INFSCALER: Enabling Efficient ML Inference Serving on Multi-Accelerator Edge Devices via Asymmetric Auto-Scaling

Borui Li, Tiange Xia, Shuai Wang, Shuai Wang✉
School of Computer Science and Engineering, Southeast University, Nanjing, China
Email: {libr, xia_tg, shuaiwang_iot, shuaiwang}@seu.edu.cn

***Abstract*—Nowadays, there is a growing trend to deploy machine learning (ML) models on edge devices. To cope with the increasing resource requirements of current ML models, multi-accelerator edge devices that integrate CPU, GPU, NPU, or TPU in a single SoC gain popularity. However, we observe that existing ML inference serving frameworks are poor in utilizing the unique hardware architecture of these edge devices. In this paper, we present INFSCALER, an efficient ML inference serving framework tailored for multi-accelerator edge devices. INFSCALER discovers the architectural bottleneck of ML models and designs a bottleneck-aware asymmetric auto-scaling technique to facilitate efficient resource allocation for ML models on the edge. Furthermore, INFSCALER capitalizes on the hardware's unified memory feature inherent to edge devices, ensuring efficient data sharing between the asymmetrically scaled model partitions. Our experimental results show that INFSCALER achieves up to 126.59% throughput improvement and 27.32% resource reduction while satisfying the latency requirements compared with the state-of-the-art inference serving approaches.**

I. INTRODUCTION

Recently, deploying machine learning (ML) models on edge computing devices rather than the cloud has gained significant attention [1], [2], [3]. Building on this momentum, various merchants such as NVIDIA [4], AMD [5], and Google [6] present multi-accelerator edge devices to provide better ML inference performance. For example, the NVIDIA Jetson series leverages System-on-Chip (SoC) architecture that integrates a CPU, GPU, and NVIDIA Deep Learning Accelerator (NVDLA) onto a single chip. These multi-accelerator devices are now widely used in typical edge environments such as autonomous driving [7], [8] and edge video analytics [9], [10].

Nevertheless, the energy and space limitation of edge environments makes it hard to match the increasing resource demand of modern ML algorithms such as large language models (LLMs). Prior attempts for efficient inference serving on resource-constrained edge devices leverage model quantization [11], [12], pruning [13] or sparsity [14]. These approaches require modifying or retraining the original model, which will lead to an accuracy drop or excessive training overhead.

If we refer to ML serving approaches without modifying the model, which are mostly designed for the cloud, the primary solutions are request batching [15], [16] and instance auto-scaling [17]. The ingress requests are organized into batches to leverage the parallel processing capabilities of hardware accelerators to improve efficiency. Once the request bursts and results in the resource usage of an inference instance exceeding a pre-defined resource cap, the cloud server will scale out, i.e., spawn new serving instances, for higher parallelism.

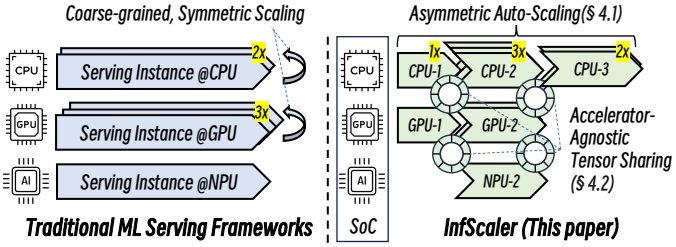

Fig. 1. Illustration of the difference between INFSCALER and existing works.

However, directly adopting these cloud-based solutions is problematic for edge environments because they do not take the unique hardware characteristics of edge devices into consideration. (1) Existing works [15], [16], [17] replicate the *whole ML model* to serve additional requests when scaling out, but edge devices may not be facilitated with enough resources for this coarse-grained scaling. For example, inferencing the `gpt2-medium` model requires ∼6GB memory. While NVIDIA Xavier NX, a typical edge device, is only equipped with 8GB which could not scale the whole model out for bursty requests. (2) Existing ML serving frameworks [18] treat accelerators such as GPU and NPU as peripherals with independent memory management. However, there is a new opportunity for multiple accelerators to cooperate more tightly due to the hardware unified memory design at the edge.

Towards the above problems, we present INFSCALER, an efficient ML inference serving framework tailored for edge devices. Specifically, INFSCALER aims to achieve better inference serving performance (i.e., throughput) while satisfying the latency constraint of requests under a given resource limitation. Considering the multi-accelerator architecture and resource-constrained nature of edge devices, the key idea of INFSCALER is scaling each group of ML operators, i.e., the building blocks of an ML model, with different scaling ratios for better resource efficiency. Unlike symmetrically scaling all operators as a whole model, we call this approach *asymmetric auto-scaling*, as shown in Fig. 1. This approach is based on our observation (cf. §II-B) that ML models inherently contain architectural bottlenecks due to the resource demands of ML operators and their affinity towards accelerators, which negatively impact overall inference performance. Selectively scaling out the bottlenecks can achieve similar, even better,

We thank all the reviewers for their valuable comments. This work was supported in part by the National Natural Science Foundation of China under Grant 62302096, 62272098, and U24B20152; in part by the Natural Science Foundation of Jiangsu Province under Grant BK20230813; and in part by the Zhishan Young Scholar Program of Southeast University under Grant 3209002402A2. Shuai Wang is the corresponding author.

