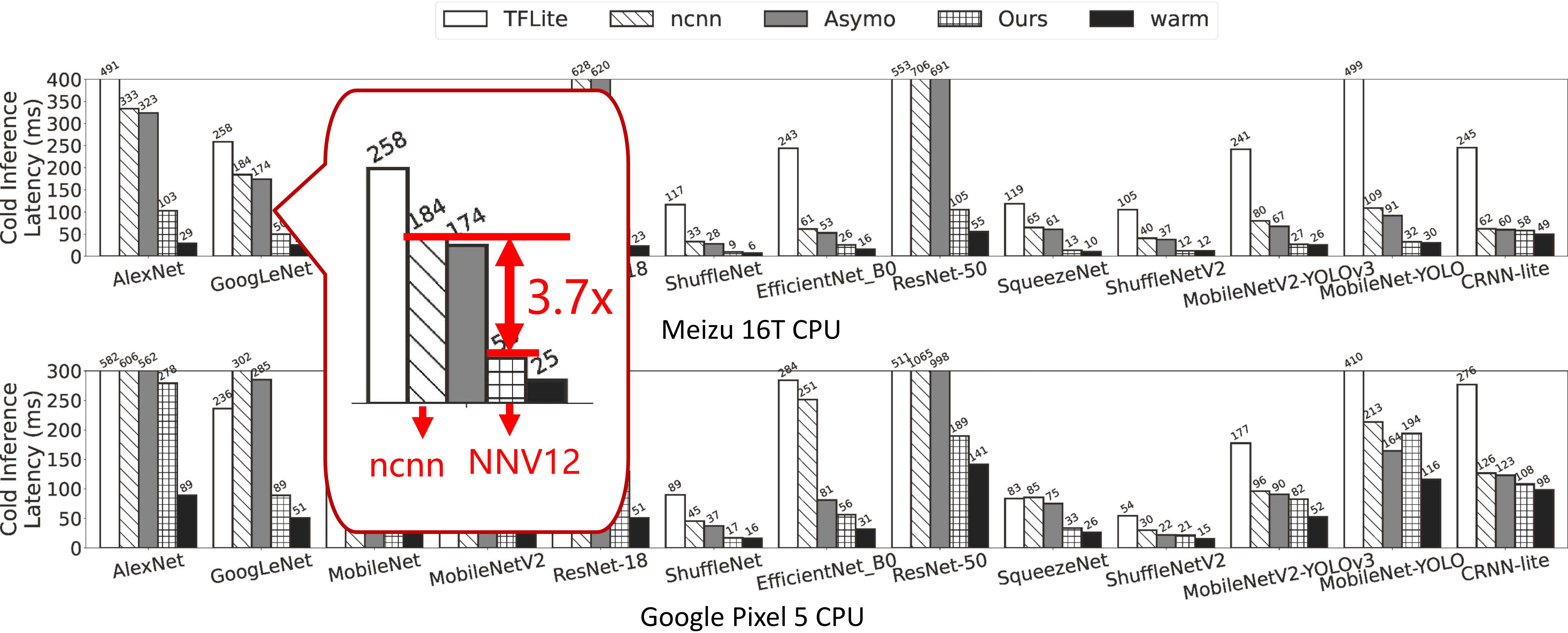
Evaluation: end-to-end performance

The cold inference latency on edge CPUs



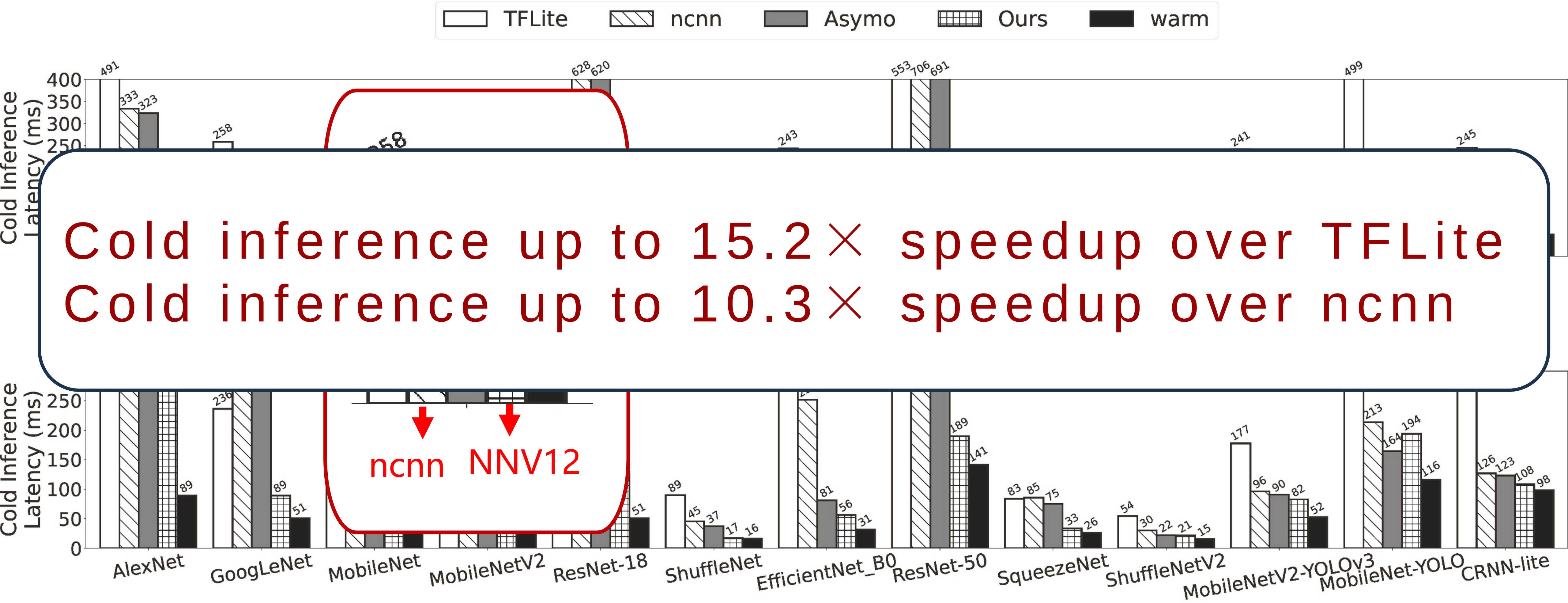
Evaluation: end-to-end performance

