A Workload-Aware DVFS Robust to Concurrent Tasks for Mobile Devices

MOBICOM’23

Chengdong Lin[1,2], Kun Wang[1], Zhenjiang Li[1], Yu Pu[2]
[1]Department of Computer Science, City University of Hong Kong, China; [2]Alibaba DAMO Academy, China

冯敏远

2025.7.18

提纲

- 作者介绍
- 研究背景与动机
- GearDVFS系统设计
- 实验评估
- 总结与讨论

提纲

- 作者介绍
- 研究背景与动机
- GearDVFS系统设计
- 实验评估
- 总结与讨论

作者简介

Chengdong Lin

院校： 香港城市大学（City University of Hong Kong）博士毕业生
导师： Zhenjiang Li

研究方向：
- Mobile Device
- Tracking System

毕业于2023年，现工作于华为（香港）

主要研究：
- [MOBICOM’23] A Workload-Aware DVFS Robust to Concurrent Tasks for Mobile Devices
- [SenSys’22] Gaze Tracking on Any Surface with Your Phone
- [MobiSys’19] Poster: When Wearable Sensing Meets Arm Tracking

提纲

- 作者介绍
- **研究背景与动机**
- GearDVFS系统设计
- 实验评估
- 总结与讨论

研究背景与动机

移动SoC的发展

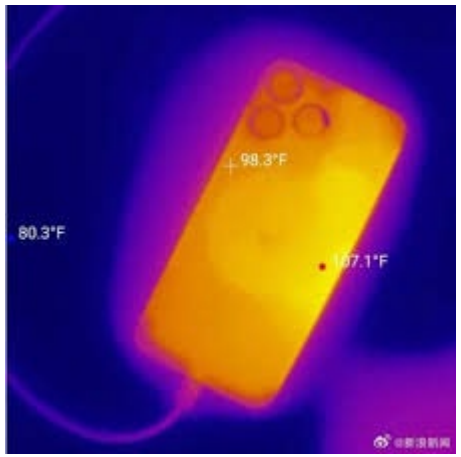
随着移动系统级芯片（SoC）性能的大幅提升，诸如深度学习、游戏和视频处理等高性能任务可以在智能手机、嵌入式板卡（如 NVIDIA Jetson、Odroid）等设备上运行。

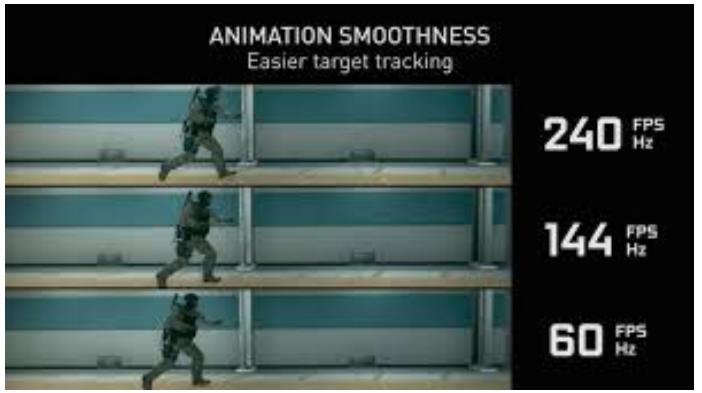
研究背景与动机

移动 SoC 上的功耗与发热挑战

但这些设备由于体积极其有限，一旦功耗过高就会产生严重的发热，不仅缩短芯片寿命，也极大影响用户体验和续航能力。因此，如何在保证足够计算能力的同时有效控制功耗与温度，成为移动平台的关键问题。

- 电池容量：移动设备的电池容量有限
- 热管理：高性能任务（如AI推理、游戏）会产生大量热量
- 用户体验：确保在不同工作负载下（如视频流、游戏）维持稳定的帧率、延迟等





研究背景与动机

DVFS 方法介绍（DVFS，Dynamic Voltage and Frequency Scaling）

DVFS作为最常用的功耗管理技术，能通过在运行时调整 CPU/GPU 频率来匹配负载，在移动设备中平衡性能与能效，对延长电池寿命和减少发热至关重要。

频率调整：处理器的时钟频率决定其计算速度。频率越高，处理器性能越强，但功耗也越高。DVFS通过降低频率来减少功耗，或在需要高性能时提高频率。

$$E \propto f \times V^2,\ V \propto f \ \Rightarrow E \propto f^3$$

研究背景与动机

现有的DVFS方法

- 应用导向DVFS：为特定应用（如目标检测）定制频率调整策略，优化单一任务的QoE。但在并发任务场景下表现不佳，因为不同任务的QoE需求难以统一。
  （zTT，MobiSys’2021，cpu、gpu联合调节）
- 应用无关DVFS：基于硬件指标（如CPU利用率，目标约80%）进行频率调整，缺乏对具体工作负载特性的感知，导致在复杂或动态工作负载下性能次优。
  （Performance、Schedutil ，Linux中的两种CPU DVFS governor）

研究背景与动机

现有的DVFS方法

- 应用导向DVFS：为特定应用（如目标检测）定制频率调整策略，优化单一任务的QoE。但在并发任务场景下表现不佳，因为不同任务的QoE需求难以统一。
  （zTT，MobiSys’2021，cpu、gpu联合调节）
- 应用无关DVFS：基于硬件指标（如CPU利用率，目标约80%）进行频率调整，缺乏对具体工作负载特性的感知，导致在复杂或动态工作负载下性能次优。
  （Performance、Schedutil，Linux中的两种CPU DVFS governor）

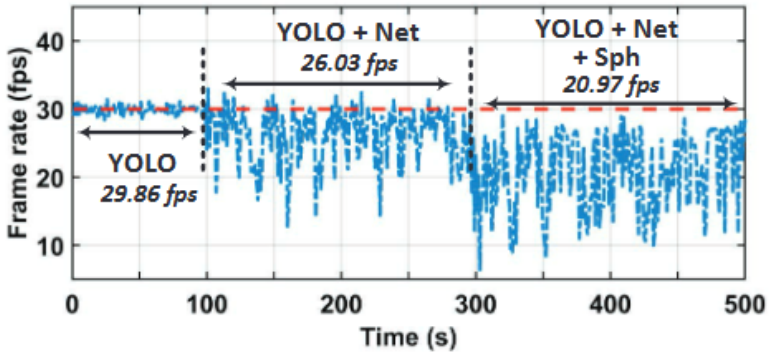


Figure 1: Frame rates achieved by zTT [41] for YOLO v3 under different workloads with the target 30 fps QoE.

