On Serving Image Classification Models

Aurora González-Vidal
University of Murcia
Murcia, Spain
aurora.gonzalez2@um.es

Alexander Isenko
Technical University of Munich
Munich, Germany
alex.isenko@tum.de

K. R. Jayaram
IBM Research
Yorktown Heights, NY, USA
jayaramkr@us.ibm.com

ABSTRACT

This paper aims to optimize model inference in interactive applications by reducing the infrastructure costs. It seeks to improve resource utilization, lower costs, and enhance the scalability and responsiveness of model serving systems. The focus is on achieving efficient inference in computer vision but has potential applications in other domains. The study involved experiments using a single GPU to analyze the impact of input image size and mini-batch size on request delivery time for image classification. Key findings include a model to estimate GPU warm-up time based on four parameters, the ratification of the existence of a linear relationship between mini-batch size and inference given one particular model, and the need to consider input size when selecting mini-batch size to avoid GPU crashes. Additionally, two mathematical models are proposed for further exploration using optimization algorithms. We also motivate the need to develop a more comprehensive mathematical model for soft and relaxed inference model serving.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning**; • **General and reference** → **Performance**.

KEYWORDS

resource allocation, GPU utilization, model inference

ACM Reference Format:
Aurora González-Vidal, Alexander Isenko, and K. R. Jayaram. 2023. On Serving Image Classification Models. In *9th International Workshop on Serverless Computing (WoSC '23), December 11–15, 2023, Bologna, Italy*. ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3631295.3631401

1 INTRODUCTION

Effective model inference has evolved into a crucial element in numerous interactive applications [1, 4, 8].

In deep learning applications, up to 90 percent of the infrastructure cost for developing and running an ML application is spent on inference — making the need for high-performance, low-cost ML inference infrastructure critical[1]. Examples in the image processing domain include (i) E-commerce and retail such as Amazon and Pinterest that use images for product search, discovery and recommendation, (ii) Social media – Instagram employs image inference to suggest filters and enhancements for photos and to identify and filter out offensive content, (iii) Autonomous vehicles use image inference from cameras, radar, and other sensors to detect objects, pedestrians, and lane markings for safe navigation, (iv) Healthcare applications use image inference for pathology and radiology, diagnosing diseases and conditions from medical images like X-rays, MRIs, and pathology slides [5] and (v) Precision Agriculture – drones equipped with cameras are used to monitor crops, assess plant health, and optimize farming practices [13].

Model serving systems, thus need to be scalable, guarantee high system goodput, and maximize resource utilization across compute units. This work is intended to set the foundations for model inference serving that is carried out in serverless computing environments, for the dynamic allocation of resources according to the load of inference requests and their deadline guarantees. There are many factors to be taken into account in such scenarios, so we will try to analyze independently what are the basics to consider and build up from there a generalizable optimization model able to assist in making scheduling decisions with deadline guarantees looking for the optimal use of the available resources.

[1]https://aws.amazon.com/ec2/instance-types/inf1/

WoSC '23, December 11–15, 2023, Bologna, Italy

ACM ISBN 979-8-4007-0455-0/23/12…$15.00
https://doi.org/10.1145/3631295.3631401

2 BACKGROUND AND RELATED WORK

2.1 Types of Inference

There are several types of inference that can be categorized based on their speed requirements:

- **"Hard" Real-time Inference** requires the fastest response time possible, often in real-time or near real-time. It is commonly used in applications where instant decisions or actions are safety critical, such as autonomous vehicles, real-time fraud detection systems, or high-frequency trading algorithms.
- **"Soft" Real-time** also requires a quick response time. However, it allows for a certain degree of latency or delay in the output without compromising the overall system reliability or safety. Soft inference is useful in applications like voice assistants or recommendation systems, where a slight delay in response is acceptable as long as the system maintains smooth interactions and user satisfaction.
- **Relaxed Inference** refers to the scenario where the time allowed for inference is large, typically in hours. It prioritizes accuracy over response time and is typically found in applications where either the computational complexity of the underlying models is high (such as complex deep-learning models or large-scale simulations) or in situations like invoice processing where SLAs are not very strict.

- **Best-effort Inference** refers to cases where there are no resources dedicated for inference and is typically seen in peer-to-peer systems and in community settings where inference happens over a wide area network.

In this work, we will focus on soft real-time and relaxed inference.