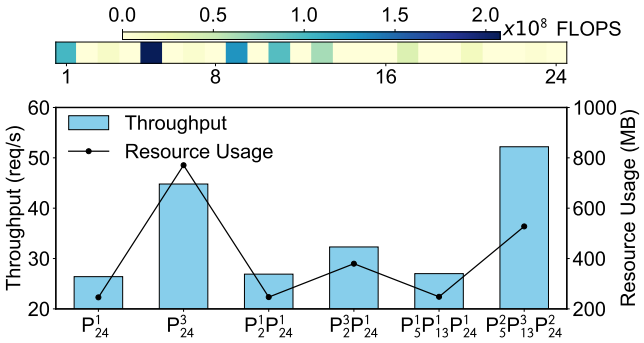

Fig. 2. Illustration of the computation complexity of AlexNet (top) and its inference performance under different settings (bottom). P_x^a denotes the model is partitioned at the end of operator #x and scaled out to a instances.

performance compared to whole model replication.

To facilitate asymmetric auto-scaling, we first identify the bottlenecks through offline profiling and partition the ML model into several groups of operators with fully aware of the bottleneck. At runtime, we perform the asymmetric auto-scaling and also take advantage of the hardware unified memory between heterogeneous accelerators to achieve efficient *accelerator-agnostic tensor sharing* between the partitioned models. Our contributions can be summarized as follows:

- We discover the architectural bottleneck of ML models and its negative impact on the inference performance. Based on this, we propose a *bottleneck-aware asymmetric auto-scaling* strategy to utilize the precious resources on the edge wisely for better performance.
- We advocate an *accelerator-agnostic tensor sharing* mechanism by taking advantage of the hardware unified memory property on edge devices.
- We implement INFSCALER and evaluate its performance extensively. Results show that INFSCALER can improve the inference throughput by 34.78%-126.59% and reduce 10.49%-27.32% resource usage while satisfying the latency constraints.

II. BACKGROUND AND MOTIVATION

A. Hardware Architecture of Edge Devices

Nowadays, edge devices are equipped with AI accelerators like GPUs, TPUs, and FPGAs to enhance performance while keeping energy efficiency in ML processing. The most prevalent edge ML devices are NVIDIA Jetson series, which are widely used in autonomous driving, VR/AR, and robotics. These devices include three generations identified as NVIDIA Jetson, Xavier and Orin. For example, on NVIDIA AGX Xavier, an 8-core Armv8.2 CPU, an integrated NVIDIA Volta GPU (iGPU), and two NVDLA accelerators are located on the same SoC, which is different from the desktop and servers.

Unified memory is originally a software concept in CUDA programming aimed at simplifying CPU RAM and dGPU VRAM management through unified addressing. NVIDIA Jetson series leverages a *hardware unified memory* architecture where all the accelerators share a common hardware main memory. From the second generation (Xavier) on, hardware cache coherence between CPU and iGPU further accelerates memory sharing. This distinct feature provides a promising starting point for our data-plane acceleration for serverless inference on edge devices.

B. Architectural Bottleneck of ML Models

ML bottlenecks. ML models include multiple computation operators such as convolution, pooling, and normalization. However, the computational complexity disparity among operators and their affinity towards heterogeneous accelerators results in some of them serving as a bottleneck of the whole model. To examplify, we use a representative ML model, AlexNet, and study its inference performance on the NVIDIA Xavier AGX multi-accelerator edge device in Fig. 2.

> **Observation 1:** ML models inherently contain bottlenecks due to the unbalanced distribution of the computation complexity, which hinders the overall inference performance.

We can see from the figure that certain operators (e.g., #1, #5, #13) exhibit notably higher complexity compared to others, resulting in their computation time being prominently longer than other parts, which lowers the overall inference throughput. We identify these layers are the "bottleneck" of AlexNet. To make things worse, the traditional auto-scaling mechanism of ML serving directly scales the whole model when confronted with burst requests. However, on resource-constrained edge devices, this coarse-grained scaling mechanism uniformly allocates additional resources to all layers, potentially resulting in a waste of precious resources. *Consequently, a novel fine-grained scaling approach is imperative for serverless ML inference on edge devices.*

Impact of asymmetric auto-scaling. Based on the above observation, we further investigate how to deal with the architectural bottleneck of ML models for better inference. We quantify the inference throughput and resource usage under the same predefined resource cap. Results in Fig. 2 show that $P_5^2P_{13}^3P_{24}^2$ achieves the best performance among the six configurations. It exhibits 16.5% throughput improvement against the second-performed baseline P_{24}^3, i.e., no partition and scaling to three instances, with 31.5% less resources used. The key takeaway of this improvement is this asymmetric scaling can leverage wiser resource usage, which is allocating more resources to the bottlenecks instead of distributing them equally. Moreover, this improvement is also fragile. Also, from the results of other partitioned baselines, we can see that it requires a prudent model partition and scaling policy to achieve optimal performance.

