THEME ARTICLE: TinyML

Hardware–Software Co-Design for Real-Time Latency–Accuracy Navigation in Tiny Machine Learning Applications

Payman Behnam, Jianming Tong, Alind Khare, Yangyu Chen, Yue Pan, Pranav Gadikar, Abhimanyu Bambhaniya, Tushar Krishna, and Alexey Tumanov, *Georgia Institute of Technology, Atlanta, GA, 30332, USA*

Tiny machine learning (TinyML) applications increasingly operate in dynamically changing deployment scenarios, requiring optimization for both accuracy and latency. Existing methods mainly target a single point in the accuracy/latency tradeoff space, which is insufficient as no single static point can be optimal under variable conditions. We draw on a recently proposed weight-shared SuperNet mechanism to enable serving a stream of queries that activates different SubNets within a SuperNet. This creates an opportunity to exploit the inherent temporal locality of different queries that use the same SuperNet. We propose a hardware–software co-design called SUSHI that introduces a novel SubGraph Stationary optimization. SUSHI consists of a novel field-programmable gate array implementation and a software scheduler that controls which SubNets to serve and which SubGraph to cache in real time. SUSHI yields up to a 32% improvement in latency, 0.98% increase in served accuracy, and achieves up to 78.7% off-chip energy saved across several neural network architectures.

Tiny machine learning (TinyML) applications simultaneously care about both the accuracy and latency of ML inference served. Examples include Intensive Care Unit (ICU) stability score prediction[1] and self-driving cars.[2] A body of work including compression- and hardware-aware neural architecture searches (NAS) significantly improved latency–accuracy tradeoffs for specific deep neural network (DNN) models. However, all of these techniques optimize for a *single* static point in the latency–accuracy tradeoff space. We claim that this is no longer sufficient for dynamic TinyML deployment conditions.

We observe that these *TinyML* applications with acute latency–accuracy sensitivity typically operate under *dynamic* deployment conditions. These include variable-query traffic patterns (e.g., emergency room versus magnetic resonance imaging in hospitals), on-device battery power level (e.g., bedside compute), and query complexity (e.g., autonomous vehicle (AV) navigation of sparse suburban versus dense urban terrain). Under dynamic deployment conditions, a choice of any single model from the latency/accuracy tradeoff space may be suboptimal as it offers a fixed tradeoff. Indeed, the highest accuracy model may result in dropped queries when transient overloads occur due to its higher latency. When the load is low, the lowest accuracy model may yield suboptimal prediction quality—both unnecessarily underperforming. The ideal is picking a "best-fit" model from the latency/accuracy tradeoff space. Thus, the ability to navigate between arbitrary points in the latency/accuracy tradeoff space in real time is intuitively required for such *TinyML* applications.

We identify one mechanism that enables this: weight-shared *SuperNets*.[3,4] This neural network construct consists of multiple convolutional neural networks sharing common weights. It simultaneously encapsulates "deep and thin" and "wide and shallow" models within the same structure. These *SuperNets* can be used to activate different *SubNets* without explicitly extracting them into different independently stored models. This is highly efficient from a systems perspective as it obviates the need to store model variants separately, and enables rapid switching between *SubNets* "activated" to serve different incoming queries in real time.

On the training side, once-for-all (OFA)[3] proposes an approach to jointly train multiple such neural network

0272-1732 © 2023 IEEE
Digital Object Identifier 10.1109/MM.2023.3317243
Date of publication 20 September 2023; date of current version 2 November 2023.

configurations simultaneously with a constant training cost. This cost is still 40–50-GPU days and suffers from an explosion of suboptimal model configurations. It also suffers from a huge search space during the deployment phase. In the proposed CompOFA,[4] we seek to reduce this search space by constraining it to model configurations close to the expected accuracy-latency Pareto frontier. We incorporate insights of compound relationships between elastic model dimensions to build CompOFA,[4] creating a design space orders of magnitude smaller.

To be able to serve such *SuperNets* efficiently, we need the support of both hardware and software. The need for real-time inference has led to a plethora of accelerators in computer architecture literature. Key optimization techniques leveraged by these accelerators involve *reusing* activations and/or weights across multiple computations, (i.e., weight-, output-, input-, row-, and hybrid versions of these[5]). These dataflow optimizations rely on neural network layers, specifically 2-D convolutions, to be compute bound. One challenge of serving *SubNets* with diverse shapes, however, is the memory-bound nature of some of the *SubNets*.[6] To ameliorate the memory pressure, we make a key enabling observation that there's a significant amount of *temporal locality* in the weights of *SubNets* used *across queries*. We identify this as an opportunity for a novel type of data reuse, which we call *SubGraph* stationary (SGS) optimization, which, to the best of our knowledge, has not been previously proposed.

