Towards Low-Cost and Energy-Aware Inference for EdgeAI Services via Model Swapping

Demetris Trihinas
Department of Computer Science
University of Nicosia
trihinas.d@unic.ac.cy

Panagiotis Michael
Department of Computer Science
University of Nicosia
michael.p15@live.unic.ac.cy

Moysis Symeonides
Department of Computer Science
University of Cyprus
msymeo03@ucy.ac.cy

***Abstract*—Over the past decade, key advancements in Artificial Intelligence (AI) and Edge Computing (EC) have led to the development of EdgeAI services to provide intelligent and low latency responses essential for mission-critical applications. However, the expansion of EdgeAI services to the network extremes can face challenges such as load fluctuations causing delays in AI inference and concerns over energy efficiency. This paper proposes "model swapping" where the model employed by the EdgeAI service is swapped on-the-fly with another readily available model so that cost and energy savings are achieved during runtime inference tasks. The ModelSwapper can achieve this by employing a low-cost algorithmic technique that explores meaningful trade-offs between the computational overhead and the model accuracy. By doing so, edge nodes adapt to load fluctuations by substituting complex models with simpler ones, thus meeting desired latency requirements, albeit with potentially higher uncertainty. Our evaluation with two EdgeAI services (object detection, NLU) demonstrates that ModelSwapper can significantly reduce energy usage and inference delays by at least 27% and 68% respectively, with only a 1% reduction in accuracy.**

***Index Terms*—Machine Learning, Edge Computing**

I. Introduction

For a while now, AI and Edge Computing have had a synergistic relationship propelled by breakthroughs in mobile computing, wireless networks, and deep learning [1]. This has led to the emergence of Edge Intelligence, or simply EdgeAI, a paradigm where compute-intensive and time consuming model training is conducted offline at the cloud, while runtime inference is executed by nodes at the network edge hosting the trained model [2] [3]. This framework is particularly advantageous for latency-sensitive applications, such as object detection for autonomous vehicles [4], natural language understanding (NLU) for humanitarian assistance services [5], and remote patient monitoring [6].

With AI gaining acceptance as a driver for new and emerging applications, the state-of-the-art in deep learning models are becoming more complex by capturing more nuances and patterns in the data to increase outputted accuracy [7]. A study by OpenAI showed that since 2012 the amount of computational effort required by such models is exponentially increasing and doubling every 4 months [8]. However, while IoT and EC hardware are vastly improving, they are not doing so at a rate capable of catering for the advancements in deep learning [9]. This need for additional computational effort is negatively impacting the user experience of resource-constraint EdgeAI services due to rising inference delays [10]. To make matters worse, more compute effort results in more energy consumption and this may result in a significant toll for EdgeAI services and the environment in general [11]. According to recent estimates, the energy footprint for data center computing has well surpassed the 1% of the global energy demand and is growing at a 4.3% annual rate [12]. Recent advancements in DL, such as GenAI, are further accelerating this growth [13] [14]. With recent projections showing that more than 50% of new IT infrastructure will be deployed at the edge by 2027 [15] it is not surprising that more initiatives in the form of policy changes (i.e., EU green deal, UK net zero) are calling for the migration to sustainable edge computing practices [16].

This brings us to the focal point of our work. Edge nodes can frequently present temporal resource constraints (i.e., low battery, disconnect from the cloud) or load fluctuations (i.e., incoming requests, background jobs) [17]. When these occur significant delays are observed as the employed model requires compute and memory capacity not currently available. To minimize these effects, model compression techniques such as quantization (i.e., reduce precision of the model parameters) and pruning (i.e., reduce number of model parameters) have been proposed for resource-constraint AI services [18]. However, while model compression methods can succeed in reducing the inference delay, they have a fixed cost and can significantly reduce accuracy even in cases where perhaps only a modest increment in uncertainty is required.

To overcome these challenges, we propose the adoption of *model swapping* for EdgeAI services. In this paradigm, an algorithmic decision-making module is employed by the EdgeAI service so that in the presence of limited resource capacity, the model in use can be swapped on-the-fly with another readily available, but less complex, model. With model swapping, EdgeAI services can achieve energy savings by employing less computational effort for runtime inference tasks and at the same time, user experience is not negatively impacted with latency bounded by using a less complex model. Moreover, in contrast to compression methods, accuracy (and complexity) reduction is only employed when the EdgeAI service is actually faced with limited resource capacity and otherwise, unaffected. Although model training is extremely resource hungry, it is also performed infrequently and may

only add up to only a small percentage of the total ML costs [19]. Hence, we focus on inference that is performed continuously and in situ by EdgeAI services that require latency guarantees. The key for accomplishing model swapping is for the edge node to maintain a model store supporting model recycling so that snapshots of pre-trained models with varying properties (i.e., layers, batch size) are available. A decision for which model to employ is performed by monitoring the load and resource availability of the edge node and opting for the model that can meet the current resource limitations, while also preserving any user-desired requirements.

The contributions of our work are:

- We provide a generalized problem description and low-cost algorithmic framework based on heuristic observations for *model swapping*. EdgeAI services can exploit this framework during the runtime inference of AI tasks to reduce their energy footprint, while also striving to maintain QoS requirements imposed by users.
- We introduce ModelSwapper a python toolkit implementing the proposed algorithmic framework that can be deployed alongside EdgeAI services to embrace on-the-fly model swapping for runtime AI inference tasks.
- We demonstrate the efficacy of ModelSwapper via 2 EdgeAI services for object detection and natural language understanding with each service employing popular DL model structures (EfficientNet, BERT) under different model configurations (network depth, width, etc) and real-world datasets (ImageNet, glue-mrpc). Evaluation artifacts are made publicly available and can be used to reproduce our results and (potentially) be used for future workload services[1].

The rest of this paper is structured as follows. Section 2 provides an in-depth problem description. Sections 3 and 4 introduce ModelSwapper and our model-swapping algorithmic framework. Section 5 provides a comprehensive experimentation analysis. Section 6 presents the related work, while Section 7 concludes the article and outlines future directions.

II. Problem Description

Let us assume that a user wants to run on an EdgeAI service a set of ML inference tasks $\mathcal{T} = t_1, t_2, ..., t_N$ with each task t_i accompanied by a minimum acceptable quality $q_{i,min}$ and maximum inference delay $\delta_{i,max}$. In turn, the provider of the EdgeAI service would like to minimize the energy consumption $\mathcal{E}$ of the EdgeAI service to meet certain sustainability goals or simply, reduce energy costs. The goal of model swapping is to design a classification function that selects from a pre-trained set of models $\mathcal{M} = m_1, m_2, ...m_K$ the model that minimizes the $\mathcal{E}$ of the EdgeAI service for $\mathcal{T}$ while obeying that the mean quality and inference delay meet user-given thresholds defined as Q and Δ, respectively.