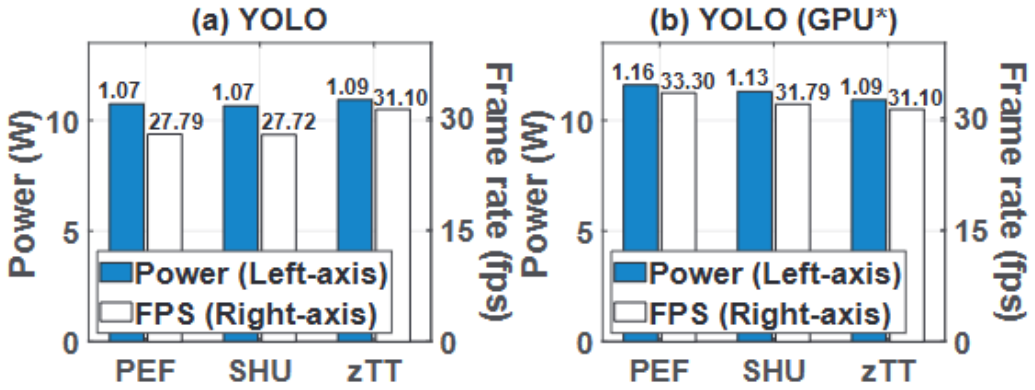
虽可在单一任务上大幅优化，但一旦遇到后台并发任务，其性能保障能力急剧下降，而多任务并发正是现代移动设备的常态

研究背景与动机

现有的DVFS方法

- 应用导向DVFS：为特定应用（如目标检测）定制频率调整策略，优化单一任务的QoE。但在并发任务场景下表现不佳，因为不同任务的QoE需求难以统一。
  （zTT，MobiSys’2021，cpu、gpu联合调节）

- 应用无关DVFS：基于硬件指标（如CPU利用率，目标约80%）进行频率调整，缺乏对具体工作负载特性的感知，导致在复杂或动态工作负载下性能次优。
  （Performance、Schedutil ， Linux中的两种CPU DVFS governor）



无GPU调试　　最优GPU频率

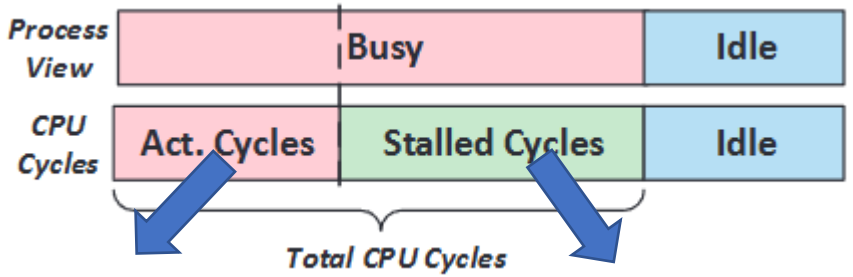
（1）挑战一：无法感知任务负载
例如目标检测任务YOLO， CPU 与 GPU 应当联合调度

研究背景与动机

现有的DVFS方法

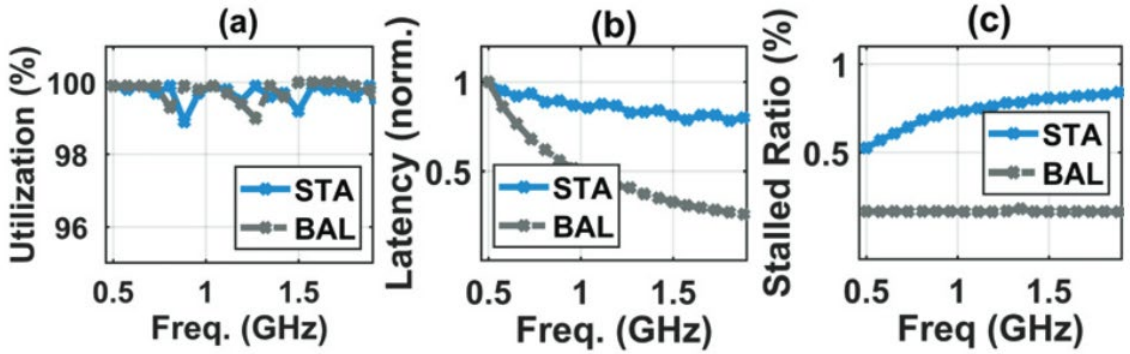
- 应用导向DVFS

- 应用无关DVFS：基于硬件指标（如CPU利用率，目标约80%）进行频率调整，缺乏对具体工作负载特性的感知，导致在复杂或动态工作负载下性能次优。
  （Performance、Schedutil，Linux中的两种CPU DVFS governor）



活跃计算
(运算、执行指令)

阻塞等待
(内存加载、I/O响应)



STA：内存密集型，频繁读写内存，阻塞等待比例高
BAL：计算与内存访问平衡，两种子状态比例相对均衡

（2）挑战二： CPU忙碌分为 活跃计算 和 阻塞等待 两个状态（“假忙” 和 “真忙”）

研究背景与动机

现有的DVFS方法的缺陷 总结 和 洞见

- **应用导向DVFS:**
  (1) 并发任务下的 QoE 崩塌, zTT在 YOLO+网络+语音三任务并发时，帧率从 29.8 fps 降到 20.9 fps。
- **应用无关DVFS:**
  (1) 缺乏对负载特征的感知，没法协调CPU 和 GPU的调频
  (2) CPU利用率“假忙”、“真忙”不可区分，提升频率对BAL（平衡型）有明显延迟改进，对STA（内存密集型）却没什么作用

如何同时支持多任务并发?

- 应用导向的方法只能针对单一任务优化。
- 应用无关的方法又缺乏对负载特征的感知。

如何做到“细粒度”的负载感知?