The efficiency of serving *SuperNets* is thus reduced to the technical challenge of managing the *SubNets* activated and *SubGraphs* reused across queries. Intuitively, the system will benefit from reusing the maximum possible *SubGraph* that offers the maximum overlap with activated *SubNets*. Addressing this problem involves careful design of the on-chip caching mechanism and the accelerator state-aware scheduler because the latency of served *SubNets* directly depends on the *SubGraph* cached on chip and is, therefore, a function of *both* control choices. Figure 1(a) illustrates that 1) latency for a deep and thin *SubNet* (blue) is lowest when a cached *SubGraph* contains more layers (matching its shape) and 2) latency for a wide and shallow *SubNet* (red) is lowest with a cached *SubGraph* with larger width layers. Thus, the software scheduler control decision must be responsible for 1) a *SubNet* selection that serves the current query with *awareness* of the current cached *SubGraph* (state) and 2) a cached-*SubGraph* update decision, which must be made by leveraging the temporal locality of weight reuse across *SubNets* served over time.

Thus, realization of the proposed SGS optimization requires the SGS-aware query schedule to be co-designed with the SGS's hardware implementation. The scheduler's goal is to decide, in real time, which *SubNet* to activate for each query and calculate which sequence of *SubGraphs* to cache over time. Finally, we generalize the SGS's optimization by proposing an abstraction that enables the query scheduling policy to retain its accelerator state awareness without any accelerator-specific information. We claim that it's possible to capture all the useful nuances of the accelerator state in a 2-D table, exposing the a priori measured latency of activating a *SubNet* i given a currently cached *SubGraph* j [Figure 2(a)]. We instantiate the concept of SGS *cross-query* optimization in **SUSHI**: our vertically integrated inference-serving stack.

TRAINING *SuperNet*

In OFA, there is a search space of *SubNets* of a teacher model with varying dimensions, like image resolution,

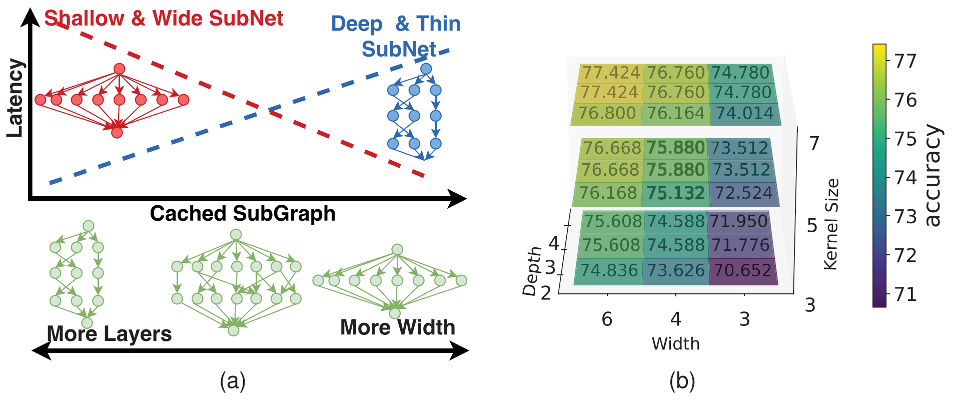

FIGURE 1. Observations for inference and training of weight-shared DNNs. (a) Latency of two different *SubNets* as a function of different cached *SubGraphs*. (b) Accuracy and latency heatmaps for varying uniform depth/width and expansion ratios and fixed different kernel size of MobV3.

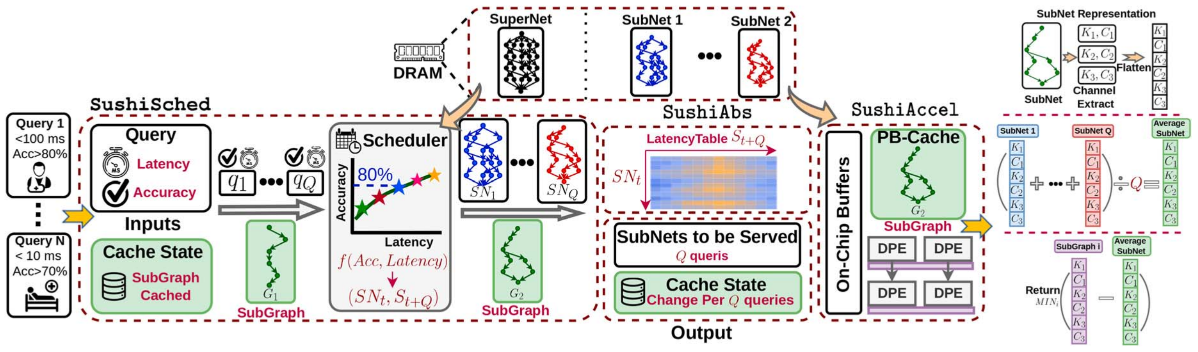

FIGURE 2. (Left) **SUSHI** overview and (right) **SUSHI** *SubGraph* selection. (a) System architecture overview. Given a stream of queries annotated with [Accuracy (Acc), Latency] pairs $q_1, .., q_Q$ and the current cache state C_1, the scheduler chooses the *SubNet* to be served SN_t for each *t*th query and next cache state C_2 after every Q queries. (b) Running average of the *SubNets* for the past Q queries. DRAM: dynamic random-access memory; PB: persistent buffer; DPE: dot product engine.

kernel size, depth, and width. These model dimensions are sampled independently of each other, allowing all combinations of dimensions, which leads to the combinatorial explosion of 10^{19} models. Obviously, such a large search space leads to complications in training and searching. In addition, during the deployment phase on the target hardware device, OFA relies on the proxy for accuracy and latency prediction, which is not precise.

