Synergy: Towards On-Body AI via Tiny AI Accelerator Collaboration on Wearables

Taesik Gong, *Member, IEEE*, SiYoung Jang, *Member, IEEE*, Utku Günay Acer, Fahim Kawsar, *Member, IEEE*, and Chulhong Min, *Member, IEEE*

***Abstract*—The advent of tiny artificial intelligence (AI) accelerators enables AI to run at the extreme edge, offering reduced latency, lower power cost, and improved privacy. When integrated into wearable devices, these accelerators open exciting opportunities, allowing various AI apps to run directly on the body. We present *Synergy* that provides AI apps with best-effort performance via system-driven holistic collaboration over AI accelerator-equipped wearables. To achieve this, Synergy provides device-agnostic programming interfaces to AI apps, giving the system visibility and controllability over the app's resource use. Then, Synergy maximizes the inference throughput of concurrent AI models by creating various execution plans for each app considering AI accelerator availability and intelligently selecting the best set of execution plans. Synergy further improves throughput by leveraging parallelization opportunities over multiple computation units. Our evaluations with 7 baselines and 8 models demonstrate that, on average, Synergy achieves a 23.0× improvement in throughput, while reducing latency by 73.9% and power consumption by 15.8%, compared to the baselines.**

***Index Terms*—On-body AI, tiny AI accelerator, accelerator composition, wearables.**

I. Introduction

THE advent of tiny artificial intelligence (AI) accelerators, such as the Analog MAX78000 [1], MAX78002 [2], Google Coral Micro [3] and GreenWaves GAP-9 [4] has brought AI closer to us than ever before, offering reduced latency, low power cost, and improved privacy. These accelerators, designed for microcontrollers (MCUs) with small form factors (e.g., MAX78000: 8 mm × 8 mm in Fig. 1), are becoming integrated into wearable devices recently [5], [6], [7], [8], e.g., smart earbuds, patches, watches, glasses, wristband, and shoes. This integration transforms wearable devices from smartphone-dependent accessories—used merely for data collection or notification alerts—into AI-capable devices.

Received 25 September 2024; revised 23 February 2025; accepted 7 April 2025. Date of publication 29 April 2025; date of current version 3 September 2025. Recommended for acceptance by X. Peng. *(Corresponding author: Chulhong Min.)*

Taesik Gong was with Nokia Bell Labs, Murray Hill, NJ 07974-0636 USA. He is now with the UNIST, Ulsan 44919, South Korea (e-mail: taesik.gong@unist.ac.kr).

SiYoung Jang, Utku Günay Acer, and Chulhong Min are with Nokia Bell Labs, Murray Hill, NJ 07974-0636 USA (e-mail: siyoung.jang@nokia-bell-labs.com; utku_gunay.acer@nokia-bell-labs.com; chulhong.min@nokia-bell-labs.com).

Fahim Kawsar is with Nokia Bell Labs, Murray Hill, NJ 07974-0636 USA, and also with University of Glasgow, G12 8QQ Glasgow, U.K. (e-mail: fahim.kawsar@nokia-bell-labs.com).

Digital Object Identifier 10.1109/TMC.2025.3564314

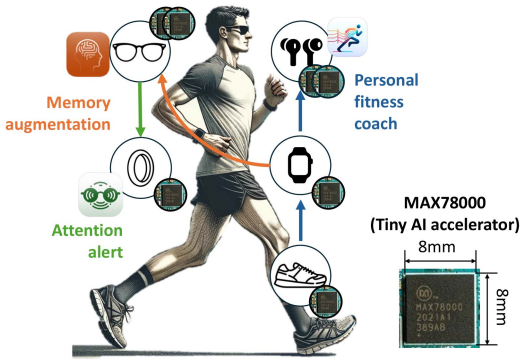

Fig. 1. Wearable computing powered by tiny AI accelerators, simultaneously running on-body AI apps.

With the proliferation of wearable devices, it is natural to expect that personal computing environments on the body will form a network of these AI-capable wearables [9]. This enables a new class of *on-body AI apps* to emerge, which enhance app functionality by leveraging on-device AI capabilities with diverse sensing capabilities and interaction methods available across wearables. These apps, running on wearable devices, continuously monitor a variety of user contexts and proactively provide context-aware services to users directly via diverse wearable interfaces. Fig. 1 shows an example scenario with three concurrent on-body AI apps: (i) *memory augmentation* detects greeting words using a smartwatch microphone and captures nearby faces using glasses-mounted cameras, (ii) *attention alert* monitors surrounding visual events through glasses and provides haptic alerts on a ring, and (iii) *personal fitness coach* analyzes exercise routines and vital signs on a smartwatch and smart shoes, providing auditory feedback via earbuds.

This computing environment, surrounded by AI-capable wearable devices, presents exciting opportunities for the runtime system. By leveraging the runtime collaboration of AI accelerators, it can support on-body AI apps with enhanced performance efficiently. The system can dynamically select the most suitable devices to execute tasks for sensing, model execution, and interaction. By strategically distributing model workloads across different AI accelerators, the system can prevent resource conflicts and ensure smooth operation, improving both individual app performance and overall system efficiency. Additionally, by splitting concurrent models and running the split chunks over multiple AI accelerators, the system maximizes

processing capacity, supporting more models simultaneously and accommodating larger models that a single AI accelerator cannot handle.

However, it is not straightforward to realize these benefits. In the current paradigm for multi-wearable programming, task-device assignment decisions are often made at the development time. These individual app-level decisions may not be optimal at runtime due to the dynamic nature of wearable devices and dependencies on resource usage between concurrent apps. One may consider existing model partitioning techniques [10], [11], [12], [13], [14], [15], [16]. While they support dynamic model splitting, they mainly focus on optimizing the splitting decision for *a single AI model*, thereby lacking a holistic view that considers (i) interdependency among different tasks—sensing, model inference, interaction on distributed devices—within an app and (ii) resource conflict across concurrent apps. Additionally, no existing partitioning work has been built for tiny AI accelerators yet, making it difficult to directly adopt them for on-body AI apps. Note that such dynamic challenges can be addressed by sending raw sensor and offloading all processing tasks to smartphones, which can be assumed to be always available. However, this incurs high communication costs to both wearable devices and the smartphone, thereby leading to increased power consumption and degraded inference throughput of AI models, as discussed in Section II-B.

We present Synergy, a first-of-its-kind runtime system designed to efficiently support on-body AI apps on tiny AI accelerator-equipped wearable devices. At its core, Synergy provides runtime orchestration that makes holistic decisions of task-device assignments for concurrent apps over distributed AI accelerators. To this end, Synergy decouples task-device assignment decisions from app logic by providing device-agnostic programming interfaces. Then, Synergy dynamically creates various execution plans (mapping logical tasks to physical devices) for each app, considering available resources, communication over devices, model splitting options, etc. Synergy selects the optimal set of execution plans to maximize the system-wide performance. In this paper, we focus on maximizing the execution throughput of AI apps, but other objectives can be adopted as well.

To this end, Synergy has four key components. (i) Synergy generates holistic collaboration plans, each representing the mapping of all logical tasks in concurrent apps to physical resources, by holistically considering resource demands from concurrent apps and resource availability of wearables. (ii) For efficient decision-making on resource-constrained MCUs, we propose a data intensity-aware pipeline accumulation method that helps Synergy reduce the exponential search cost to linear cost while providing comparable performance to the complete search. (iii) We devise a novel latency estimation model designed for tiny AI accelerators which is used to estimate and compare the throughput of holistic collaboration plans. (vi) After deployment, Synergy further improves model throughput via an adaptive task parallelization scheduler. It reduces the latency of a selected collaboration plan by maximizing parallelization opportunities of concurrent apps at runtime across distributed computation units on wearables (e.g., processors, AI accelerators, wireless chips).

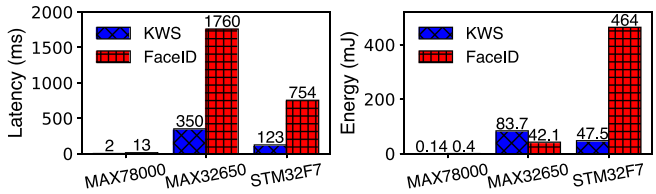

Fig. 2. Performance comparison between AI accelerator (MAX78000) and MCUs (MAX32650 and STM32F7).

We prototyped Synergy on two tiny AI accelerator platforms, MAX78000 and MAX78002. We compare Synergy with 7 baselines including state-of-the-art model partitioning techniques [10], [11], [12], [13], [14], [15], [16]. Our extensive evaluation with 8 AI models shows that Synergy consistently outperforms the baselines, with on average 23.0× throughput gain across various scenarios, while reducing latency by 73.9% and power consumption by 15.8%. Our in-depth experiments also show that Synergy effectively adapts to various runtime environment changes: the number of devices, the number of pipelines, heterogeneous device resources, different source and target mapping, and different objectives.

II. Background & Motivation

A. Tiny AI Accelerators

The integration of AI accelerators into MCUs represents a significant move towards distributed, on-device AI, ensuring enhanced user privacy and minimal latency. Although a number of tiny-scale accelerators have been proposed recently, very few products are commercially available and provide access and control over their underlying operations. In this paper, we chose Analog MAX78000 [1] and MAX78002 [2] as our primary platforms since they are the most widely used tiny AI accelerator research platforms [17], [18], [19], [20], [21], [22], [23], [24], [25] owing to the disclosed hardware details and open-source tools, enabling in-depth analysis and modification of operations.

Both MAX78000 and MAX78002 have an Arm Cortex-M4 processor with different memory capacities; 512 KB of flash and 128 KB of SRAM on MAX78000 and 2.5 MB and 384 KB on MAX78002. For acceleration, they feature a convolutional neural network (CNN) accelerator that contains 64 CNN processors specially designed for parallel convolutional operations at ultra-low power. The CNN accelerator has the dedicated memory; 512 KB of data memory, 442 KB of weight memory, and 2 KB of bias memory on MAX78000 and 1.3 MB, 2 MB, and 8 KB on MAX78002, respectively.

Recent benchmark study [21], [26] quantifies the MAX-78000's superior performance in terms of latency and energy cost. Fig. 2 shows MAX78000 significantly outperforms a conventional MCU, MAX32650 with Cortex-M4 at 120 MHz [27] and even a high-performance MCU, STM32F7 with Cortex-M7 at 216 MHz [28] in key AI models. Latency for keyword spotting (KWS) is reduced to 2.0 ms compared to 350 ms and 123 ms

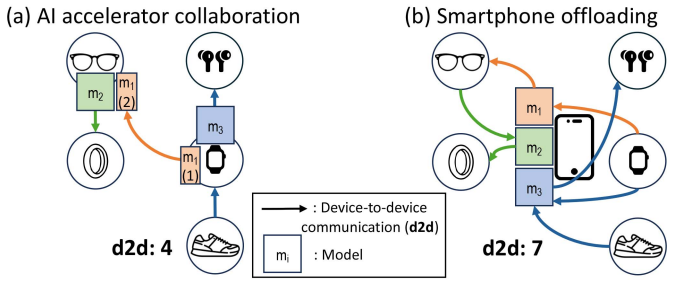

Fig. 3. Comparison between (a) AI accelerator collaboration via Synergy and (b) phone offloading.