For simplicity, and without loss of generality, let us assume that the EdgeAI service is dedicated to a single application domain (i.e., object detection) and all models in $\mathcal{M}$ are of the same architecture (e.g., BERT). Hence, the models deviate only by their complexity (e.g., layers, neurons, batch size) with $\gamma \in (0, 1)$ denoting the proportional difference from the most complex model $\mathcal{M}_\mathcal{K}$ of the model set. For example, assuming that $\mathcal{M}$ is comprised of models that differ only by network depth (layers), then if $\mathcal{M}_\mathcal{K}$ has a network depth of 10 layers, a model m_j with $\gamma = 0.3$ will have a depth of 3 layers.

Moreover, the EdgeAI service has a fixed resource capacity R and availability of these resources may vary in time (e.g., due to background tasks). Compute and memory are the two resources of interest for ML inference tasks. The availability of computational power impacts the inference delay of an inference task as these tasks are compute-intensive and enough memory must be available to fit the selected model.

With the above description, model swapping can be formalized as an optimization problem summarized as follows:

$$\arg\min_{m_j \in \mathcal{M}} \mathcal{E}(\mathcal{M}, \mathcal{T}, Q, \Delta, R), \quad j = 1, ..., |\mathcal{M}| \tag{1}$$

subject to:

$$r_{\mathcal{T},mem}(m_j) \leq R_{mem}, \; r_{\mathcal{T},comp}(m_j) \leq R_{comp} \tag{2}$$

$$\frac{1}{N} \sum_{i}^{|N|} q_i(m_j) \geq Q, \tag{3}$$

$$q_i \geq q_{i,min}, \; \forall \; t_i \in \mathcal{T} \tag{4}$$

$$\frac{1}{N} \sum_{i}^{|N|} \delta_i(m_j) \leq \Delta, \tag{5}$$

$$\delta_i \leq \delta_{i,max}, \; \forall \; t_i \in \mathcal{T} \tag{6}$$

Equation 1 highlights our optimization goal that is to select the model $m_j \in \mathcal{M}$ that minimizes the energy consumed by the node hosting the EdgeAI service. The limitation expressed in Equation 2 ensures that for $\mathcal{T}$, only models where their resource requirements can be met by the current resource availability of the edge node will be considered. Energy-efficiency is an optimization objective where trade-offs with performance must be explored. We emphasize *trade-off* as simply minimizing energy consumption means that opting for the least computational intensive model will suffice irrespective of the resulted QoS that can be heavily penalized. Hence, Equations 3-6 are in place to guarantee from the user perspective that the mean outputted quality for $\mathcal{T}$ is above the user-given threshold Q and in turn, the mean inference delay does not exceed Δ.

This ends up being an optimization problem with competing constraints. Such problems are NP-hard. However, with the adoption of various heuristics the complexity of the decision-making can resort to not creating additional overhead to the system real-time responsiveness. Approximations through heuristics are required as the longer the decision-making takes to select a model for runtime inference, the longer the delay faced by the users (or applications) submitting inference tasks to the EdgeAI service. In the next Section we show

[1] https://github.com/unic-ailab/ModelSwapper

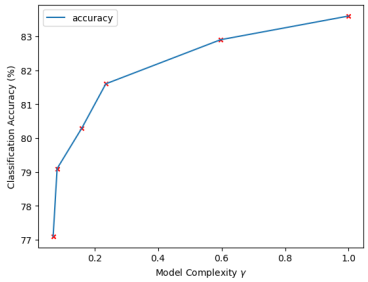

Fig. 1. Classification accuracy during model validation for 6 pre-trained EfficientNet models (avail. by TensorFlow) run with ImageNet dataset

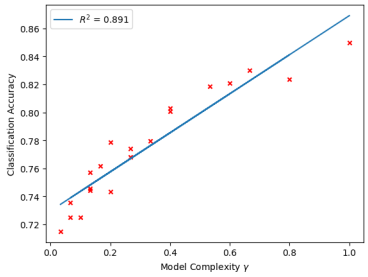

Fig. 2. Classification accuracy during model validation for 20 pre-trained BERT models (avail. by Google Research) run with glue-mrpc dataset

our heuristic-based algorithmic framework for energy-aware model-swapping with user-given QoS requirements.

III. Algorithmic Methodology

This Section provides an overview of the algorithmic methodology adopted in order to provide low-cost and energy-aware model swapping where certain assumptions and heuristics are embraced.

A. Heuristic Quantification

The first problem dimension to ease, is the resulting inference quality. Quality can have a different interpretation among users and applications. For example, classification accuracy, logarithmic loss, and confusion matrices are adopted for classification problems (e.g., object detection), while metrics such as mean square error (MSE) and R^2, are adopted for regression (e.g., price prediction) [20]. In general, given enough data during training, quality can be improved by increasing the complexity of the model γ (e.g., network depth, width), also known as model scaling, and data clarity ζ (i.e., adopting a higher resolution for images such as 224x224 vs 148x148). However, opting to increase any or both, will increase inference time and energy consumption as well [21]. Moreover, the learning ratio towards simpler models will plateau at some point (i.e., Fig 1). For example, Tan et al. [22] show that although ResNet-1000 is a lot more complex that ResNet-101, the gains in accuracy are fractional[2].

With this in mind, estimating the quality of an inference task would require a function $Q(\gamma, \zeta)$ where given the model

[2] The number followed by ResNet indicates the network depth in layers

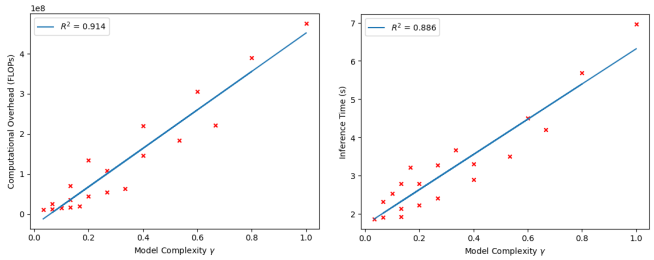

Fig. 3. Linear increment of compute load (left) and inference time (right) towards model complexity for 20 BERT models run with glue-mrpc dataset

complexity and the clarity of the input data, the resulting quality of an inference task can be estimated. However, although the expressive power of a model is (somehow) dependent on the model complexity and data clarity, a mathematical expression for estimating Q irrespective of the model domain does not exist [23]. To overcome this challenge, we will employ classification accuracy as our (expected) quality metric that is obtained during model training and make the following assumptions. Assuming the same model architecture (i.e., BERT, CNN) and a minimum data clarity (i.e., images of higher resolution can be down-scaled), a model's classification accuracy has a distinct monotonicity towards the model complexity. This is shown in both Figures 1 and 2. Thus, a model with $\gamma = 0.4$ is expected to have a higher accuracy than a model with $\gamma = 0.3$. Hence, given a quality threshold Q by the user, we can estimate the mean accuracy of each $t \in T$ and take the minimum accuracy above the threshold to then compute the inference delay and consumed energy.