We observed that in the OFA search space, there are specific trends in the accuracy and latency of the models that increase consistently with the growth of depth and width dimensions together, as Figure 1(b) shows.

Therefore, we propose that with a heuristic of growing both depth and width together, one can capture models with better accuracy and latency tradeoffs. We leverage these insights to build a new design space called *CompOFA* by removing the suboptimal configurations. We show that this tractable design space directly reduces interference in training, which allows us to reduce training duration. Note that batch normalization (BN) calibration is needed. We have to precompute BN for each *SubNet*, but BN layers are not memory/compute bound.

SERVING *SuperNet*: SYSTEM

SUSHI consists of three major components: scheduler (**SushiSched**), abstraction (**SushiAbs**), and accelerator (**SushiAccel**), as shown in Figure 2(a).

ABSTRACTION

SushiAbs provides latency estimation for the serving *SubNets* as a function of cached *SubGraph*. It implicitly makes **SushiSched** take cached-*SubGraph*-aware decisions. The abstraction needs to be space and time efficient.

To make it space efficient, the abstraction limits the set of all possible cached *SubGraphs* to a significantly smaller set $\mathcal{S}$ ($>> 10^{19\,4}$). The size of *SubGraphs* in $\mathcal{S}$ is selected to be close to the cache size. To make it time efficient, **SushiAbs** uses a lookup table with *SubNets* as rows and *SubGraphs* as columns.

SushiSched DESIGN

Here we describe the principle of **SushiSched**.

Per-Query SubNet (SN_t) *Selection.* As shown in Figure 2(a), the scheduler needs to decide from two policies, namely, 1) serve strictly better accuracy or 2) serve strictly lesser latency, which can be specified by the user. In case of strictly better accuracy, the scheduler serves a *SubNet* that has minimum latency among all the *SubNets* that have accuracy $\geq A_t$. It may be possible that the served latency might not satisfy the latency constraint of $\leq L_t$.

In case of strictly lesser latency, the scheduler serves a *SubNet* that has maximum accuracy among all the *SubNets* that have latency $\leq L_t$. Similarly, it is possible that the served accuracy might not satisfy the accuracy constraint of $\geq A_t$.

The accuracy for a given *SubNet* is fixed, whereas the latency depends on the *SubGraph* cached into the persistent buffer (PB). The scheduler employs a *Latency – Table* to get the latency values for *SubNet* given a cache state.

Encoding SubGraph *DNN Architecture.* The scheduler represents both the *SubNets* and the *SubGraphs* as a vector, as shown in Figure 2(b). The scheduler uses the number of kernels K_i and channels C_i of every layer i to create a vector of size $2N$ for $N-$ layered DNNs.

It is noteworthy that C and K are enough to encode *SubNet* dimensions. Elastic dimensions change the width or depth of *SuperNets*. The width is represented by simply changing C and K, and depth is represented by making C and K equal to zero. If both C and K are zero, that means that that layer is skipped.

Amortizing Caching Choices. The scheduler keeps a running average of the past Q *SubNets* that were served by the scheduler, as shown in Figure 2(b) (AvgNet). The running average serves as a good indicator of the kernels and the channels that were frequently used in the *SubNets* that were served for the past Q queries. The larger the C and K values for a specific layer in the AvgNet, the more frequently that that layer is used in the past Q queries.

It is noteworthy that other cache replacement policies (e.g., least recently used and first in, first out) make sense when we have multiple options to evict. They imply finer granularity of caching, whereas in SGS, caching operates at the granularity of entire *SubGraphs*. Thus, those policies are not applicable.

Predicting the Next SubGraph (S_{t+Q}). The scheduler employs the distance from the running average of the past Q queries to predict the next *SubGraph* to be cached, as shown in Figure 2(b). The scheduler caches the *SubGraph* that has the minimum distance from the average *SubNet*. A minimum distance ensures that the most frequent kernels and channels are cached into the PB. A minimum distance from average *SubNet* ensures that we are picking the best-fit *SubGraph* in terms of frequently occurring channels and kernels in the served *SubNets*.

SERVING *SuperNet*: HARDWARE

Hardware Design Challenges

To support *SGS*, we propose augmenting DNN accelerators with a custom cache called the *PB*. Introduction of the PB leads to a new design space because it competes for a finite on-chip buffer capacity [which needs to be partitioned across input activation (iAct), weight, output activation tiles, and also shared weights].

Architectural Components

In this section, we introduce components of `SushiAccel` [Figure 3(a)] and how they support all the proposed data reuse in Figure 3.