> **Observation 2:** Asymmetric auto-scaling, i.e., selectively scaling out the bottleneck with an appropriate ratio, can enhance the throughput performance with the same resources.

The above results demonstrate the potential for fine-grained, asymmetric auto-scaling on edge devices. *However, how to wisely partition the model considering bottleneck and scales out for optimized performance remains unsolved.*

III. INFSCALER OVERVIEW

A. Design Principles

Throughout our system design of INFSCALER, we adhere to the following guiding principles. **(1) User-transparent.** As an ML inference serving framework, INFSCALER should ensure users remain unaware of underlying complexities such

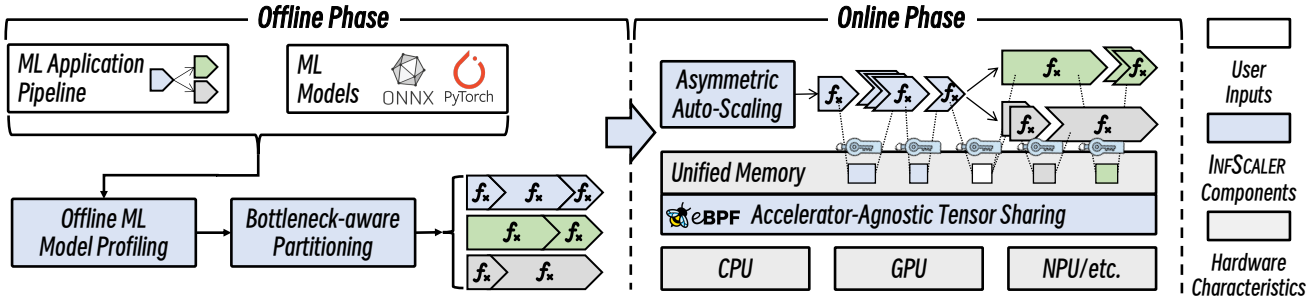

Fig. 3. System overview of INFSCALER framework.

as model partitions and fine-grained scaling mechanisms. Additionally, INFSCALER should maintain model accuracy without altering its structure to achieve this transparency. **(2) Hardware-oriented.** INFSCALER targets the ML inference on edge devices facilitated with heterogeneous accelerators such as the NVIDIA Xavier series. Hence, we should fully utilize the hardware characteristics of edge devices by design to achieve better performance. **(3) Throughput-aware.** Maximizing throughput ensures INFSCALER achieves an efficient usage of the limited resources on edge devices to process a high volume of concurrent requests. Hence, we design INFSCALER to maximize the throughput while satisfying the latency service level objective (SLO) specified by users.

B. INFSCALER *in a nutshell*

Fig. 3 depicts the bird's-eye view of INFSCALER system. It has an offline phase to generate the optimized function deployment and an online phase to further improve the performance.

Offline deployment optimization. In the offline phase, users submit their ML application to be deployed and the corresponding ML models. Similar to the existing practices such as the inference graph of KServe [19] and SageMaker Pipeline of AWS [20], INFSCALER supports submitting an ML application including multiple ML algorithms that formulate a *pipeline*. For example, an NLP pipeline needs to first run document classification, and then perform named entity detection downstream based on the previous classification results. As for ML models, INFSCALER do not require any modification to ML models and users can directly submit the *unmodified* ML models such as ONNX or PyTorch. One of the main design principles of INFSCALER is to maximize the inference throughput while satisfying the latency constraint. Hence, it is necessary to address the architectural bottlenecks of ML models during inference. A naïve idea is to allocate more resources to the bottleneck layers for better performance. However, traditional ML serving frameworks cannot precisely allocate additional resources only to the bottleneck because they leverage a coarse-grained scaling mechanism. Hence, to obtain the opportunity for precise resource allocation, INFSCALER first profiles for detailed performance metrics (e.g., FLOPs, execution time) and partitions the ML model into slices for asymmetric auto-scaling.

Online ML inference serving. During the online serving process, INFSCALER optimizes the inference performance in two ways. First, INFSCALER leverages asymmetric auto-scaling in correspondence to the ingress request variations. With the help of bottleneck-aware partitioning, INFSCALER can only scale up the bottleneck part of the model to achieve better resource efficiency. Second, INFSCALER achieves systematic support for multi-accelerator edge devices by allowing the asymmetric auto-scaling to scale part of the model to all accelerators. Furthermore, INFSCALER designs an accelerator-agnostic tensor sharing mechanism that leverages the hardware unified memory to efficiently share outputs between model partitions.

IV. DESIGN OF INFSCALER

A. Asymmetric Auto-Scaling of ML models

As we demonstrated in §II-B, ML models include (multiple) architectural bottlenecks that impede inference efficiency. In this section, we show our detailed designs about how to achieve asymmetric auto-scaling while fully aware of the bottlenecks of ML models. We first detect the bottleneck and figure out the optimal scaling-out policies. After that, we move on to the resource allocation problem with a given partition because the allocation of CPU and memory poses a significant influence on the overall performance.

