

Collaborative Inference via Dynamic Composition of Tiny AI Accelerators on MCUs

Taesik Gong*
Nokia Bell Labs
Cambridge, UK
taesik.gong@nokia-bell-labs.com

Si Young Jang*
Nokia Bell Labs
Cambridge, UK
siyoung.jang@nokia-bell-labs.com

Utku Günay Acer
Nokia Bell Labs
Antwerp, Belgium
utku_gunay.acer@nokia-bell-labs.com

Fahim Kawsar
Nokia Bell Labs
Cambridge, UK
fahim.kawsar@nokia-bell-labs.com

Chulhong Min
Nokia Bell Labs
Cambridge, UK
chulhong.min@nokia-bell-labs.com

ABSTRACT

The advent of tiny AI accelerators opens opportunities for deep neural network deployment at the extreme edge, offering reduced latency, lower power cost, and improved privacy in on-device ML inference. Despite these advancements, challenges persist due to inherent limitations of these accelerators, such as restricted onboard memory and single-device focus. This paper introduces *Synergy*, a system that dynamically composes tiny AI accelerators for multi-tenant models, effectively addressing tinyML’s critical challenges for the increasing demand for on-device AI. A key feature of *Synergy* is its *virtual computing space*, providing a unified, virtualized view of resources and enabling efficient task mapping to physical devices. *Synergy*’s runtime orchestration module ensures optimal inference across dynamic and heterogeneous accelerators. Our evaluations with 7 baselines and 8 models demonstrate that *Synergy* improves throughput by an average of 8.0× compared to baselines.

1 INTRODUCTION

The advent of tiny artificial intelligence (AI) accelerators, such as the Analog MAX78000 [23], MAX78002 [26] and Google Coral Micro [2], has brought deep neural network (DNN) model inference closer to us than ever before. These accelerators, designed for microcontrollers (MCUs) with small form factors (e.g., MAX78000: 8mm×8mm), push the boundaries of what tiny, resource-constrained devices can achieve. Instead of relying on external resources such as cloud servers, these accelerators enable on-device AI even in small sensor devices themselves, thereby offering reduced latency, lower power consumption, and enhanced privacy. With the increasing prevalence of wearable devices, we can expect a growing number of these tiny AI accelerators to be

*Equal contribution.

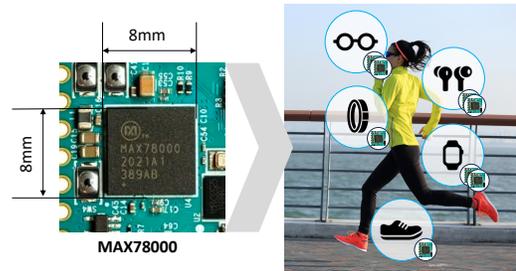


Figure 1: Tiny AI accelerator (MAX78000 [23]) and future wearable computing with these accelerators.

deployed around the human body, leading to *true on-body AI* for various applications, as shown in Figure 1.

Existing research efforts in this area, often referred to as *tinyML*, focus on compressing DNN models for operation on MCUs [8, 18, 19, 32]. However, tinyML faces several fundamental challenges. Due to the constraints of limited onboard memory, AI models must often be compressed to fit, potentially compromising their accuracy significantly. For instance, the pruned VGG (32×32) model that fits into the MCU memory achieves less than 60% top-1 accuracy [1], which is more 30% accuracy drop after compression. Additionally, even when models are specifically tailored for these devices, the number of models that can be supported is inherently limited, as existing tinyML frameworks such as TensorFlow Lite Microcontrollers [39], are generally designed to support only a single model on a single device. While there have been active research efforts on model partitioning [12, 14, 15, 17, 42] to leverage distributed devices, they mostly focus on a simple two-tier architecture (e.g., a mobile and a cloud) for a single model only, and lack the consideration of the unique characteristics of tiny AI accelerators. As wearables become more ubiquitous, there emerges an opportunity for collaboration to dynamically compose

distributed AI accelerators on the body, thereby achieving higher model accuracy and supporting multi-tenant models.

Building such a collaborative system on tiny AI accelerators presents multiple technical challenges. First, dynamically allocating computational tasks among AI accelerators requires consideration of the real-time availability of devices and model dependencies, which are often not predictable at the time of development. Second, it is important to holistically manage all models running concurrently, aiming for optimal system-wide performance due to their inter-dependencies. Third, the heterogeneity of hardware accelerators, each with its own architecture and computational capabilities, complicates the collaboration decision.

In this paper, we present *Synergy*, a system to support multi-tenant AI models through the dynamic composition of tiny AI accelerators on the body. *Synergy*'s key innovation is *virtual computing space*, which offers AI applications with a unified, virtualized view of available resources while abstracting the underlying hardware specifics. This feature allows developers to design application logic without struggling with the dynamic and heterogeneous nature of wearable device environments. For example, a conversation agent application can simply define a pipeline as *<any microphone, KeywordSpotting model, earbud>*, and *Synergy*'s runtime module dynamically distributes model execution across available AI accelerators, matching sensors and destinations to devices based on the requirements.

Synergy incorporates a holistic runtime orchestration module to guarantee the best-effort inference for multiple pipelines over dynamic and heterogeneous AI accelerators. It creates execution plans for each pipeline, mapping logical tasks to available physical devices. *Synergy* also exploits options for model splitting over distributed AI accelerators to support large models or achieve optimal parallel executions. Then, *Synergy* selects the optimal set of execution plans to maximize overall throughput. For holistic orchestration, *Synergy* has two key features: (1) jointly considering resource availability among execution plans and (2) planning end-to-end pipelines with source and target devices. For execution plan selection, we devise a simple yet effective method; based on the observation that data communication is the key bottleneck of the distribution execution, the selection prioritizes data-intensive pipelines, minimizing model splitting.

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 [12, 14, 15, 17, 42] and their adapted version for multi-tenant and end-to-end pipeline. Our extensive evaluation with 8 models demonstrates that *Synergy* consistently outperforms the baselines, with on average 8.0× throughput gain across various scenarios. Furthermore, our in-depth experiments show that *Synergy* effectively adapts to various

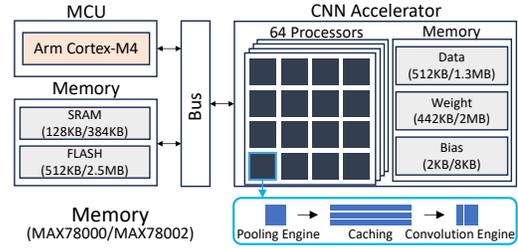


Figure 2: The architecture of MAX78000/MAX78002.

runtime environment changes: (1) the number of devices, (2) the number of pipelines, (3) heterogeneous device resources, and (4) source and target devices.

2 BACKGROUND & MOTIVATION

2.1 Tiny AI Accelerators

In recent years, AI hardware accelerators have transitioned from mobile and embedded devices to MCU devices. These accelerators, distinguished by their tiny form factors and ultra-low power efficiency, such as MAX78000 [23], MAX78002 [26], Google Coral Micro [2], Arm Ethos-U65 [7] and GreenWaves GAP-8/GAP-9 [9], are expected to be incorporated into wearable devices, marking the dawn of *true on-body AI*. The integration of such accelerators into wearables represents a significant move towards distributed, on-device AI that operates independently from cloud computing infrastructure, thereby ensuring enhanced user privacy and minimal latency. Although a number of tiny-scale accelerators have been proposed recently, only a few products are commercially available and provide comprehensive access and control over their underlying operations. In this paper, we have selected the MAX78000/MAX78002, developed by Analog Devices in 2020, as our primary platform. These accelerators are not only readily available but also offer open-source tools and documentation, facilitating in-depth analysis and modification of the internal operations.