Compute Array

Dot Product Engine (DPR). The key building block of DNN accelerators is the ability to compute *dot products*. We picked fixed-size DPEs of size nine. Larger kernels will be broken down into a serial of 3×3 kernels and get flattened across the multipliers for reduction using the adder tree. As for small kernels (e.g., 1×1), the C dimension will be flattened across multipliers to leverage input channel parallelism.

Parallelism. The number of rows indicates the total number of kernels being processed in parallel in the DPE array, i.e., kernel-level parallelism (K_P) in Figure 3(a). Although the number of columns stands for the total number of iAct channels being processed in parallel, i.e., channel-level parallelism (C_P). To leverage parallelism, **`SushiAccel`** first loads weights and then stores and forwards iActs to reuse them from top to bottom.

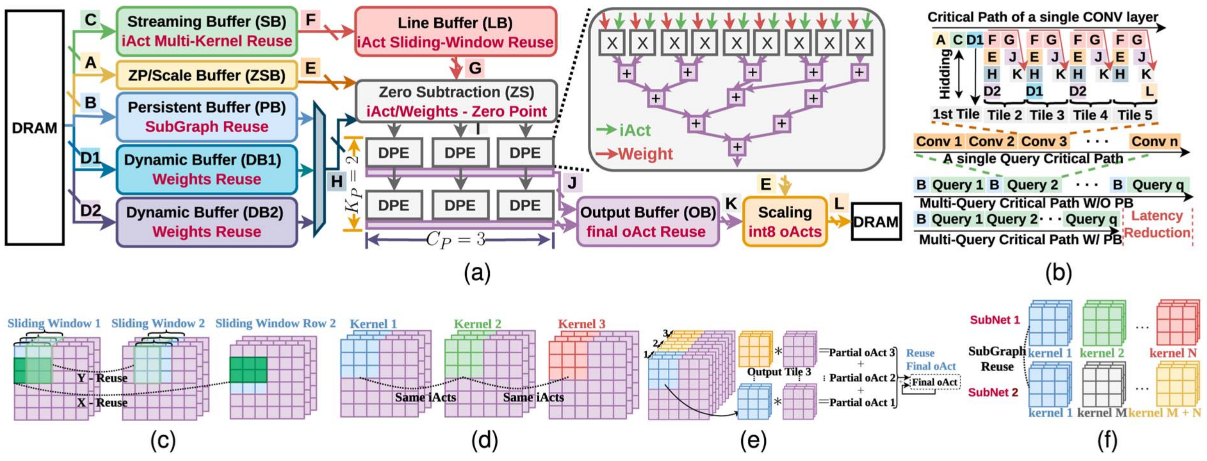

FIGURE 3. `SushiAccel` architecture and corresponding single convolutional (conv) and multiquery scheduling. (a) Overall `SushiAccel` architecture $(K_P = 2, C_P = 3)$. (b) Dataflow overview. (c) iAct sliding window reuse. (d) iAct multifilter reuse. (e) oAct reuse. (f) *SubGraph* reuse. iAct: input activation; oAct: output activation.

On-Chip Buffers and Supported Data Reuse

We designed a custom on-chip buffer hierarchy to support reuse opportunities not leveraged by the DPE array. Each data type has its own separate buffers, as illustrated by different colors in Figure 3(a).

Persistent Buffer. The PB is designed to enable *SubGraph* reuse. For example, **`SushiAccel`** loads the *SubGraph* (kernel 1) in Figure 3(f) from off-chip memory only once and stores it inside the PB such that it could be reused when switching between *SubNet* 1 and *SubNet* 2.

Dynamic Buffer (DB). The DB is a typical on-chip storage device that stores the distinct weights of the requested *SubNet*. By adopting a PB, only noncommon weights need to be fetched from the off-chip to the on-chip storage. For example, in Figure 3(f), all kernels except the common part (kernel 2 to kernel N) will be loaded into the DB when targeting at *SubNet* 1 and will be replaced by kernel M to kernel $M+N$ when switching into *SubNet* 2. The DB Is implemented as a ping-pong buffer, as indicated by DB1 and DB2 in Figure 3(a), to hide the latency of fetching distinct weights from the off-chip dynamic random-access memory (DRAM).

Zero Point (ZP)/Scale Buffer (ZSB). The ZP/ZSB stores zero point and scale for quantized inference.

Further, we implement a streaming buffer, line buffer, and an output buffer to enable *iAct reuse—multiple kernels* [Figure 3(d)], *iAct—sliding-window reuse* [Figure 3(c)], and oAct reuse [Figure 3(e)], separately.

`SushiAccel` DATAFLOW AND SCHEDULING

Latency Reduction From Interquery Scheduling

The interquery scheduling of **`SushiAccel`** is shown in Figure 3(b), where stage B indicates the movement of the common *SubGraph* from an off-chip to an on-chip PB. The latency saving of **`SushiAccel`** comes from eliminating the redundant off-chip *SubGraph* access, as illustrated in Figure 3(b), where **`SushiAccel`** reduces multiple common *SubGraph* off-chip access (stage B) to only once, compared to the design without a PB.