The second problem dimension to ease is the estimation of the inference delay. As previously mentioned, by considering models of the same model architecture, when the model complexity increases, so does the computational overhead and subsequently inference time. As shown in Figure 3 for both quantities this is almost linear. Although the relationship to inference time is not perfect, the distinct monotonicity towards model complexity is evident here as well. Hence, the inference time δ per $m \in M$ of a given task $t \in T$ can be estimated as follows:

$$\delta(m_j, t) = \frac{FLOPs_{m_j}}{\lambda \cdot PFLOPS} \tag{7}$$

In this equation, FLOPs are the Floating Point Operations required by a model from data ingestion to outputting a result. Although one can run an extensive FLOPs analysis for all the models in M (e.g., using `flopth`[3]) a linear relationship among model complexity and FLOPs upholds (Figure 3). Hence, with slight uncertainty the FLOPs of a model m_j can be estimated by multiplying γ with the FLOPs of the most complex model in M:

$$FLOPs_{m_j} = \gamma \cdot FLOPs_{M_K} \tag{8}$$

At this point we denote a common misconception. Although the number of model parameters are usually reported and

[3] https://github.com/vra/flopth

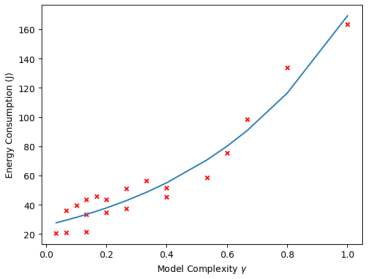

Fig. 4. Energy consumption towards towards model complexity for 20 BERT models run with glue-mrpc dataset

associated with the increase in computational effort of a model, FLOPs are not directly dependent on parameters. Parameters affect the network's depth and width, hence the model complexity, and by distributing parameters in different ways will result in different FLOPs [24]. With the FLOPs of a model in hand, we then proceed by dividing with the PFLOPS of the edge node. PFLOPS denotes the Peak Floating-Point Operations per Second (note the capital S) and this metric is processor dependent and indicates the theoretical max number of floating-point operations the edge node can process per second. This is multiplied by λ that is the current compute availability of the processor(s). For example, if only 40% of the computational power is currently available for t then the max capacity in PFLOPS must be proportionally reduced.

Having estimated the inference delay of a task, we then proceed with the estimation of its energy consumption. Energy-awareness, and subsequently green computing, is an important aspect for edge computing that can critically affect the liveness of the underlying processing infrastructure as well as the overall carbon footprint of the deployment [11]. In a geo-distributed environment, edge nodes may present not only resource heterogeneity but also different operating power levels. Power, denoted as P directly impacts energy consumption ($E = P \cdot \tau$) as the amount of energy required by a computing system to execute a specific task is calculated by multiplying the power drawn with the time (τ) required for the task to finish. In our case the time is equivalent to δ, the inference delay. Power usage is reported as the sum of $P_{idle} + P_{dyn}$, where P_{idle} denotes the load-independent power drawn by the computing system, even if no task is under execution, and P_{dyn} is load dependent. Patterson et al. identify the computational overhead (reported in FLOPs) as the key component contributing to P_{dyn} [13]. However, we denote that the relationship among FLOPs and energy is not proportional (as shown in Figure 4) when voltage scaling is supported by the underlying processor(s).

Hence, reducing the computational overhead reduces the inference delay and subsequently, energy consumption. If energy consumption is the sole metric of interest, then the most modest model in M will suffice. However, realistically users would like certain guarantees in both response quality and inference delay. As such, an energy-aware algorithm for model swapping must aim to reduce (not minimize) energy consumption and at the same time ensure certain user-given requirements are met.

Algorithm 1 ML Model Selection

Input: Model set $\mathcal{M}$, task set $\mathcal{T}$, desired quality Q and delay Δ
Output: Model $m \in \mathcal{M}$ that minimizes energy consumption
Ensure: User requirements Q and Δ for the task set

```
1: C.keys ← M //dict with candidate models
2: for each m in M do
3:     r_mem, r_comp ← getResourceAvailability()
4:     if (m.mem() ≥ r_mem && m.flops() ≥ r_comp) then
5:         C ← C − m
6:         continue
7:     end if
8:     q ← 0, δ ← 0, ϵ ← 0
9:     for each t in T do
10:        q ← q + getModelAccuracy(m, t)
11:        δ ← δ + estInfTime(m, t)
12:        p ← getOperPowerLevel()
13:        ϵ ← ϵ + estEnergy(m, δ, p)
14:    end for
15:    if (q/|T| ≤ Q && δ/|T| ≥ Δ) then
16:        C ← C − m
17:        continue
18:    end if
19:    C[m] ← e
20: end for
21: m ←argmin(C.values())
22: return m //model that minimizes energy
```

B. Model-Swapping Algorithm

Algorithm 1 summarizes our energy-aware model swapping methodology. In brief, the decision-making process receives as input the model set $\mathcal{M}$, task set $\mathcal{T}$ and the user-requested requirements for the the mean quality Q and inference delay Δ. Initially, all models of the model store can be considered as candidates for selection for the given task set $\mathcal{T}$. At this point, the current availability in memory (ram) and computational power (in flops) are extracted from the underlying edge node (lines 3-7). Models that require more resources than the currently available, are removed from consideration. Next, for all tasks $t \in \mathcal{T}$ (lines 8-14), we estimate and aggregate their model accuracy (q), inference time (δ), and energy consumption (ϵ). At this point, any models that cannot meet the mean quality and delay requirements given by the user, are removed by consideration (lines 15-18). Finally, for the models still remaining under examination, the algorithm opts for the model minimizing energy consumption and returns that model for selection (lines 21-22).

Getting the model accuracy, estimating inference time, and energy consumption per $t \in \mathcal{T}$ is of $\mathcal{O}(1)$ time. Hence, the algorithmic complexity of the model selection algorithm is $\mathcal{O}(\mathcal{M} \cdot \mathcal{T})$. The number of possible model configurations can be extremely large when one considers network depth, width, dropout rate, trained epochs and more. Practically though, a limited number of models will comprise the model store $\mathcal{M}$ due to both physical constraints of the edge node and in turn, the retraining "nightmare" that will occur each time these models must be updated due to data and/or concept drift. For example, in the NLU use-case presented in our evalu-

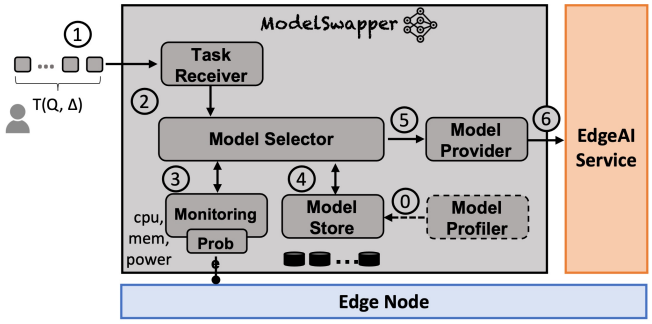

Fig. 5. High-Level Overview of ModelSwapper

ation, the number of pre-trained BERT models released by Google Research and (deemed) suitable for edge-computing are 24 [25]. Hence, the dominating factor for the algorithm complexity is the size of the task set. Therefore, the complexity can be summarized as $\mathcal{O}(\mathcal{T})$ where the the running time scales linearly towards the size of the task set.