2.2 Equipment: TPU, GPU, CPU, etc.

The composition of computing systems typically encompasses a diverse array of components, each contributing uniquely to the overall functionality. These components often include high-performance hardware such as Tensor Processing Units (TPUs) [10], Graphics Processing Units (GPUs), and Central Processing Units (CPUs), which serve as the computational backbone of these systems. Moreover, the inclusion of smaller devices such as system-on-a-chip ones (SoC) and specialized peripherals can further enhance their capabilities. As a matter of fact, the current trends to use mobile devices as inference machines are opening new possibilities for computation on the edge[16].

2.3 Related Work

Existing inference model serving systems also support dynamic batching and replica auto-scaling [7, 18], inference buffering, and auto-selection of model variants [3, 14]. The recently proposed white-box model serving systems [8] enable model-specific optimizations with model layer sharing and fine-grained GPU scheduling. Others sample a small number of configurations in order to make the selection process faster [17]. To delve a bit more into some of these works, INFaaS [14] uses an optimization algorithm to select the best model-variant for each inference query. The algorithm is based on an Integer Linear Programming (ILP) formulation that takes into account various constraints and objectives, such as minimizing cost, maximizing throughput, and meeting Service Level Objectives (SLOs), therefore it could also focus on time. Since ILP is computationally expensive, INFaaS uses a heuristic algorithm that approximates the ILP solution in a more efficient manner. The heuristic algorithm is based on a model autoscaler that estimates the current headroom in capacities of running model variants, and selects the best scaling action (replicate or upgrade/downgrade) to satisfy the constraints and minimize the objective cost function. Other approaches focus on using multiple GPUs and try to ensure bounded latency for each request and serve multiple heterogeneous ML models in a system [2].

There exists literature focusing on the input difficulty and the complexity and optimality of the scheduling problem rather than the specific system, GPU and inference parameters [12] and works that tackle both aspects of the problem at the same time [19].

3 METHODOLOGY

In this work, we have studied the inference time for image classification tasks in order to propose a preliminary mathematical model that is able to optimize the computing resources available when performing soft and relaxed inference. For that purpose, we have used a system with a single GPU to run the experiments and a single deep-learning model as well. The model was EfficientNet [15]. EfficienNet has emerged as a benchmark in the field of computer vision. This innovative family of convolutional neural network architectures employs a principled approach to scale network depth, width, and resolution, optimizing the balance between computational efficiency and model accuracy. EfficientNet identifies the ideal scaling coefficients, enabling its top-tier performance on various tasks, including image classification, object detection, and semantic segmentation. In the original paper, they introduced a coefficient named ϕ that uniformly scales network width, depth, and resolution. Giving different values to such ϕ, they create different variants of the network, from B0 to B7. As we move from B0 to B7, the models become progressively deeper, wider, and capable of handling higher-resolution images. EfficientNet-B0 was pre-trained on the ImageNet dataset and will be used in this work.

3.1 Measuring memory usage

The *torch.cuda.memory_allocated()* function is a method provided by the PyTorch library for Python, and it is used to determine the amount of GPU memory currently allocated by your PyTorch tensors and variables. This function specifically reports the amount of memory allocated in bytes on the GPU device you are currently using for your PyTorch operations. In deep learning, the data that is processed by neural networks is often stored in tensors (multi-dimensional arrays) on the GPU. When creating those tensors and performing operations on them, PyTorch allocates GPU memory to store these tensors. This memory allocation is necessary for computation. In order to monitor GPU memory during the inference process, we have computed the prior and posterior allocated memory.

3.2 General system profiling

GPU warmup was originally proposed in 2017 [6]. Besides the GPU being in a power-saving state, there can be a number of other reasons why the first launch of a kernel could be slower than further runs because the GPU starts from a cold state. GPUs have startup overhead [11], including memory allocation and driver initialization. Accounting for these delays is fundamental when a deadline needs to be achieved: just-in-time compilation, transfer of kernel to GPU memory, and cache content are some things to be taken into account.

All those parameters can be profiled. For such a purpose we have used the single file implementation of a hardware monitor from one of the authors [9]. Such a script includes 164 features obtained making use of *psutil* that enables tracking of network bandwidth, disk read/write bandwidth, disk read/write counters, context switches, average CPU load (1,5,15 min), memory utilization and process memory (resident, virtual, lib, etc.). And if a GPU is available, over *pynvml* and *torch*: framebuffer memory, bar1 memory, gpu/memory utilization, temperature, power, throttle reasons, statistics retrieved by *torch.cuda.memory_stats()*, average (small, large, all) segment size, average block size, average inactive block size, allocation rounding overhead, memory fragmentation.