Hiding Latency From Intralayer Dataflow

Within each convolution layer, **`SushiAccel`** processes a convolution layer in the granularity of weight tiles shown in Figure 3(b). Different stages (i.e., A–L) are defined in Figure 3(a), which represent the movement of specific data.

To further hide off-chip data access latency from the critical path, we implement a double-distinct weights buffer [ping-pong DBs, i.e., DB1 and DB2 shown in Figure 3(a) to hide the off-chip latency of fetching distinct weights behind the computation latency]. This is indicated by stages D1 and D2, which are hidden from stages F, G, J, and K [the red arrows in Figure 3(b)].

EXPERIMENTAL RESULTS

System Setup

Workload

We choose the weight-shared version of ResNet50 and MobV3. By utilizing Sahni et al.,[4] we obtain a sequence of six *SubNets* from ResNet50, with sizes ranging from [7.58 MB, 27.47 MB], and 7 *SubNets* from MobV3 ranging from 2.97 MB to 4.74 MB.[a] Shared weights take up 7.55 MB and 2.90 MB for ResNet50 and MobV3, separately.

Deployment Platforms

We implemented the proposed **`SushiAccel`** on ZCU104 as a small field-programmable gate array (FPGA) board. We compare our **`SushiAccel`** with the PB and without the PB and the CPU (Intel i7, 45 W).

Scheduler Simulator

We developed an analytic model that estimates the behavior of **`SushiAccel`** to explore the design space by configuring the architecture with parameters. We also developed **`SushiSched`**, which runs on the CPU.

SuperNet Training Evaluation

Although OFA requires 978 h for training, CompOFA[4] needs only 494 h (a $2\times$ reduction). In addition, CompOFA reaches a $216\times$ speedup in model search/extraction time compared to OFA.

A latency–accuracy comparison of CompOFA with a fixed/elastic kernel size and OFA networks on an Intel CPU and a Samsung cellphone is shown in Figure 4(a) and (b). The best model in CompOFA for the evaluated latency targets is as accurate as OFA, despite its significantly smaller family size and training budget. The results validate our intuition behind the CompOFA heuristic, which leads to a smaller design space without degrading the latency/accuracy of Pareto optimality.

`SushiAccel` Evaluation

We evaluate how **`SushiAccel`** contributes to latency and energy reduction. We run the 3×3 convolution layers of ResNet50. The **`SushiAccel`** on ZCU104 has an off-chip bandwidth of 9.6 GB/s, a PB size of 1.7 MB, and throughput of 0.2592 tera-floating point operations per second when running at 100 MHz.

[a]The selected *SubNets* are shown by A–F and A–G in the Figure 4(c) and (d).

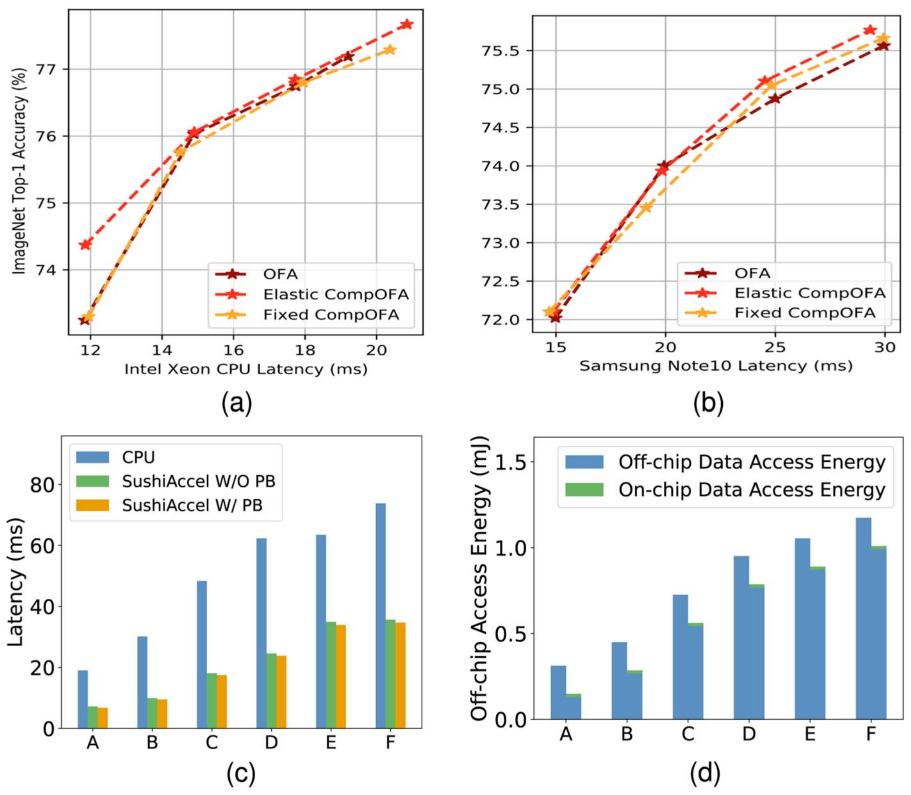

FIGURE 4. Real hardware evaluations. (top) Latency–accuracy tradeoff space and (bottom) the FPGA measurement [the left and right bars in (c) are **SushiAccel** without the PB and with the PB]. (a) Intel (CompOFA versus OFA). (b) Samsung (CompOFA versus OFA). (c) ZCU104 (Latency). (d) ZCU104 (Energy).