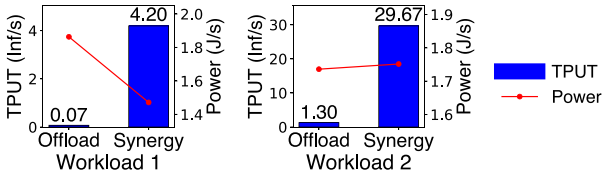

Fig. 4. Comparison of Synergy and phone offloading.

for MAX32650 and STM32F7, respectively. Energy efficiency is similarly enhanced, with MAX78000 consuming merely 0.40 mJ for face detection (FaceID), in contrast to 42.1 mJ and 464 mJ consumed by MAX32650 and STM32F7.

B. AI Accelerator Collaboration

With wearable devices integrating tiny AI accelerators, we foresee a proliferation in on-body AI apps running across distributed wearables (Fig. 1). These environments offer opportunities to enhance performance through runtime collaboration of AI accelerators as follows.

Dynamic device selection: The system can dynamically allocate tasks such as sensing, model execution, and interaction to the most suitable devices based on resource availability and conditions. For example, a higher-performance AI accelerator can be chosen for an inference task as the set of available wearable devices changes.

Strategic workload distribution: Concurrent apps often face resource conflicts. For example, assigning multiple models to the same AI accelerator can cause out-of-resource (OOR) errors if their combined size exceeds the capacity. By distributing workloads across AI accelerators, the system prevents conflicts and ensures efficient execution of apps.

Optimizing AI accelerator utilization: Tiny AI accelerators are usually optimized for single-model support. By splitting and distributing models across multiple AI accelerators, the system can maximize processing capacity. This approach supports more models simultaneously and accommodates large models that cannot fit into a single AI accelerator.

1) Why Not Offloading?: Offloading AI model execution to a smartphone might seem advantageous due to its higher processing capability. In particular, this would be true for conventional mobile apps where model execution occurs intermittently, mostly upon a user's request, and the service with the final inference output is provided on a smartphone. However, this approach can be inefficient for on-body AI apps because (i) they require continuous sensing and inference for situational services and (ii) services are directly provided through wearable devices. Fig. 3 shows an illustrative comparison when the three apps introduced in Section I are concurrently running. While the AI accelerator collaboration can support the concurrent execution of three apps with four device-to-device (d2d) communications as shown in Fig. 3(a), smartphone offloading requires seven d2d communications as shown in Fig. 3(b), as the data must pass through a smartphone regardless of the data path. Thus, such offloading incurs additional data communication and results in increased latency and energy costs. These additional transmission costs are unnecessary when apps run solely on wearables.

Continuous transmission of raw sensor data to a smartphone also causes communication and energy bottlenecks, affecting both smartphones and wearables. We compared the total throughput of model execution using Workload 1 and 2 from Section VI-B. By eliminating unnecessary links, Synergy improves throughput by 57.7× and 28.8× compared to smartphone offloading, as shown in Fig. 4. Despite frequent model execution, Synergy consumes less or comparable power due to avoiding energy-intensive data communication.

Based on these findings, we can conclude that offloading (sending raw sensor data to a smartphone) is not effective when (i) the size of raw sensor data is larger than that of the intermediate or final results, (ii) the final results need to be delivered to wearable devices for the service, and (iii) the model needs to be executed frequently for situational and proactive services, which is the case for on-body AI apps.

III. Synergy Design

A. Limitations of Existing Approaches

In the current multi-wearable programming paradigm, task-device assignments are predetermined at development time, limiting adaptability in dynamic environments with concurrent AI apps and heterogeneous wearables. This app-level assignment decision poses several challenges:

- *Lack of adaptability to resource contention:* If a memory augmentation app relies on a smartwatch's AI accelerator, it may underperform or fail due to competition from other apps or battery depletion, even when other devices have available accelerators.
- *Limited visibility into available resources:* Developers lack real-time awareness of available AI accelerators and their capabilities, making it difficult to leverage multiple devices efficiently.

Existing model partitioning techniques [10], [11], [12], [13], [14], [15], [16] primarily focus on offloading computation to powerful devices (e.g., smartphones or cloud). While they could be adapted for distributed on-body AI accelerators, they have fundamental limitations:

- *Single-model optimization:* These techniques optimize splitting decisions for individual models, without considering: (i) Resource dependencies between concurrent apps.

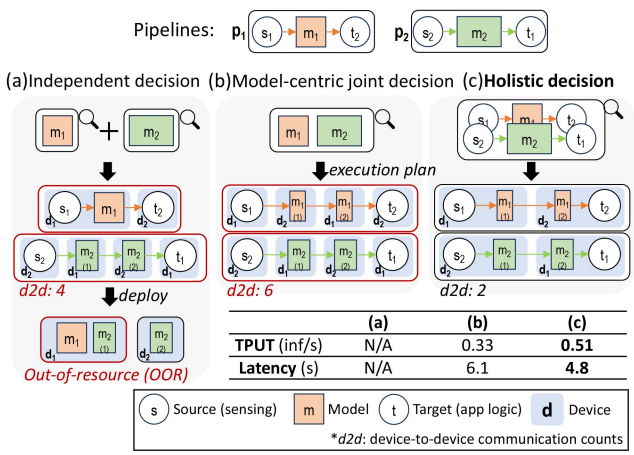

Fig. 5. Comparison of decision approaches.

(ii) Interdependencies between model inference and other tasks (e.g., sensing and interaction).

- *Incompatibility with tiny AI accelerators:* No existing method is specifically designed to partition models onto tiny AI accelerators, leading to inaccurate performance modeling in resource-constrained environments.
- *Independent decision-making:* Existing partitioning methods split and deploy each model independently, ignoring conflicts among models. This can lead to resource contention and out-of-resource (OOR) failures, particularly for memory-constrained accelerators. *Example:* As shown in Fig. 5(a), if two apps deploy model chunks to the same device (d_1), the full model of p_1's m_1 and the partial model of p_2's m_2 may exceed d_1's memory capacity, leading to OOR failures.
- *Model-centric joint decision:* Some methods extend model partitioning to account for memory constraints, filtering out unsupported cases. However, this model-centric view still remains suboptimal, as it ignores sensing and interaction costs in an app's end-to-end pipeline. *Example:* As illustrated in Fig. 5(b), even when split models of m_1 and m_2 avoid memory conflicts, long-distance execution between source and target devices increases communication overhead, degrading overall performance.

These limitations highlight the need for a holistic, system-driven approach that considers concurrent workloads, device heterogeneity, and end-to-end execution costs.

B. Our Approach: AI Accelerator Collaboration With a Holistic View

We propose shifting from app-level independent decision-making to system-driven collaboration in environments with concurrent AI applications on distributed AI accelerator-equipped wearables. Our approach introduces the following key innovations:

- *Decoupled task-device assignment*: Existing approaches tightly couple task execution with app logic, limiting flexibility. We decouple task-device assignment, allowing the runtime system to gain system-wide visibility and fine-grained control over AI accelerator collaboration.
- *Device-agnostic programming interface*: Synergy provides a programming model (Section IV-B) that enables developers to define end-to-end AI pipelines without specifying target devices. This abstraction facilitates dynamic task placement, adapting execution based on real-time resource availability.
- *Holistic collaboration and runtime optimization*: Unlike existing methods that optimize resource allocation per app, Synergy takes a holistic approach, considering both intra-app and inter-app dependencies. It constructs a system-wide collaboration plan by leveraging direct knowledge of concurrent workloads, resource availability, and task dependencies. The runtime dynamically assigns tasks to devices based on their capabilities and splits AI models across distributed AI accelerators to maximize efficiency.
- *End-to-end performance optimization*: Beyond model partitioning, Synergy optimizes task placement by minimizing communication overhead between source, execution, and target devices. This improves throughput and reduces overall latency. Fig. 5(c) illustrates an example of this holistic collaboration.

By adopting a system-driven approach, Synergy overcomes the limitations of independent per-app optimization, improving efficiency, reducing latency, and enhancing AI accelerator utilization in multi-app distributed AI environments.

C. System Scope

Moderator-initiated orchestration: In our current implementation, orchestration tasks—discovering and managing devices, creating and selecting holistic collaboration plans, and deploying them to devices—are managed by an external moderator, such as a smartphone. This is due to the limited computing capabilities of MCUs and their lower accessibility compared to smartphones, as users may not always wear their wearables. Orchestration is needed only when there is a change in apps or device availability. Once set up, runtime model inference operates independently on wearable devices. We envision a shift towards a more decentralized approach, embedding these orchestration capabilities into powerful wearables, facilitating self-sufficiency in wearable AI systems and reducing the need for external devices.

Target metric: To execute multiple models across distributed devices, various system-wide objectives can be considered, such as maximizing inference throughput, minimizing latency, and reducing energy consumption. This paper focuses on maximizing overall system-wide model inference throughput, a key quality of service metric in AI apps. To ensure fairness among models, we merge multiple app pipelines into a unified one and maximize its execution per second. Simply maximizing total throughput—the number of model executions per second—could let lower-latency models monopolize resources, causing fairness issues. This target metric can be replaced with other objectives, such as minimizing latency or energy costs. Results are reported in Section VI-C4.

Potential synergy with smartphones: While Synergy primarily focuses on tiny AI accelerators on wearable devices and their collaboration, and we show its benefits over phone offloading for on-body AI apps in Section II-B, incorporating smartphones could enhance system performance and flexibility in other scenarios. In particular, a hybrid approach leveraging both wearables and smartphones can be advantageous for cases where large AI models need to run with body signals, e.g., foundation models with IMU data [29] and optical physiological signals [30]. Another example would be when the final inference outcome is served using a mobile UI on a smartphone, thereby requiring no transmission overhead from a smartphone to wearable devices. By integrating the synergy with smartphones, Synergy could extend its applicability to a broader range of AI-driven wearable apps while maintaining seamless user experiences and efficient on-body computing. We leave this as future work.

IV. Synergy Runtime Technical Components

A. Overview

This section provides a high-level overview of our approach. We first summarize the key technical novelties that distinguish Synergy from existing solutions, followed by a step-by-step description of Synergy's operational flow in multi-wearable environments.

1) Summary of Technical Novelties: Synergy introduces several innovations that improve performance and flexibility for on-body AI applications:

- *Device-agnostic programming interface:* An abstracted interface for building on-body AI pipelines without specifying target devices at design time.
- *Holistic collaboration planning:* A global approach that considers all concurrent apps together, supporting multi-model splitting and effective resource sharing.
- *Progressive search space reduction:* A method to mitigate the combinatorial explosion of multi-pipeline orchestration, reducing complexity from exponential to near-linear in the number of pipelines.
- *Clock-cycle-based latency modeling:* A precise inference-latency estimator that captures per-layer execution time on tiny AI accelerators by analyzing hardware clock cycles.
- *Online throughput estimation:* A lightweight scheme for dynamically evaluating end-to-end performance in real time, enabling adaptive selection of the best collaboration plan.
- *Adaptive task parallelization:* Mechanisms that exploit idle computation units by overlapping tasks across multiple pipelines (inter-pipeline) and consecutive runs (inter-run).