4 EXPERIMENTS AND RESULTS

The experiments were conducted on a system running the Debian 6.1.0-10-amd64 Linux distribution with kernel version 6.1.38-1. The hardware platform was x86_64 and the system was equipped with

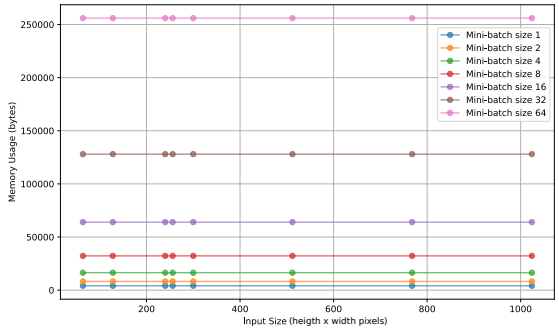

Figure 1: Memory usage using different image input sizes and mini-batch sizes

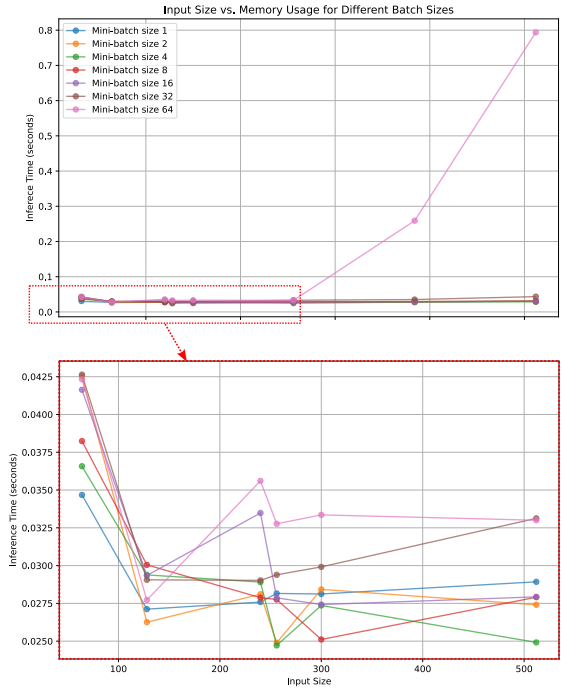

Figure 2: Inference time using different image input sizes and mini-batch sizes (up) and its zoom (down)

an NVIDIA A100 GPU with 40 GB of VRAM, and computations were performed using CUDA version 11.0.

4.1 Input size influence on memory and time

The relationship between input size and GPU performance needs to be investigated. We have systematically varied the input size while keeping other parameters constant and measuring key performance metrics, including inference time and GPU memory utilization. As can be seen in Figure 1, the allocated memory grows depending on the mini-batch size but it is indifferent to the input size. Something similar happens with the inference time according to the input size (see Figure 2), except that from a certain moment, the inference time explodes. It seems that the GPU is handling the smaller mini-batch sizes effectively. However, when the mini-batch size is very large, the GPU may experience bottlenecks due to the need to store and process a large number of samples at the same time, leading to increased latency and longer inference times. Given that circumstance, in the following subsection, we will take a single input size and test on further mini-batch sizes.

4.2 Mini-batch size influence on time and other parameters

We have measured inference time with different values of mini-batch size repeated 10 times. The upper part of Figure 3 shows the results of performing inference. It can be seen that there is a certain linear trend only disturbed by a few points. Those points correspond to the first time that a mini-batch size inference time is measured. In that sense, we understand that the GPU needs to be warmed up for every mini-batch size in order to present its regular behavior and, we noticed as well that from a certain point, the GPU warm-up inference time explodes, which can be related as well to getting close to its maximum mini-batch size. Taking out the first measurement for every mini-batch size, that could be considered warming up, we obtain an almost perfect regression fit, with a $R^2 = 0.9998$ (see below part of Figure 3), therefore we can think that the time after warming up is highly predictable.