Offline bottleneck-aware partitioning. Designing a wise partition algorithm requires us to consider the following trade-off, where more partitions lead to an excessive overhead for inter-partition communication but fewer partitions are not fine-grained enough for efficient resource allocation. According to §II-B, we find that computation complexity can be a source of inference bottlenecks. To address the partition granularity and overhead tradeoff, the main idea of INFSCALER's partitioning is to balance the computation complexity among the operators in a partition. The intuition here is each partition is also a minimum resource allocation unit. Operators inside a partition scale up and down at the same time, which means the resources are allocated evenly in a partition. If the partition contains several operators with massive different complexities, the largest operator will further become the new bottleneck in the partition that impedes the overall performance.

Therefore, our bottleneck-aware partitioning problem could be formulated as an optimization problem that minimizes the variance of the computation complexity of each group while keeping the latency satisfies the user-specified SLO:

$$\arg\min_{P_1,\cdots,P_k} \sum_{k=1}^{K} \sigma(P_k)$$
$$\text{s.t.} \begin{cases} \hat{T_o} + (K-1)T_p \leq SLO \\ P_k = \{c_n | n \in [p_{k-1}+1, p_k], c_n > \overline{C_N}\} \\ P_1 \cup \cdots \cup P_K = C_N \end{cases}, \quad (1)$$

Algorithm 1: Asymmetric auto-scaling policy

Input: $\mathcal{B} = \{B_1, \cdots, B_K\}$: set of bottleneck indicator;
$A_k^m \in \mathcal{A}_k$, u_k^m: hardware affinity and expected resource usage of partition k on accelerator m;
U^m, R^m: resource used and limit of accelerator m;
Output: Number of scaled instance $\mathcal{S} = \{S_1, \cdots, S_K\}$

1 **foreach** *SLO unsatisfaction detected* **do**
2 $\quad k^* \leftarrow \arg\max_{k=1}^{K} B_k, B_k^* \leftarrow \max_{k=1}^{K} B_k$;
3 $\quad$ **foreach** $A_{k^*}^m \in sort(\mathcal{A}_{k^*})$ **do**
4 $\quad\quad$ **if** $U^m \leftarrow U^m + r_{k^*}^m < R^m$ **then**
5 $\quad\quad\quad B_{k^*} \leftarrow \frac{B_{k^*} S_{k^*}}{S_{k^*}+1}$;
6 $\quad\quad\quad U^m \leftarrow U^m + r_{k^*}^m$; $S_{k^*} \leftarrow S_{k^*} + 1$;
7 $\quad\quad\quad$ **break**;

which partitions an ML model with N operators into K partitions. $\sigma()$ stands for the calculation of standard variance. P_k and p_k are the computation complexity set and the index of the last operator of partition k. $c_n \in C_N$ is the computation complexity of n-th operator. $\hat{T}_o$ means the predicted model execution time, T_p denotes the overhead introduced by partitioning. Instead of using all layers, INFSCALER only considers the representative layers whose complexity is bigger than the model average in this optimization (constraint #1). This is because compared with the bottleneck layers, these small layers have a very limited impact on the throughput and resource allocation.

Online asymmetric auto-scaling policy. By "asymmetric," we mean that (1) the scaling factors of each partition can differ from one another, and (2) partitions can be scaled to different types of accelerators.

Algorithm 1 depicts the auto-scaling policy. Once detected the serving performance cannot meet the user-specified latency SLO, INFSCALER starts auto-scaling. First, we identify the partition k with the maximum *bottleneck indicator* B_k (line 2), which is defined as the sum of the computation complexity of each operator in the partition $\sum P_k$, serving as the target for scaling out. This is because the partition with the highest B_k has the greatest computational complexity and thus requires scaling first. Then, INFSCALER determines on which accelerator to scale partition k with the help of the profiled hardware affinity, e.g., execution latency, and checks if the accelerator has enough resources to accommodate it (lines 3, 4). If yes, we reduce the bottleneck indicator of partition k to adjust its future priority in scaling (line 5), record the resource usage, and finally perform scaling (line 6). With this asymmetric auto-scaling policy, INFSCALER dynamically scales out the partition that needs more resources the most, which leads to better resource efficiency and inference performance.

B. Accelerator-Agnostic Tensor Sharing

Although asymmetric auto-scaling brings intrinsic performance gains, it also incurs more Inter-partition data transmission. Therefore, it is imperative to meticulously devise a systematic data flow design for sharing the input and output tensors of each function.

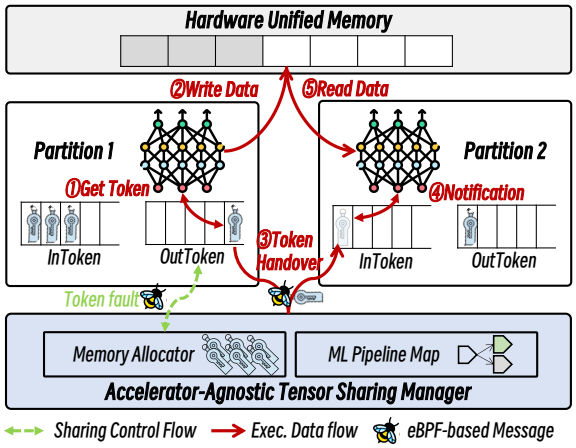

Fig. 4. Illustration of the accelerator-agnostic tensor sharing.