IV. ModelSwapper

This Section provides a high-level overview of ModelSwapper. Figure 5 showcases how ModelSwapper fits within the ecosystem of an EdgeAI service.

ModelSwapper is implemented as a lightweight python library that can be added alongside existing EdgeAI services and is designed so that users interacting with the EdgeAI service are agnostic to the existence of the "model swapping". Hence, users do not need to change the business logic of their applications. Specifically, users submitting a set of inference tasks to the EdgeAI service continue doing so and optionally, can provide mean quality (Q) and inference delay requirements (Δ) for the submitted set of tasks (T).

We next describe the key components of ModelSwapper and their interactions. A typical flow for ModelSwapper starts with the user submitting a set of tasks for inference ①. The recipient component of the task set is the *Task Receiver*. This component will accept the set of tasks and parse the submitted request to extract if there are any accompanied requirements for the mean quality and inference delay of the task set. When finished, the *Model Selector* will run the model swapping algorithm described in the previous section. To do so though, it must interact with two other components ②.

The first component is the *Monitoring* that interacts with the underlying edge node to extract the current availability in computational power and memory ③. The monitoring component features 2 implementations of monitoring probes so that data extraction can be automated and not require any additional coding for service operators unless a custom probe is something they would want to develop. In brief, the monitoring features a probe for Netdata[4], an open and popular monitoring tool, and cAdvisor[5], another popular monitoring tool designed for containerized execution environments. In addition, the monitoring is also tasked with extracting the node's current power operating level. For this, ModelSwapper embeds a monitoring plugin that supports the extraction of the power drawn by the underlying edge node from various smart meters. The probing interface can easily be extended to support other power metering libraries and devices while users with no direct access to the underlying processor can use the probe interface to provide estimated power measurements inferred by a model. The second component is the *Model Store* where the set of pre-trained models reside along with an index providing elaborate metadata about each model ④. This includes each model's configuration differences (i.e., layers, headings) and their resource requirements. With this information in hand, the *Model Selector* can run the model-swapping algorithm each time a new set of inference tasks is received. Upon determining the most suitable model, the response is given to the *Model Provider* module ⑤ which will render the availability of model from the model store and notify the EdgeAI service accordingly ⑥.

Finally, we note that ModelSwapper presents one more component, the *Model Profiler*, that runs during the initial deployment ⓪. Specifically, the *Model Profiler* is tasked with computing the normalized model complexity γ and the FLOPs for each $m \in \mathcal{M}$. For the latter, the operators of the EdgeAI service have 3 options: (i) they may provide the FLOPs in a csv format; (ii) request the FLOPs to be computed for each model; or (iii) compute the FLOPs only for the most complex model and for the rest use the model complexity (Equation 8) to obtain an estimate.

V. Evaluation

This section introduces an evaluation of the efficacy and efficiency of ModelSwapper over two use-cases, multiple model configurations, and datasets. We note that for our evaluation we used popular and publicly available pre-trained models (and datasets) to support both the reproducibility and verification of the results.

A. Use-Cases and Datasets

The first use-case focuses on object detection where the EdgeAI service is tasked with accepting a set of inference tasks where objects must be detected (and classified) within the provided images. The model architecture embraced for this use-case is the popular convolutional neural network EfficientNet [22]. The classic EfficientNet version has 8 variants (B0-B7) where each model differs in network depth, width, resolution, and dropout rate. For the model store we employ the first 6 models (B0-B5) from the pre-trained model library of Keras[6] and note that we omit the final two variants (B6, B7) that are too complex for the edge computing testbed. For reference, the normalized model complexity (γ) for the variants B0-B5 are [0.070, 0.082, 0.160, 0.235, 0.597, 1.0]. For the inference tasks we will embrace the popular and open `ImageNet-tiny`[7] dataset (hosted on huggingface)

[4] https://www.netdata.cloud/
[5] https://github.com/google/cadvisor
[6] https://keras.io/api/applications/efficientnet/
[7] https://huggingface.co/datasets/zh-plus/tiny-imagenet

where the validation set features 10,000 colored 64×64 images classified into 200 label classes.

The second use-case focuses on natural language understanding where the EdgeAI service is tasked with accepting a set of text-based inference tasks that use a language model to semantically interpret the meaning of the provided text. The language model architecture embraced for the second use-case is the popular BERT architecture that is a transformers model pre-trained on a large corpus of English data (wikipedia) in a self-supervised fashion [26]. For the model store we do not embrace the classic pre-trained BERT models (i.e., BERT-base, BERT-large) but rather, we employ the light-weight models introduced by Google Research[8] that are deemed suitable for edge computing. These models differ in network depth (layers) and hidden embedding sizes. We note that the available lightweight models provided by Google Research are 24 but we employ that first 20 (up to 10 layers), excluding the final 4 models that overwhelm our edge computing testbed. As the models considered amount to 20, we omit presenting the model complexity vector here and refer readers to the plots introduced in Section 3. For the inference tasks we will embrace the popular and open `glue-mrpc` dataset that was released by Microsoft Research[9] and includes a corpus of sentence pairs automatically extracted from online news sources, with human annotations for whether the sentences in the pair are semantically equivalent.

B. Testbed

We run our experiments on a HP Proliant DL380 G9 server with an Intel Xeon E5-2680 processor embedding 44 cores clocked at 2.50GHz and 176GB memory. We embrace Fogify to cut and shape the server into an edge host for the EdgeAI service, streamline experimentation, and inject background load [27]. Specifically, a Fogify template is used to configure an edge server for the EdgeAI service with 13 GFLOPS processing power and 8GB memory. For reference this is x2 the performance of a Raspberry Pi 4 and on par with an nvidia Jetson TX1 based on the linpack bechmark[10]. To show that different models should be considered when the EdgeAI is under different load fluctuations, we utilize Fogify's compute-capping feature that restricts the available computational power of the edge server. With this, we can mimic background processes that affect the resource availability for the submitted inference tasks.

C. Methods for Comparison

For a comprehensive evaluation, we will compare ModelSwapper against the following methods:

- **Base**: this method will be our baseline by adopting the most complex, and accurate, model ($\gamma = 1.0$) per use-case for all given inference tasks.
- **Rand**: this method will perform a random selection from the model store for the EdgeAI service to utilize during inference irrespective of the underlying computational availability.
- **Prune**: this method is similar to the Baseline, but with the model having undergone pruning to reduce the feature space by 50%. This number was opted to show similar performance gains with ModelSwapper.