The cold inference latency on edge CPUs



Evaluation: end-to-end performance

The cold inference latency on edge CPUs



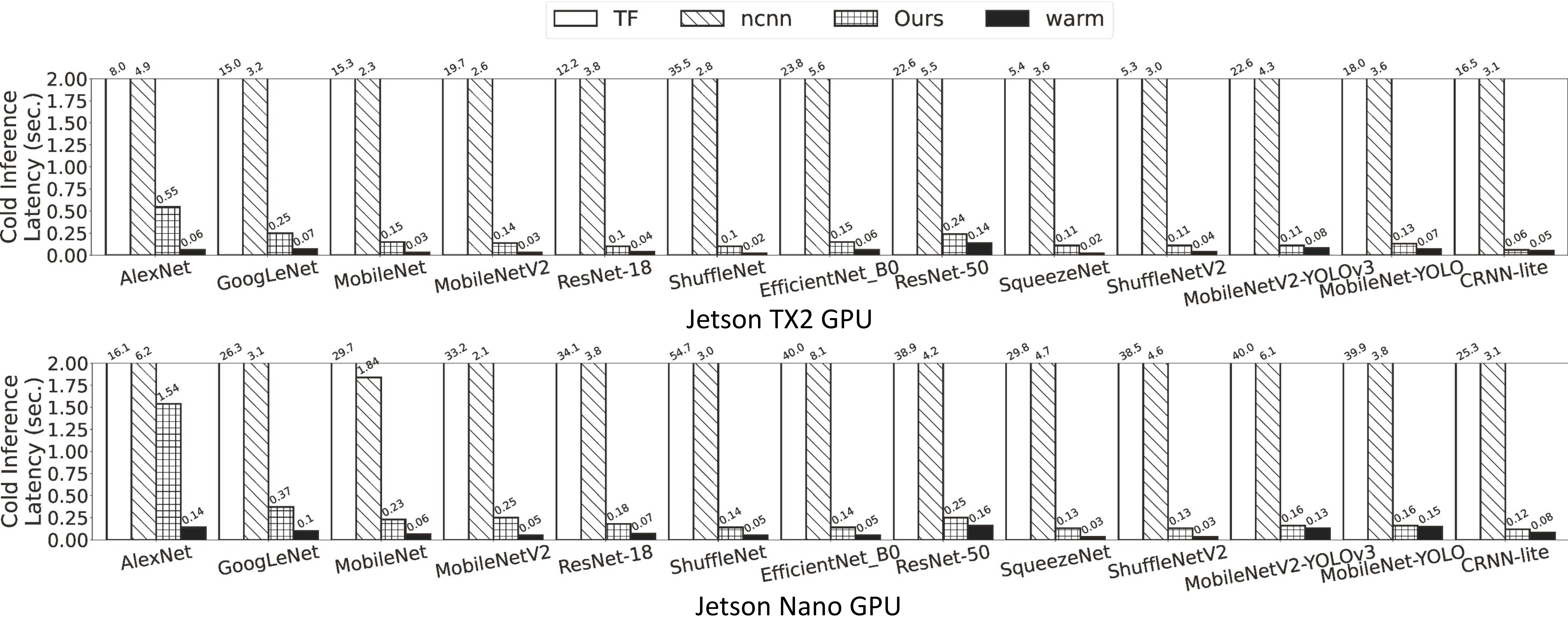
Google Pixel 5 CPU

Cold inference up to 15.2× speedup over TFLite