These novelties overcome limitations of per-app approaches by jointly managing on-body resources. Next, we describe how Synergy integrates these techniques into its operational flow.

2) Operational Flow: Fig. 6 depicts the runtime stages of Synergy:

App development & installation stage: On-body AI apps are developed using the *device-agnostic programming interface* (Section IV-B). Developers only need to specify the pipeline's logical tasks without worrying about which wearables will execute them. Once the app is installed, the pipeline description is registered with a moderator that manages device assignments.

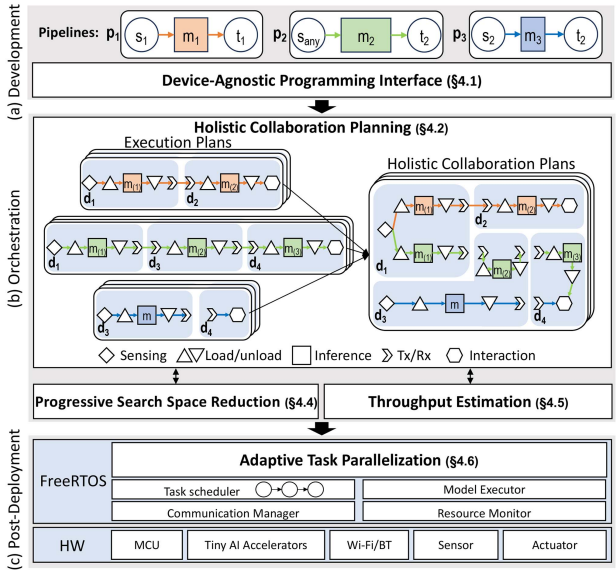

Fig. 6. An illustration of Synergy's operational flow.

Orchestration stage: When running apps or available devices change, Synergy triggers *holistic orchestration* by generating holistic collaboration plans (Section IV-C). It maps all tasks in concurrent apps to devices by matching task requirements with device capabilities and potentially splitting model tasks across distributed AI accelerators. Because exhaustively searching for the best split and device assignment can be computationally intractable on resource-constrained MCUs, Synergy applies *progressive search space reduction* (Section IV-D) to prune infeasible or suboptimal plans. Once feasible candidates are formed, Synergy employs a *novel latency estimation model* and an *online throughput estimation* method (Section IV-E) to select the plan expected to yield the highest overall throughput.

Post-deployment stage: As multiple pipelines execute under a single holistic collaboration plan, the entire execution cycle completes when all pipelines finish. This can create idle periods on certain computation units (e.g., MCUs, AI accelerators, wireless chips). To further boost throughput at runtime, Synergy incorporates *adaptive task parallelization* (Section IV-F), which overlaps tasks across pipelines (inter-pipeline) and across multiple runs (inter-run) to reduce idle time and improve overall system performance.

By coordinating these stages under a single, unified framework, Synergy effectively balances task distribution, model-splitting decisions, and parallelization strategies to achieve high performance and resource utilization across multiple on-body AI accelerators.

B. Device-Agnostic Programming Interface

We structure on-body AI apps as a directed acyclic graph (DAG) of tasks for several reasons. First, DAGs align well

with the architecture of many AI-centered apps. Second, this structure represents data and device dependencies among tasks, providing Synergy visibility and controllability of concurrent apps. Third, DAGs inherently abstract app logic as a series of tasks, simplifying the division and allocation of these tasks across distributed devices.

In this work, we categorize tasks into three types: *sensing*, *model*, and *interaction*. Sensing tasks are specified with requirements such as sensor type, resolution, and position. Interaction tasks are described with requirements like interface type and physical location. Currently, Synergy supports two types of requirements: designated device and sensor type for sensing tasks, and designated device or interface type for interaction tasks. For model tasks, apps specify an AI model to be executed. For example, a memory augmentation app pipeline could be *(microphone, KeywordSpotting, camera on glasses)*, and an attention alert app could be *(camera on glasses, MobileNet, haptic)*. In this paper, we support three tasks for a pipeline, but it can be expanded to a DAG, e.g., conditional inference or fusion models.

C. Holistic Collaboration Planning

Execution plan creation: As an initial step to exploit available collaboration options, Synergy creates various *execution plans* for each app to run over distributed devices. An execution plan abstracts the task-device assignment of a pipeline, serving as the basis for holistic orchestration. The key to obtaining diverse execution plans from each pipeline is (i) exploring various combinations of model splitting layers across distributed AI accelerators and (ii) flexibly mapping source and target tasks to suitable devices.

An execution plan is defined as a sequence of tuples, each containing a task and an assigned device (Fig. 6(b)). Synergy supports seven task types: (i) sensing, (ii) data loading to accelerator memory, (iii) (partial) model inference, (vi) data unloading from accelerator memory, (v) Tx (transmitting data), (vi) Rx (receiving data), and (vii) interaction. For two wearables, smart glasses and a smart ring, a pipeline can be described as (*camera on glasses*, *EfficientNet*, *haptic feedback*); EfficientNet has 29 layers. Then, one execution plan example would be [*camera* $\rightarrow$ *data loading* $\rightarrow EfficientNet^{0:19}$ $\rightarrow$ *data unloading* $\rightarrow$ *Tx to ring*] for the glasses and [*Rx from glasses* $\rightarrow$ *data loading* $\rightarrow EfficientNet^{19:29}$ $\rightarrow$ *data unloading* $\rightarrow$ *haptic*] for the smart ring, where $EfficientNet^{i:j}$ refers to the model's layer i to j.

Runnable holistic collaboration plan generation: The next step is to generate *holistic collaboration plans*, each integrating execution plans from all pipelines (one execution plan per each app). It gives Synergy visibility over resource competition and dependencies across devices (see Fig. 6(b)). A key consideration is filtering out plans that cause OOR cases. We consider three memory constraints: (i) weight memory, (ii) bias memory, and (iii) the maximum number of layers supported (e.g., 442 KB, 2 KB, 32 for MAX78000 [1] and 1.3 MB, 2 MB, 128 for MAX78002 [2]). A collaboration plan is *runnable* if the total weight memory, bias memory, and number of layers for assigned tasks do not exceed the AI accelerators' capacities.

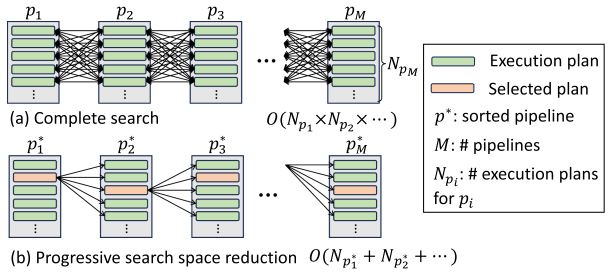

Fig. 7. Comparison between complete search (a) and our progressive pipeline selection (b).

D. Progressive Search Space Reduction

We propose a *progressive search space reduction* method to reduce the intractable search space for the holistic collaboration plan selection. For each pipeline p, the number of execution plans available N_p can be modeled as: $N_p = \sum_{d=1}^{D} {}_D\mathrm{P}_d \cdot {}_{L-1}\mathrm{C}_{d-1} \cdot D^2$, where D and L denote the number of available devices and model layers, respectively. ${}_D\mathrm{P}_d$ represents d-permutation of D, which means possible device orders to use, ${}_{L-1}\mathrm{C}_{d-1}$ is $d-1$ combinations of $L-1$, which is model splitting candidates, and D^2 is all source/target device mappings, respectively. This search space grows exponentially for holistic collaboration planning with concurrent pipelines, i.e., $O(N_{p_1} \times N_{p_2} \times \ldots)$. For instance, for three small models (a 9-layer KWS, a 14-layer SimpleNet, and a 19-layer UNet in Section VI-A) with three MAX78000 devices, the number of holistic collaboration plans that can be generated is $1,971 \times 4,941 \times 9,261 = 90,190,202,571$. It is evident that a complete search algorithm considering all combinations requires substantial computational overhead. To mitigate this, we propose *data intensity-aware execution plan accumulation*, shown in Fig. 7. This approach arranges execution plans in sequence via our data intensity metric and selects an execution plan for one app at a time, building on the collaboration plan integrated with previously selected execution plans from earlier apps, thereby reducing complexity to $O(N_{p_1} + N_{p_2} + \cdots)$.

Pipeline prioritization: The key to data intensity-aware execution plan accumulation is determining the sequence of pipelines. Synergy prioritizes data-intensive pipelines for execution plan selection, meaning pipelines that require the transmission of larger sensing data and intermediate outputs. The intuition is that optimizing data-intensive pipelines yields greater overall benefits for the system. To understand the relationship between data intensity (i.e., data size) and latency, we conducted a preliminary measurement. Fig. 8 shows layer-wise latency for inference, memory, and communication, alongside output data size using UNet [31] as an example. Compared to inference latency (1.5 ms), both memory latency (10.6 ms) and communication latency (6869.1 ms) are significantly higher, being $7\times$ and $4579\times$ greater, respectively. This latency gap between inference and communication is notable in tiny AI accelerators designed for fast inference. Moreover, the output size varies significantly across different layers, with the lowest layer latency at 4426.2 ms and the highest at 161864.5 ms, indicating a $36\times$ difference. These findings suggest that (i) the end-to-end latency

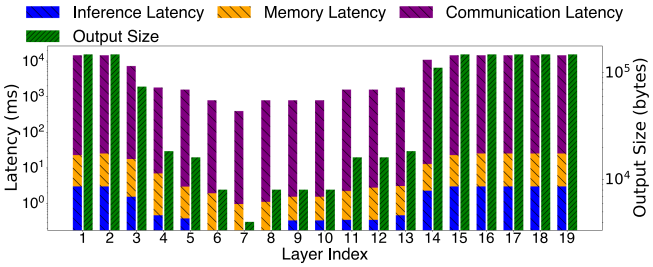

Fig. 8. Layer-wise latency analysis for UNet. The y-axis is in the logarithmic scale. Since the scale is logarithmic, the total height of each stacked bar represents the geometric sum of its components, making small values appear relatively smaller while highlighting differences across orders of magnitude.

TABLE I
WORKLOAD AND MODEL DESCRIPTION USED IN THIS STUDY

Workload	Pipeline	Models	Model Size	Input Size	Avg. Out Size
1	1	ConvNet5	71158	28×28×1	14031
	2	ResSimpleNet[37]	381792	32×32×3	11217
	3	UNet[31]	279084	48×48×48	74547
2	4	KWS[35]	169472	128×128×1	7976
	5	SimpleNet[38]	166448	32×32×3	9237
	6	WideNet[38]	313700	32×32×3	10091
3	7	EfficientNetV2[39]	627220	32×32×3	66468
4	8	MobileNetV2[36]	821164	32×32×3	296318

Sizes are all in bytes.

