OSDI 26

Unleash All Cores: Scalable Asymmetry-aware DNN Inference on Mobile CPUs

Qianlong Sang · Puyi He · Huanghuang Liang · Yili Gong · Chuang Hu · Xiaobo Zhou · Dazhao Cheng

Wuhan University
University of Macau



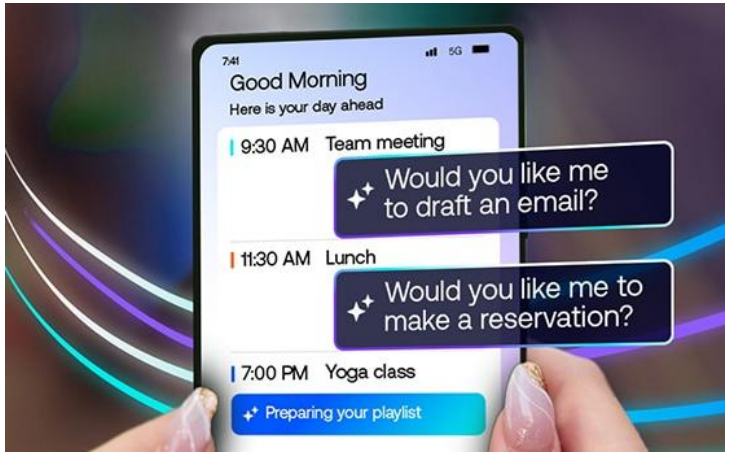


On-device AI powers interactive experiences

OSDI '26



Smart photography



Personal AI assistant



Real-time translation

Underlying Models:

- Face detection: MobileNet
- Region editing: BiSeNet
- Image enhancement: U-Net
- Aesthetic improve: StyleGAN

.....

- Wake detection: DS-CNN
- Text generation: Conformer
- Speech recognition: RNN-T
- Speech generation: VITS

.....

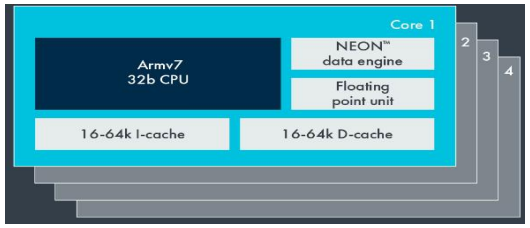
- Translation: mBART
- Fluency improve: Small LLM translation heads

.....

Mobile CPUs remain the universal backend

OSDI '26

Homogeneous CPU

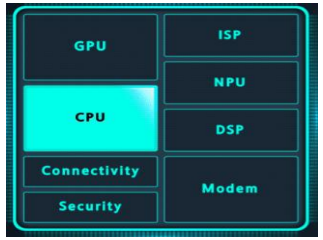


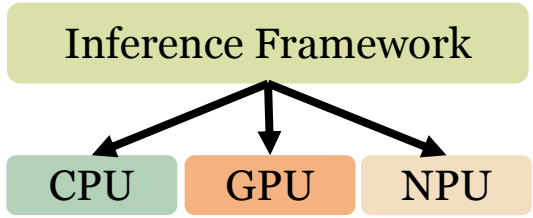
Heterogeneous CPU

- powerful cluster (big)
- efficient cluster (little)



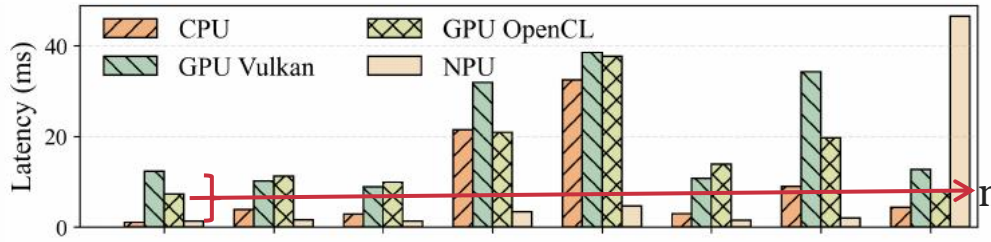
Heterogeneous SoC





We got CPU, GPU and NPU on mobile devices, so why CPU?

CPUs provide the broad and efficient execution path for DNN models.

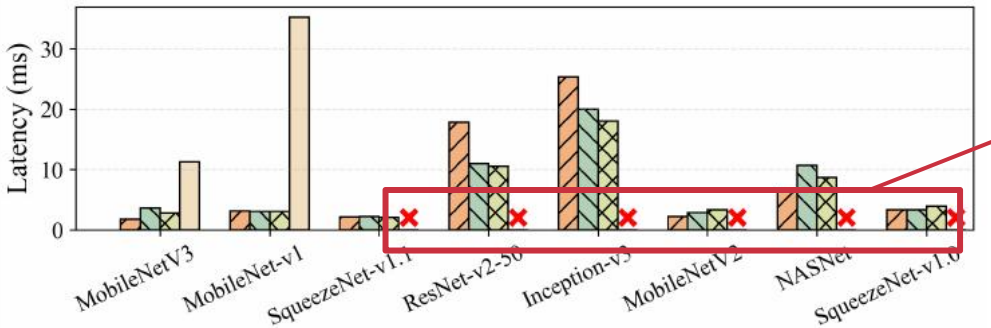


Google Pixel 9 (2024)

mobile CPU up to **11×** faster than mobile GPU

Mobilenet V3

GPUs can lose on lightweight models due to launch and transfer overhead.



Oneplus 12 (2024)

NPUs have vendor-specific operator gaps and fallbacks.

Mobile CPUs are shared and interference-prone

OSDI '26

Competing workloads

Inference does not own the CPU.