Latency Evaluation

The real board results are shown in Figure 4(c), where the **SushiAccel** without and with the PB achieves a [1.81×, 3.04×] and [1.87×, 3.17×] speedup, respectively, over the CPU. And the *SubGraph* reuse could boost extra [2.86%, 6.48%] speedup for **SushiAccel** running on the ZCU104 for a single query.

Energy Evaluation

Energy spent on data movement has been proven to dominate energy consumption in DNN accelerators. We estimate the overall off-chip energy for several platforms in Figure 4(d). We estimate the off-chip energy by profiling the DRAM data access and compute it as *NumberAccess* × *EnergyPerAccess*. Compared to with and without the PB design, the proposed *SubGraph* reuse can save [14%, 52.6%] and [43.6%, 78.7%] off-chip data access energy of ResNet50 (MobV3), respectively, for different *SubNets*!

SushiSched Evaluation

Figure 5(a) and (b) show that **SushiSched** is able to serve queries with strictly lesser latency and/or better accuracy for ResNet50. The blue dots represent the served queries by employing **SushiSched**. We observed similar trends for MobV3.

End-to-End **SUSHI** Evaluation

In this section, we compare the latency–accuracy tradeoff results among **SUSHI** without the PB, **SUSHI** with the PB (state-unaware caching), and **SUSHI**. The blue dots in Figure 5(c) and (d) illustrate how **SUSHI** serves random queries.[b]

For ResNet50, in all cases in Figure 5(c), **SUSHI** with the PB consistently outperforms **SUSHI** without the PB. And **SUSHI** increases the serving accuracy up to 0.98% given the same latency. It is also able to decrease the latency by 2.4 ms given the same accuracy. For MobV3, **SUSHI** offers better accuracy latency except for a few points, which is slightly worse than with the PB. Due to the small size of MobV3, a big portion of *SubNet* can be fit into the PB. Thus, for a few queries, it may be

[b]Due to the overlapping, only limited points in the figures are visible.

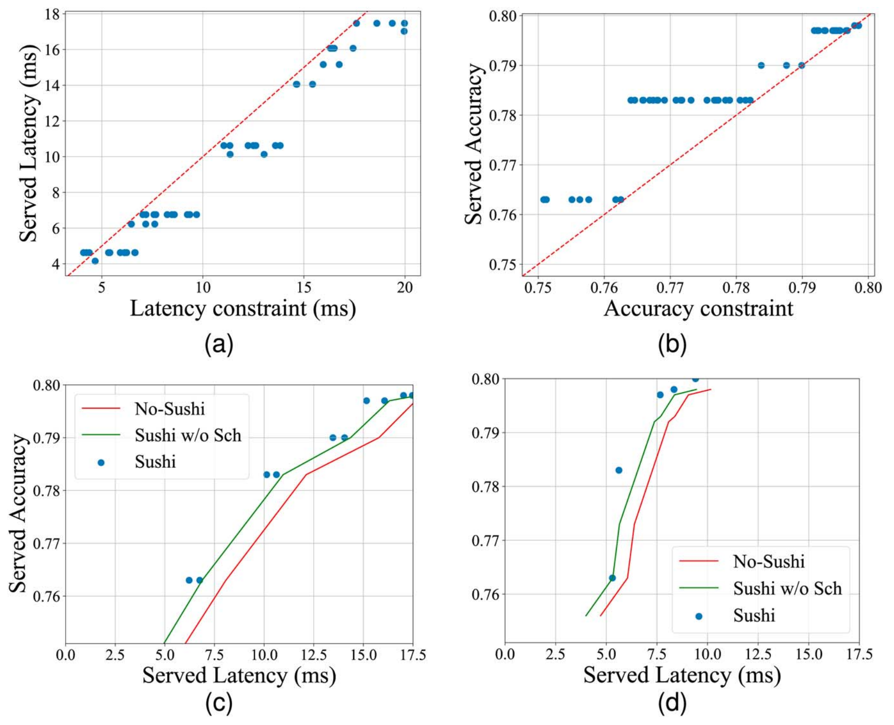

FIGURE 5. (top) Serving strictly lesser latency and higher accuracy. (a) Lesser latency for ResNet50. (b) Higher accuracy for ResNet50. (bottom) Comparing SUSHI latency-accuracy trade0ff space with design without SUSHI and SUSHI without scheduler for (c) ResNet50 and (d) MobV3. sch: scheduling.

beneficial to use the current content rather than update the PB at every "Q" queries.

ABLATION STUDY

PB Size

When the PB size is less than the minimum SubNet size, increasing the PB size results in further latency reduction for **SushiAccel** [Figure 6(a) and (b)]. Even with a fixed PB size, the amount of improvement can be changed depending on what is in the PB (up to 7×). **SushiSched** can help us to provide the *SubGraph* that can provide maximum latency reduction.