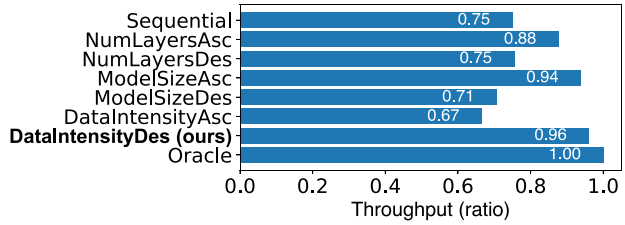

Fig. 9. Comparison among different pipeline prioritization strategies against complete search (Oracle).

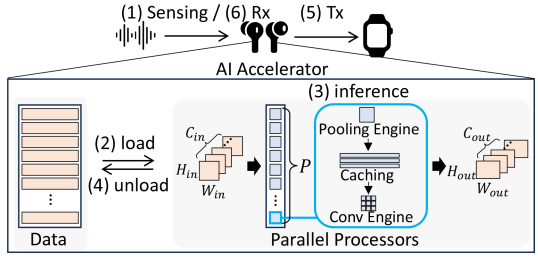

Fig. 10. Tasks involved in a pipeline for latency modeling (1)-(6) from the viewpoint of a device.

of a pipeline varies greatly depending on the model split in the execution plan, (ii) latency is strongly related to the size of data exchanged, and (iii) prioritizing data-intensive pipelines could offer more optimization opportunities due to fewer resource conflicts with other pipelines.

Pipeline arrangement and accumulation: We define a pipeline's data intensity as the average data size of transmission over all execution plans. Specifically, given the input size $\texttt{In}^{\texttt{size}}$ and output size $\texttt{Out}_l^{\texttt{size}}$ for each layer $l \in L$, the data intensity of a pipeline is defined as: $(\texttt{In}^{\texttt{size}} + \sum_l \texttt{Out}_l^{\texttt{size}})/(L+1)$. The pipelines are sorted in descending order with this metric. Then, for each pipeline, Synergy generates runnable holistic collaboration plans by combining each execution plan of the current pipeline with the previously selected execution plans from earlier pipelines. Then, Synergy selects the execution plan of which holistic collaboration plan is expected to have the highest system-wide throughput based on our online throughput estimation (Section IV-E).

Comparison with other prioritization methods: We investigate (i) how significantly our selection reduces the search space compared to a complete search, and (ii) how close our prioritization method is to the optimal selection compared to other alternatives. We compare Synergy with seven baselines: the complete search (Oracle), ascending order of data intensity (DataIntensityAsc), descending/ascending order of model size (ModelSizeDes/ModelSizeAsc), and descending/ascending order of the number of model layers (NumLayersDes/NumLayersAsc), and without prioritization (Sequential). Except for Oracle, the six baselines use the execution plan accumulation method but with different prioritization objectives. In this experiment, we assume three concurrent pipelines with two MAX78000 devices and consider all possible combinations of three pipelines out of eight (Table I). Fig. 9 shows the relative throughput ratio compared to Oracle by averaging the throughput over all pipeline combinations. The results validate our design choice. Synergy outperforms the other alternatives, showing only a 3.9% degradation in overall throughput compared to the complete search. Additionally, our execution plan accumulation approach results in a 5576× reduction in search space compared to Oracle. Note that the other prioritization methods have the same search space as ours because they are all based on our progressive pipeline selection.

E. Throughput Estimation for Distributed Tiny AI Accelerators

For holistic collaboration plan selection, evaluating the throughput of a plan is necessary. To achieve this, we first model the execution latency for each task and estimate throughput based on this latency model. Fig. 10 illustrates the operation of the five types of tasks ((1)-(6)) involved in a device. We present (i) our novel latency estimation model for model inference tasks, which reflects the unique characteristics of tiny AI accelerators (Section IV-E1), and (ii) the latency estimation for other tasks (Section IV-E2). We then introduce our technique for estimating the system-wide throughput of a holistic collaboration plan (Section IV-E3).

1) Latency Modeling for Tiny AI Accelerators: For latency estimation of model splitting options, existing studies rely on measurement-driven approaches: online latency measurement by running each model layer on the target device directly [12], [13] or offline latency modeling with model configurations [10], [14], [15], such as learning a regression model based on the number of model parameters and latency. The first approach—direct runtime measurement for each layer—is accurate and feasible for two-tier architectures (e.g., a smartphone and a cloud server) with limited cases but entails significant overhead in our scenario involving multiple models and resource-constrained

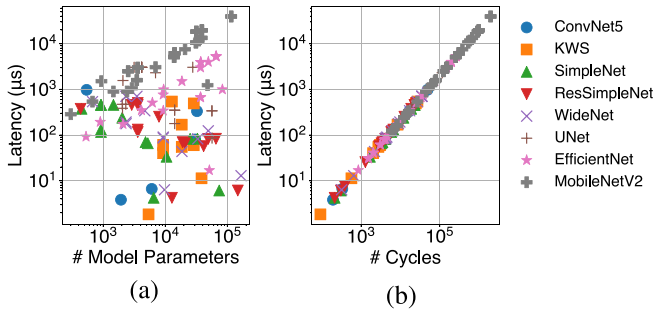

Fig. 11. Correlations of latency with (a) trainable parameters (left) and (b) accelerator clock cycles (right). Each dot means a different layer within a model.

devices. The second approach—latency modeling with offline profiling—mitigates this issue but has critical limitations in our environment. First, its scalability is highly limited, requiring profiling of all possible models and AI accelerators before app deployment. Second, due to the unique characteristics of AI accelerators, existing configuration-based latency modeling may not work. Fig. 11(a) shows the relationship between the number of model parameters and the corresponding inference latency of each layer in eight models on MAX78000. Due to hardware-level optimization in tiny AI accelerators, there is a weak correlation between these two variables.

To address these issues, we propose a *clock cycle-based latency model*. Unlike measurement-driven approaches, our model calculates the number of clock cycles required for arithmetic operations of a given layer on AI accelerators, reflecting the internal operation of convolution accelerators on tiny AI accelerators. The number of clock cycles is, by design, proportional to the latency, as the model inference task runs solely on dedicated hardware (tiny AI accelerator). This approach would be inaccurate for general-purpose processors because their use is shared with other system components or apps. Specifically, the latency within the AI accelerator can be represented as follows:

$$\mathcal{L}_{\texttt{ai\_acc}} = \mathcal{L}_{\texttt{load}} + \mathcal{L}_{\texttt{inf}} + \mathcal{L}_{\texttt{unload}}, \tag{1}$$

where $\mathcal{L}_{\texttt{inf}}$ is the inference latency with the AI accelerator, and $\mathcal{L}_{\texttt{load}}/\mathcal{L}_{\texttt{unload}}$ is the data (un)loading latency between the processor (Arm Cortex-M4) and AI accelerator via SRAM.

Inference latency modeling: To model $\mathcal{L}_{\texttt{inf}}$ ((3) in Fig. 11), we calculate the number of clock cycles $\mathbb{C}$ at AI accelerators. We can estimate the latency of the inference by dividing the number of clock cycles per each layer $\mathbb{C}_l$ with the clock frequency $\mathcal{F}_l$ of the AI accelerator that processing layer l, i.e., $\mathcal{L}_{\texttt{inf}} = \sum_l \frac{\mathbb{C}_l}{\mathcal{F}_l}$. Specifically, given the shape of an input $(H_{\texttt{in}}, W_{\texttt{in}}, C_{\texttt{in}})$ and the corresponding output $(H_{\texttt{out}}, W_{\texttt{out}}, C_{\texttt{out}})$ of a layer, representing the height, width, and number of channels respectively, the number of clock cycles for fully connected layers (MLP) and convolutional layers (CNN) under *sequential* processors such as Arm Cortex-M4 are as follows:

$$\hat{\mathbb{C}}_{\texttt{MLP}} = H_{\texttt{in}} \cdot W_{\texttt{in}} \cdot C_{\texttt{in}} \cdot C_{\texttt{out}} \tag{2}$$

$$\hat{\mathbb{C}}_{\texttt{CNN}} = K^2 \cdot H_{\texttt{in}} \cdot W_{\texttt{out}} \cdot C_{\texttt{in}} \cdot C_{\texttt{out}} \tag{3}$$

K refers to the size of the kernel. Note that tiny AI accelerators have (i) parallel computation units that parallelize operations across channels and (ii) convolution engines that process the convolution with a single clock cycle. Given the number of parallel convolutional processors, P, the clock cycles on AI accelerators are reduced to:

$$\mathbb{C}_{\texttt{MLP}} = H_{\texttt{in}} \cdot W_{\texttt{in}} \cdot \left\lceil \frac{C_{\texttt{in}}}{P} \right\rceil \cdot C_{\texttt{out}} \tag{4}$$

$$\mathbb{C}_{\texttt{CNN}} = H_{\texttt{in}} \cdot W_{\texttt{out}} \cdot \left\lceil \frac{C_{\texttt{in}}}{P} \right\rceil \cdot C_{\texttt{out}} \tag{5}$$

Fig. 11(b) shows the correlation between the clock cycles of every layer in eight models and the corresponding inference latency. Unlike Fig. 11(a), our clock cycle-based modeling shows a strong correlation by design. When we translate the number of clock cycles to the estimated latency with the clock frequency, the results show that the gap between the estimated and measured latency is less than 1%.

For memory operations, such as data loading and unloading between SRAM and accelerator memory ((2) and (4) in Fig. 11), we use a measurement-driven approach. We estimate $\mathcal{L}_{\texttt{load}}$ and $\mathcal{L}_{\texttt{unload}}$ by profiling them with different data sizes and creating a linear regression model. Since data communication between two memories occurs through the central bus at a dedicated data transmission rate, the runtime latency of memory operations is proportional to the data size. As memory latency is not affected by the model architecture, we can model $\mathcal{L}_{\texttt{load}}$ and $\mathcal{L}_{\texttt{unload}}$ using a few data samples of different sizes.

2) Latency Modeling for Other Tasks: Besides operations on tiny AI accelerators, on-body AI apps involve sensing and communication (Rx and Tx) tasks ((1), (5), (6) in Fig. 11). For sensing tasks, we measure the latency for camera and audio inputs with different parameters (e.g., different sampling rates) during the profiling phase and estimate the latency by matching the app's requirements to the profiles. For communication latency, similar to memory operations on tiny AI accelerators, we estimate it by dividing the data size by the bandwidth of the wireless channel. Advanced methods [10], [12] can address wireless transmission latency fluctuations, but this is beyond the scope of this paper.

3) Throughput Estimation of Holistic Collaboration Plan: The next step is to estimate the throughput of a holistic collaboration plan (a collection of execution plans from multiple apps). To this end, Synergy initially estimates the end-to-end latency of a holistic collaboration plan based on the latency estimation of tasks in the plan. Since a holistic collaboration plan is structured as a DAG with multiple source tasks and target tasks, its end-to-end latency can be defined as the longest path from any source task to any target task; the path's length is the cumulative latency of all tasks within the path. Then, the system-wide throughput is calculated by taking the inverse of the end-to-end latency and then multiplying this value by the number of pipelines.