D. Experiment Runs for Use-Case 1

The following configurations are specific to this experiment. We split the validation set of the ImageNet dataset (10K images) into batches with each batch considered a set of inference tasks ($\mathcal{T}$) submitted by users in a streaming modality. For brevity, we note that the batches are fixed in size. In subsequent experiments we will work with different $\mathcal{T}$'s to study the overhead of the algorithmic process. Hence, for this run the batch size is set to 20 so that each batch is comprised of 20 randomly selected images. In addition, each batch should be annotated with a user-desired mean accuracy and inference delay. For the mean inference delay we select randomly between the 10 and 40s. For the classification accuracy, we consider a random selection among 0 (no model is excluded), $> 77\%$ (excludes 1 model), and $> 81\%$ (excludes 3 models). For fairness, we maintain the randomly assembled batches so that for each method evaluated, the comparison is performed over the same set of images and user requirements.

Each experiment run is set with a max duration of 30min and during execution the computational availability of the edge node fluctuates in time. As previously mentioned, for this we embrace Fogify that is configured to vary the computational availability of the edge node between 20% and 90% by deploying a computational-intensive background process. Similar to batched images, the derived computational availability of the edge node is pre-recorded so that all methods under evaluation are stressed, for fairness, in the same manner.

Figure 6 depicts the results of this experiment run. Within this figure, we observe (top-to-bottom) for each method under comparison the chronological evolution of the compute utilization, inference delay per batch, energy consumed per batch (Kjoules), and the classification accuracy per batch. We also note that the computational availability of the edge node and how it fluctuates in time is shown within the top plot (dashed line). From these plots we can make the following observations. As expected, and irrespective of the adopted model, the computational capacity made available is almost exhausted despite fluctuations that are dependent on the images randomly selected per batch. It is immediately evident that for the Baseline adopting the most complex EfficientNet model this results in significant inference delays as the computational availability of the edge node fluctuates with the mean reported delay at 36.2s (std 15.8) and some delays reaching 78-80s per batch. In turn, significant energy is consumed with the energy per batch well within the range of 0.23-0.71 KJoule and a mean of 0.41 KJoule. For the Baseline, the mean classification accuracy per batch is 68.46% (std 2.78). Moreover, from the plots we observe that a random selection of the model outputs truly random results with the inference delay and energy

[8] https://github.com/google-research/bert/

[9] https://www.microsoft.com/en-us/download/details.aspx?id=52398

[10] https://www.top500.org/project/linpack/

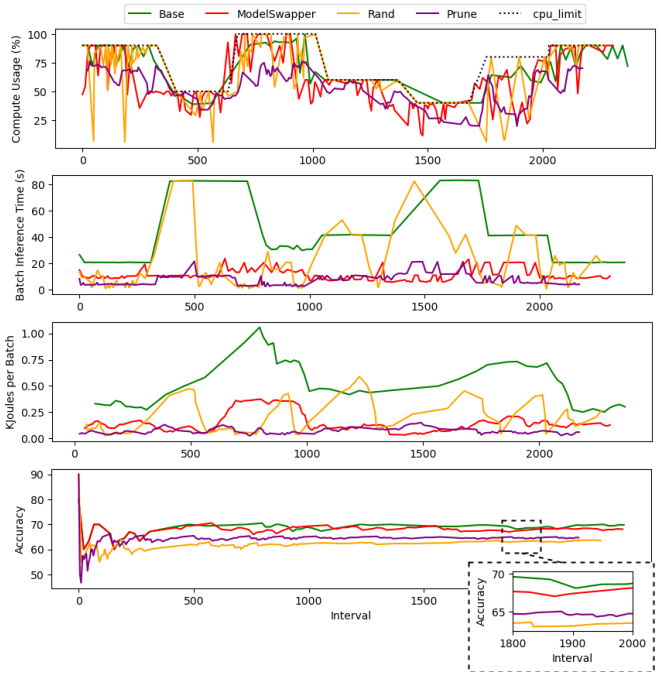

Fig. 6. UC1 performance evaluation with EfficientNet model store

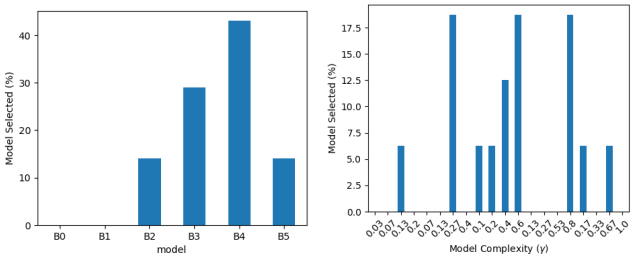

Fig. 7. ModelSwapper diverse selectivity for UC1 (left) and UC2 (right)

consumption presenting extreme variability, while the accuracy is significantly hampered, increasing the uncertainty by 7.4% on average (std 3.27).

In contrast to a fixed or random model selection, *by monitoring the computational availability of the edge node and striving to maintain user-given QoS requirements, we observe that ModelSwapper achieves significant energy savings with only a slight accuracy degradation*. Specifically, we observe that the inference delay per batch remains relatively stable irrespective of the computational availability. The mean inference delay per batch is 11.1s (std 2.9), a 68% reduction in comparison to the Baseline. This results in significant energy savings of 27% with the energy consumed per batch within the range of 0.03-0.36 KJoule and a mean of 0.13 KJoule. Moreover, these savings are achieved with the classification accuracy per batch dropping by only 0.88% on average. We note here that the percentage reduction in energy (27%) is less than the latency reduction (68%) as energy savings cannot be harvested when the selected model (to swap) still leaves the device completely stressed to meet the given latency request. Moreover, Figure 6 also depicts the experimentation

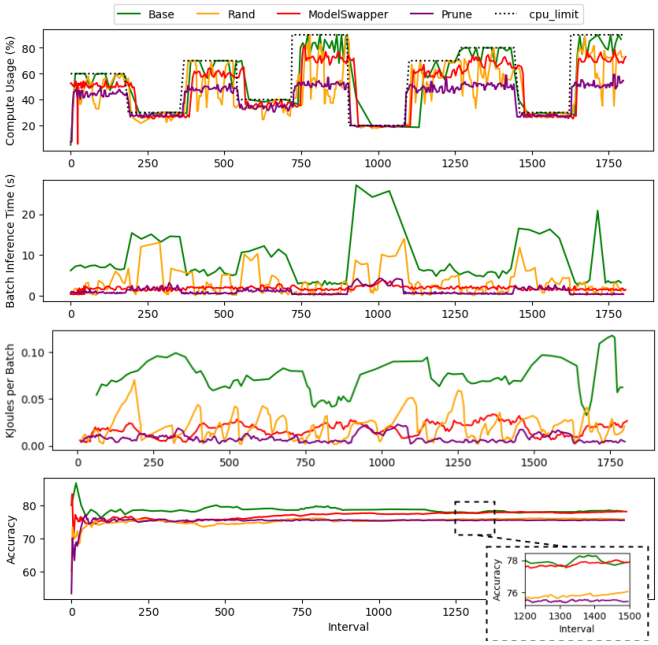

Fig. 8. UC2 performance evaluation with BERT model store

run with the EdgeAI service adopting parameter pruning so that the Baseline model employs compression to reduce the feature vector of the model. From the results, we observe that Pruning achieves similar results to ModelSwapper in terms of performance (inference delay, energy consumed). However, the classification accuracy significantly suffers and is even lower than the Random model selection. This is attributed to the fact that compression is a fixed cost imposed throughout the entire run irrespective of the edge node computational availability.