The architecture of MAX78000 and MAX78002 are illustrated in Figure 2. These devices comprise two primary processing modules: a dual-core MCU and a convolutional neural network (CNN) accelerator. The MCU features an Arm Cortex-M4 processor operating at a maximum frequency of 100 MHz for MAX78000 and 120 MHz for MAX78002. Both are equipped with a CNN accelerator with 64 convolutional parallel processors, specially designed for running neural networks at ultra-low-power in parallel. Each processor has a pooling engine, input cache, and convolution engine supporting simultaneous processing up to a 3 by 3 kernel.

Benchmark of MAX78000: Recent benchmark study [24] quantitatively illustrates the MAX78000's superior performance in terms of latency and energy consumption. As shown in Figure 3, the MAX78000 significantly outperforms

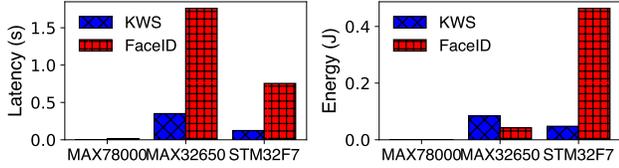


Figure 3: Performance comparison between AI accelerator (MAX78000) and MCUs (MAX32650 and STM32F7).

conventional microcontroller, MAX32650 with Cortex-M4 at 120 MHz [22] and high-performance controller, STM32F7 with Cortex-M7 at 216 MHz [36] in key AI tasks. Latency for keyword spotting (KWS) is reduced to 2.0 ms compared to 350 ms and 123 ms 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.

2.2 Practical Limitations

Limited memory size of AI accelerators: One of the primary limitations of tiny AI accelerators is the constraint imposed by the memory size. For example, MAX78000 and MAX78002 have 442 KB and 2 MB of weight memory, respectively. This limited memory capacity becomes a significant barrier when deploying state-of-the-art neural networks that require large memory footprints. For example, even traditional models such as VGG16 [35] can require up to 100 MB, exceeding the tiny accelerator’s memory capacity by two orders of magnitude. This gap forces to reduce the model size at the expense of accuracy and generalizability of the model. In many cases, to fit NNs into the limited memory of an MCU, the model must be substantially compressed, resulting in significant accuracy loss (e.g., less than 60% top-1 accuracy of VGG (32×32) or over-specialization (e.g., the ability to detect only a few object classes).

Lack of support for multi-model, distributed inference: Another limitation of these accelerators is the lack of system support for multi-tenancy models and collaborative inferences over distributed devices. This implies that today’s tiny AI accelerators do not efficiently handle the execution of multiple AI models concurrently on a single accelerator. Moreover, current tiny ML models do not fully exploit the potential of distributed accelerators. The ability to conduct collaborative inferences, where computational tasks are intelligently divided and processed across multiple accelerators, is hardly explored.

Why not offloading? A possible solution to address these limitations is to offload AI model execution to the cloud or nearby edge devices. However, this approach may not be viable for proactive AI applications, which often require continuous or frequent execution of AI models. Offloading to a cloud server involves transmitting sensitive data, which

raises significant privacy concerns. Additionally, wearables frequently lack cellular connectivity, making this option less feasible. An alternative could be to use smartphones as offloading targets. However, this reliance on smartphones for executing AI models does not ensure seamless service, as continuous connectivity to the smartphone cannot be guaranteed (e.g., exercising without smartphones, battery outages, etc.). Furthermore, continuous transmission of raw sensor data for multiple applications to a smartphone could lead to communication and energy bottlenecks, affecting not only the wearables but also the smartphones.

2.3 Collaboration of Tiny AI Accelerators

To address these challenges, the concept of AI accelerator collaboration emerges as a promising solution. As wearables become increasingly ubiquitous, it is likely that multiple such devices, each equipped with its own AI accelerator, will be worn on the body as illustrated in Figure 1. This scenario opens up the possibility of distributing AI tasks across these accelerators, harnessing their collective computational power. This distributed and collaborative approach has the potential to overcome the memory and processing constraints of individual devices, enabling more complex, accurate, and multi-tenant AI models.

However, realizing effective collaboration among these distributed accelerators brings multiple challenges. First, in a multi-device context, there is inherent variability in the set of available resources. This variability arises from differences in the types and number of wearable devices each user possesses, leading to a dynamic and heterogeneous pool of accelerators. Such diversity requires a highly adaptable system capable of efficiently managing a wide range of on-body computing environments, each with its own computational demand and capabilities. Second, in a multi-model context, the optimal distribution of models across various accelerators is not only a function of the individual model’s requirements but also interdependent across all models. This necessitates a holistic orchestration strategy that can intelligently allocate resources and balance workloads, taking into consideration the collective needs and interactions of all models.

3 SYNERGY DESIGN

3.1 Virtual Computing Space

To address the challenges introduced in §2.3, a significant shift in programming and computing is needed. We propose the creation of a *virtual computing space*, an abstract layer offering a unified, virtualized view of distributed computational resources across wearable devices with AI accelerators. This space hides the heterogeneity and dynamic nature of physical devices, providing a consistent computational environment for AI models and a device-agnostic programming

abstraction (§4.2). AI tasks are deployed in this virtual space, where underlying orchestration maps virtual resources to physical ones based on real-time availability and device capabilities (§5). This ensures efficient resource use, operational continuity despite fluctuating device availability, and seamless scaling as devices join or leave the system.

3.2 System Overview

We propose Synergy, designed to support the best-effort inference of multiple models by dynamically composing tiny AI accelerators over distributed wearable devices. Synergy takes the execution pipeline of an AI model as input and provides model results as output. The system is distinguished by two key features. First, Synergy creates a virtual computing space to facilitate seamless service across dynamic device environments, offering a unified view of distributed resources and providing device-agnostic programming abstractions (§4.2). Second, given pipelines, Synergy explores various task distribution options, including sensor/actuator mapping and model splitting, across available resources. It then strategically maps virtual resources to physical ones, aiming to maximize system-wide performance.

3.3 Design Principles

Device-agnostic pipeline programming: Traditional ML frameworks were mostly designed to run the model on the device where the model is running. However, in a dynamic computing environment, it becomes nearly impossible for developers to specify particular hardware resources to run the model. To address this, Synergy offers *device-agnostic* programming abstraction, decoupling the execution logic from the physical hardware resources.

Support for multi-tenancy: Multi-tenancy is critical for enhancing the utility and adaptability of resource-constrained wearables across diverse applications. To overcome the challenge of limited memory and processing power in MCU-equipped wearables, Synergy constructs a collaborative runtime by dynamically composing distributed AI accelerators and enables the execution of various AI models.

Providing best-effort performance: Synergy is designed to offer best-effort inferences rather than strictly meeting each application’s specific requirements. This approach is essential given the inherent resource limitations of wearable devices, which differs from the cloud computing with abundant resources available. Synergy prioritizes providing the best-effort performance within available resources.