- 需要额外的硬件元数据去分辨CPU 和 GPU 负载比例 、 “活跃计算”与“阻塞等待”。

研究背景与动机

现有的DVFS方法的缺陷 总结 和 洞见

- **应用导向DVFS:**
  (1) 并发任务下的 QoE 崩塌, zTT在 YOLO+网络+语音三任务并发时，帧率从 29.8 fps 降到 20.9 fps。
- **应用无关DVFS:**
  (1) 缺乏对负载特征的感知，没法协调CPU 和 GPU的调频
  (2) CPU利用率“假忙”、“真忙”不可区分，提升频率对BAL（平衡型）有明显延迟改进，对STA（内存密集型）却没什么作用

洞见：移动 SoC 中，随时都在产生大量性能监控数据，如温度、频率、处理器利用率等 —— 如果把它们拼成一个多维特征，就能对任务动态和并发关系做出精确刻画。

如何[illegible]

- 应用[illegible]
- 应用无关的方法又缺乏对负载特征的感知。

如何做到“细粒度”的负载感知?

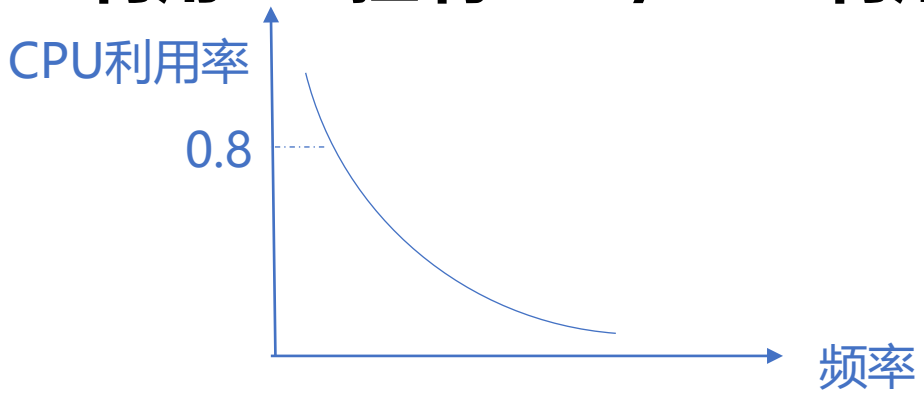
- 需要额外的硬件元数据去分辨CPU 和 GPU 负载比例 、 “活跃计算”与“阻塞等待”。

提纲

- 作者介绍
- 研究背景与动机
- **GearDVFS系统设计**
- 实验评估
- 总结与讨论

GearDVFS系统设计

GearDVFS的核心思想：利用 RL 控制CPU/GPU利用率接近目标值（如80%）



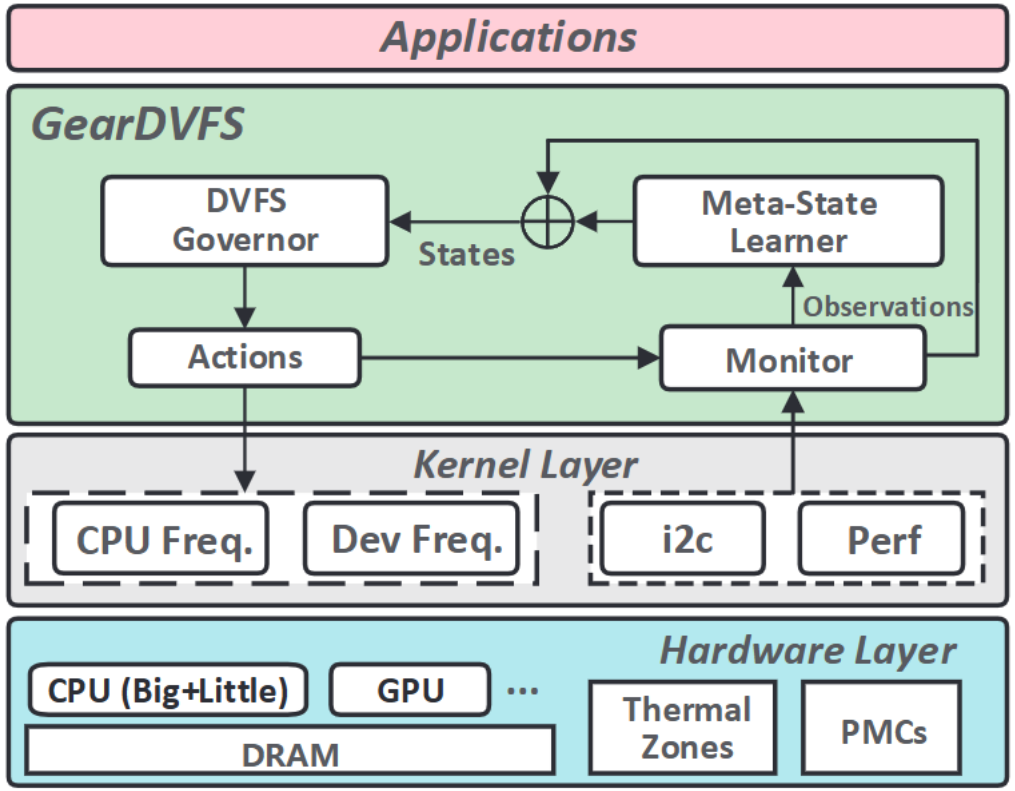
在周期T内，最小化 M个处理器实际利用率与目标值的偏差 $\min \frac{1}{T\times M}\sum_{t=1}^{T}\sum_{i=1}^{M}|u_i(t)-u_i^g|,$