Cold inference up to 10.3× speedup over ncnn

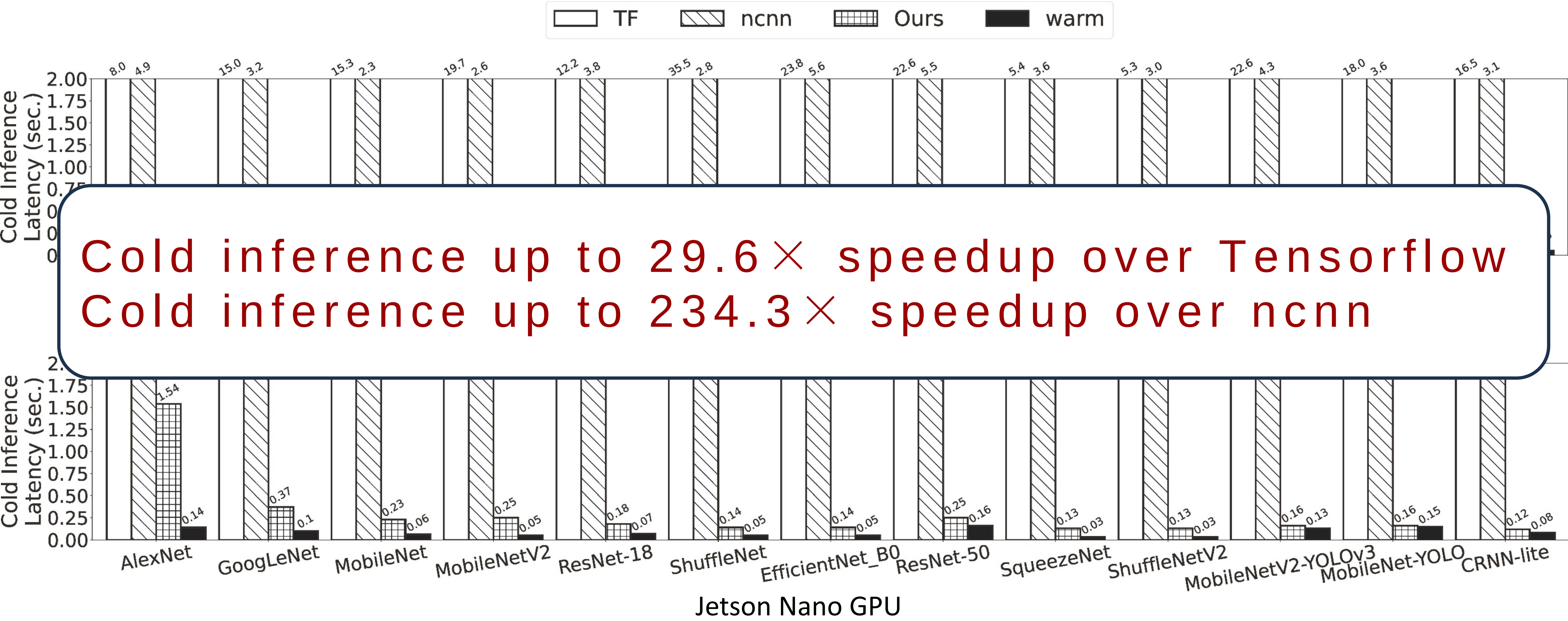
Evaluation: end-to-end performance

The cold inference latency on edge GPUs