Impact of PB Hit Ratio on Latency

Due to the novel concept of SGS cache reuse, we define the cache hit ratio as a fraction of *SubGraph* weights in the PB that were used (hit) by a served *SubNet*, relative to *SubNet* size. More precisely, the **SUSHI** cache hit ratio is *size_of_used_SubGraph_in_PB/size_of_served_SubNet*. Intuitively, higher temporal locality should lead to a higher cache hit ratio as more of the cached *SubGraph* weights are reused across queries. This property directly stems from the weight-shared nature of the activated *SubNets*. Cache hit ratio becomes a function of the workload, quality of **SushiSched** decisions, and the properties of **SushiAccel** (such as PB size). We validate that better temporal locality of the workload leads to a better cache hit ratio and correlates to higher latency performance improvement. We consider *MinNetTrace* (*MaxNetTrace*), a set of queries annotated with latency that only the smallest (largest) *SubNet* can satisfy. We also define a step function that steps through *SubNets* such that each next *SubNet* is a superset of the previous one, starting from minimum *SubNet* to maximum *SubNet*.

Figure 6(c) shows that for a given model architecture, all workloads consistently perform between the "guardrails" set by the *MinNetTrace* and the *MaxNetTrace*. This is useful as it defines the range of operation for **SushiSched**. The traces that activate the exact same *SubNet* exhibit maximum locality. Further, we show that the step trace has a higher hit ratio than a random trace, with queries randomly annotated. Importantly, this correlates with the better latency performance in Figure 6(d) and (e). Thus, **SUSHI** offers a convenient method of evaluating its relative performance by using

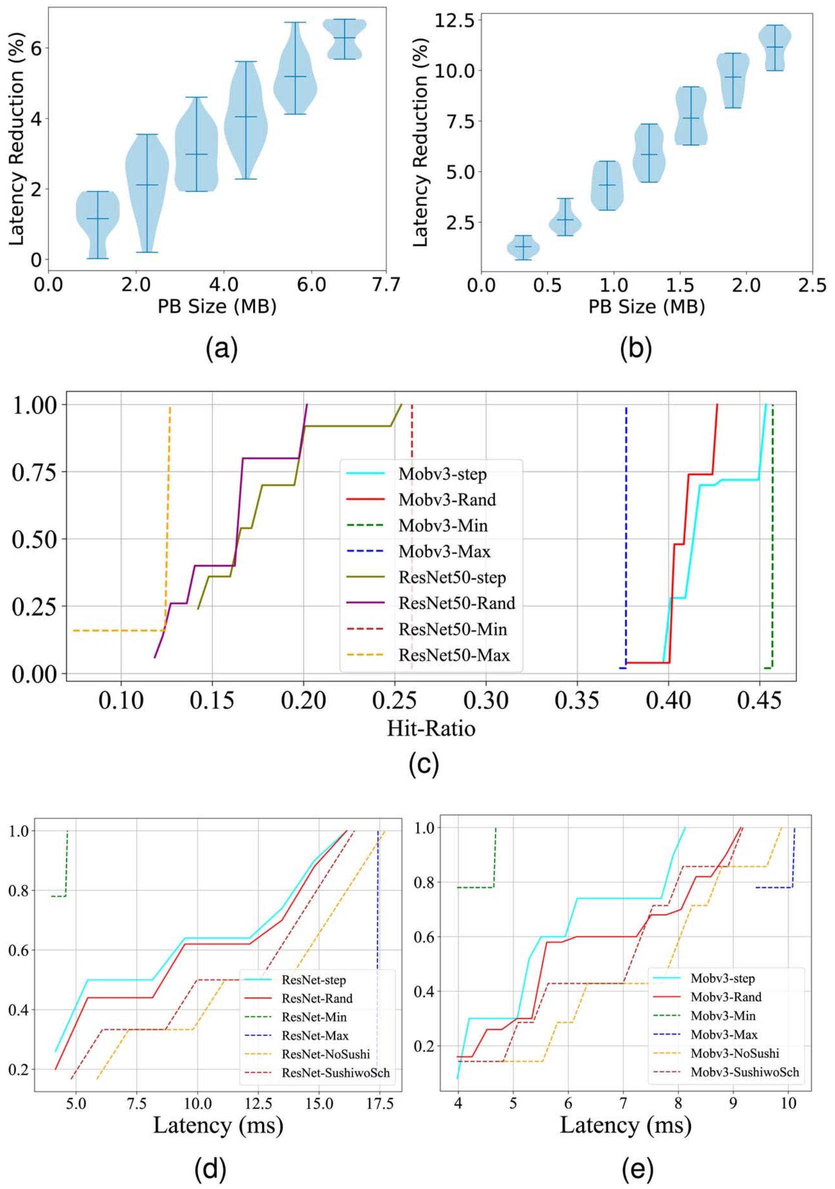

FIGURE 6. Ablation study. Latency reduction by increasing (top) PB size and (bottom) effect of higher temporal locality and model architecture on hit ratio and latency. (a) PB size ResNet50. (b) PB size MobV3. (c) Hit ratio CDF (lower right is better). (d) ResNet50 latency CDF. (e) MobV3 latency CDF.

the newly defined SGS cache hit ratio. We observe that for a stream of queries (e.g., 50), **SUSHI** improves the latency by 29% (32%) compared to **SUSHI** without a scheduler for ResNet50 (MobV3).