频率调节不超出硬件手册规定的可行范围 $f_i^{min} \le f_i(t) \le f_i^{max},\ \forall i,t,$

每个处理器的温度不超过其热调节阈值 $c_i(t) \le c_i^{thermal},\ \forall i,t.$

GearDVFS系统设计

GearDVFS的核心思想：利用 RL 控制CPU/GPU利用率接近目标值（如80%）

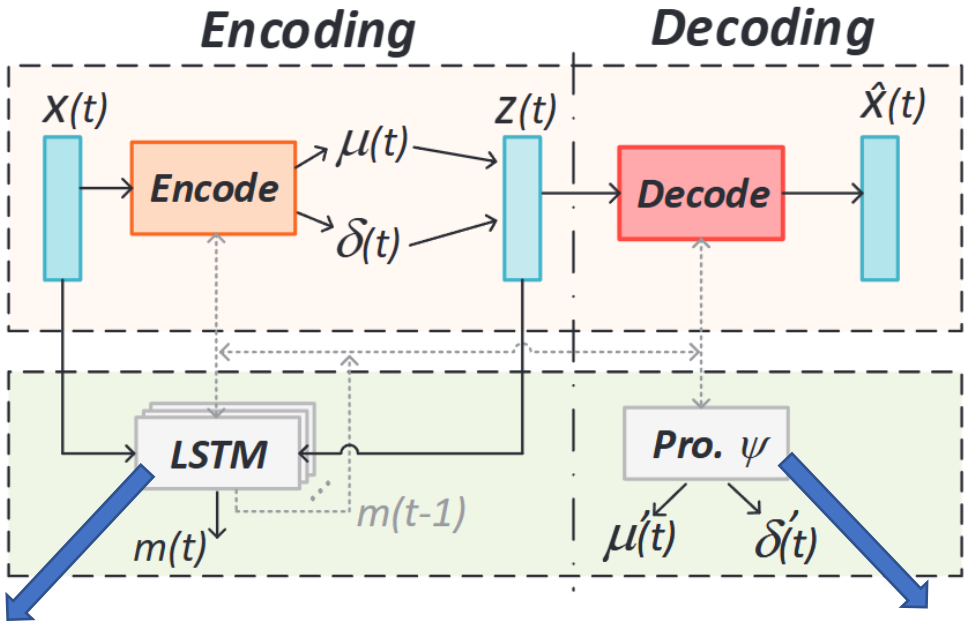


1. 如何感知复杂负载——Meta-state；
2. 如何决策频率——Action；
3. 如何定义好坏——Reward；
4. 如何持续适应新场景——Governor Update。

训练通常在每个时间窗口（如100毫秒）内进行。系统根据当前的状态选择频率调整动作，并通过反馈的奖励不断调整Q值函数。

GearDVFS系统设计

- **GearDVFS设计一：Meta-state**



- CPU 活跃计算利用率
- CPU 阻塞等待利用率
- GPU 利用率
- 各处理器当前频率
- 各处理器温度

$$z(t) \mid x(t) \sim \mathcal{N}(\mu(t), \sigma^2(t)) \quad p(z(t)) = \mathcal{N}(z(t)\,;\, \mu_{\text{prior}}(t), \sigma^2_{\text{prior}}(t))$$

$$L_{\text{MSE}} = \frac{1}{T}\sum_{t=1}^{T} \text{MSE}(x(t), \hat{x}(t)) = \frac{1}{T}\sum_{t=1}^{T} \| x(t) - \hat{x}(t) \|^2$$

$$L_{KL} = \frac{1}{T}\sum_{t=1}^{T} \text{KL}(p(z(t) \mid x(t)) \,\|\, p(z(t))) \qquad L = L_{\text{MSE}} + L_{KL}$$

长短时记忆网络: 捕捉时序特征

能够利用过去的硬件观测信息来更准确地预测未来的负载走势，从而生成更有效的 meta-state（元状态）

先验分布动态估计：避免过拟合

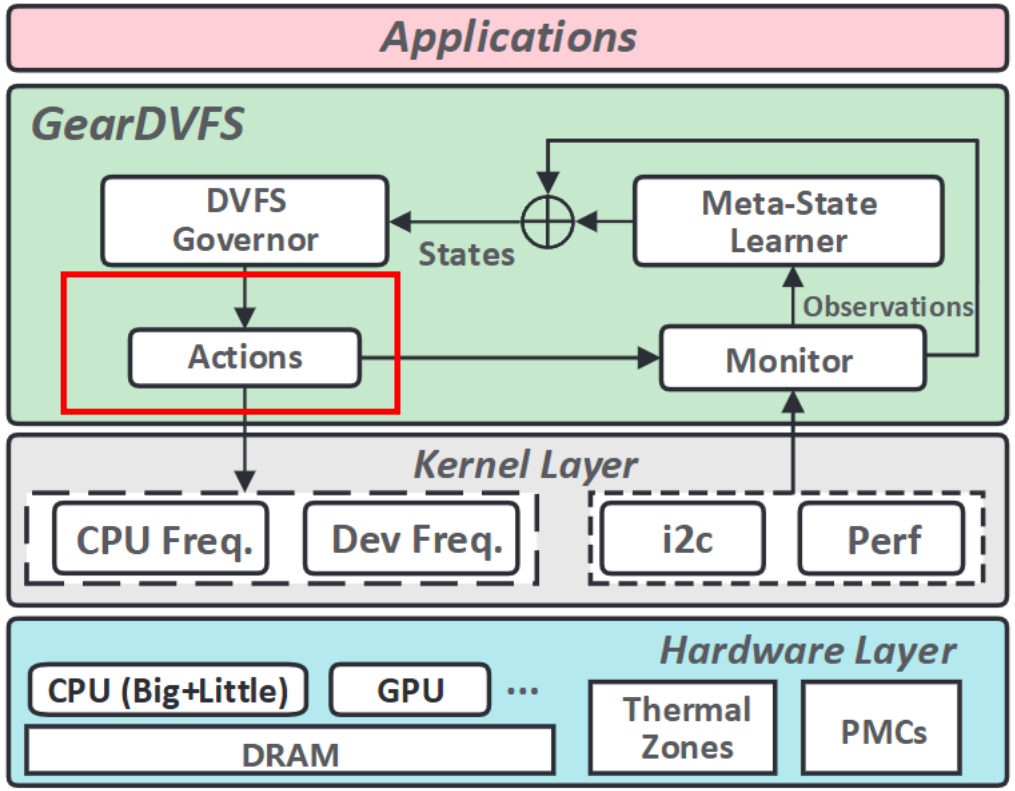
为后验分布 p(z(t) | x(t)) 提供动态参考，从而正则化模型、避免过拟合并更好地捕捉负载的时序特征。