3.4 System Scope

Maximizing overall throughput: To support the execution of multiple models across distributed devices, various system-wide objectives can be considered, such as maximizing the

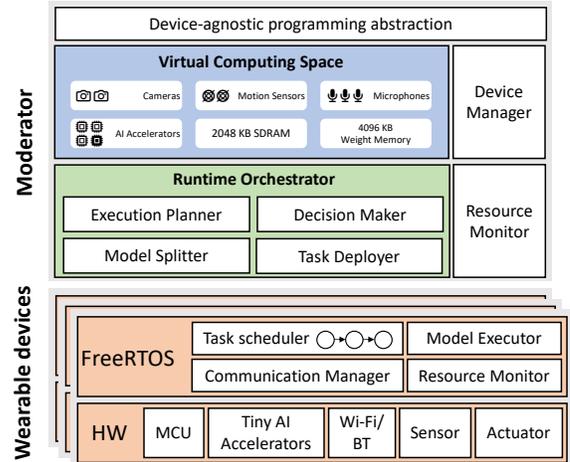


Figure 4: Synergy architecture.

total throughput of models, minimizing the total latency of models, and minimizing the overall energy consumption of devices. It is also possible to balance these metrics for fairness, such as maximizing the minimum throughput of models. In this paper, we focus on maximizing the total throughput of models as it is a widely used quality-of-service (QoS) metric for AI applications. Also, this objective also generally leads to the reduction of the inference latency and energy cost, discussed in §5.3 and §7.2.

Moderator-initiated orchestration: In our current implementation, an external moderator such as a smartphone is needed due to the limited capabilities of MCUs. For instance, MCUs often lack programming capabilities to compile pre-trained models to run on themselves (details in §5.5). The moderator is responsible for discovering and managing resources, making orchestration decisions, segmenting AI models, and deploying them to devices. This process is necessary only when there is a change in the models or device configuration. Once set up, runtime model inference operates independently on wearable devices, without further reliance on the moderator. We envision a shift towards a more decentralized approach, where these capabilities are embedded in powerful wearable devices. Such a development would facilitate self-sufficiency in wearable AI systems, reducing the need for external devices.

4 SYNERGY

4.1 System Architecture

Figure 4 illustrates the architecture of Synergy, which is designed to facilitate the dynamic deployment of models across AI accelerators. The moderator has two key roles: constructing a virtual computing space and performing runtime orchestration. It monitors nearby devices and their resources to create and maintain a unified, abstract view of the distributed

resources. It allows model execution logic to be specified in a format of a pipeline, without mapping physical resources through device-agnostic programming abstractions. The runtime orchestrator is in charge of making distribution decisions dynamically for the registered pipelines. It assesses the available physical resources, generates potential distribution candidates (including sensor/interface mapping and model splitting across various AI accelerators) (§5.1 and §5.2), and selects the most efficient option based on estimated cost (§5.3 and §5.4). Following this selection, the runtime orchestrator splits the model pipeline across the distributed AI accelerators and dispatches tasks to the devices (§5.5). Once these tasks are deployed, the end-to-end pipelines operate collaboratively on the wearables, independent of the moderator.

4.2 Programming Abstraction

Synergy introduces a paradigm shift in programming abstraction for distributed wearable environments. Unlike conventional model serving platforms such as Tensorflow Serving [38], which primarily focus on the execution of AI models, our system empowers applications to specify comprehensive end-to-end pipelines. In addition to the model execution, applications can specify the entire pipeline, starting from data acquisition (sensing) to the data destination (where the model output is processed). This feature enables applications to implement output-processing logic independently to underlying AI accelerators.

Given the dynamic and heterogeneous nature of wearable environments, developers cannot predetermine which specific devices will be used at development time. To address this, we decouple the specification of pipeline logic from the physical hardware. For model execution, applications simply specify the model to be executed; Synergy then dynamically assigns this execution over available AI accelerators, in a way of maximizing throughput. Moreover, Synergy supports distributed execution of models when it proves beneficial, splitting the model across distributed AI accelerators to leverage their collective computational power.

For sensing operations, our system dynamically maps the appropriate sensor device to the operation based on the application’s requirements. These requirements can include sensor type, resolution, position, and other relevant criteria. The target devices for model outputs can be similarly described, with requirements such as interface type, physical location, etc. Currently, Synergy supports two types of requirements, (designated device and sensor type) for source devices and (designated device or interface type) for target devices. Similarly, target devices can be specified as a designated device or interface type.

We provide two primary system functions: **add**(*pipeline*) and **remove**(*pipeline*). A pipeline is described using three

parameters: sensing requirement, model, and target requirement. For example, a pipeline for conversational agents can be described as (*microphone, KeywordSpotting, earbud*) for keyword spotting, and for life-logging applications, it could be (*camera on glasses, MobileNet, storage*).

5 RUNTIME ORCHESTRATION

Synergy performs runtime holistic orchestration when Synergy receives a pipeline request from an application or detects changes in the availability of devices. Figure 5 shows the high-level operational flow, which comprises four stages.

Execution plan generation (§5.1): Synergy generates execution plan candidates for each pipeline based on the resources currently available. Here, an execution plan details which tasks in the pipeline run on which devices. This enables Synergy to leverage a wide variety of resource use options and achieve the best-effort performance.

Runnable Joint Plan Generation (§5.2): Synergy creates joint plan candidates as a Directed Acyclic Graph (DAG) by combining execution plan candidates from different pipelines.

Joint Plan Selection (§5.3): By estimating the throughput of each joint plan (§5.4), Synergy selects the best joint plan, expected to deliver the highest throughput.

Task Deployment (§5.5): Synergy allocates a set of tasks to distributed devices according to the chosen joint plan. This includes setting up data sources or destinations if data exchange between devices is required. Subsequently, each wearable device executes the assigned tasks according to the assigned schedule.

5.1 Execution Plan Generation

For each pipeline, we generate execution plan candidates. Each execution plan determines (a) where to put source and target tasks and (b) where to split the model on which devices, resulting in a set of tasks mapped to each of the devices. For model execution, the system explores various combinations of splitting layers over different AI accelerators. For source and target tasks, it performs device mapping based on matching the application’s requirements with the device’s capabilities. This end-to-end pipeline planning considering source and target is crucial to reduce device-to-device communication. Suppose there are two devices, a smart watch (d_1) with a microphone and AI accelerator and a smart ring (d_2) with a haptic interface and AI accelerator. If a pipeline is described as (*microphone, KeywordSpotting (KWS), haptic*) and the KWS model has 9 layers, one execution plan example would be [$(microphone \rightarrow d_1) \rightarrow (KWS^{0:4} \rightarrow d_1) \rightarrow (KWS^{4:9} \rightarrow d_2) \rightarrow (haptic \rightarrow d_2)$], where $KWS^{i:j}$ refers to the execution of the model from layer i to j . The edge between nodes represents data exchange. If the devices in the source and

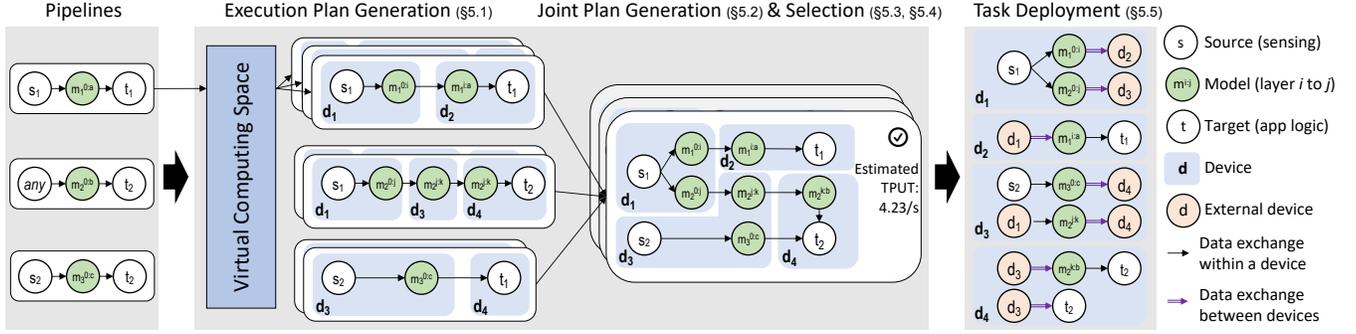


Figure 5: Operational flow of Synergy.

destination nodes with an edge are different, we add the data transmission and reception tasks. Similarly, we add the task of data loading to and unloading from the input memory of an AI accelerator when the model task is involved.