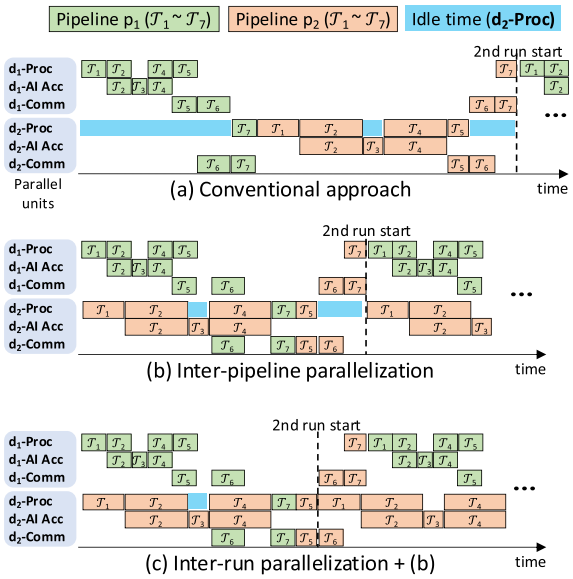

Fig. 12. Comparison between the conventional approach (a) and our proposed adaptive parallelization mechanisms (b, c) across two devices (d_1 and d_2).

F. Adaptive Task Parallelization

After selecting the best holistic collaboration plan, Synergy deploys the tasks and their connection to the corresponding devices. As existing model partitioning techniques [10], [11], [12], [13], [14], [15], [16] focused on a single run for a single model, a straightforward way to execute a holistic collaboration plan with concurrent pipelines over multiple runs (continuous inference over time) is to run each pipeline sequentially—immediately run another after the completion of the preceding pipeline—in a continuous manner, as shown in Fig. 12(a). However, since tasks in the plan runs over different computation units (processors, AI accelerator, and communication module) from distributed devices, such a sequential execution yields long idle time for unused units. The challenge is how to enhance performance by effectively utilizing these diverse computation units while keeping the sequential dependency of tasks within each pipeline.

To further enhance the system-wide throughput at runtime, we propose a method for *adaptive task parallelization*, which consists of two parallelization strategies. First, Synergy maximizes parallelization opportunities by concurrently executing tasks among different pipelines (*inter-pipeline parallelization*), which utilizes idle parallel computation units (Fig. 12(b)). Note that, when multiple tasks from different pipelines are competing with the same computation units, the later-arriving tasks need to wait until the completion of the earlier tasks. Second, we extend this concept to tasks from successive runs within a pipeline (*inter-run parallelization*) as shown in Fig. 12(c); for example, the model inference in the second run does not need to wait for the completion of the pipeline in the first run when the AI accelerator is idle and the sensor data for the second run is ready.

To enable these strategies, Synergy employs separate task queues for each type of computation unit, alongside a dedicated scheduler. At runtime, on each computation unit, its scheduler is responsible for dequeuing tasks from its queue and initiating their execution. Upon task completion, the scheduler adds the next task to its corresponding queue. This structured approach maximizes the utilization of computation units, thereby significantly improving the throughput of the holistic collaboration plan at runtime.

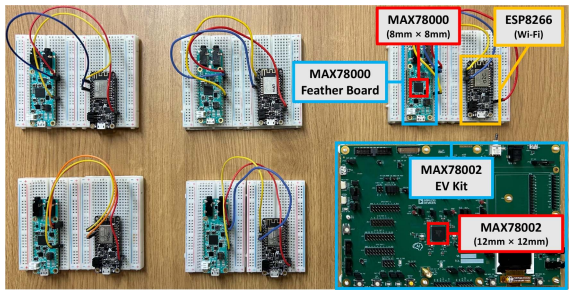

Fig. 13. Hardware setup.

V. Prototype Implementation

We prototyped Synergy on the off-the-shelf MAX78000 feather board [32] and MAX78002 Evaluation Kit [33], which are a development platform for the MAX78000 [1] and MAX78002 [2], respectively (Fig. 13). Note that although the development platform is bulky, the actual size of the accelerators is tiny (e.g., 8 mm × 8 mm for MAX78000).

For wireless communication, we interfaced MAX78000/MAX78002 with an ESP8266 Wi-Fi module [34]. To ensure that the communication process via Wi-Fi does not disrupt the data transmissions with MAX78000/MAX78002, we have employed a round-robin scheduling algorithm on ESP8266. This mechanism is designed to alternate the data flow: transmitting data to the wireless communication channel for outward transmission and redirecting to the serial communication channel upon receiving data from the Wi-Fi interface. Additionally, we employ lightweight Internet Protocol a compact, standalone implementation of the TCP/IP protocol tailored for low-power devices, consuming a mere 40 KB of memory.

The software system of Synergy is implemented on a FreeRTOS with C language. We abstract sensor reading, inference, and networking functionality as individual tasks, and each task is scheduled on top of the FreeRTOS scheduler. Once a joint plan is selected based on the target metric, the model is split to allow parts of the model to run on different AI accelerators. To this end, we *synthesize* the partial model for a given range of layers for a designated AI accelerator. Synthesizing is the process of generating device-specific code from pre-trained models. This involves analyzing the pre-trained model and mapping inputs, weights, and outputs to memory and processor appropriately. In this study, we generate C codes from pre-trained PyTorch models for deployment on MAX78000 and MAX78002 devices.

VI. Evaluation

A. Experimental Setup

1) Workload: To demonstrate the effectiveness of Synergy orchestrating and executing on-body AI apps, we design four

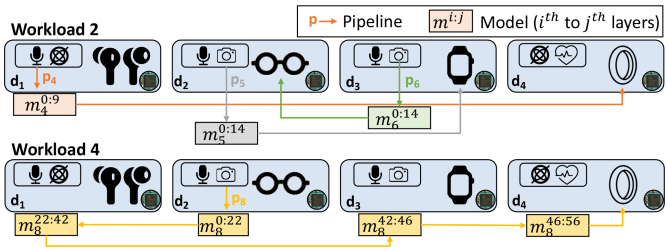

Fig. 14. Example execution plans (workload 2, 4).

workloads with eight pipelines and eight different models (see Table I). For the workload design, we considered two scenarios: concurrent apps, each using a different model (type, size) for different requirements (Workload 1 and 2) and a single app with a large model for enhanced capability (Workload 3 and 4). For the device setup, we used four MAX78000 devices, representing four smart wearables; an earbud, glasses, a watch, and a ring. We then assign source and target devices and a model based on the app scenarios. Fig. 14 shows example execution plans of Workload 2 and 4. Pipeline 4 (keyword spotting), captures audio from an earbud (d_1), runs KWS [35], and sends results to a ring (d_4). Similarly, in Workload 4, pipeline 8 (object detector) captures images on glasses (d_2), runs MobileNetV2 [36] across devices, and sends results to a ring (d_4). For each pipeline, various execution plans are generated by leveraging the four AI accelerators, including splitting models at different layers and assigning chunks to different devices.

2) Baselines: To the best of our knowledge, there are no existing studies that address model splitting in multi-pipeline scenarios across multiple devices. Therefore, we devised several baselines based on different rationales and adapted state-of-the-art splitting algorithms to evaluate their effectiveness in our environments. We consider 7 baselines. The first 4 are heuristic baselines that consider the resource usage of other pipelines when selecting the holistic collaboration plan. The last 3 are based on state-of-the-art algorithms.

MinDev: This heuristic aims to avoid model splitting across devices as much as possible. The rationale is that using fewer devices would reduce communication overhead between devices, thereby increasing throughput. For each pipeline, it selects an execution plan that uses the minimum number of devices to run its model, considering the resource usage of previously selected plans, similar to Synergy.

MaxDev: In contrast, MaxDev focuses on maximizing model splitting over distributed AI accelerators, with the rationale that more devices could enhance task parallelization, thereby improving throughput. Unlike MinDev, MaxDev selects an execution plan that splits the model to all available devices.

PriMinDev: Prioritized MinDev (PriMinDev) enhances MinDev by prioritizing splitting points and device assignment order. For each pipeline, it selects an execution plan that minimizes intermediate output sizes from devices, while using the fewest possible devices. When selecting the device, it prioritizes MAX78002 over MAX78000 to reduce splitting.

PriMaxDev: PriMaxDev is the same as PriMinDev except that it considers execution plans that involve all devices.

IndModel: State-of-the-art model partitioning methods [10], [11], [12], [13], [14] primarily determine the optimal split execution plan based on metric estimations. However, these methods are designed for single-model optimization and do not directly extend to our multi-model scenario. We categorize them under IndModel and adapt them for our use case. Specifically, IndModel selects the best-split execution plan for each pipeline independently, without a holistic view. The final collaborative execution plan is then formed by aggregating these independently selected plans.

JointModel: IndModel may lead to out-of-resource (OOR) errors if the cumulative plan exceeds available resources. To prevent this, JointModel, a multi-tenant version of IndModel, conducts a joint resource assessment similar to Synergy.

IndE2E: IndE2E advances IndModel by incorporating source and target devices alongside model splitting. Each pipeline independently selects the execution plan expected to yield the highest throughput by considering end-to-end (E2E) latency, from sensing to output delivery. However, it does not account for the resource usage of other pipelines.

3) Performance Metrics: We consider three performance metrics: throughput, latency, and energy. Throughput is defined as the total number of model inferences per second, while latency is the time taken to execute an end-to-end holistic collaboration plan, respectively. For energy, we report the average power consumption for the execution of a holistic collaboration plan. We measured power consumption with Monsoon power monitor [40].

B. Overall Performance

Fig. 15 shows the result of different methods for the four workloads. In terms of throughput, Synergy consistently outperforms all baselines owing to its holistic decision-making process and adaptive task parallelization. Synergy on average shows 23.0× higher throughput than the baselines. In Workload 1 and 2 where multiple pipelines run concurrently, IndModel (state-of-the-art model partitioning methods) results in OOR situations. In Workload 3 and 4, Synergy continues to exhibit 1.8× and 2.2× higher throughput than the second best (IndE2E) respectively. The results indicate two key points. First, Synergy boosts throughput without OOR failures when multiple pipelines exist (Workload 1 and 2). Second, Synergy supports large models through the collaboration of distributed AI accelerators (Workload 3 and 4).