训练完成后：$S_{MS} = \langle m(t), x(t) \rangle$

增强决策时的状态表达

GearDVFS系统设计

- **GearDVFS设计二：Action**



对于 DVFS 而言，动作被定义为设置每个 CPU 和 GPU 处理器的频率。采取不同的动作会导致不同的设备状态，并由奖励函数进行评估。

假设系统中共有 M 个处理器，第 i 个处理器有 n_i个可选频率级别，则总动作数为

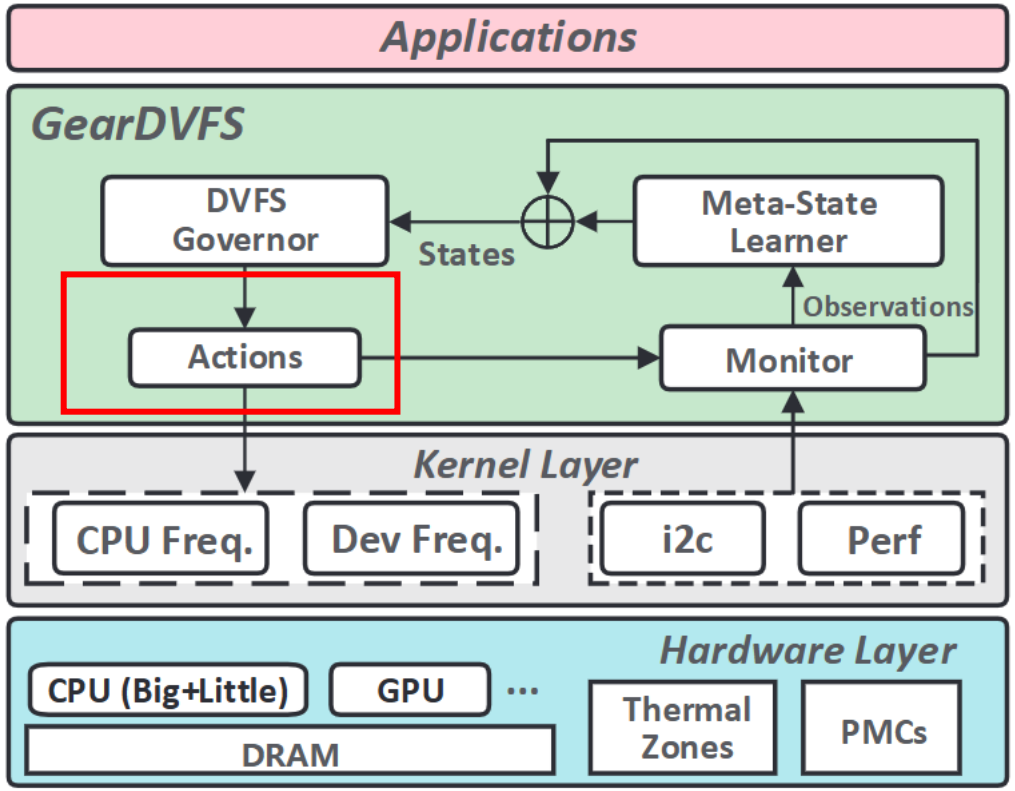
$$\prod_{i=1}^{M} n_i,$$

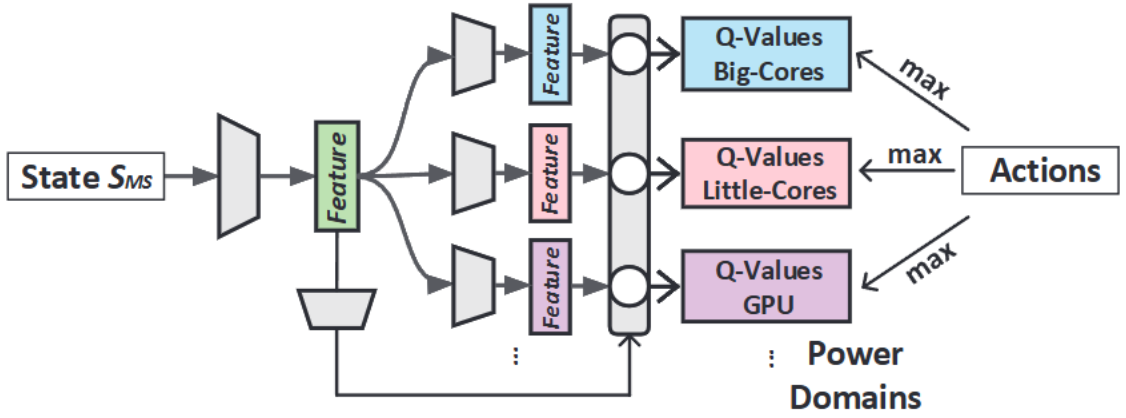
这对强化学习Q网络来说，会非常耗时又难以收敛。

现有方法是每次只采样动作空间的一个子集，虽然降低了训练复杂度，却因大多数组合未被探索而难以获得最优性能

GearDVFS系统设计

- **GearDVFS设计二：Action**



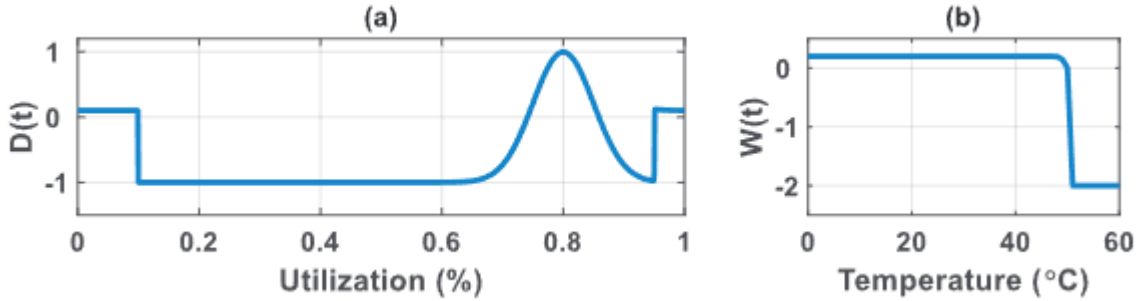


先将State输入一组**共享特征提取层**，抽取 CPU、GPU等各域的公共特征；然后将这些公共特征分别送入对应的**特征提取分支**，每个分支负责一个域的动作评估。

$$\prod_{i=1}^{M} n_i, \quad \Rightarrow \quad \sum_{i=1}^{M} n_i,$$

GearDVFS系统设计

- **GearDVFS设计三：Reward**



u^g = 80%　　T_{thre} = 50°C

DVFS 设计目标是让各处理器利用率贴近预设目标，同时避免过热

奖励组成：
1.利用率奖励 $D_i(t)$
2.温控奖励 $W_i(t)$)

$$R(t) = \frac{1}{M}\sum_{i=1}^{M}[D_i(t) + W_i(t)]$$

$$D_i(t) = \begin{cases} \lambda, & u_i(t) \notin [u_i^{min}, u_i^{max}], \\ u + v \times e^{-\frac{(u_i(t)-u_i^g)^2}{w^2}}, & otherwise, \end{cases}$$