Evaluation: end-to-end performance

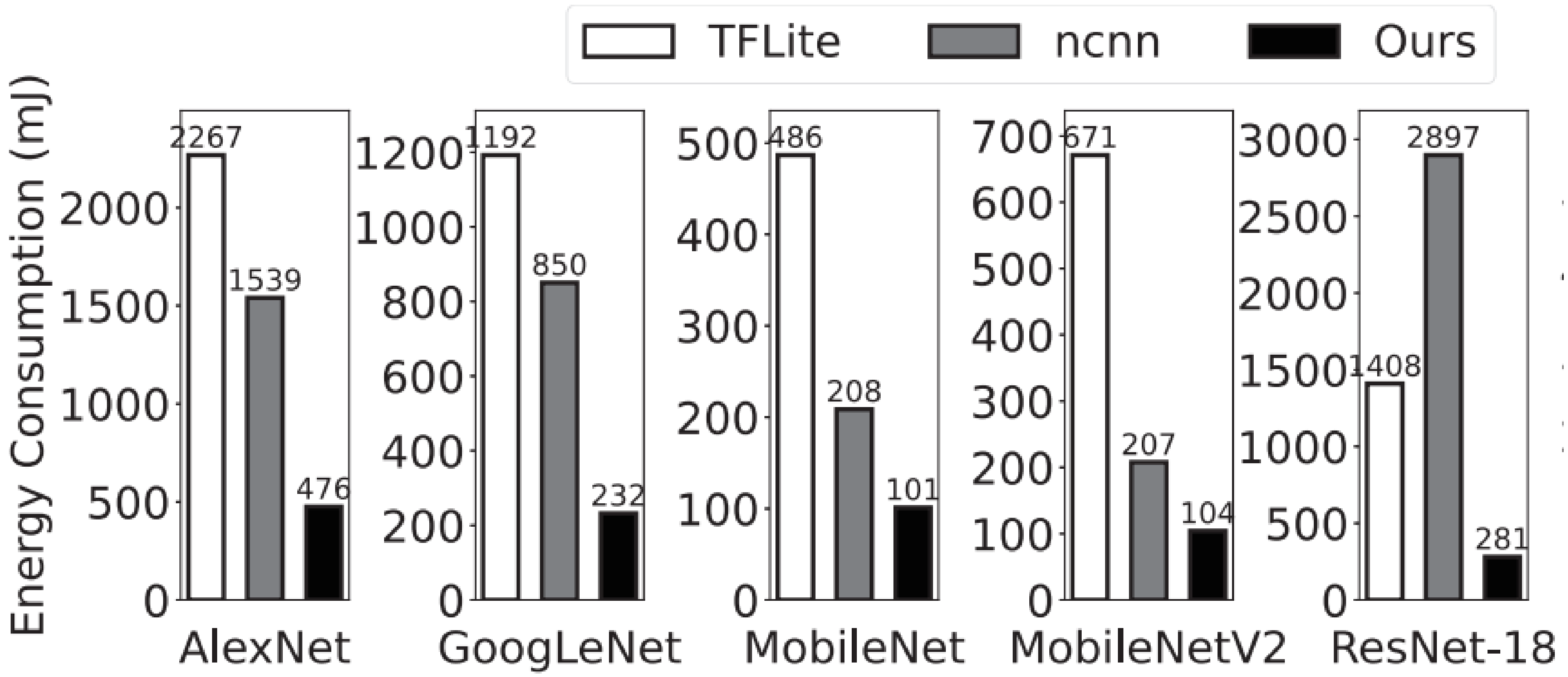
The cold inference latency on edge GPUs



Cold inference up to 29.6× speedup over Tensorflow
Cold inference up to 234.3× speedup over ncnn