5.2 Runnable Joint Plan Generation

The next step is to generate a set of runnable joint execution plans. The joint execution plan can be simply created by combining execution plan candidates from different pipelines. We combine execution plans to give Synergy visibility over resource competition and dependency across pipelines. It also contributes to identifying the operation-sharing opportunities between pipelines. When duplicate nodes are found, Synergy merges them and allows sharing of the operation output with the subsequent nodes.

Here, one important task is to filter out unsupported joint plans when AI accelerators currently available do not have sufficient capability. More specifically, we consider three key constraints pertaining to MAXIM AI accelerators: (1) the weight memory, (2) the bias memory, and (3) the maximum number of layers. If a set of model tasks is assigned to an AI accelerator, we consider it if the total weight memory, bias memory, and number of layers of assigned model tasks are below the capacity of the AI accelerator. If all AI accelerators used in a joint plan are supportable, we consider the joint plan to be runnable.

To demonstrate the importance of (1) jointly considering resource availability in the previous paragraph and (2) end-to-end pipeline planning with source and target devices (§5.1), we conducted an experiment where two pipelines (KWS [16] and EfficientNetV2 [37]) are deployed on two MAX78000 devices. We consider two variants of Synergy: (1) Synergy without joint resource consideration (Synergy w/o Joint), which identifies an optimal candidate for each pipeline individually and integrates them into a final execution plan. (2) Synergy excluding source and target considerations (Synergy w/o Src&Tgt), which focuses solely on model splitting

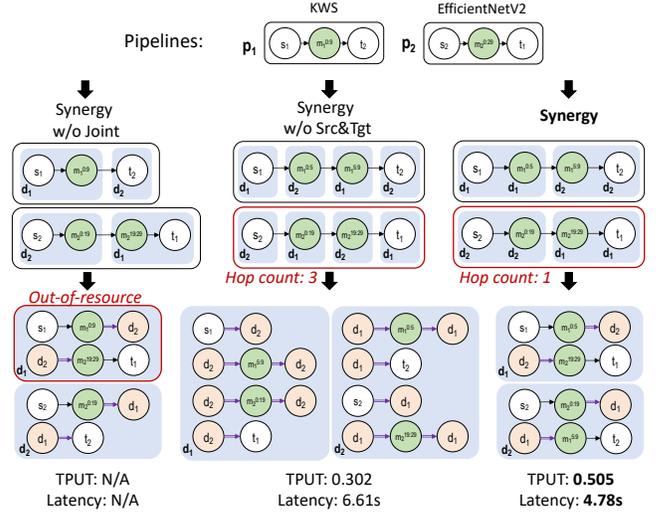


Figure 6: Comparison of Synergy with (1) Synergy without joint resource consideration and (2) Synergy without source and target consideration.

jointly and assigns source and target devices after combining the candidates. Figure 6 depicts the execution plans and tasks allocated to each device. Notably, Synergy w/o Joint encountered an out-of-resource (OOR) error on d_1 as both p_1 and p_2 assigned the large chunk of each model. This aspect highlights the necessity of accounting for joint resource constraints. On the other hand, Synergy w/o Src&Tgt exhibited reduced throughput and increased latency compared to Synergy. It made the same decision as Synergy for the model splitting; however, source and target nodes are assigned to different devices from the ones where the relevant model execution is running, leading to unnecessary communication overhead. This result demonstrates the importance of considering source and target devices in the planning phase to mitigate device-to-device communication costs.

Algorithm 1 Synergy Multi-Pipeline Execution Planning

Input: List of pipelines P , List of devices D , Metric m **Output:** Best joint execution plan E^*

```
1:  $P \leftarrow \text{SortByDataSize}(P), E^* \leftarrow \emptyset$ 
2: for each pipeline  $p \in P$  do
3:    $E_p \leftarrow \text{ExecutionPlanGeneration}(p, D)$   $\triangleright$  §5.1
4:    $E'_p \leftarrow \text{JointPlanGeneration}(E_p, E^*)$   $\triangleright$  §5.2
5:   if  $m == \text{throughput}$  then
6:      $E^* \leftarrow \arg \max_{e_i \in E'_p} \text{Throughput}(E^*, e_i)$   $\triangleright$  §5.4
7:   else if  $m == \text{latency}$  then
8:      $E^* \leftarrow \arg \min_{e_i \in E'_p} \text{Latency}(E^*, e_i)$   $\triangleright$  §5.4
9:   end if
10: end for
```

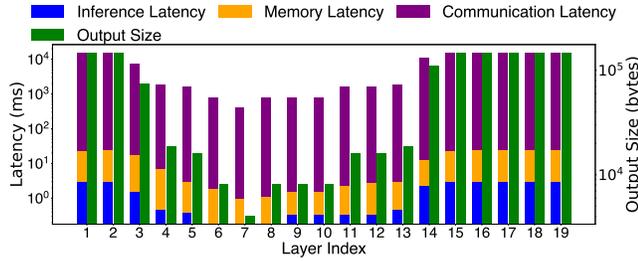


Figure 7: Layer-wise latency analysis for UNet. The y-axis is in the logarithmic scale.

5.3 Joint Plan Selection

Since a pipeline’s execution plan could compete the resources of AI accelerators with other pipelines’ plans and have performance dependency with each other, it is important to consider pipelines’ execution plans altogether (as a joint plan) to achieve the best performance. However, consideration of all possible combinations between pipelines and devices would require intractable computational overhead. To address this, we devise a simple yet effective orchestration algorithm. Algorithm 1 shows a high-level description. Its key idea is to make a selection decision sequentially pipeline by pipeline. For instance, the decision for the execution plan of the second pipeline is made by considering the remaining resources of AI accelerators after the first pipeline’s assignment, and so forth.

Understanding layer-wise latency: To devise an effective planning algorithm, it is essential to understand the composition of latency at each layer. Figure 7 illustrates the layer-wise inference latency, memory latency, communication latency, and output data size for UNet [31]. We found two important characteristics. First, compared to the inference latency (1.5 ms), both the memory latency (10.6 ms) and communication latency (6869.1 ms) are significant, which are 7 \times and 4579 \times higher, respectively. This latency gap between

inference and communication is a notable characteristic of AI accelerators specially designed for fast inference. Second, the output size varies across different layers, with the lowest layer latency of 4426.2 ms and the highest of 161864.5 ms, indicating a 36 \times difference. These findings collectively suggest that the primary bottleneck in running a pipeline across multiple AI accelerators lies in the data communication latency. Consequently, an efficient execution plan should strategically select split points to minimize the total data transmission size. It is important to note that minimizing data communication cost also contributes to minimizing the overall energy cost, as communication latency is much higher than inference latency. In addition, Wi-Fi itself draws higher power than model inference on AI accelerators; 25 mW~56 mW during the model inference on MAX78000 [30] and 215 mW for Tx and 60 mW for Rx on ESP8266 [28], which is the Wi-Fi module used in our prototype.