To improve data sharing efficiency, numerous works [21], [17] leverage the zero-copy technique for tensor sharing. Zero-copy means transferring solely an address pointer to shared data instead of transferring all the data itself. Nonetheless, these prior works of zero-copy data sharing devote less attention to the following two unique problems in INFSCALER. (1) The asymmetric auto-scaling technique leads the tensor sharing problem to a multi-producer, multi-consumer (MPMC) memory sharing scenario. For example, once a partition of an ML model scales out asymmetrically to multiple instances, its subsequent partition should acquire data from both the scaled instances, i.e., multi-consumer, and its preceding partition should share data with both instances, i.e., multi-consumer. For example, when a partition of an ML model scales out asymmetrically across multiple instances, it acts as multiple data producers to distribute data to subsequent partitions. by acquiring data from all involved instances. Conversely, its preceding partition also needs to share data to multiple consumers. (2) To make things worse, the data producers and consumers can reside on heterogeneous accelerators. Hence, INFSCALER proposes an *accelerator-agnostic tensor sharing* mechanism to address the aforementioned problems.

eBPF-based event-driven sharing. Towards the MPMC zero-copy sharing problem and preventing the data race between multiple instances, INFSCALER adopts an event-driven sharing mechanism. Specifically, to share the data between instances, the data producer should acquire a token (i.e., memory address) from INFSCALER before it can proceed with writing data. After completing the write operation, the producer then sends this token to the consumer as an event to trigger the reading.

This token-based sharing has two merits. On the one hand, compared with the widely-used polling-based zero-copy sharing approach [17], it reduces the frequency of memory access and achieves lower runtime overhead which is suitable for edge environments. On the other hand, it transforms the ad-hoc usage of shared memory into centralized management, which gives us the chance to holistically manage the load balancing among different scaled instances on heterogeneous accelerators. We further optimize the centralized scheduling overhead by leveraging eBPF for all the control flow communications. An eBPF map is introduced to intercept the

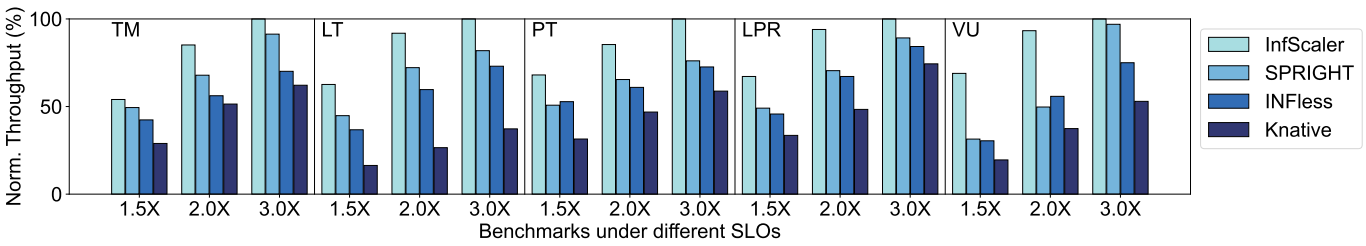

Fig. 5. Comparison of the normalized throughput of INFSCALER under different latency SLO settings against baselines. Here, 1.5× denotes that the SLO is set to 1.5 times of its normal inference time, which we refer to as high pressure. Similarly, 2.0× and 3.0× stands for medium and low pressures, respectively.

token fault for sidecar control flow and the token handover for execution control flow of the inter-partition data sharing. Here raises a new question: why not implement the memory sharing manager directly in an eBPF function? This is because eBPF functions reside in the kernel, which poses strict constraints on execution time. If we push all the logic in the memory sharing manager into an eBPF function, it incurs much overhead for the kernel and may expand the attack surface. Moreover, eBPF functions are not adaptive to the function deployment changes. As INFSCALER is a general ML serving system, applications deployed on top of our framework are versatile. Implementing the memory sharing manager in a separate service in user space is more adaptive for ever-changing applications.

Lazy sharing mechanism. Towards the data sharing between heterogeneous accelerators, we implement the above zero-copy mechanism on the hardware unified memory which all accelerators can access directly. Furthermore, INFSCALER adopts a lazy allocation mechanism to prevent excessive allocations of limited hardware unified memory on edge devices. Lazy means instead of allocating all the shared memory at once, INFSCALER allocates the memory when instances report a memory shortage. We argue that this approach will reach a dynamic balance of the input and output between consecutive partitions.

The whole tensor sharing process is illustrated in Fig. 4. The tensor sharing manager handles the overall new memory allocation and inter-instance token exchange. When partition 1 tends to share the output with partition 2, it first checks if the `OutToken` queue is empty. If there is a token, it writes data to the corresponding region and hands over the token to the consumer. If the `OutToken` is empty, partition 1 will send a token fault to the tensor sharing manager. The sharing manager will select an idle memory region and send its address back for further usage.

V. EVALUATION

A. Experimental Setup

Testbed. We conduct the experiments on the NVIDIA Xavier AGX development board, which is a widely used multi-accelerator edge device.

