NeuOS: A Latency-Predictable Multi-Dimensional Optimization Framework for DNN-driven Autonomous Systems

Soroush Bateni and Cong Liu, *University of Texas at Dallas*

https://www.usenix.org/conference/atc20/presentation/bateni

This paper is included in the Proceedings of the 2020 USENIX Annual Technical Conference.

July 15–17, 2020

978-1-939133-14-4

Open access to the Proceedings of the 2020 USENIX Annual Technical Conference is sponsored by USENIX.

NeuOS: A Latency-Predictable Multi-Dimensional Optimization Framework for DNN-driven Autonomous Systems

Soroush Bateni and Cong Liu
The University of Texas at Dallas

Abstract

Deep neural networks (DNNs) used in computer vision have become widespread techniques commonly used in autonomous embedded systems for applications such as image/object recognition and tracking. The stringent space, weight, and power constraints seen in such systems impose a major impediment for practical and safe implementation of DNNs, because they have to be latency predictable while ensuring minimum energy consumption and maximum accuracy. Unfortunately, exploring this optimization space is very challenging because (1) smart coordination has to be performed among system- and application-level solutions, (2) layer characteristics should be taken into account, and more importantly, (3) when multiple DNNs exist, a consensus on system configurations should be calculated, which is a problem that is an order of magnitude harder than any previously considered scenario. In this paper, we present NeuOS, a comprehensive latency predictable system solution for running multi-DNN workloads in autonomous systems. NeuOS can guarantee latency predictability, while managing energy optimization and dynamic accuracy adjustment based on specific system constraints via smart coordinated system- and application-level decision-making among multiple DNN instances. We implement and extensively evaluate NeuOS on two state-of-the-art autonomous system platforms for a set of popular DNN models. Experiments show that NeuOS rarely misses deadlines, and can improve energy and accuracy considerably compared to state of the art.

1 Introduction

The recent explosion of computer vision research has led to interesting applications of learning-driven techniques in autonomous embedded systems (AES) domain such as object detection in self-driving vehicles and image recognition in robotics. In particular, deep neural networks (DNNs) with generally the same building blocks have been dominantly applied as effective and accurate implementation of image recognition, object detection, tracking, and localization towards enabling full autonomy in the future [60, 50]. For example, using such DNNs alone, Tesla has recently demonstrated that a great deal of autonomy in self-driving cars can be achieved [33]. Another catalyzer for the feasibility of DNN-driven autonomous systems in practice has been the advancement of fast, energy-efficient embedded platforms, particularly accelerator-enabled multicore systems such as the NVIDIA Drive AGX and the Tesla AI platforms [44, 22].

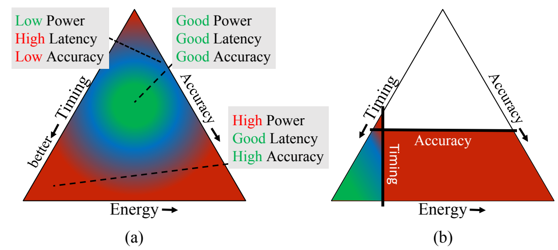


Figure 1: Ternary depiction of the 3D optimization space.

Autonomous systems based on embedded hardware platforms are bounded by stringent Space, Weight, and Power (SWaP) constraints. The SWaP constraints require system designers to carefully take into account energy efficiency. However, DNN-driven autonomous embedded systems are considered mission-critical real-time applications and thus, require predictable latency[1] and sufficient accuracy[2] (of the DNN output) in order to pass rigorous certifications and be safe for end users [46]. This causes a challenging conflict with energy efficiency since accurate DNNs require a tremendous amount of resources to be feasible and to be timing-predictable, and are by far the biggest source of resource consumption in such systems [5]. This usually results in less complicated (and less resource-demanding) DNN models to be designed and used in these systems, reducing accuracy considerably.

Fig. 1(a) shows a hypothetical three-dimensional space between latency, power, and accuracy mapped to a ternary plot [55] (where (Energy + Timing + Accuracy) has been normalized to 3). Each dot in Fig. 1(a) represents a configuration with a unique set of latency, power, and accuracy characteristics. The power consumption is usually

[1]Latency from each system component (including the DNNs) in AES will add up to the reaction latency between when a sensor observes an event and when the system externally reacts to that event, such as by applying the breaks in a self-driving vehicle. The faster a system reacts, the more likely it is for the system to avoid a disaster, such as an accident. However, policymakers might adopt a reasonable reaction time, such as 33ms or even 300ms [48, 27, 12, 14, 9, 4] as "safe enough".