- **Foreground UI / rendering** — frame-critical work
- **Camera and media playback** — continuous compute
- **Sync, download, app services** — bursty background work
- **OS daemons** — unpredictable system work

Shared AMP CPU

big cores: B0 DNN, B1 BG

LITTLE cores: L0 BG, L1 DNN

Background work can occupy either cluster

Shared last-level cache

Shared memory bandwidth

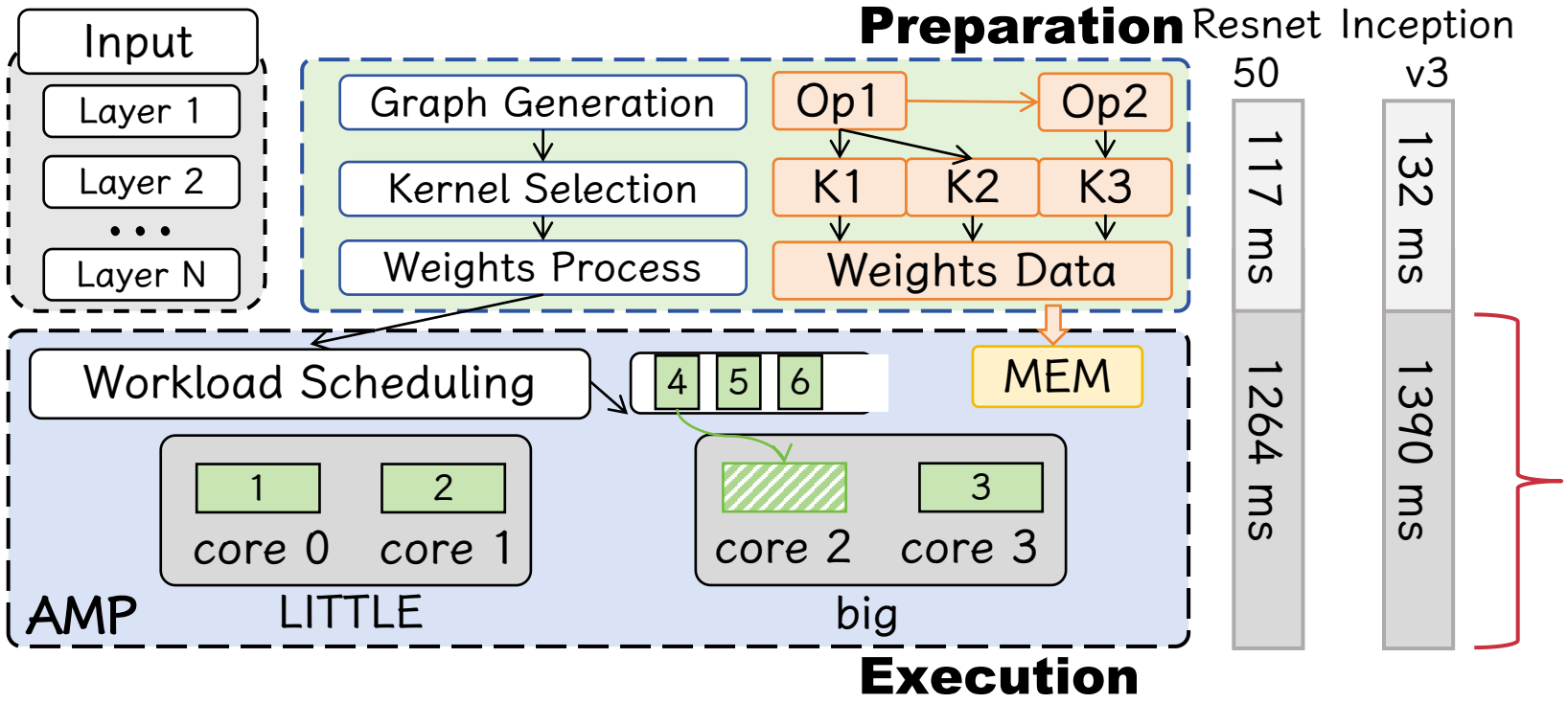
What an inference thread sees

Its effective speed changes over time.

- **Preemption** — the thread stops running
- **Cache pressure** — working data is displaced
- **Memory contention** — each access takes longer

Same core ≠ constant speed

DNN inference on mobile devices



Inference Pipeline on Mobile Devices

That's our focus

Model

The complete inference graph and its tensor dependencies.

MobileNetV2

Operator

One mathematical function in that graph, such as Conv2D, GEMM, or ReLU.

Conv2D

Kernel

A hardware-tuned implementation of an operator (not the OS kernel).

GEMM / Winograd

Workload

The schedulable unit: contiguous output-tensor region plus a kernel pointer.