For the other metrics, we first found that Synergy reduces latency significantly by 73.9% on average, compared to the baselines. This is because maximizing throughput is tightly related to reduced latency; to maximize the throughput, the pipeline should be streamlined across multiple processors, which in turn reduces the end-to-end latency. Interestingly, we found that Synergy reduces power consumption by 15.8% on average, despite its significant throughput improvement of 23.0× that naturally entails more use of computation units. This is because the major source of power consumption is data transmission between devices,

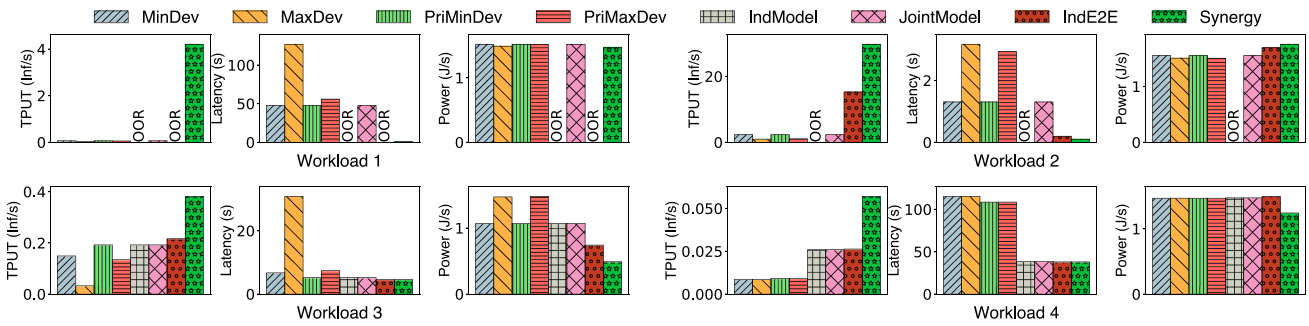

Fig. 15. Throughput, latency, and power consumption result for four different workloads.

TABLE II
ABLATION STUDY OF SYNERGY

JRC	STT	PSR	ATP	Workload 1			Workload 2		
				TPUT (inf/s)	Latency (s)	Power (J/s)	TPUT (inf/s)	Latency (s)	Power (J/s)
				OOR	OOR	OOR	OOR	OOR	OOR
✓				0.06	47.63	1.52	2.30	1.31	**1.55**
✓	✓			0.92	3.25	1.59	15.28	0.20	1.69
✓	✓	✓		2.72	1.10	1.61	15.28	0.20	1.69
✓	✓	✓	✓	**4.20**	**0.86**	**1.47**	**29.67**	**0.10**	1.75

The bold values represent the best performance for each performance metric. For TPUT, higher is better. For Latency and Power, lower is better.

and maximizing throughput in Synergy naturally minimizes data communication cost.

1) Ablation Study: We perform an ablative study to understand the effectiveness of each technical component of Synergy. Table II shows the result. JRC refers to the joint resource consideration among pipelines, and STT refers to device mapping accounting for source and target tasks (Section IV-C). PSR means progressive search space reduction (Section IV-D), and ATP means adaptive task parallelization (Section IV-F). If none of these are applied, it is the same as the state-of-the-art approach (IndModel). If all are checked, it becomes Synergy. Overall, the throughput and latency improve as each of our components is added. Improving IndModel by addressing resource conflicts among pipelines to avoid OOR (JRC) is the same as JointModel in our baselines. Note that merely resolving the resource conflicts is still far from Synergy's performance. Incorporating other technical components, STT, PSR, and ATP shows 6.9, 7.6, and 14.3× higher throughput compared to JRC-only, respectively. Similarly, the latency was reduced by 93.0%, 97.3%, and 98.0% compared to JRC-only, owing to the parallelization of the pipelines. We also noted that utilizing all components increased power consumption by around 12.9% compared to JRC-only, due to the increased use of computation units. These findings are aligned with the earlier analysis of the overall performance.

C. In-Depth Analysis

1) Runtime Environment Changes: We investigate how Synergy adapts to changes in the runtime environment.

Number of devices: In this experiment, we explore Synergy's scalability to the change of the number of MAX78000 devices. We increase the number of devices from two to five while running the same set of four pipelines with ConvNet5, KWS, SimpleNet, and ResSimpleNet. Fig. 16(a) shows the result. Interestingly, Synergy significantly outperforms the baselines as the number of devices increases. This is due to Synergy's strategic consideration of accelerator assignments, effectively minimizing communication overhead between source and target devices. Conversely, except for IndE2E, the throughput gains for all other baselines are not notable, even with additional devices. Another interesting observation is that using more devices does not always lead to higher throughput. In the case of Synergy, the throughput saturates once the number of devices reaches 4. This is because, beyond optimal distribution, further splitting of models fails to contribute to additional throughput gains.

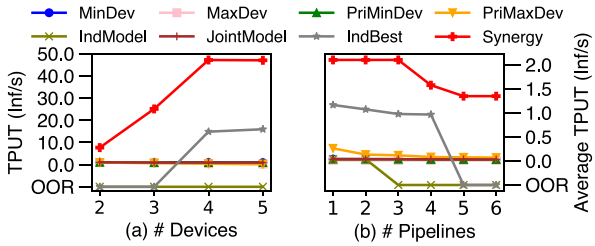

Fig. 16. Impact of runtime environment changes: (a) number of devices and (b) number of pipelines.

Number of pipelines: We assessed Synergy's performance with a varying number of pipelines, incrementally increasing from one to six: UNet, ConvNet5, SimpleNet, KWS, ResSimpleNet, and WideNet, using four MAX78000 s. To understand resource competition, we report the *average* throughput across pipelines, i.e., the ratio of completed pipelines per second to the total number of pipelines. Fig. 16(b) shows a downward

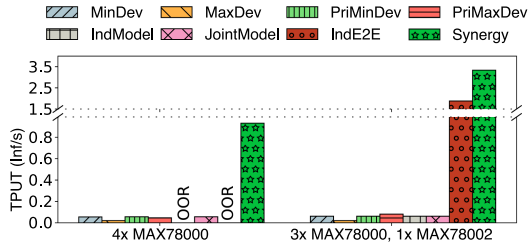

Fig. 17. Effect of accelerator composition.

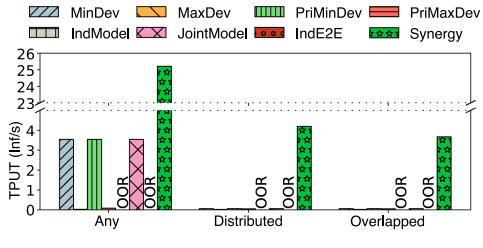

Fig. 18. Effect of source and target mappings.

TABLE III
COMPARISON AMONG DIFFERENT OBJECTIVES

Workload 1	TPUT (inf/s)	Latency (s)	Power (J/s)
TPUT-max	**4.20**	**0.86**	1.47
Latency-min	3.15	**0.86**	1.42
Power-min	0.19	27.17	**1.22**

Workload 2	TPUT (inf/s)	Latency (s)	Power (J/s)
TPUT-max	**29.67**	**0.10**	1.37
Latency-min	**29.67**	**0.10**	3.21
Power-min	1.37	3.21	**1.06**

The bold values represent the best performance for each performance metric. For TPUT, higher is better. For Latency and Power, lower is better.

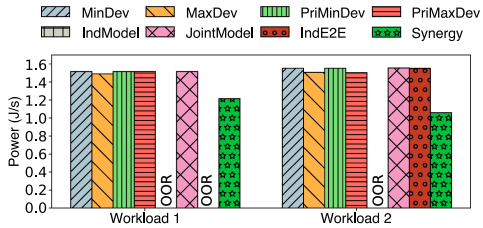

Fig. 19. Effect of the objective: minimizing power consumption.

trend in average throughput as the number of pipelines increases for all baselines due to competition for resources. Nonetheless, Synergy consistently outperforms the baselines, achieving an average throughput of 1.35 with six pipelines, 19.4× higher than the second best (PriMaxDev).

2) Composing Heterogeneous Accelerators: To examine Synergy's effectiveness in heterogeneous accelerator resources, we conducted an experiment where one of four MAX78000 devices was substituted with more resource-capable MAX78002. The workload is comprised of three pipelines: ConvNet5, UNet, and EfficientNet. Fig. 17 shows the results in two different setups. The inclusion of a higher-resource device generally led to an improvement in throughput. With four MAX78000 s, Synergy achieves a total throughput of 0.93, outperforming the second best (JointModel) by a factor of 16.4. This throughput further increases to 3.33 when incorporating one MAX78002. PriMinDev, which allocates all models exclusively to the single MAX78002, results in a significantly low throughput of 0.06. This discrepancy demonstrates why simply offloading model execution to a more powerful device (MAX78002) is not always the best approach. It also underscores the need to account for communication overhead between source and target devices. Additionally, we observed that while IndE2E results in an OOR condition with four MAX78000 s, it achieves the second-highest throughput when using the MAX78002. This suggests that IndE2E performs well in resource-rich environments but fails to manage resources carefully in resource-constrained settings.

3) Source and Target Mapping: We assessed source and target mapping effects in three scenarios: (i) any device can be source or target (Any), (ii) source/target devices are evenly allocated (Distributed), and (iii) the same device is used as both source and target across pipelines (Overlapped). The Distributed setup matches Workload 1 from Section VI-B, while the Overlapped and Any scenarios only differ in their source and target device mappings. As shown in Fig. 18, the Overlapped scenario has the lowest throughput due to communication bottlenecks, while the Any scenario has the highest throughput by distributing communication costs. The result indicates Synergy's robustness in execution planning, considering both source and target factors.

4) Different Objectives: While Synergy defaults to maximizing system-wide throughput (TPUT-max), it can also minimize latency (Latency-min) or power consumption (Power-min). Table III demonstrates Synergy's ability to achieve these different objectives effectively. Each objective excels in its metric, though identical latencies do not ensure the same holistic collaboration plan. Notably, maximizing throughput provides a balanced performance with low latency and near-minimal power use. For example, in workload 1, TPUT-max achieves 22.1× higher throughput with only 1.2× more power consumption compared to Power-min. This is because maximizing throughput reduces latency and communication overhead, the primary power consumption source.

We further investigate the effectiveness of Synergy's orchestration by taking Power-min as an example. Fig. 19 shows the average power consumption for Workload 1 and Workload 2 when Power-min is set as an objective. The baselines also select the execution plan, prioritizing the minimization of power consumption. The results show that Synergy executes multiple workloads while minimizing power consumption and avoiding OOR.

VII. RELATED WORK

TinyML: TinyML represents a research field in machine learning techniques, aiming to bring AI capabilities to the most resource-constrained devices, such as MCUs. These devices typically have tens to hundreds of kilobytes of SRAM, and most of the research efforts in this domain have focused on minimizing model size. Existing studies have explored mainly

three techniques: model pruning [41], model quantization [42], and neural architecture search (NAS) [43], [44]. Model pruning [41] reduces model complexity by identifying and eliminating unnecessary parameters that contribute minimally to output accuracy. Model quantization [42] decreases both model size and computational intensity of operations by reducing the precision of the numerical values used to represent a model's parameters. NAS [43], [44] is an automated method for discovering the best neural network architectures that are tailored to the resource constraints of a specific device, while balancing efficiency and accuracy. In contrast, our work focuses on supporting multiple or large AI models without compromising accuracy, by dynamically composing distributed AI accelerators. However, the combined power of tiny AI accelerators may still be insufficient to support unmodified, off-the-shelf large AI models. We envision that Synergy can benefit from these TinyML techniques to accommodate larger AI models.