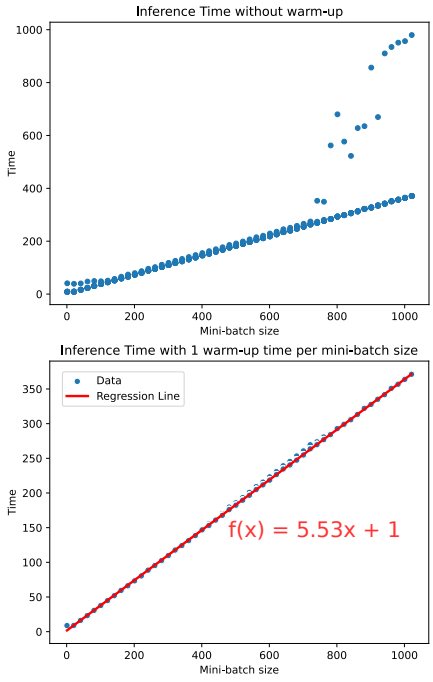

Figure 3: Inference time using different mini-batch sizes without considering warm-up (above) and considering warm-up (below)

In order to estimate the time that the GPU would take to warm up, we have collected information about the system prior to the

inference in each step making use of the already introduced repository [9] in Section 3.2. We have applied Random Forest in order to estimate such a value. One of the key advantages of Random Forest is its ability to assess the importance of features in making predictions. In Figure 4 we see how much the most important features contribute to the overall predictive performance and we selected the 4 most important that are related to the system in general, the mini-batch size and the memory of the GPU:

- process/involuntary_proc_ctx_switches: number of involuntary context switches that occur within the operating system. A high number of involuntary context switches can indicate potential performance bottlenecks or contention for CPU resources,
- Mini-batch size,
- 03_gpu_mem/requested_bytes.all.current: current or real-time number of bytes of GPU memory that have been requested or allocated, and
- 03_gpu_mem/allocated_bytes.large_pool.current: By creating large memory pools, the memory manager reduces the number of calls to the CUDA memory APIs, improving runtime performance. This metric indicates the current amount of GPU memory, in bytes, that has been allocated or reserved within a specific large memory pool on a GPU.

With these 4 variables, we have trained a Random Forest algorithm that estimates the warm-up time with a Coefficient of Variation of the Root Mean Square Error (CVRMSE) = 22 %, and a Mean Absolute Percentage Error (MAPE) = 12.5 %.

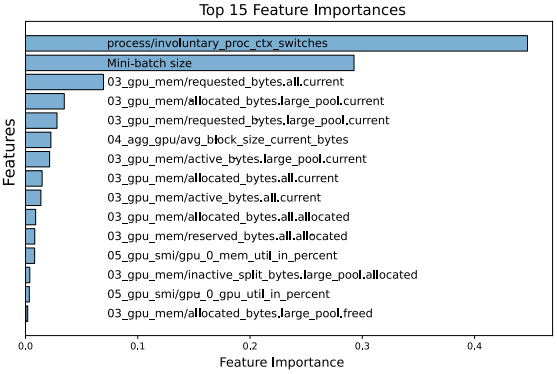

Figure 4: 15 most important features to determining first inference time / warm up

5 MATHEMATICAL MODELS

Given the results that have helped in understanding how the models can work, two different optimization models are proposed.

5.1 Soft Real-time Inference

In this use case, our goal is to minimize the number of GPUs that are needed to deliver a response within a certain time considering again the constraints of available GPUs, warm-up times, and linear batch size-latency relationships. The decision variables are as follows:

- t_i: The number of times GPU_i is used (an integer).
- mbs_i: The mini-batch size chosen for GPU GPU_i (an integer).
- N_G: The number of GPUs to be used (an integer)

The constants:

- T: The total available time. This should not be exceeded by any of the GPUs, given that they work in parallel (a decimal number).
- N: The number of images that need to be processed in total in the given time (an integer).
- $NGPU$: The maximum number of GPUs available (an integer)
- M_i: The maximum number of times GPU_i can be used (a constant)
- $Size_i$: The images' input size for GPU_i

Before the optimization process, it is imperative to understand the relationship between the latency, denoted as L_i, for GPU_i given a minibatch size denoted as mbs_i. According to our experiments, this latency exhibits a linear dependence on the minibatch size, and the relationship can be expressed as $L_i(\text{mbs}_i) = a_i \cdot \text{mbs}_i + b_i$. Here, a_i and b_i are coefficients representing the slope and intercept of the linear relationship for GPU_i, respectively.

Furthermore, the determination of the warm-up time, denoted as W_i, for GPU_i needs to be investigated as well. Unlike the latency, the warm-up time does not adhere to a strictly linear relationship and can deviate based on the current system parameters. Therefore, the warm-up time W_i is influenced by various factors beyond the minibatch size.