Pipeline ordering: As we make a decision sequentially, the order of pipelines to investigate plays a key role to determine the system-wide performance. Based on our observation in layer-wise analysis (Figure 7), we found that data communication to share intermediate outputs between devices is the key bottleneck of the execution. To this end, we sort the pipelines in descending order based on their data intensity to ensure that models with higher data intensity occupy device resources first for better decisions. Specifically, given the input size In^{size} and output size Out_l^{size} for each layer $l \in L$, we define the data intensity of the model as its average data size, $(In^{\text{size}} + \sum_l Out_l^{\text{size}})/(L + 1)$.

Holistic approach for plan selection: The selection method takes a list of pipelines and available devices as input and provides a joint plan as output. For each pipeline in the sorted pipeline list, it first generates runnable candidate plans based on the previous decision. Then, the algorithm computes the target metric for each candidate plan and selects the best plan. This selection considers the joint resource constraints imposed by the previously selected plan for the current pipeline. This holistic approach ensures that the overall system efficiency is optimized, rather than focusing on individual pipeline performance. Finally, the algorithm returns the best joint execution plan (a set of selected plans), which maps all tasks to the devices for running pipelines.

5.4 Selection Policy

As introduced in §3.4, in this paper, we target maximizing the overall throughput of pipelines as an objective. However, we can also easily adopt different system policies (e.g., minimizing the latency or minimizing the energy consumption) by defining a different cost function. We present the latency estimation of a task, which acts as a primitive unit

for the cost function, and then explain how we model the throughput function using the task latency.

5.4.1 Task Latency. To estimate the throughput of a joint plan, it is important to estimate the latency of each task in the execution plan. The execution plan can have six types of tasks: (1) sensing, (2) data reception from other devices, (3) data loading to AI accelerator memory, (4) model inference for a given range of layers, (5) data unloading from AI accelerator memory, (6) data transmission to other devices. We explain how to estimate the latency for each task.

Sensing (1): We measure sensing latency for camera and audio inputs during the profiling phase, which is then used for latency computation. **Memory** (3,5): We model memory latency by quantifying the time taken for memory copy operations between the microcontroller’s memory and the accelerator’s memory. **Inference** (4): The inference latency is estimated by considering the number of clock cycles in conjunction with the accelerator’s clock frequency. Note that we can calculate the number of required clock cycles with the operations of layers and the parallel processors within AI accelerators. For example, the number of clock cycles required for CNN layers in MAX78000 devices can be calculated as follows: $(In^{col} + Out^{col} \cdot Out^{ch}) \cdot In^{row} \cdot \left\lceil \frac{In^{ch}}{N} \right\rceil$, where In is input data, Out is output data, col is the column, row is the row, ch is the channel, and N is the number of parallel processors, respectively. In^{ch} are executed in parallel by the N number of parallel processors in accelerators. Kernel sizes are not considered here, as each processor has a kernel accelerator within it. **Communication** (2,6): We account for the latencies involved in UART and Wi-Fi communications between different devices. The details of our networking implementation for UART and Wi-Fi are described in Section 6.

The latency numbers of the above four tasks are calibrated for our target devices. We validated that the error rate of this latency estimation (excluding data transmission) is less than around 1% (e.g., 11 μ s gap between the estimated and measured latency for the inference of ConvNet5 on MAX78000). Since wireless transmission latency fluctuates, we take an average and update it periodically.

5.4.2 Throughput Estimation. As we combine execution plans, we assume that the execution cycle is complete if all pipelines are finished. That is, we consider the number of execution cycles per second as throughput (i.e., the sum of inferences of all pipelines per second). Specifically, given a joint plan, we calculate the throughput by assuming all tasks in parallel across independent computation units, such as microcontroller (MCU), AI accelerator, and Wi-Fi module, to reflect how the tasks operate at runtime.

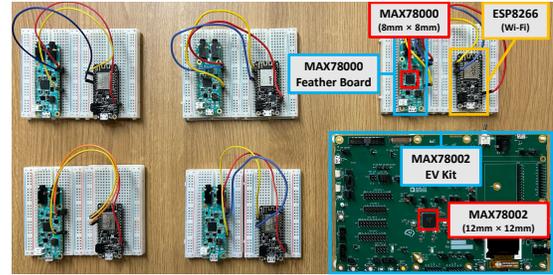


Figure 8: Hardware setup.

5.5 Task Deployment

Once a joint plan is selected, model splitting is needed to allow parts of the model to be executed 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 in servers. 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 AI devices. Then, Synergy constructs a DAG for each device by combining tasks assigned to the very device. It also adds external device as a source or a destination (as shown in Figure 5) if data exchange between different devices is needed.

6 PROTOTYPE IMPLEMENTATION

We prototyped Synergy on the off-the-shelf MAX78000 feather board [25] and MAX78002 Evaluation Kit [27], which are a development platform for the MAX78000 [23] and MAX78002 [26], respectively, as shown in Figure 8. Both platforms have a variety of peripherals such as a camera, a microphone, LED, and push buttons to help quick proof-of-concepts and software development on top of MAX78000 and MAX78002. Note that although the development platform is bulky for the wearable form factor, the actual size of the accelerators is tiny (e.g., 8mm×8mm for MAX78000 and 12mm×12mm for MAX78002).

For wireless communication, we interfaced MAX78000/MAX78002 with an ESP8266 Wi-Fi module [6]. The interface between AI accelerators and ESP8266 transfers data via serial communication over UART configured at a 115200 baud rate. 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

Table 1: Models used in this study. Sizes are all in bytes.

Models	# Layers	Sensing	Model Size	Input Size	Avg. Out Size
ConvNet5	5	Image	71158	28×28×1	14031
KWS	9	Audio	169472	128×128×1	7976
SimpleNet	14	Image	166448	32×32×3	9237
WideNet	14	Image	313700	32×32×3	10091
ResSimpleNet	17	Image	381792	32×32×3	11217
UNet	19	Image	279084	48×48×48	74547
EfficientNetV2	29	Image	627220	32×32×3	66468
MobileNetV2	56	Image	821164	32×32×3	296318

interface. Also, since typical TCP/IP stack is too resource-intensive for MCUs, we implement lightweight internet protocol (lwIP), which is a small independent implementation of TCP/IP protocol suite at under 40 kB memory with only essential functionalities.

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.

7 EVALUATION

7.1 Evaluation Settings

7.1.1 Models. In our experiment, we use eight different models: ConvNet5, KWS [16], SimpleNet [10], ResSimpleNet [11], WideNet [10], UNet [31], EfficientNetV2 [37], and MobileNetV2 [34]. We quantized these models with 8-bit integers and synthesized them for compatibility with MAX78000 and MAX78002. Note that the synthesized EfficientNetV2 and MobileNetV2 fit to a single MAX78002, but do not fit into a single MAX78000 device due to resource constraints.

7.1.2 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 joint 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. The overall operation is similar to MinDev but 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: Similar to PriMinDev, Prioritized MaxDev (PriMaxDev) also emphasizes optimal splitting points and device assignment order. The key difference is to consider execution plans that involve all devices.

IndModel: Inspired by state-of-the-art model partitioning algorithms [12, 14, 15, 17, 42], IndModel selects the best-split execution plan based on metric estimations, but without considering holistic planning. IndModel independently selects the optimal execution plan for each pipeline and forms the final joint plan by aggregating the 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 state-of-the-art algorithms, conducts a holistic assessment. It primarily focuses on optimally splitting multiple models, considering resource usage across pipelines and aligning with Synergy’s decision-making process.

All aforementioned six baselines focus on optimal model distribution over AI accelerators and map source and target devices post-model splitting. If multiple candidates exist for source and target devices, these baselines choose the ones that minimize the total number of source and target devices. **IndBest:** IndModel advances state-of-the-art partitioning algorithms (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 latency, from sensing to output delivery. However, it does not account for the resource usage of other pipelines.