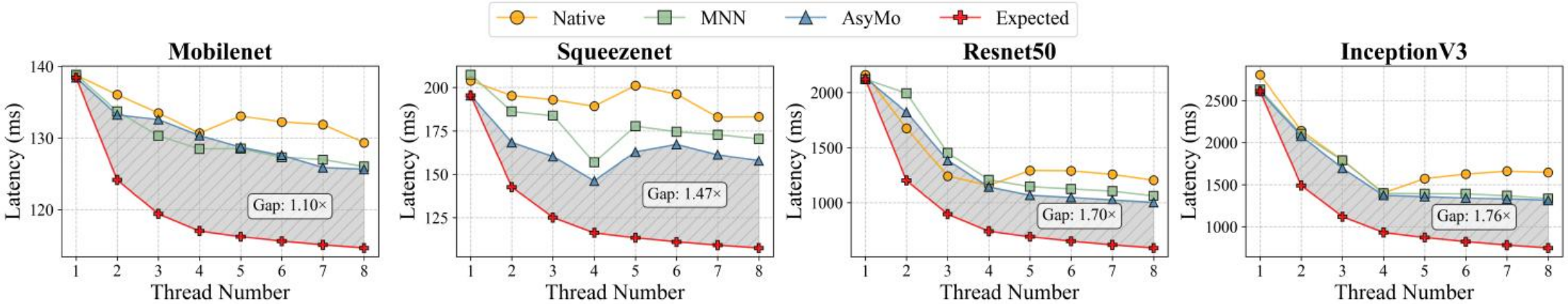
$\text{output}[h_0 : h_1, w_0 : w_1]$

LITTLE cores can make inference 37% slower

OSDI '26

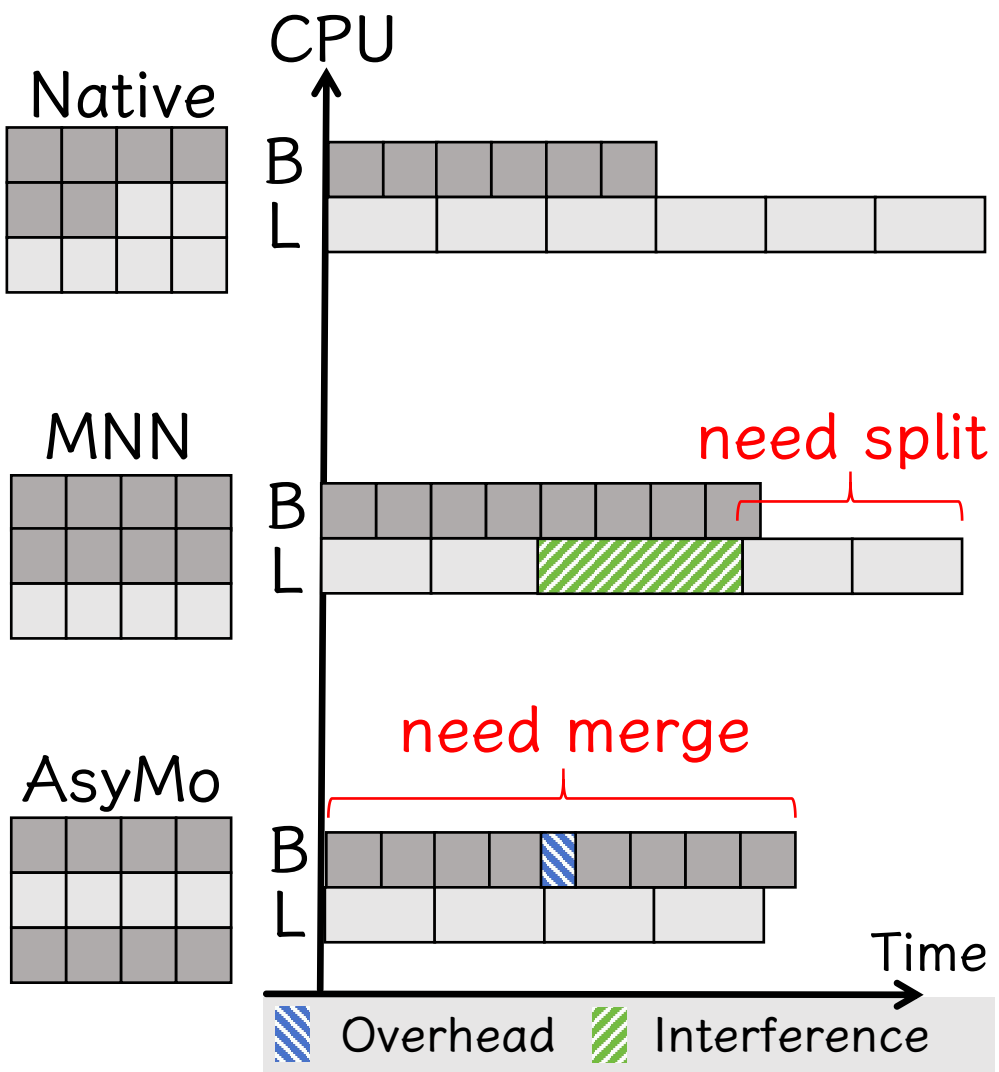
Related work:

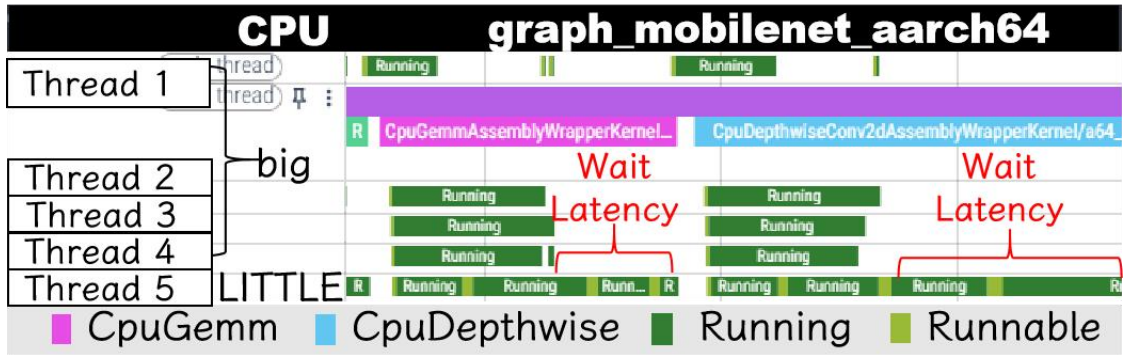
Name	Author	Design
Native	Arm Compute Library, Arm	Uses fixed-size, evenly distributed workloads.
MNN	Alibaba	Statically partitions workloads according to a predefined big-to-LITTLE compute-capacity ratio.
AsyMo	Manni Wang et al., MobiCom 2021	Uses a cost model to partition operators into fine-grained tasks and dynamically dispatch them across asymmetric cores.
Expected	Theoretical projection, constructed in this paper	Uses Amdahl's Law, the profiled parallelizable workload fraction, and big/LITTLE capacity ratios to estimate ideal latency scaling .