Baselines. We compare INFSCALER against the following baselines: (1) *INFless* [15]: An ML inference serving framework that leverages auto-scaling and batching to improve resource efficiency. (2) *SPRIGHT* [17]: A state-of-the-art application serving framework that focuses on efficient memory sharing mechanism for pipelined applications. (2) *Knative* [18]: A widely-used open-source ML serving system based on Kubernetes. We apply our partitioned and configured functions on Knative without the hardware-oriented memory sharing to see how it performs.

TABLE I
APPLICATION BENCHMARKS USED FOR EVALUATION.

Benchmark	ML Pipeline	Involved Models
Traffic Monitoring (TM)	❶→❷	Vehicle detection(❶SSD), Vehicle model recognition(❷ShuffleNet)
Language Translation (LT)	❶→❷→❸	Semantic feature extraction(❶Transformer), Cross-language conversion(❷Transformer), Target statement decoding(❸Transformer)
Person Tracking (PT)	❶→❷	Human detection(❶TinyYOLOv3), Person identity finding(❷VGG)
License Plate Recognition (LPR)	❶→❷	Image super-resolution(❶ResNet100), Character recognition(❷MobileNet)
Video Understanding (VU)	❶→❷	Video content analysis(❶Inception), Content Understanding(❷BERT)

Benchmarks. We excerpt five production ML applications from [15], [22] as benchmarks in our evaluation, as shown in Table I. Furthermore, to evaluate INFSCALER under real-world workloads, we replay the Microsoft Azure Functions (MAF) dataset [23] which records approximately 46,000 requests of Azure cloud services.

B. Evaluation Results

Throughput optimization. Fig. 5 shows the results of how INFSCALER optimizes the model inference throughput compared with existing approaches. We can see from the result that INFSCALER achieves 34.78%, 48.14%, and 126.59% average improvement on throughput compared with SPRIGHT, INFless, and Knative. We can also observe that INFSCALER perfoms better in harsh workloads. For example, under the low workload pressure (SLO=3.0x), INFSCALER achieves 23.87% throughput improvement against the best-performed counterparts, SPRIGHT. When it comes to high workload pressure (SLO=1.5x), INFSCALER can achieve 40.82% performance gain. This is because higher pressure makes efficient resource usage more critical; even slight inefficiencies can lead to substantial performance degradation.

We then elaborate more on the different improvements against different approaches. The most noticeable improvement is when compared with Knative and INFless. INFless leverages batch mechanism to improve resource efficiency, but its coarse-grained auto-scaling hinders the performance on edge devices. On the other side, SPRIGHT achieves the best performance among the baselines. As we all know SPRIGHT does not include a partitioning algorithm, it focuses on the

TABLE II
RESOURCE USAGE (MEMORY IN MB) COMPARISON AGAINST BASELINES UNDER THE REAL-WORLD MAF WORKLOAD [23].

Benchmark	INFSCALER	SPRIGHT	INFless	Knative
TM	**7570.85**	9732.45	8489.23	10416.96
LT	**13808.67**	14735.2	15848.44	N/A
PT	**10935.03**	13979.08	12216.01	13383.16
LPR	**5573.82**	6882.41	7007.03	6993.59
VU‡	**15706.17**	N/A	N/A	N/A

‡ Under the real-world workload, the state-of-the-arts fails to meet the SLO of VU application while INFSCALER succeed.

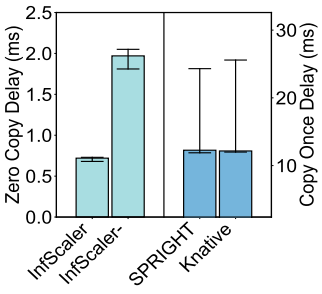

Fig. 6. Tensor sharing overhead against baselines.

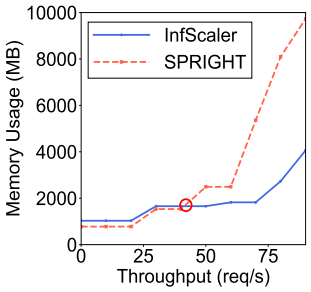

Fig. 7. Latency comparison among INFSCALER and baselines.

efficient data sharing between different serving instances. The ML applications we used as baselines are composed of several steps, which are suitable for SPRIGHT. However, similar to INFless, it also leverages coarse-grained scaling which misses the opportunity to achieve efficient resource usage.

Resource usage efficiency. We then move on to evaluate INFSCALER under real-world workloads and compare the resource usage efficiency against baselines. Table II shows the resource usage, i.e., memory, comparison against baselines. The workload pressure is set to medium, i.e., SLO=2.0x. We can see from the table that INFSCALER achieves 10.49%-27.32% of resource usage reduction against state-of-the-arts. More importantly, INFSCALER is the only framework that finishes the VU application under the SLO constraint. We attribute this improvement to the asymmetric auto-scaling which facilitates an unbalanced but efficient resource usage of different parts of ML models. Furthermore, the second-best approach varies between SPRIGHT and INFless, suggesting that solely focusing on scaling and data sharing may not always yield optimal performance. In addition to asymmetric auto-scaling, INFSCALER also optimizes the tensor sharing overhead, ultimately resulting in improved performance.