CONCLUSION

SUSHI is a vertically integrated inference-serving stack that takes advantage of the temporal locality induced by queries on the same weight-shared *SuperNet*. **SUSHI** proposes a novel *SGS* optimization for *SuperNet* inference across queries. **SUSHI** is a vertically co-designed and integrated scheduler and FPGA implementation with a layer of abstraction in between that generalizes the design. **SUSHI** can be integrated into state-of-the-art ML inference-serving frameworks and enables better latency/accuracy tradeoffs when serving a stream of queries with latency–accuracy constraints in real time.

ACKNOWLEDGMENT

This material is based on work partially supported by the National Science Foundation under Grant CCF-2029004, Semiconductor Research Corporation, and the Qualcomm Innovation Fellowship Award. Additional support was provided by a sponsored research award by Cisco Research.

REFERENCES

1. S. Hong et al., "HOLMES: Health OnLine model ensemble serving for deep learning models in intensive care units," in *Proc. 26th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining (KDD)*, 2020, pp. 1614–1624, doi: 10.1145/3394486.3403212.
2. I. Gog et al., "D3: A dynamic deadline-driven approach for building autonomous vehicles," in *Proc. 17th Eur. Conf. Comput. Syst. (EuroSys)*, 2022, pp. 453–471, doi: 10.1145/3492321.3519576.
3. H. Cai et al., "Once for all: Train one network and specialize it for efficient deployment," in *Proc. ICLR*, 2022.
4. M. Sahni et al., "CompOFA: Compound once-for-all networks for faster multi-platform deployment," in *Proc. ICLR*, 2021.
5. H. Kwon et al., "MAESTRO: A data-centric approach to understand reuse, performance, and hardware cost of DNN mappings," *IEEE Micro*, vol. 40, no. 3, pp. 20–29, May/Jun. 2020, doi: 10.1109/MM.2020.2985963.
6. P. Behnam et al., "Subgraph stationary hardware-software inference co-design," in *Proc. MLSys*, 2023.

PAYMAN BEHNAM is a Ph.D. candidate majoring in electrical and computer engineering at the Georgia Institute of Technology, Atlanta, GA, 30332, USA. His research interests are at the intersection of machine learning, systems, and hardware. He is the co-corresponding author of this article. Contact him at paymabn.behnam@gatech.edu.

JIANMING TONG is a Ph.D. candidate majoring in computer science at the Georgia Institute of Technology, Atlanta, GA, 30332, USA. His research interests are focused on AI acceleration and privacy-perserving AI. He is the co-corresponding author of this article. Contact him at jianming.tong@gatech.edu.

ALIND KHARE is a Ph.D. candidate majoring in computer science at the Georgia Institute of Technology, Atlanta, GA, 30332, USA. His research interests are at the intersection of machine learning and systems. Contact him at alind.khare@gatech.edu.

YANGYU CHEN was a master's student in electrical and computer engineering at the Georgia Institute of Technology, Atlanta, GA, 30332, USA. His research interests include computer architecture and deep neural network accelerators. Contact him at yangyuchen@gatech.edu.

YUE PAN is a Ph.D. candidate majoring in computer science and engneering at the University of California, San Diego. His research interests include computer architecture and domain-specific accelerators. Contact him at yup014@ucsd.edu.

PRANAV GADIKAR is a master's student majoring in computer science at the Georgia Institute of Technology, Atlanta, GA, 30332, USA. His research interest is focused on systems for machine learning. Contact him at pranav.gadikar@gatech.edu.

ABHIMANYU BAMBHANIYA is a Ph.D. candidate majoring in electrical and computer engineering at the Georgia Institute of Technology, Atlanta, GA, 30332, USA. His research interests include artificial intelligence accelerators and deep neural networks. He is a Graduate Student Member of IEEE. Contact him at abambhaniya3@gatech.edu.

TUSHAR KRISHNA is an associate professor of electrical and computer engineering/computer science at the Georgia Institute of Technology, Atlanta, GA, 30332, USA. His research interests include computer architecture, on-chip networks, and deep learning accelerators. Krishna received his Ph.D. degree in electrical engineering and computer science from MIT. He is a Senior Member of IEEE. Contact him at tushar@ece.gatech.edu.

ALEXEY TUMANOV is an assistant professor of computer science/electrical and computer engineering at the Georgia Institute of Technology, Atlanta, GA, 30332, USA. His research interest is focused on systems support for soft real-time machine learning and distributed machine learning frameworks. He received his Ph.D. degree from Carnegie Mellon University. Contact him at atumanov@gatech.edu.