Model partitioning: While there are very few attempts for partitioning AI models over multiple MCU-equipped devices, there have been active research efforts for layer-wise model partitioning over resource-limited embedded and mobile devices. In common, these methods [10], [11], [12], [13], [14], [15], [16] allocates few initial layers of a DNN on mobile/embedded device and the latter in edge or cloud server. Intermediate output from the initial layer execution is transmitted to powerful resources such as the cloud or nearby edge device where the subsequent part of the model is executed. They adapt the splitting layer depending on network status and server load. While Synergy also adopts vertical partitioning for distributed inferences, existing model partitioning studies lack the (i) end-to-end perspective of the on-body AI app execution (sensor and interface operations as well as model execution) and (ii) consideration of concurrent execution of multi-tenant models. Our experiments demonstrate that Synergy outperforms these studies by solving the challenges.

DNN workload distribution: An application pipeline often consists of a (conditional) sequence of multiple DNN models. Several distributed systems have been proposed to balance model workloads across distributed devices [45], [46], [47]. Their key difference with model partitioning methods is to treat a model as a primitive execution unit and focus more on scheduling *model* execution over distributed devices (without model splitting). For example, a face classification task can be executed on a different device following object detection on the initial device, thereby leveraging distributed resources from multiple devices [45], [46]. Synergy shares the same high-level objective of distributing multiple model workloads into multiple devices. However, our target environment is on-body devices with AI accelerators, which brings resource challenges. To overcome the limited resources of these devices, we adopt model partitioning on top of workload distribution and devise a tailored solution for tiny AI accelerators.

Model serving systems: Several platforms, such as TensorFlow serving [48], Sagemaker [49], and Azure ML [50], have been proposed to facilitate model inference serving by offering containerized environments for model execution in diverse devices. Research platforms, such as Velox [51] and Clipper [52], focus on low-latency prediction serving, together with optimizing cloud server performance. However, in wearable computing, the main challenge lies in dynamically composing distributed tiny AI accelerators. Our *device-agnostic programming interface* addresses this by mapping software logic to physical resources and optimizing performance by considering pipeline interdependencies and resource usage.

Middleware for body sensor networks: Several decades ago, body sensor networks (BSNs) received attention for recognizing human contexts using sensory signals from multiple wearable devices. Early work focused on fusion techniques to effectively and robustly concatenate multiple data streams [53], [54], [55], [56], [57] when different devices introduced several challenges, e.g., time synchronization, missing data, and varying sampling rates. Another focus was handling dynamic device availability [58], [59], [60], [61], [62], i.e., when the set of available devices changes over time. To tackle this problem, many studies have explored the effects of different characteristics on recognition accuracy, e.g., types, compositions, and placements of sensors, and have proposed methods to dynamically select the best sensor based on predefined parameters.

Although these works provide a foundation for sensing and processing in multi-device environments, their consideration of sensing pipelines and target devices has been limited to support on-body AI apps we envision. First, Sensing pipelines in BSNs was simple, so pipeline partitioning across devices was not actively studied. Second, due to insufficient processing capabilities, wearable devices were primarily treated as sensor data streamers, with most processing assumed to be run on a smartphone. However, we argue that on-body sensing pipelines have evolved with the introduction of numerous DNN-based AI models, and wearable devices have begun to feature on-device AI capabilities, with the emergence of tiny AI accelerators. This insight necessitates revisiting existing strategies for on-body sensing and processing in multi-wearable environments. To the best of our knowledge, our work is the first to efficiently support on-body AI applications on wearable devices equipped with tiny AI accelerators.

VIII. Conclusion

We presented Synergy, a novel system that supports on-body AI apps through the collaboration of tiny AI accelerators on wearables. Synergy's device-agnostic programming interface simplifies integrating diverse AI applications. Its runtime dynamically distributes model execution tasks to available device resources for optimal inference. Our evaluation showed Synergy achieves higher throughput than baselines.

Future work aims to enhance collaboration among AI accelerators by expanding from layer-wise model splitting to include channel-wise splitting [63], [64], [65], [66], [67], [68], enabling Synergy to handle larger inputs and optimize performance through a combination of both techniques [69]. Additionally, we plan to extend Synergy to support more complex model architectures, such as sensor fusion and conditional chaining of multiple models, to meet real-world application demands. Lastly, we aim to enrich the device-agnostic programming interface by adding

features like detailed taxonomy for sensor and target device mapping and supporting customized functions, thereby providing developers with greater flexibility in designing on-body AI apps.

REFERENCES

[1] Analog MAX78000. Accessed: Nov. 30, 2023. [Online]. Available: https://www.analog.com/en/products/max78000.html

[2] Analog MAX78002. Accessed: Nov. 30, 2023. [Online]. Available: https://www.analog.com/en/products/max78002.html

[3] Google Coral Micro. Accessed: Nov. 30, 2023. [Online]. Available: https://coral.ai/products/dev-board-micro/

[4] Greenwaves Technology. Accessed: Nov. 30, 2023. [Online]. Available: https://greenwaves-technologies.com/low-power-processor/

[5] A. N. Balaji and L.-S. Peh, "AI-on-skin: Towards enabling fast and scalable on-body AI inference for wearable on-skin interfaces," in *Proc. ACM Hum.-Comput. Interaction*, vol. 7, no. EICS, pp. 1–34, 2023.

[6] OmniBuds. Accessed: Mar. 15, 2024. [Online]. Available: https://omnibuds.tech/

[7] Shift Moonwalkers. Accessed: Mar. 15, 2024. [Online]. Available: https://shiftrobotics.io/

[8] TWS Processor with GAP9. Accessed: Mar. 15, 2024. [Online]. Available: https://greenwaves-technologies.com/tws_processor/

[9] C. Min et al., "An AI-native runtime for multi-wearable environments," 2024. [Online]. Available: https://arxiv.org/abs/2403.17863

[10] Y. Kang et al., "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," *ACM SIGARCH Comput. Archit. News*, vol. 45, no. 1, pp. 615–629, 2017.

[11] C. Hu, W. Bao, D. Wang, and F. Liu, "Dynamic adaptive DNN surgery for inference acceleration on the edge," in *Proc. IEEE Conf. Comput. Commun.*, 2019, pp. 1423–1431.

[12] S. Zhang et al., "Towards real-time cooperative deep inference over the cloud and edge end devices," in *Proc. ACM Interactive, Mobile, Wearable Ubiquitous Technol.*, vol. 4, no. 2, pp. 1–24, 2020.

[13] 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 *Proc. 26th Annu. Int. Conf. Mobile Comput. Netw.*, 2020, pp. 1–15.

[14] H.-J. Jeong, H.-J. Lee, C. H. Shin, and S.-M. Moon, "IONN: Incremental offloading of neural network computations from mobile devices to edge servers," in *Proc. ACM Symp. Cloud Comput.*, 2018, pp. 401–411.

[15] H. Wang, B. Guo, J. Liu, S. Liu, Y. Wu, and Z. Yu, "Context-aware adaptive surgery: A fast and effective framework for adaptative model partition," in *Proc. ACM Interactive, Mobile, Wearable Ubiquitous Technol.*, vol. 5, no. 3, pp. 1–22, 2021.

[16] A. Banitalebi-Dehkordi, N. Vedula, J. Pei, F. Xia, L. Wang, and Y. Zhang, "Auto-split: A general framework of collaborative edge-cloud AI," in *Proc. 27th ACM SIGKDD Conf. Knowl. Discov. Data Mining*, 2021, pp. 2543–2553.

[17] J. Moosmann, M. Giordano, C. Vogt, and M. Magno, "TinyissimoYOLO: A quantized, low-memory footprint, TinyML object detection network for low power microcontrollers," in *Proc. IEEE 5th Int. Conf. Artif. Intell. Circuits Syst.*, 2023, pp. 1–5.

[18] T. Rüegg, M. Giordano, and M. Magno, "KP2Dtiny: Quantized neural keypoint detection and description on the edge," in *Proc. IEEE 5th Int. Conf. Artif. Intell. Circuits Syst.*, 2023, pp. 1–5.

[19] A. Bakar et al., "Protean: An energy-efficient and heterogeneous platform for adaptive and hardware-accelerated battery-free computing," in *Proc. 20th ACM Conf. Embedded Netw. Sensor Syst.*, 2022, pp. 207–221.

[20] L. Caronti, K. Akhunov, M. Nardello, K. S. Yıldırım, and D. Brunelli, "Fine-grained hardware acceleration for efficient batteryless intermittent inference on the edge," *ACM Trans. Embedded Comput. Syst.*, vol. 22, no. 5, pp. 1–19, 2023.

[21] A. Moss, H. Lee, L. Xun, C. Min, F. Kawsar, and A. Montanari, "Ultra-low power DNN accelerators for IoT: Resource characterization of the max78000," in *Proc. 20th ACM Conf. Embedded Netw. Sensor Syst.*, 2022, pp. 934–940.

[22] T. Gong, F. Kawsar, and C. Min, "DEX: Data channel extension for efficient CNN inference on tiny AI accelerators," in *Proc. 38th Annu. Conf. Neural Inf. Process. Syst.*, 2024, pp. 43925–43951.

[23] C. Jeon, T. Gong, J. Yi, F. Kawsar, and C. Min, "TinyMem: Boosting multi-DNN inference on tiny AI accelerators with weight memory virtualization," in *Proc. 26th ACM Int. Workshop Mobile Comput. Syst. Appl.*, New York, NY, USA, 2025, pp. 1–6, doi: 10.1145/3708468.3711888.

[24] S. Jang, F. Kawsar, and C. Min, "Thermal characterization of AI applications on AI accelerators-equipped microcontrollers," in *Proc. ACM Workshop Body-Centric Comput. Syst.*, New York, NY, USA, 2024, pp. 17–22, doi: 10.1145/3662009.3662020.

[25] Y. Huang, T. Gong, S. Jang, F. Kawsar, and C. Min, "Energy characterization of tiny AI accelerator-equipped microcontrollers," in *Proc. 2nd ACM Int. Workshop Hum.-Centered Sens. Netw. Multi-Device Syst.*, New York, NY, USA, 2024, pp. 1–6, doi: 10.1145/3698388.3699628.

[26] Cutting the AI Power Cord: Technology to Enable True Edge Inference. Accessed: Nov. 30, 2023. [Online]. Available: https://cms.tinyml.org/wp-content/uploads/talks2020/tinyML_Talks_Kris_Ardis_and_Robert_Muchsel_-201027.pdf

[27] Analog MAX32650. Accessed: Nov. 30, 2023. [Online]. Available: https://www.analog.com/en/products/max32650.html

[28] STM32F7 Series. Accessed: Nov. 30, 2023. [Online]. Available: https://www.st.com/en/microcontrollers-microprocessors/stm32f7-series.html

[29] A. M. Das, C. I. Tang, F. Kawsar, and M. Malekzadeh, "PRIMUS: Pretraining IMU encoders with multimodal self-supervision," in *Proc. NeurIPS Workshop Time Ser. Age Large Models*, 2024.