Overhead analysis. The overhead of INFSCALER mainly comes from the tensor sharing latency and the additional resources used for supporting multiple instances after partition.

We first evaluate the tensor sharing overhead in Fig. 6. INFSCALER– indicates the latency without our eBPF-based control-plane optimization. We can see that the data transferring overhead is reduced to less than 1 ms with our accelerator-agnostic tensor sharing mechanism, which is neglectable compared with tens or hundreds of milliseconds of inference latency. As for the resource overhead, we present a detailed measurement of the resource usage under different throughputs in Fig. 7. Comparing with SPRIGHT, the best-performed baseline, we observe that at low throughputs, the memory overhead leads to excessive resource usage because SPRIGHT does not include a partitioning strategy. As the throughput increases, the benefit of our asymmetric auto-scaling outweighs the overhead. We also argue that this runtime overhead could be further reduced with the runtime sharing approaches [24], which we identify as the future direction of INFSCALER.

VI. RELATED WORK

INFSCALER builds upon various projects and research. In this section, we summarize the existing works on ML inference serving that shed light on the design of INFSCALER.

To optimize the inference latency and achieve cost efficiency, some existing works use adaptive batching [25], [16], [26] while others focus on how to allocate resources of each ML serving instances [27]. The most related work to INFSCALER is INFless [15]. Instead of "patching" existing ML serving systems such as AWS Lambda [28] or Azure Functions [29], INFless presents a domain-specific inference system leveraging adaptive batching and cold-start mitigation mechanism to support ML inference tasks better. Compared with existing works, INFSCALER adopts an orthogonal view on inference serving, which is optimizing the structural bottleneck of AI models with auto-scaling. Another difference is INFSCALER targets the inference on edge devices that exhibit different hardware characteristics against cloud servers.

Due to the limited resources of edge devices, plenty of research on edge ML inference focuses on reducing resource usage or involving additional computing devices [1]. One of the well-studied optimizations is model pruning [13], [30]. It could reduce the model size and execution time to adapt to edge or IoT devices. For example, DyGNN [13] presents a software-hardware co-optimization approach for efficient GNN pruning and inference. However, model pruning approaches mostly need re-training or sacrifice accuracy limiting its utilization. There are also many works leveraging edge-cloud collaboration to improve inference performance. Several works [31], [32] split the ML model and distribute the ML partitions to edge and cloud for better latency and energy. Orthogonal to the above optimization attempts, INFSCALER takes the advantages of the auto-scaling technique and proposes an asymmetric auto-scaling approach with the help of bottleneck identification to make a wiser usage of limited edge resources.

VII. CONCLUDING REMARKS

This paper presents INFSCALER, a machine learning (ML) inference serving framework designed to address the challenges of inference on edge devices. By identifying architectural bottlenecks within ML models and implementing a bottleneck-aware asymmetric auto-scaling technique, INFSCALER facilitates fine-grained resource allocation on the edge, resulting in significant throughput improvements while adhering to latency SLOs. Leveraging the unified memory characteristic of multi-accelerator edge devices, INFSCALER efficiently reduces the partitioning overhead by an accelerator-agnostic tensor sharing mechanism. The experimental results underscore the effectiveness of INFSCALER by achieving 69.8% throughput improvement on average and up to 27.3% resource usage reduction, showcasing its potential to enhance edge-based ML inference capabilities while meeting stringent performance requirements.

REFERENCES

[1] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1738–1762, 2019.

[2] Y. Gong, P. Zhao, Z. Zhan, Y. Wu, C. Wu, Z. Kong, M. Qin, C. Ding, and Y. Wang, "Condense: A framework for device and frequency adaptive neural network models on the edge," in *Proc. of ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023.

[3] A. Mascolini, S. Gaiardelli, F. Ponzio, N. Dall'Ora, E. Macii, S. Vinco, S. Di Cataldo, and F. Fummi, "Varade: a variational-based autoregressive model for anomaly detection on the edge," in *Proc. of ACM/IEEE Design Automation Conference (DAC)*, 2024.

[4] "NVIDIA Jetson Xavier," https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-xavier-series/.

[5] "AMD Zynq UltraScale+ MPSoCs," https://www.amd.com/en/products/adaptive-socs-and-fpgas/soc/zynq-ultrascale-plus-mpsoc.html.

[6] "Google Edge TPU," https://cloud.google.com/edge-tpu.

[7] Y. He, L. Ma, Z. Jiang, Y. Tang, and G. Xing, "Vi-eye: Semantic-based 3d point cloud registration for infrastructure-assisted autonomous driving," in *Proc. of ACM MobiCom*, 2021, pp. 573–586.

[8] S. Shi, J. Cui, Z. Jiang, Z. Yan, G. Xing, J. Niu, and Z. Ouyang, "Vips: Real-time perception fusion for infrastructure-assisted autonomous driving," in *Proc. of ACM MobiCom*, 2022, pp. 133–146.

[9] R. Bhardwaj, Z. Xia, G. Ananthanarayanan, J. Jiang, Y. Shu, N. Karianakis, K. Hsieh, P. Bahl, and I. Stoica, "Ekya: Continuous learning of video analytics models on edge compute servers," in *Proc. of USENIX NSDI*, 2022, pp. 119–135.