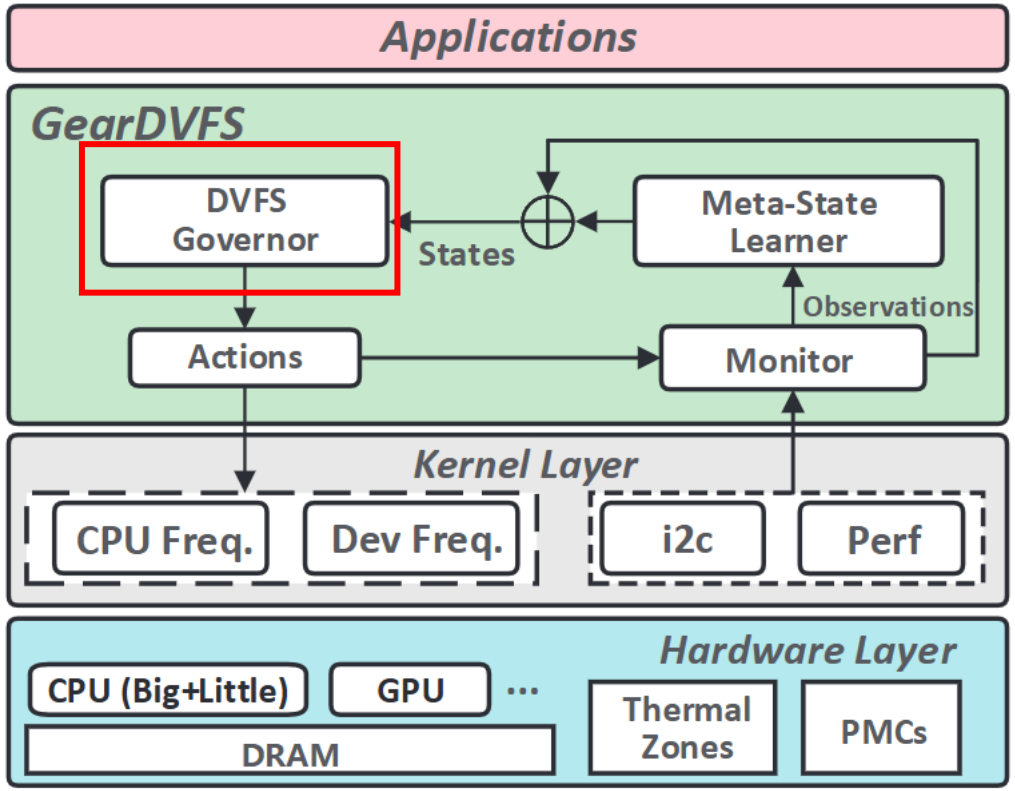
$$W_i(t) = \begin{cases} 0.2\tanh\left(T_{\text{thre}} - T_C(t)\right), & T_C(t) < T_{\text{thre}}, \\ -2, & T_C(t) \geq T_{\text{thre}}, \end{cases}$$

- 越靠近目标利用率，奖励越高（接近 +1）
- 距离越大，奖励 快速下降（可到 -1）
- 极端场景：当利用率太低或太高，哪怕用最小/最大频率也达不到目标，统一给一个小正奖励 （经验值 0.1），避免 RL 过度惩罚