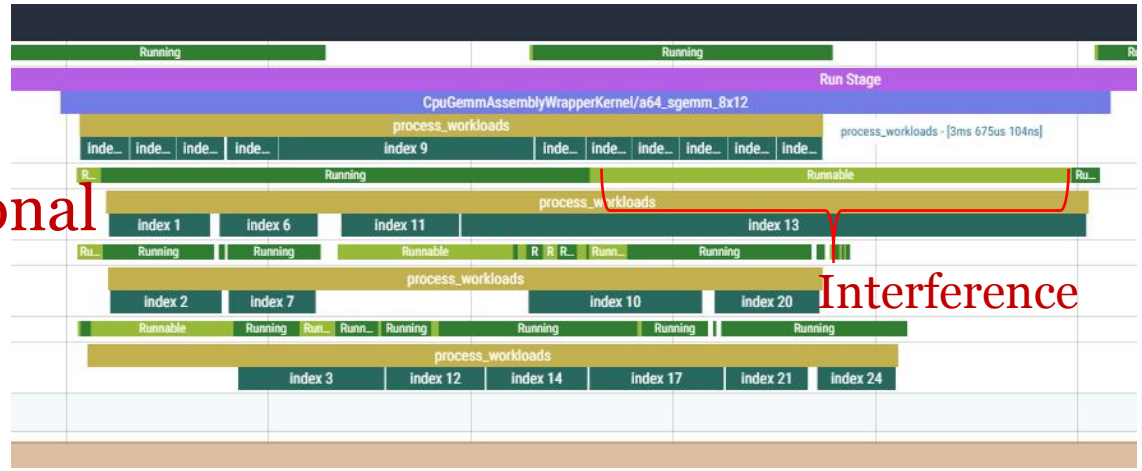


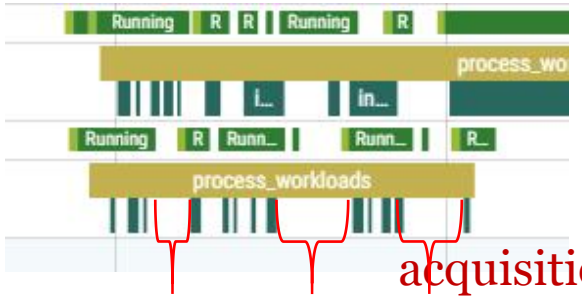
More compute capacity creates a longer critical path.

OB1: Static workload size is inefficient under interference









OB2: Kernel-core affinity

DNN affinity adds a kernel choice

One operator can have multiple optimized implementations.

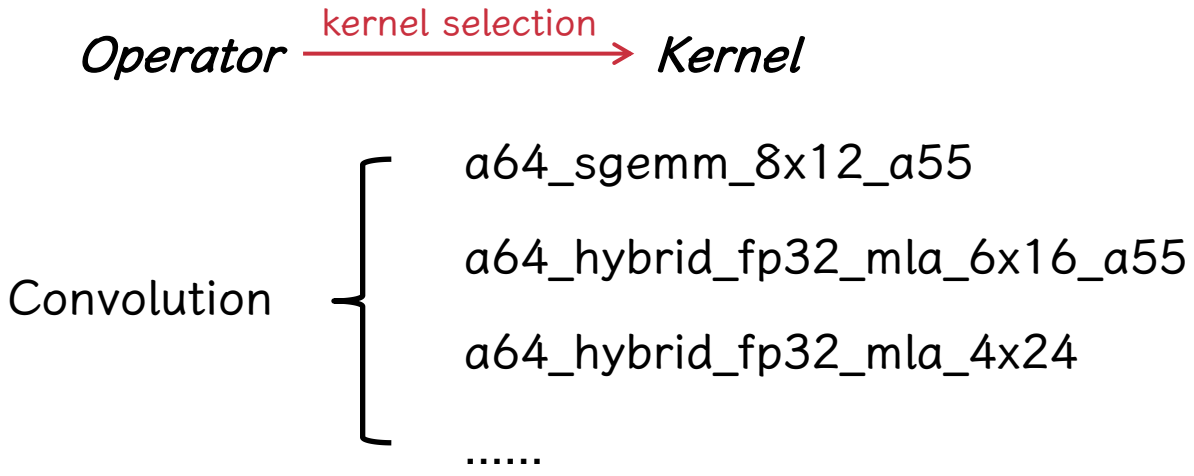


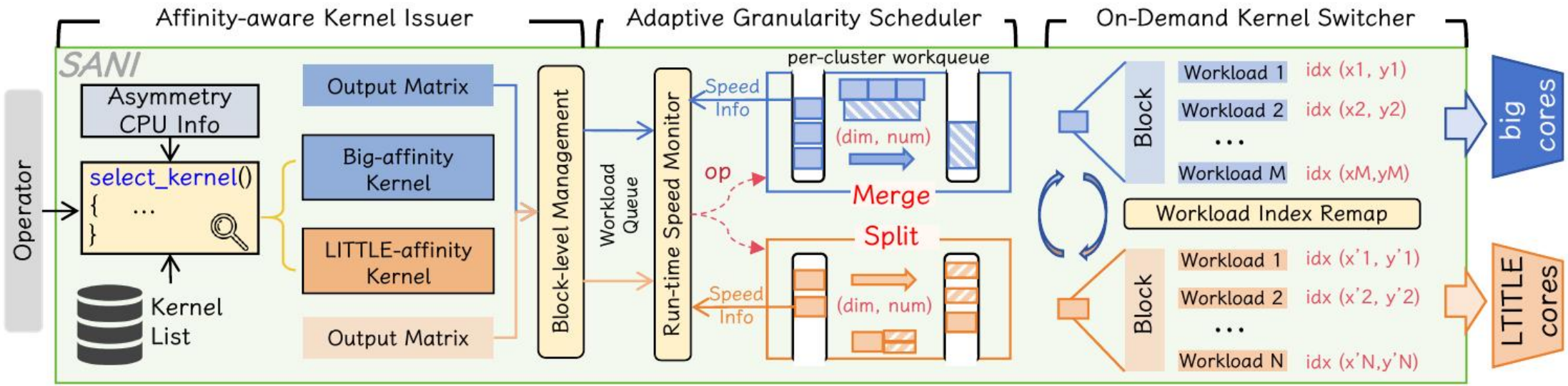
Table 1: Kernel execution time (ms) across core types. h1: *hybrid*_6 × 16, sg: *sgemm*_8 × 12, d1: *depthfirst*_2 × 2, d2: *depthfirst*_3 × 3. B/W: Best/Worst ratio.