[30] A. Pillai, D. Spathis, F. Kawsar, and M. Malekzadeh, "PaPaGei: Open foundation models for optical physiological signals," in *Proc. 13th Int. Conf. Learn. Representations*, 2025. [Online]. Available: https://openreview.net/forum?id=kYwTmlq6Vn

[31] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional networks for biomedical image segmentation," in *Proc. 18th Int. Conf. Med. Image Comput. Comput.-Assist. Intervention*, Munich, Germany, Springer, Oct. 5–9, 2015, pp. 234–241.

[32] Analog MAX78000FTHR. Accessed: Nov. 30, 2023. [Online]. Available: https://www.analog.com/en/design-center/evaluation-hardware-and-software/evaluation-boards-kits/max78000fthr.html

[33] Analog MAX78002EVKIT. Accessed: Nov. 30, 2023. [Online]. Available: https://www.analog.com/en/design-center/evaluation-hardware-and-software/evaluation-boards-kits/max78002evkit.html

[34] Adafruit HUZZAH ESP8266. Accessed: Nov. 30, 2023. [Online]. Available: https://www.adafruit.com/product/2821

[35] Analog Keywords Spotting. Accessed: Nov. 30, 2023. [Online]. Available: https://www.analog.com/en/design-notes/keywords-spotting-using-the-max78000.html

[36] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 4510–4520.

[37] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.

[38] S. H. Hasanpour, M. Rouhani, M. Fayyaz, and M. Sabokrou, "Lets keep it simple, using simple architectures to outperform deeper and more complex architectures," 2016, *arXiv:1608.06037*.

[39] M. Tan and Q. Le, "EfficientNetV2: Smaller models and faster training," in *Proc. Int. Conf. Mach. Learn.*, PMLR, 2021, pp. 10 096–10 106.

[40] Monsoon solutions. Accessed: Mar. 15, 2024. [Online]. Available: https://www.msoon.com/high-voltage-power-monitor

[41] E. Liberis and N. D. Lane, "Differentiable neural network pruning to enable smart applications on microcontrollers," in *Proc. ACM Interactive, Mobile, Wearable Ubiquitous Technol.*, vol. 6, no. 4, pp. 1–19, 2023.

[42] M. Rusci, A. Capotondi, and L. Benini, "Memory-driven mixed low precision quantization for enabling deep network inference on microcontrollers," in *Proc. Mach. Learn. Syst.*, vol. 2, pp. 326–335, 2020.

[43] E. Liberis, Ł. Dudziak, and N. D. Lane, "μNAS: Constrained neural architecture search for microcontrollers," in *Proc. 1st Workshop Mach. Learn. Syst.*, 2021, pp. 70–79.

[44] I. Fedorov, R. P. Adams, M. Mattina, and P. Whatmough, "Sparse: Sparse architecture search for CNNs on resource-constrained microcontrollers," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 4977–4989.

[45] X. Zeng, B. Fang, H. Shen, and M. Zhang, "Distream: Scaling live video analytics with workload-adaptive distributed edge intelligence," in *Proc. 18th Conf. Embedded Netw. Sensor Syst.*, 2020, pp. 409–421.

[46] S. Y. Jang, B. Kostadinov, and D. Lee, "Microservice-based edge device architecture for video analytics," in *Proc. IEEE/ACM Symp. Edge Comput.*, 2021, pp. 165–177.

[47] Z. Dong, Y. Lu, G. Tong, Y. Shu, S. Wang, and W. Shi, "Watchdog: Real-time vehicle tracking on geo-distributed edge nodes," *ACM Trans. Internet Things*, vol. 4, no. 1, pp. 1–23, 2023.

[48] Google. Accessed: Nov. 30, 2023. [Online]. Available: https://www.tensorflow.org/tfx

[49] Amazon SageMaker. Accessed: Nov. 30, 2023. [Online]. Available: https://aws.amazon.com/sagemaker/?nc1=h_ls
[50] Azure Machine Learning. Accessed: Nov. 30, 2023. [Online]. Available: https://azure.microsoft.com/en-gb/products/machine-learning
[51] D. Crankshaw et al., "The missing piece in complex analytics: Low latency, scalable model management and serving with velox," 2014, *arXiv:1409.3809*.
[52] D. Crankshaw, X. Wang, G. Zhou, M. J. Franklin, J. E. Gonzalez, and I. Stoica, "Clipper: A low-latency online prediction serving system," in *Proc. 14th USENIX Symp. Netw. Syst. Des. Implementation*, 2017, pp. 613–627.
[53] F. J. Ordóñez and D. Roggen, "Deep convolutional and LSTM recurrent neural networks for multimodal wearable activity recognition," *Sensors*, vol. 16, no. 1, 2016, Art. no. 115. [Online]. Available: https://www.mdpi.com/1424-8220/16/1/115
[54] L. Peng, L. Chen, Z. Ye, and Y. Zhang, "Aroma: A deep multi-task learning based simple and complex human activity recognition method using wearable sensors," in *Proc. ACM Interactive, Mobile, Wearable Ubiquitous Technol.*, vol. 2, no. 2, pp. 74:1–74:16, Jul. 2018, doi: 10.1145/3214277.
[55] S. Yao, S. Hu, Y. Zhao, A. Zhang, and T. Abdelzaher, "DeepSense: A unified deep learning framework for time-series mobile sensing data processing," in *Proc. 26th Int. Conf. World Wide Web*, 2017, pp. 351–360, doi: 10.1145/3038912.3052577.
[56] S. Yao, Y. Zhao, S. Hu, and T. Abdelzaher, "QualityDeepSense: Quality-aware deep learning framework for Internet of Things applications with sensor-temporal attention," in *Proc. 2nd ACM Int. Workshop Embedded Mobile Deep Learn.*, New York, NY, USA, 2018, pp. 42–47, doi: 10.1145/3212725.3212729.
[57] Y. Vaizman, N. Weibel, and G. Lanckriet, "Context recognition in-the-wild: Unified model for multi-modal sensors and multi-label classification," in *Proc. ACM Interactive, Mobile, Wearable Ubiquitous Technol.*, vol. 1, no. 4, pp. 168:1–168:22, Jan. 2018, doi: 10.1145/3161192.
[58] P. Zappi et al., "Activity recognition from on-body sensors: Accuracy-power trade-off by dynamic sensor selection," in *Wireless Sensor Networks*, R. Verdone, Ed. Berlin, Heidelberg: Springer, 2008, pp. 17–33.
[59] S. Kang et al., "SeeMon: Scalable and energy-efficient context monitoring framework for sensor-rich mobile environments," in *Proc. 6th ACM Int. Conf. Mobile Syst., Appl., Serv.*, New York, NY, USA, 2008, pp. 267–280, doi: 10.1145/1378600.1378630.
[60] S. Kang et al., "Orchestrator: An active resource orchestration framework for mobile context monitoring in sensor-rich mobile environments," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun.*, 2010, pp. 135–144.
[61] Y. Lee, C. Min, Y. Ju, S. Kang, Y. Rhee, and J. Song, "An active resource orchestration framework for pan-scale, sensor-rich environments," *IEEE Trans. Mobile Comput.*, vol. 13, no. 3, pp. 596–610, Mar. 2014.
[62] M. Keally, G. Zhou, G. Xing, J. Wu, and A. Pyles, "PBN: Towards practical activity recognition using smartphone-based body sensor networks," in *Proc. 9th ACM Conf. Embedded Netw. Sensor Syst.*, New York, NY, USA, 2011, pp. 246–259, doi: 10.1145/2070942.2070968.
[63] J. Mao, X. Chen, K. W. Nixon, C. Krieger, and Y. Chen, "MoDNN: Local distributed mobile computing system for deep neural network," in *Proc. IEEE Des. Automat. Test Europe Conf. Exhib.*, 2017, pp. 1396–1401.
[64] J. Mao et al., "MeDNN: A distributed mobile system with enhanced partition and deployment for large-scale DNNs," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, 2017, pp. 751–756.
[65] L. Zeng, X. Chen, Z. Zhou, L. Yang, and J. Zhang, "CoEdge: Cooperative DNN inference with adaptive workload partitioning over heterogeneous edge devices," *IEEE/ACM Trans. Netw.*, vol. 29, no. 2, pp. 595–608, Apr. 2021.
[66] Z. Zhao, K. M. Barijough, and A. Gerstlauer, "DeepThings: Distributed adaptive deep learning inference on resource-constrained IoT edge clusters," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 37, no. 11, pp. 2348–2359, Nov. 2018.
[67] X. Hou, Y. Guan, T. Han, and N. Zhang, "DistrEdge: Speeding up convolutional neural network inference on distributed edge devices," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, 2022, pp. 1097–1107.
[68] C. Hu and B. Li, "Distributed inference with deep learning models across heterogeneous edge devices," in *Proc. IEEE Conf. Comput. Commun.*, 2022, pp. 330–339.
[69] B. Pang et al., "AdaMEC: Towards a context-adaptive and dynamically combinable DNN deployment framework for mobile edge computing," *ACM Trans. Sensor Netw.*, vol. 20, no. 1, pp. 1–28, 2023.

Taesik Gong (Member, IEEE) received the PhD degree in computer science from KAIST in 2023. He is an assistant professor with the Department of Computer Science and Engineering at UNIST. During the PhD degree, he interned with Google Research, Microsoft Research, and Nokia Bell Labs. Before joining UNIST, he was a research scientist with Nokia Bell Labs and a visiting scholar with the University of Cambridge. He is also a recipient of the Google PhD Fellowship. His research interests include on-device AI, human-centered AI, and ubiquitous computing.

SiYoung Jang (Member, IEEE) received the PhD degree from the School of Computing, KAIST, South Korea. He is a research scientist with Nokia Bell Labs, Cambridge U.K. His research interests lie broadly in designing efficient and scalable machine learning solutions for real-world applications, with a focus on mobile computing, on-device AI, and networked edge AI systems.

Utku Günay Acer received the PhD degree from the Electrical, Computer and Systems Engineering department of Rensselaer Polytechnic Institute, Troy, NY. He is a principal research scientist with Nokia Bell Labs in Antwerp, Belgium. His research interests lie broadly in pervasive systems and edge computing. His current work focuses on collaborative and distributed sensing with on-device ML inference for next-generation mobile, wearable, and embedded devices.

Fahim Kawsar (Member, IEEE) currently leads Pervasive Systems Research with Nokia Bell Labs, Cambridge. He holds a mobile systems professorship in computing science from the University of Glasgow. He studies the forms and intelligence of emerging mobile, IoT, and wearable devices.

Chulhong Min (Member, IEEE) received the PhD degree in computer science from KAIST in 2016. He is a principal research scientist leading the Device Systems team, Nokia Bell Labs in Cambridge, U.K. His current research explores next-generation sensory systems to realize multi-modal, multi-device, and multi-sensory functionalities for collaborative and interactive services. Broadly, his research interests include mobile systems, edge computing, on-device AI, and IoT.