- 温度低于阈值时，奖励缓慢衰减
- 温度超阈值时，强烈惩罚（例如-2）

GearDVFS系统设计

- **GearDVFS设计四： Governor Update**



在设计好状态、动作和奖励后，GearDVFS 经训练即可在每个时刻选择最优动作以获取最高奖励。

训练完成后，GearDVFS 在运行时可能会遇到与训练环境不同的新“上下文”，例如全新的负载变化、更高或更低的温度等。

可以利用实际运行数据来仅对已定好的网络结构中的少量参数进行再训练，快速更新调度器

提纲

- 作者介绍
- 研究背景与动机
- GearDVFS系统设计
- **实验评估**
- 总结与讨论

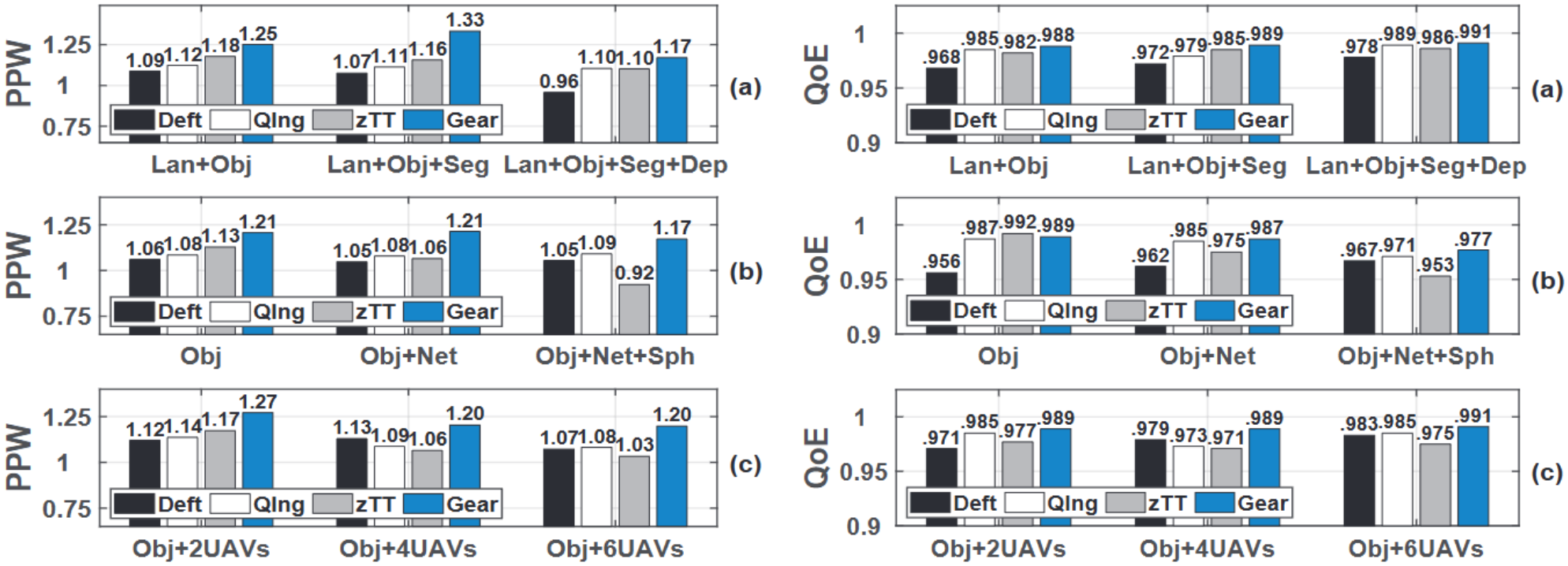
实验评估

实验设置

- **硬件平台:** Jetson NX、Odroid XU3、Jetson Nano、Raspberry Pi 4B、Redmi Note 9
- **数据集:**
  1. 自动驾驶：车道检测、目标检测、语义分割和深度估计，负载偏平衡但更倾向计算密集
  2. 机器人：YOLO v3（目标识别）、视频上传（网络任务）、QuartzNet（语音识别）。其中 1)、3) 计算密集，2) 内存密集。
  3. 无人机地面站：YOLO v3 和多路视频接收，分别对应计算与内存密集型负载。
  4. 手机应用：抖音、绝地求生和 Zoom，均以计算密集为主。
- **Baseline:**
  1. 默认（deft）： cpu — schedutil, GPU — simple_onedemand
  2. Q-Learning（Qlng）:一种强化学习算法，仅利用处理器利用率来控制 CPU 频率，不考虑负载上下文；GPU 仍然沿用 simple_ondemand 默认策略。
  3. zTT：面向应用的 DVFS 方法，在调节频率的同时加入了对热节流的处理
- **评估指标:**
  1. 性能功耗比（PPW）：定义为 QoE（服务质量体验）与功耗的比值，值越大表示能效越高
  2. 利用率（Util）： 本实验将 CPU 和 GPU 的目标利用率均设为 80%。评估实际利用率与 80%的接近程度。