Operator	Convolution			DepthWiseConv		
Kernel / Core	**h1**	**sg**	**B/W**	**d1**	**d2**	**B/W**
big	**2958**	3226	1.09×	92	**69**	1.34×
LITTLE	4509	**4371**	1.03×	**291**	345	1.19×

The best kernel changes by cluster; a mismatch costs over 34% for DepthWiseConv.

Our solution: SANI

OSDI '26



01 Issue affine kernels

Prepare a kernel for each cluster.

02 Adapt task granularity

Merge for fast threads; split for slow ones.

03 Switch on migration

Remap indices and preserve affinity.

Block, kernel and workload

OSDI '26

MECHANISM 1

1 Operator and kernel tiles

Operator: mathematical computation over an output tensor

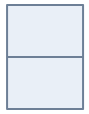
Output tensor (4 × 4 example)

big-affine kernel



native tile: 1 × 2

LITTLE-affine kernel



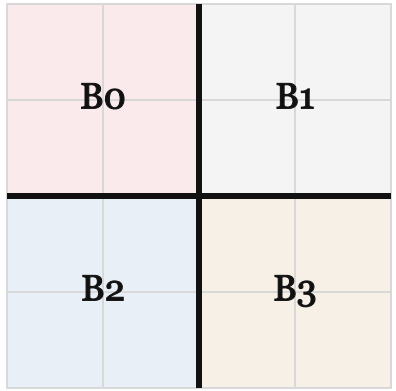
native tile: 2 × 1

2 Our "block" unification for kernels

Block: kernel-agnostic partition of the output tensor for efficient management

LCM of native tile dimensions

LCM(1 × 2, 2 × 1) = 2 × 2 unified block



Same region IDs for both kernels

A block is logical; it does not carry a kernel pointer.

3 Workloads are executable tasks

Workload = contiguous region + kernel pointer

W = { start[ ], end[ ], split_dim, kernel* }

Workload size

Tensor sub-region covered by one W; measured by its extents across dimensions, not bytes.

MERGE



2 adjacent workloads → 1 larger workload

SPLIT



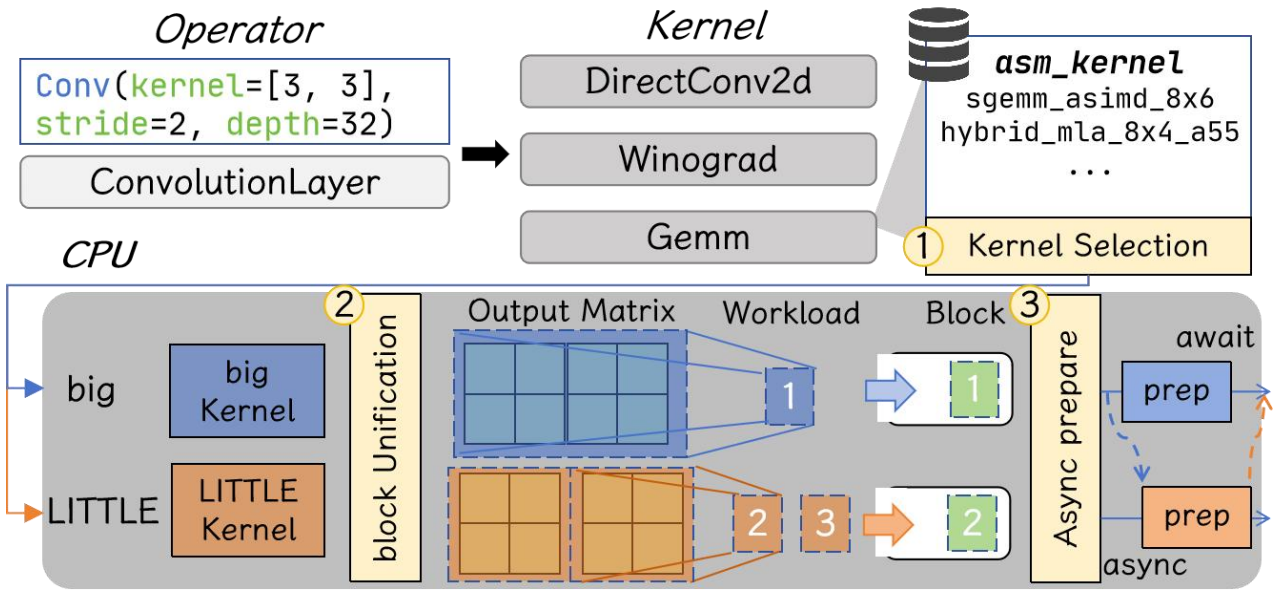
1 large workload → smaller workloads

Size changes, kernel stays the same

Each cluster gets its most efficient kernel

OSDI '26

MECHANISM 1



A unified block is the common scheduling unit above different native kernel tiles.

01 **Select**

Score compute, memory, and instruction compatibility.

02 **Unification**

Use the LCM of native tile sizes to define blocks.

03 **Prepare**

Build both kernels asynchronously at model load.