Finally, Figure 7 (left) showcases the diversity of the model swapping process. For the fluctuating computational availability of the EdgeAI service, we observe that ModelSwapper favors the EfficientNet B4 model 43%, B3 29% and models B2 and B5 14%, while models B1 and B2 are not selected as they never meet the user-given QoS requirements.

E. Experiment Runs for Use-Case 2

UC2 features a similar experiment setting with the previous. For brevity, we denote only the different configurations. For this experiment, we set the batch size to 15 so that each batch is comprised of 15 randomly selected sentences. In turn, the user-given mean inference delay per batch is randomly selected within the range of 1s and 4s. For the classification accuracy, we consider a random selection among 0 (no model is excluded), $> 72.5\%$ (excludes 3 models), and $> 75\%$ (excludes 6 models). As a reminder, for fairness across experiment runs, we maintain the randomly assembled batches so that for each method evaluated, the comparison is performed over the same set of sentences and user requirements.

Figure 8 depicts the results of UC2 where we output the same performance metrics as in the previous use-case. For both the Baseline and the Random execution we observe a similar pattern as with before. The Baseline employing the most

complex model ($\gamma = 1.0$) utilizes the computational resources completely and when availability is low, the inference delay per batch is significantly hampered. Specifically, the mean inference delay of the Baseline is 7.10s (std 3.61) with batches run over low compute availability incurring inference delays well within the 17-22s range. This significantly impacts energy consumption, with the reported energy per batch between 0.03-0.12 KJoule with a mean of 0.07 (std 0.02) KJoule. With 20 models to select from, the Random execution can, in several instances, reduce the inference time and energy consumption per batch. However, this (again) comes with a significant reduction in accuracy. For UC2, the mean classification accuracy per batch of the Random execution is reduced to 74.3% (std 1.4) in comparison to the Baseline with a mean accuracy of 79% (std 1.05).

In contrast, ModelSwapper is able to both reduce the inference delay per batch and it's variability, exploit significant energy savings, and do so while keeping the classification accuracy in line with the Baseline execution. Specifically, the mean inference delay per batch is 1.74s (std 0.43) that amounts to a 74% reduction in comparison to the Baseline. In turn, the mean energy consumed per batch is 0.017 KJoule (std 0.007) amounting to a 29% reduction in the total energy consumption. Most importantly, and as shown in Figure 8 (bottom plot), these savings are achieved with a minor reduction in classification accuracy of only 1.1%. Moreover, and similar to UC1, employing Pruning produces similar inference and energy savings with ModelSwapper. However, accuracy takes a significant hit, performing even worse that a random selection, as the compression is applied irrespective of the computational (un-) availability.

Finally, Figure 7 (right) showcases the diversity of the model swapping process for UC2. We note that in this figure the x-axis depicts the model complexity as the 20 BERT models do not have a naming convention. We observe that ModelSwapper performs a diverse selection from the model store to meet both the computational load of the edge node and the QoS demands of the userbase. The selection opts for 9 different models with some from the low-, mid-, and high-complexity spectrum.

F. Overhead Study

In this experimentation, focus is given in assessing the additional overhead imposed by the model swapping algorithmic framework to the EdgeAI service.

Although the algorithmic process is decoupled from the actual inference process of the EdgeAI service, we run this experimentation under realistic conditions. Specifically, we deploy ModelSwapper alongside UC1 that employs the EfficientNet models and performs inference on the ImageNet-tiny dataset. For the evaluation we run the ModelSwapper algorithm under different inference task set sizes ($\mathcal{T}$) and number of pre-trained models made readily available in the model store ($\mathcal{M}$). We consider $\mathcal{T}$ within the range of 10 and 10K, with the later denoting that the entire ImageNet-tiny validation set is considered a single batch. In turn, we assess the impact of 3 different model stores, where $\mathcal{M}$ = 20, 40, and 60. To generate different model configurations we use the EfficientNet model builder to randomly output models with different network depth, width, and dropout rate. We note that, for the majority of these model configurations, accuracy is low but is expected as a byproduct of having to generated up to 60 different models.

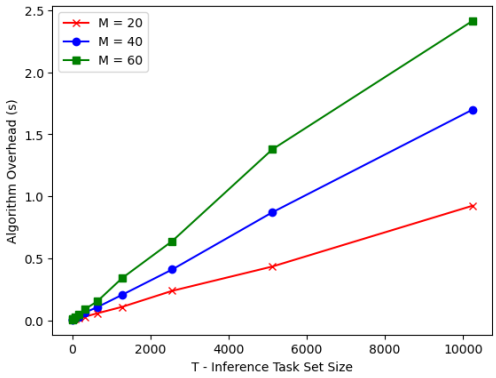

Fig. 9. ModelSwapper algorithm overhead analysis

Figure 9 depicts the results of this experimentation. The results confirm the complexity analysis performed in Section III-B. Specifically, we observe a linear increment in the running time of the model swapping algorithmic process as $\mathcal{T}$ increases. We also observe that adding more models to the model store $\mathcal{M}$, has a multiplier effect. Finally, we note that in comparison to the total inference time of a batch, the algorithmic overhead for small (and more realistic) batch sizes (< 100), is under 1% and as $\mathcal{T}$ increases, the ratio drops well under 0.1% for more than 1000 images in the batch.

VI. Related Work

The advent of open and publicly accessible ML/DL frameworks are facilitating the realization of EdgeAI services by easing the design and deployment of deep neural networks (DNNs) on resource-constrained devices [2]. Examples of such frameworks include TensorFlow-Lite[11], Glow[12], and ONNX[13]. With AI technology maturing and capable of training more accurate and versatile models, efficient inference serving out in the "wild" is becoming a key interest of both the systems and AI communities. Relevant inference serving tools for geo-distributed and edge computing realms are Clipper [28] and Nexus [29] with both enabling multi-client request serving and the definition of app-level SLOs for inference execution. In turn, ClockWork [30] is a distributed model serving system that mitigates tail latency for inference with the scheduling reordering the execution of inference tasks based on their targets and avoiding interference on worker nodes.