7.2 Overall Performance

We conducted our primary experiments using two types of workloads: (1) multiple small pipelines, each capable of running on a single device, and (2) a single large pipeline that requires multiple devices. For the experiment, we devise four workloads, two for each type: Workload 1 (ConvNet5, ResSimpleNet, UNet), Workload 2 (KWS, SimpleNet, WideNet), Workload 3 (EfficientNetV2) and Workload 4 (MobileNetV2). By default, we use four MAX78000 devices, ensuring an even distribution of source and target devices.

Figure 9 shows the throughput of different baselines on four scenarios. The experimental results show that Synergy consistently outperforms all baselines in various workloads due to its holistic decision-making process, which includes end-to-end pipeline metric estimations and joint resource

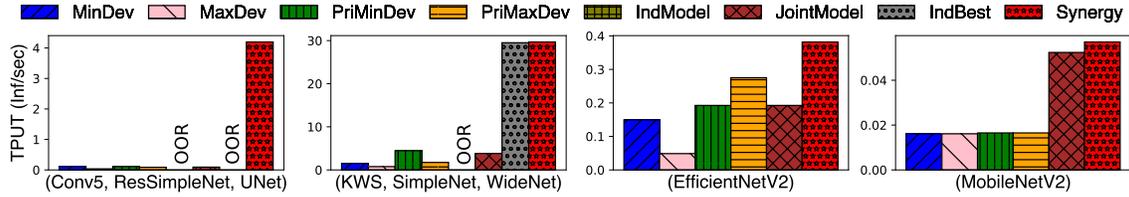


Figure 9: Overall performance: total throughput of different workloads; higher is better.

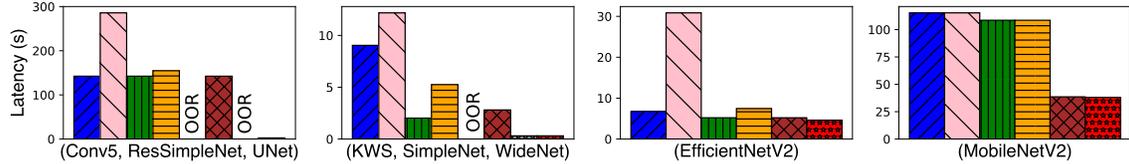


Figure 10: Overall performance: average end-to-end latency of different workloads; lower is better.

constraints. In Workload 1, while IndModel (representative of current model partitioning techniques) results in an OOR situation, Synergy achieves a total throughput of 4.20, which is 38.4 times higher than the next best (PriMinDev). In Scenario 2, IndModel still leads to OOR, yet IndBest, when incorporating source and target devices with model splitting, shows comparable throughput to Synergy. In Workload 3 and 4, Synergy continues to exhibit 1.4 times and 1.1 times higher throughput than the next best (PrivMaxDev and IndModel), respectively. Note that we do not report IndBest and IndModel in Workload 3 and 4 because IndBest is the same as Synergy and IndModel is the same as JointMode due to the presence of only one pipeline. The results indicate two key points. First, Synergy significantly boosts throughput when there are more distribution options with multiple pipelines and devices (Workload 1 and 2). Second, Synergy supports large models even on top of resource-constrained MCU environments through the dynamic composition of distributed AI accelerators (Workload 3 and 4).

We further analyze the performance characteristics of the baselines. MinDev generally outperforms MaxDev, suggesting that maximizing splitting layers is less effective due to increased communication overhead. The performances of PriMinDev and PriMaxDev improve with prioritization strategies, highlighting the benefits of choosing split points with minimal size of outputs. However, PriMaxDev outperforms PriMinDev in Workload 3 (EfficientNetV2), implying that minimizing device usage can sometimes result in sub-optimal split points due to resource limitations, leading to decreased performance. This finding emphasizes the necessity of a comprehensive approach in execution planning. Interestingly, the individually best model partitioning plan (IndModel) is not as effective as anticipated due to the absence of holistic execution planning. While IndBest, which includes source and target mapping with model splitting,

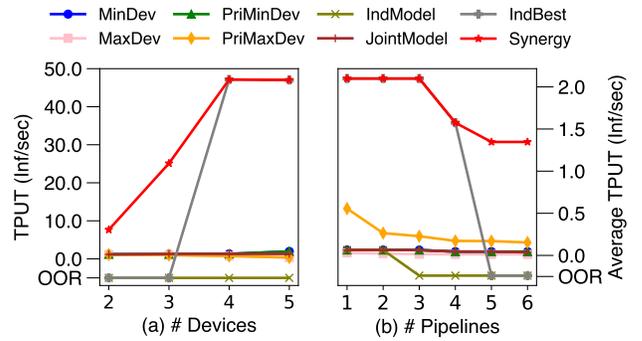


Figure 11: Impact of runtime environment changes: (a) number of devices 8, (b) number of pipelines.

can match Synergy’s effectiveness, it leads to OOR situations in some cases, e.g., Workload 2. Although JointModel reliably manages OOR cases by considering joint resource constraints, it falls short of Synergy as it does not fully account for end-to-end communication costs, including source and target device involvement.

Figure 10 shows the associated latency results. While the primary goal was to maximize total throughput, significant latency improvements were also observed. For instance, in Workload 1, Synergy not only offers 38.4 times higher throughput compared to PriMinDev but also reduces the average latency of pipeline executions by 55.2 times. Similarly, in Workload 3, Synergy achieves 1.39 times higher throughput and 1.61 times lower latency than PriMaxDev. In the subsequent experiments, we report only throughput results as the overall trend is similar across the experiments.

7.3 In-Depth Analysis

7.3.1 Runtime Environment Changes. We investigate how Synergy adapts to changes in the runtime environment. We consider two scenarios: variations in (a) the number of devices and (2) the number of pipelines.

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.

Figure 11 (a) shows that all baselines generally achieve higher overall throughput with an increase in available accelerators. This is a result of having more resources available for executing plans more effectively. 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 IndBest, the throughput gains for all other baselines are not notable, even with additional devices. Specifically, IndModel consistently faces OOR issues in all scenarios. While IndBest achieves comparable throughput in scenarios with sufficient accelerator resources (such as when 4 or 5 devices are available), it also leads to OOR in resource-constrained settings. 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.

Number of pipelines: We assess Synergy’s performance with a varying number of pipelines. The number of pipelines incrementally increased from one to six, following the sequence: UNet, ConvNet5, SimpleNet, KWS, ResSimpleNet, and WideNet. We kept the device set as four MAX78000s. To understand the effect of resource competition among pipelines, we report the *average* throughput across pipelines, i.e., the ratio of the number of completed pipelines per second to the total number of pipelines. Figure 11 (b) reveals a downward trend in average throughput as the number of pipelines increases for all baselines, due to the competition for accelerator resources among the models. Nevertheless, Synergy consistently outperforms the baselines significantly. Remarkably, even with six models, Synergy achieves an average throughput of 1.35, 30.1 times higher than the next best (PriMinDev). IndBest achieves comparable performance with a lower number of models but faces OOR issues when the count reaches five pipelines.

7.3.2 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.

Figure 12 shows the results in two different setups: one with four MAX78000s and another with three MAX78000s

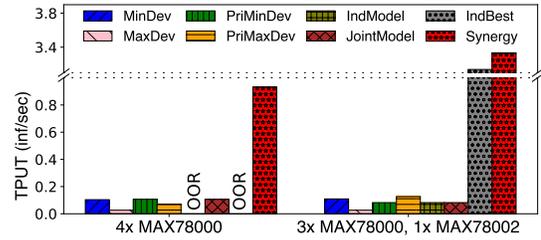


Figure 12: Effect of accelerator composition.