Fast threads do more; slow threads do less

OSDI '26

MECHANISM 2

> **With great power comes great responsibility.**
>
> — *Spider-Man*

Translated into SANI:

FAST THREADS	→	larger workloads size (reduce acquisitionoverhead)
SLOW THREADS	→	smaller workloads size (avoid becoming bottleneck)

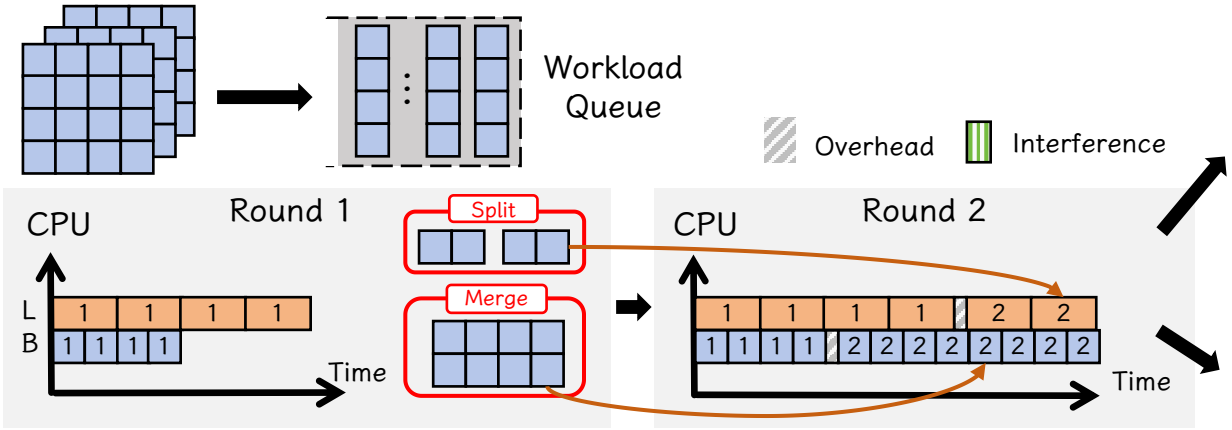
More power → more blocks.

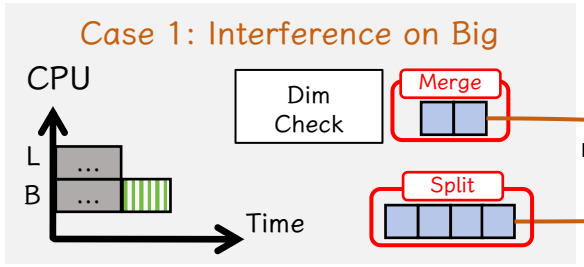
Fast threads do more; slow threads do less

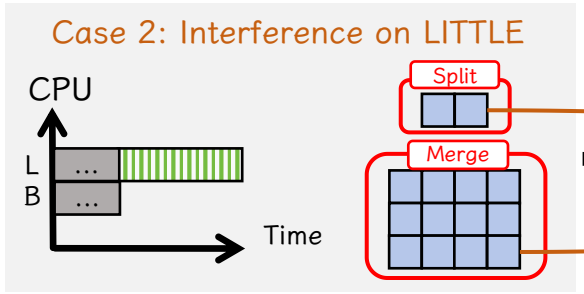


OSDI '26

MECHANISM 2







01

Measure

Completion time

02

Rank

Fastest ↔ slowest

03

Resize

Merge ↔ split

04

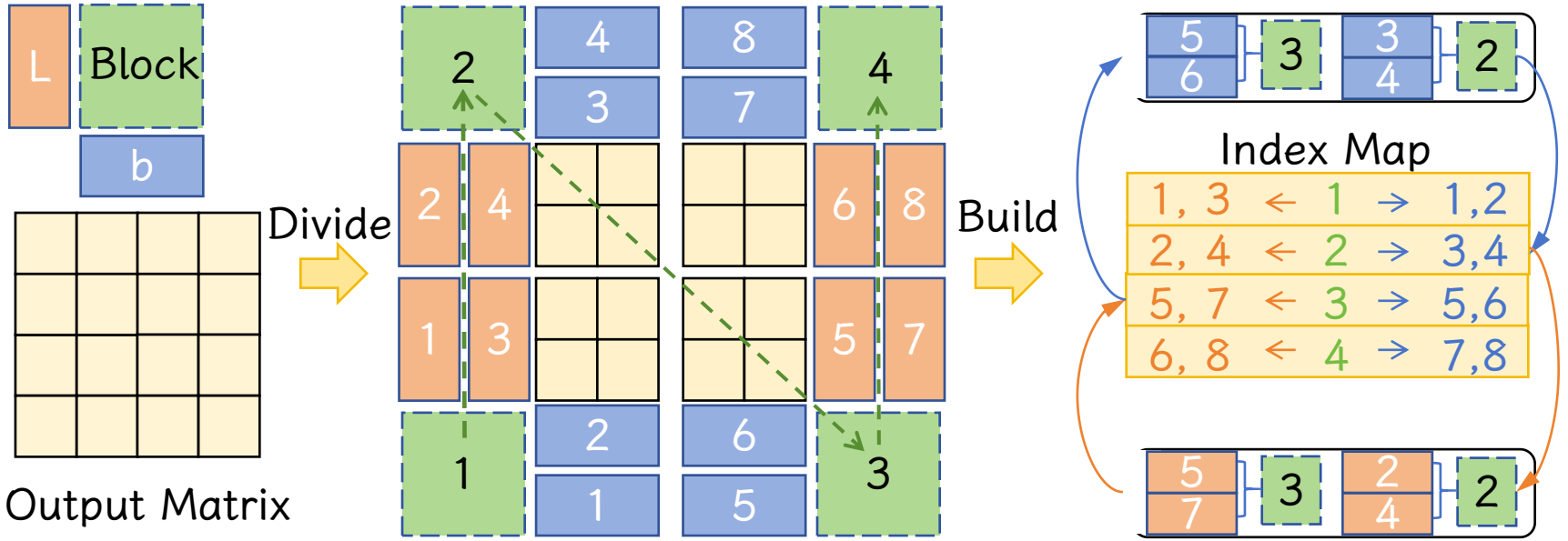
Dispatch

Next round

Feedback reflects both core asymmetry and transient OS interference.

Migration remap

MECHANISM 3



The same output region maps to different workload indices under different kernel layouts.

01 Steal a block

A cluster queue drains and work migrates.

02 Remap indices

A prebuilt table finds the destination workloads.

03 Swap kernel

The payload now points to the target-affine kernel.