Efficiency for EdgeAI inference serving boils down to achieving desired latency and, more recently, energy saving requirements. For the former several works propose the introduction of model compression techniques to achieve faster

[11] https://www.tensorflow.org/lite
[12] https://github.com/pytorch/glow
[13] https://onnx.ai/

execution [18]. Examples of model compression techniques include quantization (i.e., reduce precision of the model parameters) [31] and pruning (i.e., reduce number of model parameters) [32]. While model compression methods can succeed in reducing the inference delay, they have a fixed cost and can significantly reduce accuracy even in cases where perhaps only a modest increment in uncertainty is required. To compensate, model adaptation can be embraced. For example, Teerapittayanon et al. introduce BranchyNet [33], where exit points are added to the (layered) network structure so that samples can exit the network early if the classification is confident, thus reducing the cost of inference. In turn, Bateni et al. [34] and Venkataramani et al. [35] propose ApNet and AxNN, respectively. These frameworks are designed to guarantee model execution under strict latency requirements by introducing layer-based approximation methods for DNNs that explore the trade-off between the approximation degree and the resulting execution reduction.

To avoid fixed costs, model selection methods can be employed. A notable model selection framework is Jellyfish, proposed by Nigate et al. [36]. Jellyfish selects a subset of available DL models to run a batch of inference tasks with the intent of achieving high accuracy guarantees through the ensemble of the models. As this incurs a high (resource) cost, the batch size must be dynamically adjusted to reduce the network latency incurred by remote and distant clients. Similary, Salmani et al. propose InfAdapter [37], a framework that also selects a subset of model variants that run on compute workers (within a Kubernetes cluster) to meet latency SLOs and maximize an objective function of accuracy and cost defined as a computational budget.

Closer to our work are the following frameworks. Shu et al. introduce IF-CNN [10] that embeds a CNN-based model, denoted as the recognition predictor (RP), to select among 3 models the most efficient to run a given set of inference tasks. To select among the 3 models the RP outputs a prediction for the top-1 accuracy of each model and the one above a user-given threshold with the lowest complexity is selected. Scaling to more than 3 models is a challenge as the training process must learn the top-1 probabilities per model and due to the inference delay of the RP itself, significant gains are observed only when the task set is large. On the other hand, Marco et al. [38] introduce a lightweight ML framework capable of selecting among models the most accurate for inference based on the feature set of the input sample. The feature set is app-dependent, e.g., for image classification brightness, hue and edge length, and for machine translation number of words and word length; are deemed the most important during the training process of the model selector. A key observation made by the authors is that selecting the most suitable model for the given input can potentially also reduce inference time. Finally, Park et al. [39] introduce Big/Little, a framework for image classification that selects for inference tasks among two available model variants (little, big), in an attempt to reduce energy consumption on edge nodes without significantly impacting accuracy. To achieve this, Big/Little computes the score margin, a metric that represents the difference between the 1st and 2nd score at the last classification layer and if the margin is above a threshold (e.g. 0.7) then the "little" model can be run with the expectation that the accuracy will not be significantly affected.

VII. Conclusions and Future Directions

With the pressing need to move towards sustainable practices for Edge Computing, AI cannot be the exception. Our work tackles energy-aware model swapping by introducing ModelSwapper. When resource constraint EdgeAI services are faced with pressing loads, the algorithmic mechanism of ModelSwapper is designed to select among a set of pre-trained models a less computationally complex model that is best suited to reduce energy consumption for the runtime inference tasks imposed by users to the EdgeAI service. With model swapping, energy savings are harvested by employing less computational effort and at the same time, user experience is not negatively impacted as ModelSwapper balances among energy saving and user-desired QoS requirements in the form of inference delay and accuracy. To showcase both the efficacy and efficiency of ModelSwapper, we perform a comprehensive evaluation featuring EdgeAI services for object detection and NLU, with each service employing popular model structures (EfficientNet, BERT), under different model configurations (network depth, width, etc) and real-world datasets. The evaluation demonstrates that ModelSwapper can significantly reduce energy usage and inference delays by at least 68% and 27% respectively, with only a 1% reduction in accuracy.

The future directions of our work are two-fold. First, the algorithmic framework for energy-aware model swapping will be extended to acknowledge DL models of different structure where the relationship among model complexity towards efficiency (flops per watt) and accuracy can significantly differ between model structures. Second, the algorithmic framework will be extended to propose the computational placement of a set of inference tasks by considering, other than local execution, the placement to other neighboring edge nodes.

Acknowledgement

This work is part of AdaptoFlow that has indirectly received funding from the European Union's Horizon Europe research and innovation action programme, via the TRIALSNET Open Call issued and executed under the TrialsNet project (Grant Agreement no. 101017141).

References

[1] H. Hua, Y. Li, T. Wang, N. Dong, W. Li, and J. Cao, "Edge computing with artificial intelligence: A machine learning perspective," *ACM Comput. Surv.*, vol. 55, no. 9, jan 2023. [Online]. Available: https://doi.org/10.1145/3555802

[2] R. Singh and S. S. Gill, "Edge AI: A survey," *Internet of Things and Cyber-Physical Systems*, vol. 3, pp. 71–92, 2023.

[3] S. Deng, H. Zhao, W. Fang, J. Yin, S. Dustdar, and A. Y. Zomaya, "Edge intelligence: The confluence of edge computing and artificial intelligence," *IEEE Internet of Things Journal*, vol. 7, no. 8, pp. 7457–7469, 2020.

[4] A. Balasubramaniam and S. Pasricha, "Object detection in autonomous vehicles: Status and open challenges," 2022.

[5] K. Mabuntham and W. Marurngsith, "Mobile edge nlu with on-device inference for humanitarian assistance during disasters," in *2022 IEEE 5th International Conference on Electronics and Communication Engineering (ICECE)*. IEEE, 2022, pp. 239–243.

[6] J. P. Queralta, T. N. Gia, H. Tenhunen, and T. Westerlund, "Edge-ai in lora-based health monitoring: Fall detection system with fog computing and lstm recurrent neural networks," in *2019 42nd international conference on telecommunications and signal processing (TSP)*. IEEE, 2019, pp. 601–604.

[7] C.-J. Wu, R. Raghavendra, U. Gupta, B. Acun, N. Ardalani, K. Maeng, G. Chang, F. Aga, J. Huang, C. Bai, M. Gschwind, A. Gupta, M. Ott, A. Melnikov, S. Candido, D. Brooks, G. Chauhan, B. Lee, H.-H. Lee, B. Akyildiz, M. Balandat, J. Spisak, R. Jain, M. Rabbat, and K. Hazelwood, "Sustainable AI: Environmental Implications, Challenges and Opportunities," in *Proceedings of Machine Learning and Systems*, D. Marculescu, Y. Chi, and C. Wu, Eds., vol. 4, 2022, pp. 795–813.

[8] D. Amodei and D. Hernandez, "AI and Compute," https://openai.com/research/ai-and-compute, 2018.

[9] A. Gujarati, S. Elnikety, Y. He, K. S. McKinley, and B. B. Brandenburg, "Swayam: distributed autoscaling to meet slas of machine learning inference services with resource efficiency," in *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference*, ser. Middleware '17, New York, NY, USA, 2017, p. 109–120.