To comprehensively model the warm-up time W_i as a function of mbs_i and additional system parameters, a detailed analysis of the specific system characteristics and their impact on W_i is required. The analysis we realized showed that by profiling 4 main elements of the system, we can estimate the warm-up time.

The functions:

- L_i: Latency per mbs_i for GPU_i
- W_i: Warm-up time for GPU_i
- MB_i: The maximum mini-batch size for GPU_i (a function of $Size_i$).

Then, the optimization problem can be formulated as follows:

$$
\begin{aligned}
\min \quad & N_G \\
\text{s.t.} \quad & \text{Maximum}_i(W_i(\text{mbs}_i) + t_i \cdot L_i(\text{mbs}_i)) \leq T \\
& \sum_i (t_i + 1) \cdot \text{mbs}_i \geq N \\
& 1 \leq \text{mbs}_i \leq MB_i \quad \text{for all } i \\
& 0 \leq t_i \leq M_i \quad \text{for all } i \\
& 1 \leq N_G \leq NGPU
\end{aligned}
\tag{1}
$$

5.2 Relaxed inference

In a relaxed inference use case, our goal is to maximize the number of images processed in a given time while considering the constraints of available GPUs, warm-up times, and linear batch size-latency relationships. The definitions with regard to decision variables, constraints, constants and functions are almost the same as in 5.1, except that we do not need to minimize the number of GPUs that are used in this scenario. We are considering that a number of GPUs are reserved for this task and we are interested in the

operation of the system in order to maximize the throughput. The optimization problem can be formulated as follows:

$$\begin{aligned} \max \quad & NGPU \times \sum_i (t_i + 1) \cdot \text{mbs}_i \\ \text{s.t.} \quad & \text{Maximum}_i(W_i(\text{mbs}_i) + t_i \cdot L_i(\text{mbs}_i)) \leq T \\ & 1 \leq \text{mbs}_i \leq MB_i \quad \forall i \\ & 0 \leq t_i \leq M_i \quad \forall i \end{aligned} \tag{2}$$

Solving this optimization problem will give us information about how to use the available GPUs optimally, meaning with what mini-batch size and how many times, according to our limits.

6 CONCLUSIONS AND FUTURE WORK

The purpose of this research was to establish a foundation for exploring an optimal way of serving AI models for inference in different scenarios. We have performed an analysis of how the input of the images and the mini-batch size influence the time of delivery in the task of image classification using a single GPU and 1 deep learning model and provided a mathematical formulation as a starting point.

We are currently exploring several research directions. The first involves determining the optimal and maximum mini-batch size, considering both the system's status and the input size of the images. This determination is crucial for efficient and effective processing. Additionally, the inclusion of model and image load times is essential to better manage the system's resource allocation.

Handling multiple models within a single GPU tackling issues related to concurrency is another future work line. There will also be a focus on incorporating limits related to cost and energy consumption, ensuring that the system operates within predefined constraints. Furthermore, extending the system's capabilities to handle immediate hard inference and no-limit inference scenarios is part of the future work.

Finally, we will explore diverse scenarios, such as heterogeneous serving, combining TPU, GPU, CPU, and other devices, to improve versatility. Another important aspect to be considered is the integration of a model that estimates workflow, providing insights into process optimization. The research will also delve into scenarios where GPUs operate in a "sequential" manner, rather than in parallel, to address specific needs.

To further augment the system's capabilities, there will be an emphasis on integrating it with another system that can select deep learning models based on user requirements, like delivery time, data distribution, and signal-to-noise ratio. Lastly, the project will explore adapting its outcomes to serve Large Language Models effectively. These proposed enhancements collectively aim to create a inference serving system that is more versatile, efficient, and adaptable to a wide range of applications.

ACKNOWLEDGMENTS

This study forms part of the ThinkInAzul program and was supported by MCIN with funding from European Union NextGenerationEU (PRTR-C17.I1) and by Comunidad Autónoma de la Región de Murcia - Fundación Séneca. This work has been supported as well by ONOFRE PID2020-112675RB-C44 (MCIN) and PERSEO PDC2021-121561-I00, financed by MCIN/AEI /10.13039/501100011033 and by the European Union Next GenerationEU/ PRTR and by the HORIZON-MSCA-2021-SE-01-01 project Cloudstars (g.a. 101086248)

REFERENCES

[1] Ahsan Ali, Riccardo Pinciroli, Feng Yan, and Evgenia Smirni. 2022. Optimizing inference serving on serverless platforms. *Proceedings of the VLDB Endowment* 15, 10 (2022).