Evaluation spans five SoCs and six models

OSDI '26

EVALUATION

5 PLATFORMS

Pixel 9 · Tensor G4
Find X3 Pro · SM8350
Redmi K60 · SM8475
OnePlus Ace 5 Ultra· Dimensity 9400
Odroid XU4 · Exynos 5422

6 MODELS

MobileNetV2
SqueezeNet
ShuffleNet
ResNet-50
Inception-V3
GPT-2

3 BASELINES

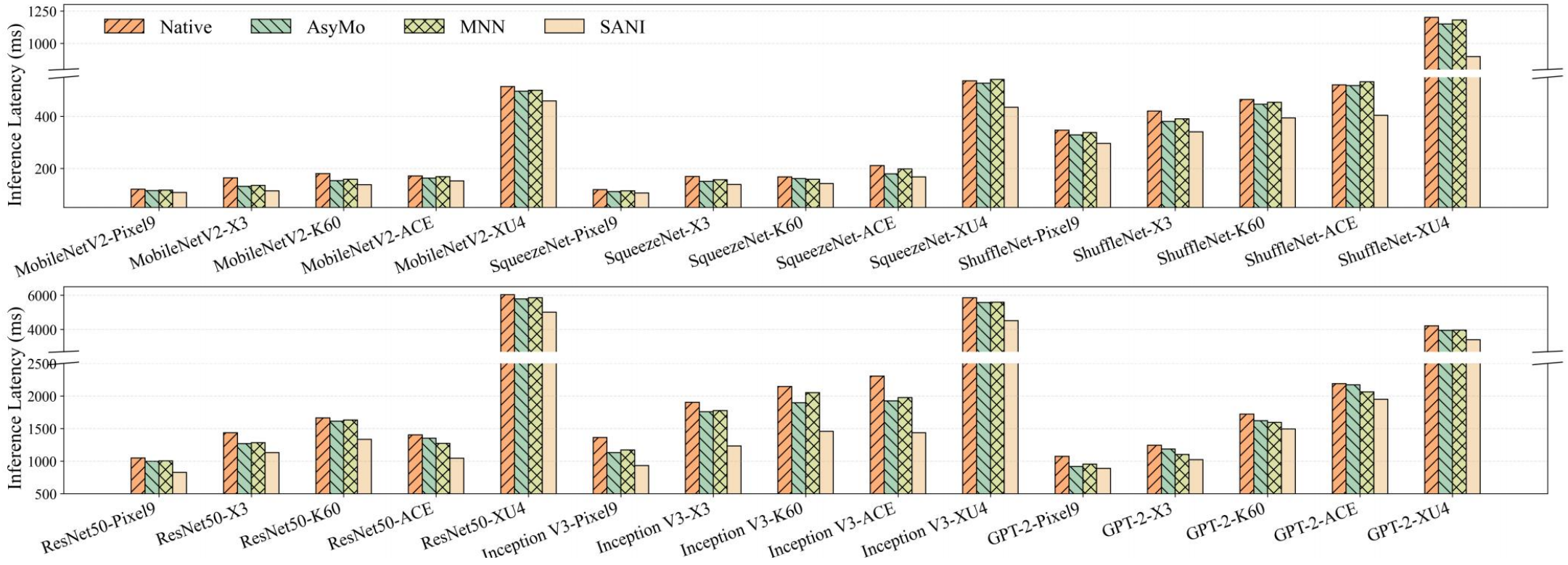
Native · Arm-CL
AsyMo · dynamic fine-grain
MNN · proportional static

Metrics: end-to-end latency · energy · scaling efficiency

SANI cuts average latency 17.6–23.7%

OSDI '26

RESULTS



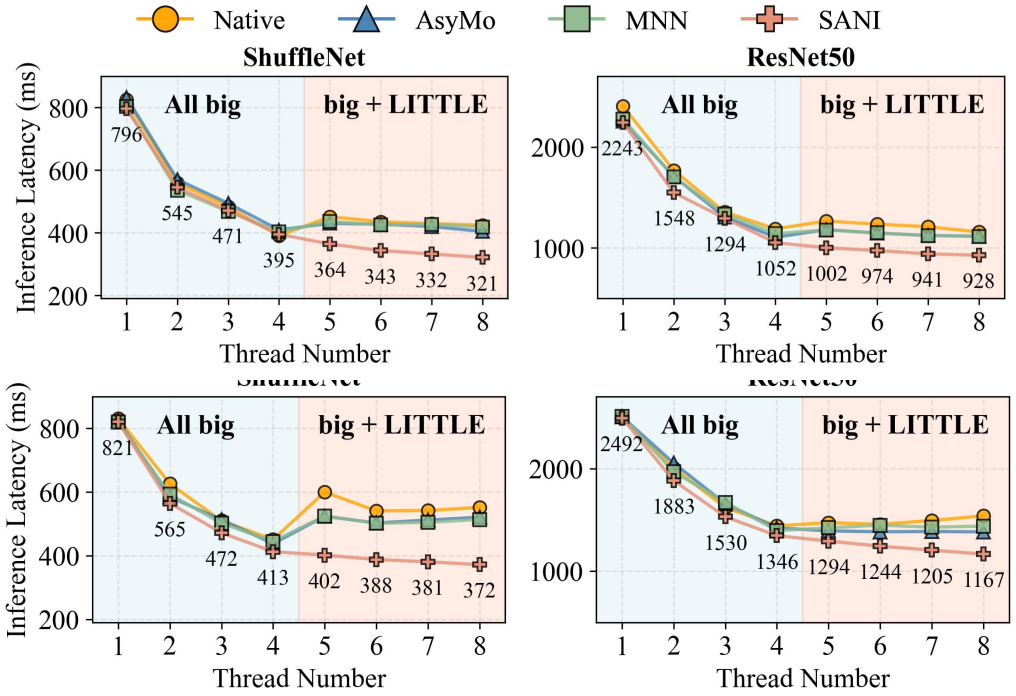
17.6–23.7% lower average latency than Native across five commercial and development SoCs