[10] G. Shu, W. Liu, X. Zheng, and J. Li, "IF-CNN: Image-Aware Inference Framework for CNN With the Collaboration of Mobile Devices and Cloud," *IEEE Access*, vol. 6, pp. 68 621–68 633, 2018.

[11] D. Trihinas, L. Thamsen, J. Beilharz, and M. Symeonides, "Towards energy consumption and carbon footprint testing for ai-driven iot services," in *2022 IEEE International Conference on Cloud Engineering (IC2E)*, 2022, pp. 29–35.

[12] E. Masanet, A. Shehabi, N. Lei, S. Smith, and J. Koomey, "Recalibrating global data center energy-use estimates," *Science*, vol. 367, no. 6481, pp. 984–986, 2020.

[13] D. A. Patterson, J. Gonzalez, Q. V. Le, C. Liang, L. Munguia, D. Rothchild, D. R. So, M. Texier, and J. Dean, "Carbon emissions and large neural network training," *CoRR*, vol. abs/2104.10350, 2021.

[14] J. Vincent, "How much electricity does ai consume?" 2024.

[15] G. Inc, "Gartner Says 50% of Critical Enterprise Applications Will Reside Outside of Centralized Public Cloud Locations Through 2027," https://tinyurl.com/4ud5tsp6, 2023.

[16] L. Saint-Martin, J. Delesse, and J. Tual, "Study on the economic potential of far edge computing in the future smart internet of things – final study report," *Publications Office of the European Union*, 2023.

[17] D. Trihinas, G. Pallis, and M. Dikaiakos, "Low-cost adaptive monitoring techniques for the internet of things," *IEEE Transactions on Services Computing*, 2018.

[18] J. Kim, S. Chang, and N. Kwak, "Pqk: model compression via pruning, quantization, and knowledge distillation," *arXiv preprint arXiv:2106.14681*, 2021.

[19] R. Desislavov, F. Martínez-Plumed, and J. Hernández-Orallo, "Trends in ai inference energy consumption: Beyond the performance-vs-parameter laws of deep learning," *Sustainable Computing: Informatics and Systems*, vol. 38, p. 100857, 2023.

[20] G. Varoquaux and O. Colliot, "Evaluating machine learning models and their diagnostic value," *Machine Learning for Brain Disorders*, pp. 601–630, 2023.

[21] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "Mnasnet: Platform-aware neural architecture search for mobile," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 2820–2828.

[22] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International conference on machine learning*. PMLR, 2019, pp. 6105–6114.

[23] M. Raghu, B. Poole, J. Kleinberg, S. Ganguli, and J. Sohl-Dickstein, "On the expressive power of deep neural networks," in *international conference on machine learning*. PMLR, 2017, pp. 2847–2854.

[24] R. Desislavov, F. Martínez-Plumed, and J. Hernández-Orallo, "Compute and energy consumption trends in deep learning inference," *CoRR*, vol. abs/2109.05472, 2021. [Online]. Available: https://arxiv.org/abs/2109.05472

[25] I. Turc, M. Chang, K. Lee, and K. Toutanova, "Well-read students learn better: The impact of student initialization on knowledge distillation," *CoRR*, vol. abs/1908.08962, 2019.

[26] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2019.

[27] M. Symeonides, Z. Georgiou, D. Trihinas, G. Pallis, and M. D. Dikaiakos, "Fogify: A fog computing emulation framework," in *2020 IEEE/ACM Symposium on Edge Computing (SEC)*, 2020, pp. 42–54.

[28] D. Crankshaw, X. Wang, G. Zhou, M. J. Franklin, J. E. Gonzalez, and I. Stoica, "Clipper: A Low-Latency online prediction serving system," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. Boston, MA: USENIX Association, Mar. 2017, pp. 613–627.

[29] H. Shen, L. Chen, Y. Jin, L. Zhao, B. Kong, M. Philipose, A. Krishnamurthy, and R. Sundaram, "Nexus: a gpu cluster engine for accelerating dnn-based video analysis," in *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, ser. SOSP '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 322–337. [Online]. Available: https://doi.org/10.1145/3341301.3359658

[30] A. Gujarati, R. Karimi, S. Alzayat, W. Hao, A. Kaufmann, Y. Vigfusson, and J. Mace, "Serving DNNs like clockwork: Performance predictability from the bottom up," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, Nov. 2020, pp. 443–462. [Online]. Available: https://www.usenix.org/conference/osdi20/presentation/gujarati

[31] N. Rotem, J. Fix, S. Abdulrasool, G. Catron, S. Deng, R. Dzhabarov, N. Gibson, J. Hegeman, M. Lele, R. Levenstein, J. Montgomery, B. Maher, S. Nadathur, J. Olesen, J. Park, A. Rakhov, M. Smelyanskiy, and M. Wang, "Glow: Graph lowering compiler techniques for neural networks," 2019.

[32] H. Wang, Z. Zhang, and S. Han, "Spatten: Efficient sparse attention architecture with cascade token and head pruning," *CoRR*, vol. abs/2012.09852, 2020. [Online]. Available: https://arxiv.org/abs/2012.09852

[33] S. Teerapittayanon, B. McDanel, and H. T. Kung, "Branchynet: Fast inference via early exiting from deep neural networks," *CoRR*, vol. abs/1709.01686, 2017. [Online]. Available: http://arxiv.org/abs/1709.01686

[34] S. Bateni and C. Liu, "Apnet: Approximation-aware real-time neural network," in *2018 IEEE Real-Time Systems Symposium (RTSS)*, 2018, pp. 67–79.

[35] S. Venkataramani, A. Ranjan, K. Roy, and A. Raghunathan, "Axnn: Energy-efficient neuromorphic systems using approximate computing," in *2014 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, 2014, pp. 27–32.

[36] V. Nigade, P. Bauszat, H. Bal, and L. Wang, "Jellyfish: Timely inference serving for dynamic edge networks," in *2022 IEEE Real-Time Systems Symposium (RTSS)*, 2022, pp. 277–290.

[37] M. Salmani, S. Ghafouri, A. Sanaee, K. Razavi, M. Mühlhäuser, J. Doyle, P. Jamshidi, and M. Sharifi, "Reconciling high accuracy, cost-efficiency, and low latency of inference serving systems," in *Proceedings of the 3rd Workshop on Machine Learning and Systems*, ser. EuroMLSys '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 78–86. [Online]. Available: https://doi.org/10.1145/3578356.3592578

[38] V. S. Marco, B. Taylor, Z. Wang, and Y. Elkhatib, "Optimizing deep learning inference on embedded systems through adaptive model selection," *ACM Trans. Embed. Comput. Syst.*, vol. 19, no. 1, feb 2020.

[39] E. Park, D. Kim, S. Kim, Y.-D. Kim, G. Kim, S. Yoon, and S. Yoo, "Big/little deep neural network for ultra low power inference," in *2015 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2015, pp. 124–132.