[10] M. Xu, T. Xu, Y. Liu, and F. X. Lin, "Video Analytics with Zero-streaming Cameras," in *Proc. of USENIX ATC*, 2021, pp. 459–472.

[11] Z. Yu, Z. Wang, Y. Li, R. Gao, X. Zhou, S. R. Bommu, Y. Zhao, and Y. Lin, "Edge-llm: Enabling efficient large language model adaptation on edge devices via unified compression and adaptive layer voting," in *Proc. of ACM/IEEE Design Automation Conference (DAC)*, 2024.

[12] H. Wang, J. Gu, Y. Ding, Z. Li, F. T. Chong, D. Z. Pan, and S. Han, "Quantumnat: quantum noise-aware training with noise injection, quantization and normalization," in *Proceedings of the 59th ACM/IEEE design automation conference*, 2022, pp. 1–6.

[13] C. Chen, K. Li, X. Zou, and Y. Li, "Dygnn: Algorithm and architecture support of dynamic pruning for graph neural networks," in *Proc. of ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021.

[14] H. Tang, S. Yang, Z. Liu, K. Hong, Z. Yu, X. Li, G. Dai, Y. Wang, and S. Han, "Torchsparse++: Efficient training and inference framework for sparse convolution on gpus," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, 2023, pp. 225–239.

[15] Y. Yang, L. Zhao, Y. Li, H. Zhang, J. Li, M. Zhao, X. Chen, and K. Li, "INFless: A native serverless system for low-latency, high-throughput inference," in *Proc. of ACM ASPLOS*, 2022, pp. 768–781.

[16] A. Ali, R. Pinciroli, F. Yan, and E. Smirni, "Batch: Machine learning inference serving on serverless platforms with adaptive batching," in *Proc. of IEEE SC*, 2020, pp. 1–15.

[17] S. Qi, L. Monis, Z. Zeng, I.-c. Wang, and KK. Ramakrishnan, "SPRIGHT: Extracting the server from serverless computing! high-performance eBPF-based event-driven, shared-memory processing," in *Proc. of ACM SIGCOMM*, 2022, pp. 780–794.

[18] "Knative Project," https://knative.dev/.

[19] "KServe Project," https://github.com/kserve/kserve.

[20] "AWS SageMaker Serverless Endpoint," https://docs.aws.amazon.com/sagemaker/latest/dg/serverless-endpoints.html.

[21] M. Yu, T. Cao, W. Wang, and R. Chen, "Following the data, not the function: Rethinking function orchestration in serverless computing," in *Proc. of USENIX NSDI*, 2023, pp. 1489–1504.

[22] S. S. Shubha and H. Shen, "AdaInf: Data Drift Adaptive Scheduling for Accurate and SLO-guaranteed Multiple-Model Inference Serving at Edge Servers," in *Proc. of ACM SIGCOMM*, New York, NY, USA, Sep. 2023, pp. 473–485.

[23] M. Shahrad, R. Fonseca, Í. nigo Goiri, G. Chaudhry, P. Batum, J. Cooke, E. Laureano, C. Tresness, M. Russinovich, and R. Bianchini, "Serverless in the Wild: Characterizing and Optimizing the Serverless Workload at a Large Cloud Provider," in *Proc. of USENIX ATC*, 2020, pp. 205–218.

[24] J. Li, L. Zhao, Y. Yang, K. Zhan, and K. Li, "Tetris: Memory-efficient serverless inference through tensor sharing," in *Proc. of USENIX ATC*, 2022.

[25] C. Zhang, M. Yu, W. Wang, and F. Yan, "MArk: Exploiting Cloud Services for Cost-Effective, SLO-Aware Machine Learning Inference Serving." in *Proc. of USENIX ATC*, 2019, pp. 1049–1062.

[26] A. Ali, R. Pinciroli, F. Yan, and E. Smirni, "Optimizing inference serving on serverless platforms," *Proc. of ACM VLDB*, vol. 15, no. 10, 2022.

[27] M. Yu, Z. Jiang, H. C. Ng, W. Wang, R. Chen, and B. Li, "Gillis: Serving large neural networks in serverless functions with automatic model partitioning," in *Proc. of IEEE ICDCS*, 2021, pp. 138–148.

[28] "AWS Lambda," https://aws.amazon.com/lambda.

[29] "Azure Function," https://azure.microsoft.com/en-us/products/functions/.

[30] S. Yao, Y. Zhao, A. Zhang, L. Su, and T. Abdelzaher, "DeepIoT: Compressing deep neural network structures for sensing systems with a compressor-critic framework," in *Proc. of ACM SenSys*, 2017, pp. 1–14.

[31] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," in *Proc. of ACM ASPLOS*, vol. 45, 2017, pp. 615–629.

[32] L. Capogrosso, E. Fraccaroli, S. Chakraborty, F. Fummi, and M. Cristani, "Mtl-split: Multi-task learning for edge devices using split computing," in *Proc. of ACM/IEEE Design Automation Conference (DAC)*, 2024.