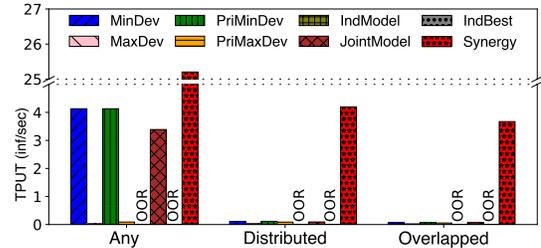


Figure 13: Effect of source and target mappings.

and one MAX78002. The inclusion of a higher-resource device generally led to an improvement in throughput. Leveraging its holistic planning approach, which takes into account the resource capabilities of each device, Synergy surpasses the baselines by efficiently utilizing the additional resources. With four MAX78000s, Synergy achieves a total throughput of 0.94, outperforming the next best (JointModel) by a factor of 8.8. This throughput further increases to 3.33 when incorporating one MAX78002. Interestingly, while IndBest matches Synergy in scenarios with sufficient accelerator resources, it falls behind by 6% in throughput. PriMinDev, which allocates all models exclusively to the single MAX78002, results in a significantly lower throughput of 0.08 compared to Synergy. This discrepancy highlights the importance of consideration for the communication overhead associated with source and target devices.

7.3.3 Source and Target Mapping. To assess the effect of source and target mapping, we conducted experiments across three scenarios with varying configurations of source and target devices: (1) "overlapped," where source and target devices are identical across pipelines, (2) "distributed," where source and target devices of pipelines are evenly allocated among the devices, and (3) "any," where source and target devices can be any device within the set. The "distributed" setup is the same as Workload 1 from §7.2, while the "overlapped" and "any" scenarios only differ in their source and target device mappings.

As shown in Figure 13, throughput is typically at its lowest in the "overlapped" scenario, where a single device becoming a communication bottleneck limits parallelization opportunities. In contrast, the "any" scenario exhibits the highest throughput, benefiting from the distribution of communication costs across various source and target devices.

Specifically, Synergy achieves total throughputs of 3.67, 4.20, and 16.80 in the "overlapped," "distributed," and "any" scenarios, respectively. These findings suggest that Synergy’s performance exhibits insensitivity to source-target device configurations due to its comprehensive execution planning, which accounts for both source and target factors.

8 DISCUSSION

Expanding pipeline architecture: Currently, Synergy is designed to support a straightforward pipeline architecture, consisting of a single data source, one AI model, and a single output target. However, real-world applications often demand more complex configurations. For instance, sensor fusion requires integrating multiple data sources for a more holistic analysis, and complex information processing might require the conditional chaining of multiple models. Looking ahead, our future work aims to expand the system’s capabilities to support these sophisticated pipeline architectures.

Energy cost investigation: In this study, our primary objective was to maximize throughput while not explicitly addressing energy consumption, because we assume that data communication accounts for a significant portion of energy usage due to its high power draw and long duration as reported in §5.3. Nevertheless, this offers only a limited perspective on the power behavior of Synergy. Future research will aim to provide an in-depth analysis of the energy costs associated with tiny AI accelerators and peripherals on MCUs.

Enhancing collaboration among AI accelerators: We leverage collaboration among tiny AI accelerators by supporting layer-wise model splitting, dividing a model along its layers. Future plans include expanding collaboration strategies to include channel-wise splitting [20, 21, 40], where neural network models are segmented by channel within the same layer for parallel processing across multiple accelerators. This approach enables Synergy to handle models with larger inputs, overcoming current data memory limitations. Additionally, combining layer-wise and channel-wise splitting would offer more flexible and efficient collaboration.

9 RELATED WORK

TinyML: TinyML represents a research field of 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 been focused on minimizing model size. Existing studies have focused primarily on three techniques: model pruning [19], model quantization [32], and neural architecture search (NAS) [8, 18]. In contrast, our work focuses on supporting multiple or large AI models without compromising the accuracy, by

dynamically composing distributed AI accelerators. However, the combined power of tiny AI accelerators might still be insufficient to support unmodified, off-the-shelf large AI models. We envision that Synergy can benefit from these TinyML techniques to cover 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 a layer-wise model partitioning over resource-limited embedded and mobile devices. In common, these methods [12, 14, 15, 17, 42] allocates few initial layer 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 the powerful resources such as 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, there are three key differences in terms of the techniques: (1) As target platforms, we focus on multiple on-body AI accelerators and explore multiple layer splitting options, while the existing studies mainly consider only one splitting, e.g., between a mobile device and a cloud. (2) We identify and address the resource competition and dependency issues when dealing with multiple concurrent models. (3) While these studies focus only on the *model* distribution mostly on a smartphone environment, we identify that the data production (source) and consuming (target) devices can also be dynamic in wearable environments and devise an orchestration technique that considers these aspects holistically.

DNN workload distribution: An application pipeline often consists of a (conditional) sequence of multiple DNN models. When multiple processing devices are available, several studies have been proposed to enhance the inference throughput by distributing model processing into different devices and parallelizing their execution [5, 13, 41]. 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 [13, 41]. 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 [38], Sagemaker [33], and Azure ML [29], have been proposed to facilitate model inference serving by

offering containerized environments for model execution in diverse devices. Research platforms, such as Velox [3] and Clipper [4], 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 solution, the *virtual computing space*, addresses this by mapping software logic to physical resources and optimizing performance by considering pipeline interdependencies and resource usage.

10 CONCLUSION

We presented Synergy, a novel system for the dynamic composition of tiny AI accelerators on the body. Synergy addressed the problem of dynamic and heterogeneous wearable environments with the concept of *virtual computing space*, which simplifies the integration of diverse AI applications on wearable devices. Then, Synergy’s runtime dynamically distributes model execution tasks into device resources available for the best-effort inference. Our extensive evaluation showed that Synergy consistently shows higher throughput than existing model partitioning techniques.