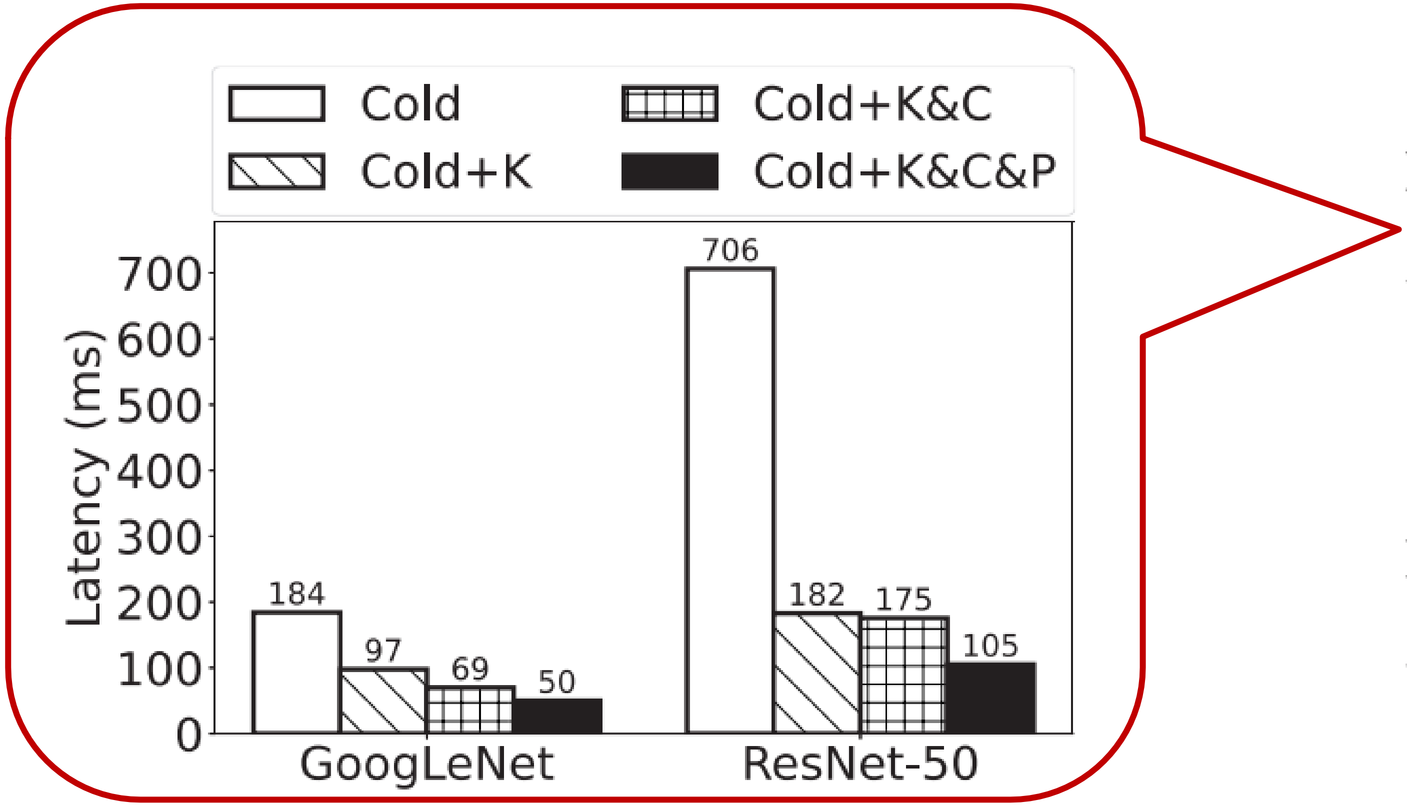
Evaluation: end-to-end performance

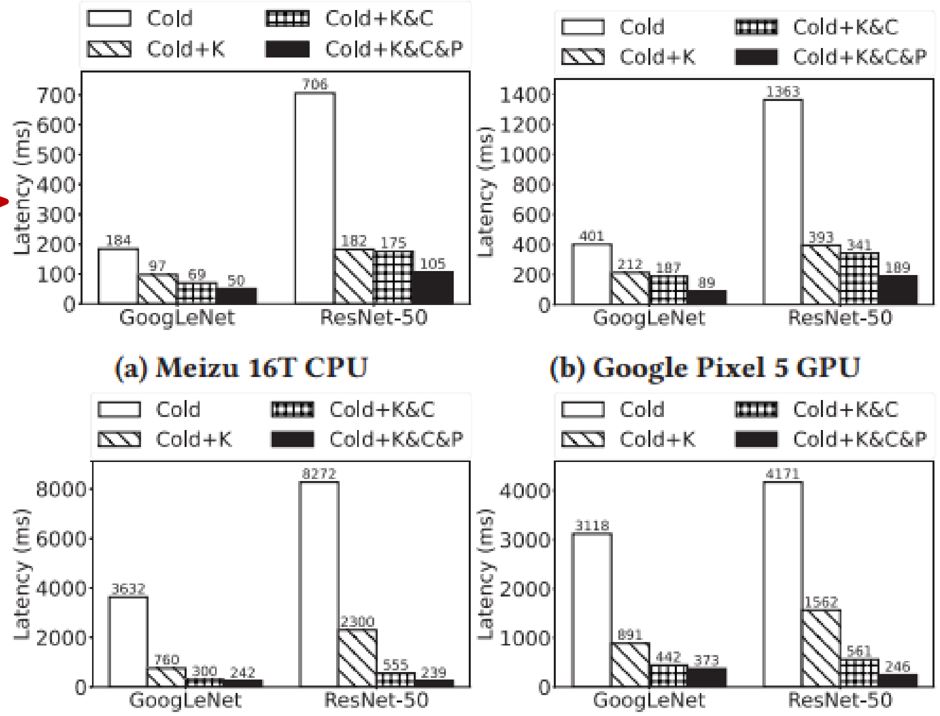
Reduce energy consumption by 1.6×-5.0×



Evaluation: ablation

- K: kernel selection;
- C: caching the post-transformed weights (and shaders);
- P: kernel execution pipeline





(a) Meizu 16T CPU
(b) Google Pixel 5 GPU
(c) Jetson TX2 GPU
(d) Jetson Nano GPU

Conclusion

- ✓ NNV12: A DNN inference framework boosting DNN cold inference on devices
- ✓ Implement a prototype and evaluate its efficiency
- ✓ Open source: https://github.com/UbiquitousLearning/NNV12

Thanks!

Boosting DNN Cold Inference on Devices

Rongjie Yi[1], Ting Cao[2], Ao Zhou[1], Xiao Ma[1], Shangguang Wang[1], Mengwei Xu[1]

[1] State Key Laboratory of Networking and Switching Technology
[2]Microsoft Research

Code:https://github.com/UbiquitousLearning/NNV12

yirongjie@bupt.edu.cn