[2] Seungbeom Choi, Sunho Lee, Yeonjae Kim, Jongse Park, Youngjin Kwon, and Jaehyuk Huh. 2022. Serving heterogeneous machine learning models on Multi-GPU servers with Spatio-Temporal sharing. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*. 199–216.

[3] 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.

[4] Antonia Creswell, Murray Shanahan, and Irina Higgins. 2022. Selection-inference: Exploiting large language models for interpretable logical reasoning. *arXiv preprint arXiv:2205.09712* (2022).

[5] Aparna Gopalakrishnan, Narayan P Kulkarni, Chethan B Raghavendra, Raghavendra Manjappa, Prasad Honnavalli, and Sivaraman Eswaran. 2022. PriMed: Private federated training and encrypted inference on medical images in healthcare. *Expert Systems* (2022), e13283.

[6] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. 2017. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677* (2017).

[7] Arpan Gujarati, Sameh Elnikety, Yuxiong He, Kathryn S McKinley, and Björn B Brandenburg. 2017. Swayam: distributed autoscaling to meet slas of machine learning inference services with resource efficiency. In *Proceedings of the 18th ACM/IFIP/USENIX middleware conference*. 109–120.

[8] Arpan Gujarati, Reza Karimi, Safya Alzayat, Wei Hao, Antoine Kaufmann, Ymir Vigfusson, and Jonathan Mace. 2020. Serving DNNs like clockwork: Performance predictability from the bottom up. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. 443–462.

[9] Alexander Isenko. 2023. *Basic Hardware Monitor*. https://github.com/cirquit/py-hardware-monitor

[10] Yuriy Kochura, Yuri Gordienko, Vlad Taran, Nikita Gordienko, Alexandr Rokovyi, Oleg Alienin, and Sergii Stirenko. 2020. Batch size influence on performance of graphic and tensor processing units during training and inference phases. In *Advances in Computer Science for Engineering and Education II*. Springer, 658–668.

[11] Pouya Kousha, Bharath Ramesh, Kaushik Kandadi Suresh, Ching-Hsiang Chu, Arpan Jain, Nick Sarkauskas, Hari Subramoni, and Dhabaleswar K Panda. 2019. Designing a profiling and visualization tool for scalable and in-depth analysis of high-performance GPU clusters. In *2019 IEEE 26th International Conference on High Performance Computing, Data, and Analytics (HiPC)*. IEEE, 93–102.

[12] Zichong Li, Lan Zhang, Mu Yuan, Miaohui Song, and Qi Song. 2023. Efficient Deep Ensemble Inference via Query Difficulty-dependent Task Scheduling. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*. IEEE, 1005–1018.

[13] Najmeh Razfar, Julian True, Rodina Bassiouny, Vishaal Venkatesh, and Rasha Kashef. 2022. Weed detection in soybean crops using custom lightweight deep learning models. *Journal of Agriculture and Food Research* 8 (2022), 100308.

[14] Francisco Romero, Qian Li, Neeraja J Yadwadkar, and Christos Kozyrakis. 2021. INFaaS: Automated model-less inference serving. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. 397–411.

[15] Mingxing Tan and Quoc Le. 2019. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*. PMLR, 6105–6114.

[16] Stylianos I Venieris, Ioannis Panopoulos, and Iakovos S Venieris. 2021. OODIn: An optimised on-device inference framework for heterogeneous mobile devices. In *2021 IEEE International Conference on Smart Computing (SMARTCOMP)*. IEEE, 1–8.

[17] Luping Wang, Lingyun Yang, Yinghao Yu, Wei Wang, Bo Li, Xianchao Sun, Jian He, and Liping Zhang. 2021. Morphling: fast, near-optimal auto-configuration for cloud-native model serving. In *Proceedings of the ACM Symposium on Cloud Computing*. 639–653.

[18] Chengliang Zhang, Minchen Yu, Wei Wang, and Feng Yan. 2019. MArk: Exploiting Cloud Services for Cost-Effective,SLO-Aware Machine Learning Inference Serving. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*. 1049–1062.

[19] Hong Zhang, Yupeng Tang, Anurag Khandelwal, and Ion Stoica. 2023. SHEPHERD: Serving DNNs in the Wild. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. 787–808.