实验评估

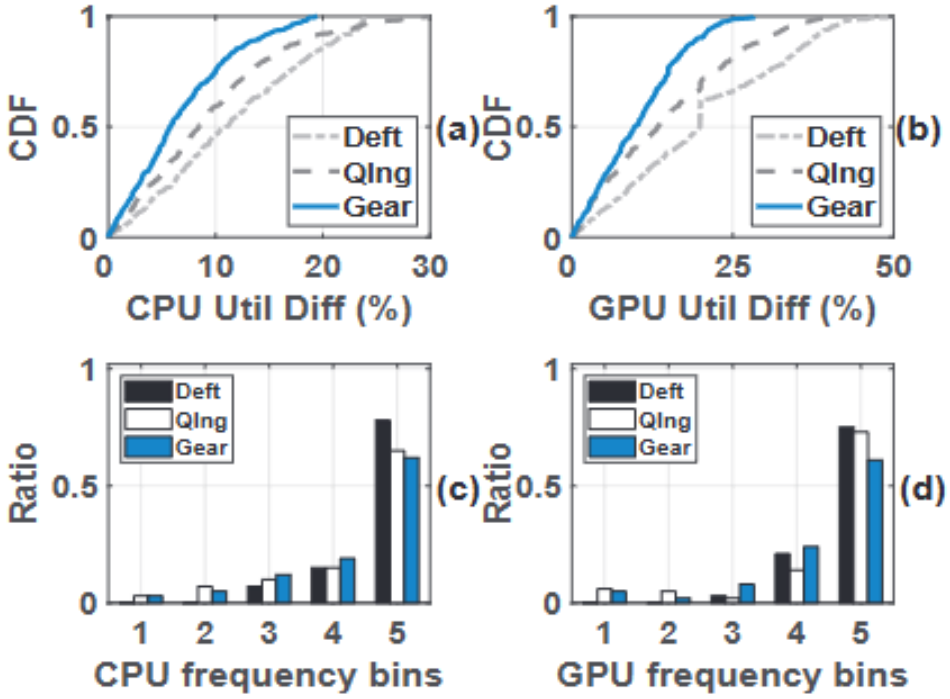
核心结果 I：总体性能功耗比(PPW) 和 QoE指标

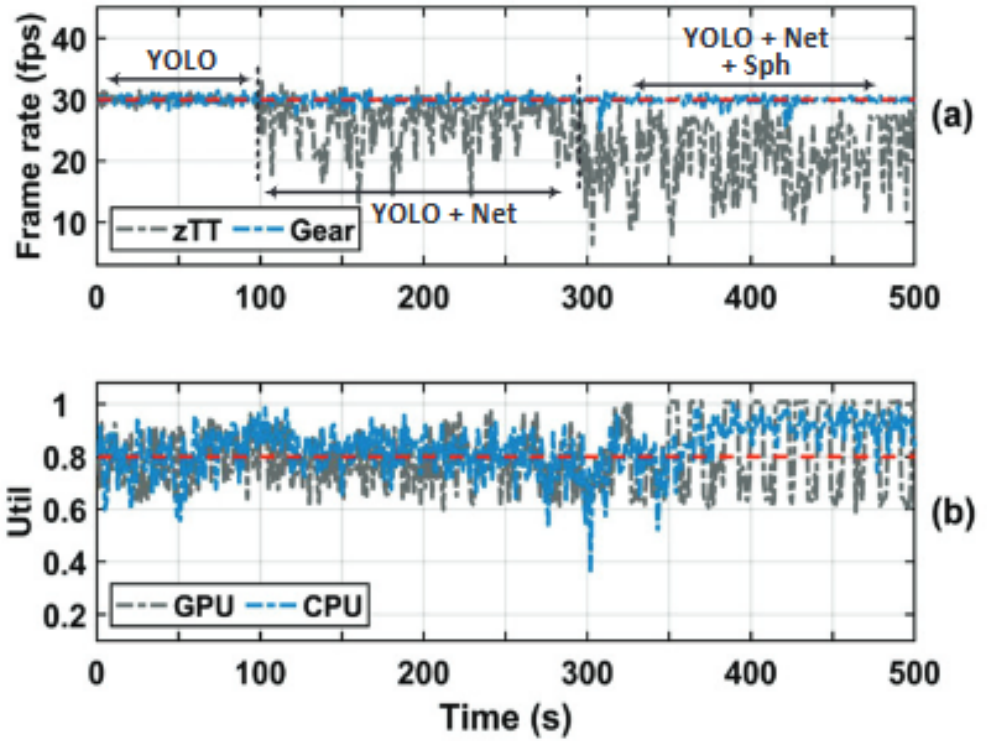


车道检测（Lan）、目标检测（Obj）、语义分割（Seg）、深度估计（Dep）和无人机视频流数目（UAVs)

实验评估

核心结果 II：CPU、GPU利用率 和 帧率维持

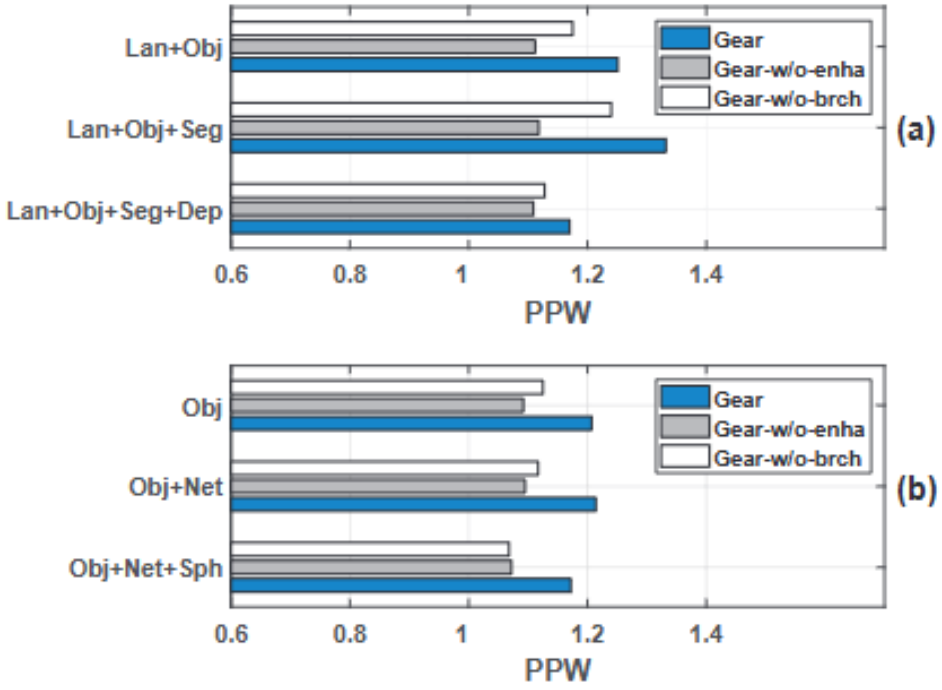


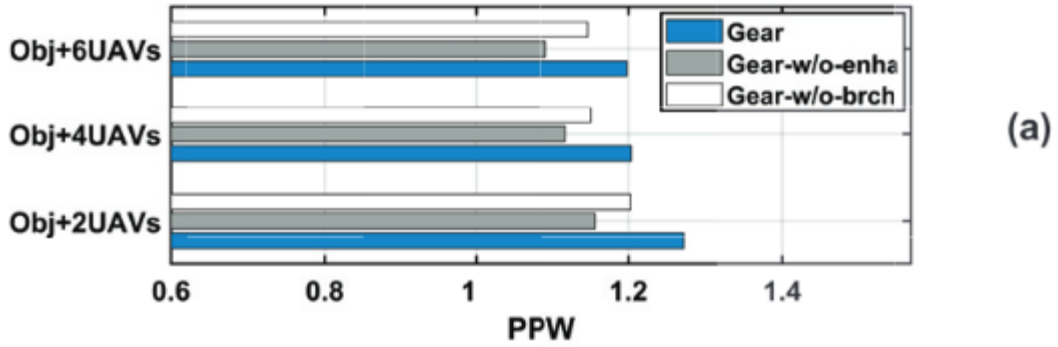


- CPU和GPU利用率差异的累积分布函数
- CPU和GPU频率选择的分布情况

实验评估

消融实验





Gear-w/o-enha：移除LSTM和先验分布动态估计（但保留主元状态学习器），导致元状态质量下降；
Gear-w/o-brch：移除分支式网络结构，仅抽取动作空间的一个子集。

车道检测（Lan）、目标检测（Obj）、语义分割（Seg）、深度估计（Dep）和无人机视频流数目（UAVs）

提纲

- 作者介绍
- 研究背景与动机
- GearDVFS系统设计
- 实验评估
- **总结与讨论**

总结与讨论

总结：

GearDVFS，一种在强化学习框架下结合元状态学习（通过编码–解码与 LSTM 从硬件元数据提取工作负载上下文）与分支式网络高效探索大规模频率调节动作空间的通用 DVFS 方案

能否提高：

- 高训练开销、 慢速适应：Gear训练从零开始收敛大约需要 2100 s，适应新场景时需约 650 s。
- 没有考虑不同任务的实际QoS需求。

我们的工作: 针对AI模型对手机进行调度的场景优化，考虑应用负载，计算密集型还是内存密集型，考虑多应用并发场景。

Q&A

2025年7月18日

冯敏远