SANI keeps scaling after LITTLE cores join

OSDI '26

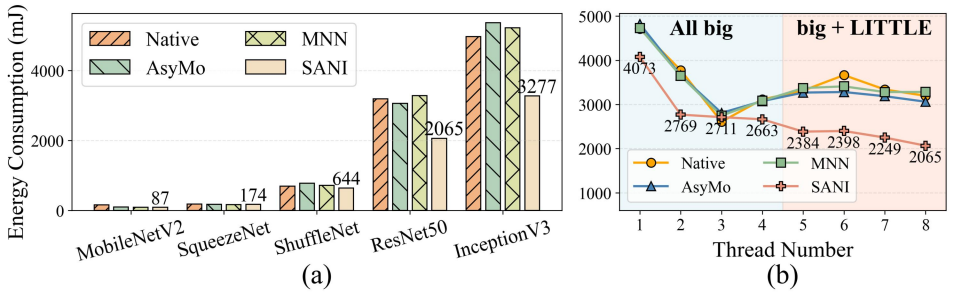
More heterogeneous cores keep helping

Pixel 9



OPPO Find X3 Pro

SANI saves more energy on heavier models



34.1–35.3%

less energy vs Native

ResNet-50 and Inception-V3

Mixed big + LITTLE execution compounds lower latency with better energy per operation.

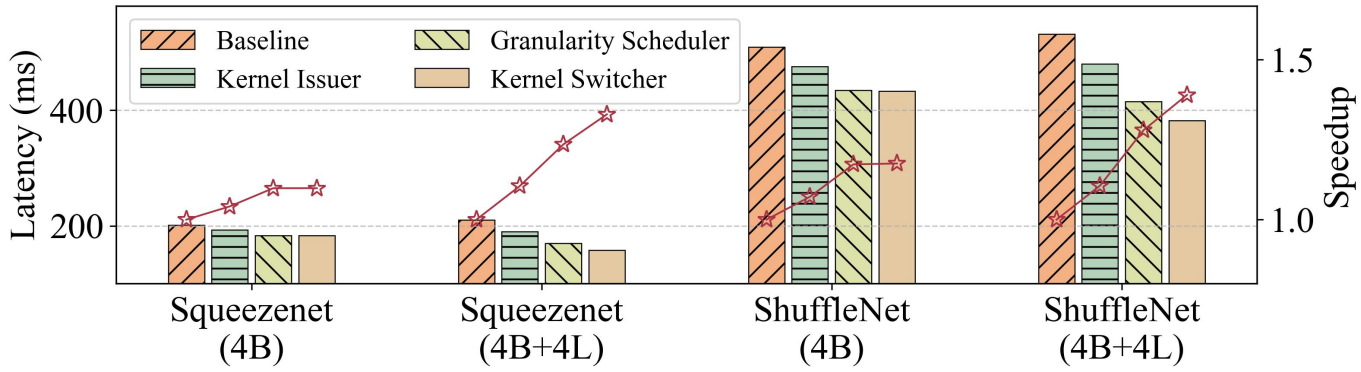
SANI adapts workload size to the capacity threads actually deliver.

Source: SANI Figures 13–14 and §5.2. Results shown for Pixel 9 / Find X3 Pro and MobileNetV2 interference scenarios.

Each mechanism contributes at low overhead

OSDI '26

ABLATION



Issuer — 9.5–9.6%
dual-kernel issue on 4B+4L

Scheduler — +10.5–13.5%
adaptive granularity on 4B+4L

Switcher — +7.1–8.0%
affinity after migration on 4B+4L

The switcher matters only when asymmetric clusters exchange work.

Model	Async Prep	Merge & Split	Kernel Trans
MobileNetV2	3.8	2.2	0.08
Squeezenet	3.2	1.1	0.25
Shufflenet	11	2.2	0.18

< 3 ms online · < 2 MB peak
ShuffleNet merge/split + transforms · all models memory

Async preparation is one-time model-load work. Merge/split and kernel transforms are online.

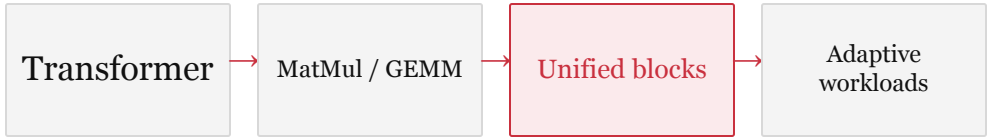
Transformers fit SANI; production LLMs fit NPUs

OSDI '26

EVALUATION · SCOPE AND GENERALITY

WHY THE DESIGN CARRIES OVER

SANI is operator-shape specific, not model-class specific.



16.1%
lower GPT-2 latency

The same MatMul and GEMM kernels also dominate larger transformers.

Applicable whenever CPU inference is the dominant or fallback path.

WHY CPU IS NOT THE PRIMARY LLM TARGET

Qwen-1.5-4B · Pixel 9 · four big cores

22.78	6.10
prefill tokens/s	decode tokens/s

> 1000 prefill tokens/s reported by recent on-device NPU work

NPUs also deliver substantially better energy per token, making them the right production target when supported.

Mechanism generality does not imply that CPUs are the best backend for every model scale.

Source: SANI §5.2 and §5.5. GPT-2, Qwen-1.5-4B, and NPU throughput values are taken from the paper.

Takeaways

OSDI '26

CONCLUSION

Key Observation

- Static workload sizes cannot balance task-acquisition overhead and barrier waiting under core asymmetry and runtime interference.
- Different core types prefer different kernel implementations; one universal kernel leaves performance untapped.

SANI USES ALL CORES

Affinity-aware kernels, adaptive granularity, and on-demand switching work together across big and LITTLE cores.

MAKE ASYMMETRY PAY

SANI reduces average latency by 17.6–23.7% and energy by up to 39%, with less than 3 ms online overhead.

SANI turns asymmetric mobile CPUs from a bottleneck into an advantage.

Q&A