REFERENCES

- [1] Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Gutttag. 2020. What is the State of Neural Network Pruning?. In *Proceedings of Machine Learning and Systems*, I. Dhillon, D. Papailiopoulos, and V. Sze (Eds.), Vol. 2. 129–146. https://proceedings.mlsys.org/paper_files/paper/2020/file/6c44dc73014d66ba49b28d483a8f8b0d-Paper.pdf
- [2] Coral Micro [n. d.]. Google Coral Micro. <https://coral.ai/products/dev-board-micro/>. Accessed: 30 Nov. 2023.
- [3] Daniel Crankshaw, Peter Bailis, Joseph E Gonzalez, Haoyuan Li, Zhao Zhang, Michael J Franklin, Ali Ghodsi, and Michael I Jordan. 2014. The missing piece in complex analytics: Low latency, scalable model management and serving with velox. *arXiv preprint arXiv:1409.3809* (2014).
- [4] Daniel Crankshaw, Xin Wang, Guilio Zhou, Michael J Franklin, Joseph E Gonzalez, and Ion Stoica. 2017. Clipper: A low-latency online prediction serving system. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. 613–627.
- [5] Zheng Dong, Yan Lu, Guangmo Tong, Yuanchao Shu, Shuai Wang, and Weisong Shi. 2023. Watchdog: Real-time vehicle tracking on geo-distributed edge nodes. *ACM Transactions on Internet of Things* 4, 1 (2023), 1–23.
- [6] ESP8266 [n. d.]. Adafruit HUZZAH ESP8266. <https://www.adafruit.com/product/2821>. Accessed: 30 Nov. 2023.
- [7] Ethos-U65 [n. d.]. Arm Ethos-U65. <https://www.arm.com/products/silicon-ip-cpu/ethos/ethos-u65>. Accessed: 30 Nov. 2023.
- [8] Igor Fedorov, Ryan P Adams, Matthew Mattina, and Paul Whatmough. 2019. Sparse: Sparse architecture search for cnns on resource-constrained microcontrollers. *Advances in Neural Information Processing Systems* 32 (2019).
- [9] GAP8/GAP9 [n. d.]. Greenwaves Technology. <https://greenwaves-technologies.com/low-power-processor/>. Accessed: 30 Nov. 2023.
- [10] Seyyed Hossein Hasanpour, Mohammad Rouhani, Mohsen Fayyaz, and Mohammad Sabokrou. 2016. Lets keep it simple, using simple architectures to outperform deeper and more complex architectures. *arXiv preprint arXiv:1608.06037* (2016).
- [11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.
- [12] Chuang Hu, Wei Bao, Dan Wang, and Fengming Liu. 2019. Dynamic adaptive DNN surgery for inference acceleration on the edge. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 1423–1431.
- [13] Si Young Jang, Boyan Kostadinov, and Dongman Lee. 2021. Microservice-based edge device architecture for video analytics. In *2021 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE Computer Society, 165–177.
- [14] Hyuk-Jin Jeong, Hyeon-Jae Lee, Chang Hyun Shin, and Soo-Mook Moon. 2018. IONN: Incremental offloading of neural network computations from mobile devices to edge servers. In *Proceedings of the ACM symposium on cloud computing*. 401–411.
- [15] Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor Mudge, Jason Mars, and Lingjia Tang. 2017. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. *ACM SIGARCH Computer Architecture News* 45, 1 (2017), 615–629.
- [16] KWS20 [n. d.]. Analog Keywords Spotting. <https://www.analog.com/en/design-notes/keywords-spotting-using-the-max78000.html>. Accessed: 30 Nov. 2023.
- [17] Stefanos Laskaridis, Stylianos I Venieris, Mario Almeida, Ilias Leonatiadis, and Nicholas D Lane. 2020. SPINN: synergistic progressive inference of neural networks over device and cloud. In *Proceedings of the 26th annual international conference on mobile computing and networking*. 1–15.
- [18] Edgar Liberis, Lukasz Dudziak, and Nicholas D Lane. 2021. μ NAS: Constrained neural architecture search for microcontrollers. In *Proceedings of the 1st Workshop on Machine Learning and Systems*. 70–79.
- [19] Edgar Liberis and Nicholas D Lane. 2023. Differentiable Neural Network Pruning to Enable Smart Applications on Microcontrollers. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 6, 4 (2023), 1–19.
- [20] Jiachen Mao, Xiang Chen, Kent W Nixon, Christopher Krieger, and Yiran Chen. 2017. Modnn: Local distributed mobile computing system for deep neural network. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. IEEE, 1396–1401.
- [21] Jiachen Mao, Zhongda Yang, Wei Wen, Chunpeng Wu, Linghao Song, Kent W Nixon, Xiang Chen, Hai Li, and Yiran Chen. 2017. Mednn: A distributed mobile system with enhanced partition and deployment for large-scale dnns. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 751–756.
- [22] MAX32650 [n. d.]. Analog MAX32650. <https://www.analog.com/en/products/max32650.html>. Accessed: 30 Nov. 2023.
- [23] Max78000 [n. d.]. Analog MAX78000. <https://www.analog.com/en/products/max78000.html>. Accessed: 30 Nov. 2023.
- [24] MAX78000 Performance [n. d.]. Cutting the AI Power Cord: Technology to Enable True Edge Inference. https://cms.tinymt.org/wp-content/uploads/talks2020/tinyML_Talks_Kris_Ardis_and_Robert_Muchsel_-201027.pdf. Accessed: 30 Nov. 2023.
- [25] Max78000FTHR [n. d.]. Analog MAX78000FTHR. <https://www.analog.com/en/design-center/evaluation-hardware-and-software/evaluation-boards-kits/max78000fthr.html>. Accessed: 30 Nov. 2023.
- [26] Max78002 [n. d.]. Analog MAX78002. <https://www.analog.com/en/products/max78002.html>. Accessed: 30 Nov. 2023.
- [27] Max78002EVKIT [n. d.]. Analog MAX78002EVKIT. <https://www.analog.com/en/design-center/evaluation-hardware-and-software/evaluation-boards-kits/max78002evkit.html>. Accessed: 30 Nov. 2023.

- [28] Manan Mehta. 2015. ESP8266: A Breakthrough in wireless sensor networks and internet of things. *International Journal of Electronics and Communication Engineering & Technology* 6, 8 (2015), 7–11.
- [29] ML as a Service [n. d.]. Azure Machine Learning. <https://azure.microsoft.com/en-gb/products/machine-learning>. Accessed: 30 Nov. 2023.
- [30] Arthur Moss, Hyunjong Lee, Lei Xun, Chulhong Min, Fahim Kawsar, and Alessandro Montanari. 2022. Ultra-low Power DNN Accelerators for IoT: Resource Characterization of the MAX78000. In *Proceedings of the 20th ACM Conference on Embedded Networked Sensor Systems*. 934–940.
- [31] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention—MICCAI 2015: 18th International Conference, Munich, Germany, October 5–9, 2015, Proceedings, Part III 18*. Springer, 234–241.
- [32] Manuele Rusci, Alessandro Capotondi, and Luca Benini. 2020. Memory-driven mixed low precision quantization for enabling deep network inference on microcontrollers. *Proceedings of Machine Learning and Systems* 2 (2020), 326–335.
- [33] SageMaker [n. d.]. Amazon SageMaker. https://aws.amazon.com/sagemaker/?nc1=h_ls. Accessed: 30 Nov. 2023.
- [34] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 4510–4520.
- [35] K Simonyan and A Zisserman. 2015. Very deep convolutional networks for large-scale image recognition. *3rd International Conference on Learning Representations (ICLR 2015)*, 1–14.
- [36] STM32F7 [n. d.]. STM32F7 Series. <https://www.st.com/en/microcontrollers-microprocessors/stm32f7-series.html>. Accessed: 30 Nov. 2023.
- [37] Mingxing Tan and Quoc Le. 2021. Efficientnetv2: Smaller models and faster training. In *International conference on machine learning*. PMLR, 10096–10106.
- [38] TensorFlow Serving [n. d.]. Google. <https://www.tensorflow.org/tfx>. Accessed: 30 Nov. 2023.
- [39] TF Lite Micro [n. d.]. TensorFlow Lite for Microcontrollers. <https://www.tensorflow.org/lite/microcontrollers/>. Accessed: 30 Nov. 2023.
- [40] Liekang Zeng, Xu Chen, Zhi Zhou, Lei Yang, and Junshan Zhang. 2020. Coedge: Cooperative dnn inference with adaptive workload partitioning over heterogeneous edge devices. *IEEE/ACM Transactions on Networking* 29, 2 (2020), 595–608.
- [41] Xiao Zeng, Biyi Fang, Haichen Shen, and Mi Zhang. 2020. Distream: scaling live video analytics with workload-adaptive distributed edge intelligence. In *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*. 409–421.
- [42] Shigeng Zhang, Yinggang Li, Xuan Liu, Song Guo, Weiping Wang, Jianxin Wang, Bo Ding, and Di Wu. 2020. Towards real-time cooperative deep inference over the cloud and edge end devices. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 4, 2 (2020), 1–24.