[2]We should mention here that there is currently no established standard to connect DNN accuracy to the safety of a particular system, such as DNNs in self-driving vehicles. In this paper, we assume the more accurate the DNN, the safer the system is.

adjusted at system-level via dynamic voltage/frequency scaling (DVFS) [5, 21]. The accuracy adjustment is done at application-level via DNN approximation configuration switching (see Sec. 4 for details). Note that both DVFS and DNN configuration adjustments would impact runtime latency. This figure highlights three configurations with various levels of latency, power consumption, and accuracy tradeoff that might or might not be acceptable given the current performance constraints. Choosing the best three-dimensional trade-off optimization point is a significant challenge given the vast and complex DVFS and accuracy configuration space.

Although all autonomous systems are required to be latency predictable in nature, the constraints on power and accuracy may vary based on the type of autonomous system (e.g., highly constrained power for drones and maximum accuracy requirement for autonomous driving). To illustrate one such variation, note Fig. 1(b), which shows a constraint on latency, and a constraint on accuracy imposed in the configuration space limiting the possible configurations considerably.

Challenges specific to DNN-driven AES. In addition to the aforementioned optimization problem, DNNs are constructed from layers, where each layer responds differently to DVFS changes and has unique approximation characteristics (as we shall showcase in Sec. 3.1). In order to meet a latency target with optimized energy consumption and accuracy, each layer requires a unique DVFS and approximation configuration, whereas existing approaches such as Poet [23] and JouleGuard [21] deal with DNNs as a black-box. Moreover, system-level DVFS adjustments and application-level accuracy adjustments happen at two separate stages. Without smart coordination, the system might fall in a negative feedback loop, as we shall demonstrate in Sec. 3.2. This coordination needs to happen at layer boundaries, making the problem at least an order of magnitude harder than previous work.

Furthermore, existing techniques mostly focus on single-tasking scenarios [5, 3, 20] whereas AES generally require multiple instances of different DNNs. As we shall motivate in Sec. 3.3 using a real-world example, these DNNs need to communicate and build a cohort on a layer-by-layer basis to avoid greedy and inefficient decision-making. Moreover, system-level and application-level coordination in this multi-DNN scenario is much harder than isolated processes considered in previous work.

Finally, existing approaches [13, 5] optimize latency performance on a best-effort basis (e.g., by using control theory) that can overshoot a latency target (as demonstrated in Sec. 3.2). A better solution should include proven real-time runtime strategies such as LAG analysis [51].

Contribution. In this paper, we present NeuOS[3], a comprehensive timing-predictable system solution for multi-DNN workloads in autonomous embedded systems. NeuOS can manage energy optimization and dynamic accuracy adjustment for DNNs based on specific system constraints via smart coordinated system- and application-level decision-making.

NeuOS is designed fundamentally based on the idea of multi-DNN execution by introducing the concept of cohort, a collective set of DNN instances that can communicate through a shared channel. To track this cohort, we address how latency, energy, and accuracy can be measured and propagated efficiently in the multi-DNN cohort.

Besides the fundamental goal of providing latency predictability (i.e., meeting deadlines for processing each DNN instance), NeuOS addresses the challenge of balancing energy at system level and accuracy at application level for DNNs, which has never been addressed in literature to the best of our knowledge. Balancing three constraints at various execution levels in the multi-DNN scenario requires smart coordination 1) between system level and application level decision making, and 2) among multiple DNN instances.

Towards these coordination goals, we introduce two algorithms in Sec. 4.2 that are executed at the layer completion boundary of each DNN instance: one algorithm that can predict the best system-level DVFS configuration for each DNN member of the cohort to meet deadline and minimize power for that specific member in the upcoming layer, and one algorithm that decides what application level approximation configuration is required for others if any one of these system-level DVFS decisions were chosen. These two algorithms effectively propagate all courses of action for the next layer in order to meet the deadline. Based on these two algorithms, we propose an optimization problem in Sec. 4.3 that can decide the best course of action depending on the system constraint, and minimize system overhead. This method is effective because 1) it introduces an identical decision-making among all DNN instances in the cohort and solves the coordination problem between system-level and application-level decision making, and 2) provides adaptability to three typical scenarios imposing different constraints on energy and accuracy.

Implementation and Evaluation. We implement a system prototype of NeuOS and extensively evaluate NeuOS using popular image detection DNNs as a representative of convolutional deep neural networks used in AES. The evaluation is done under the following conditions:

- **Extensible in terms of architecture.** We fully implement NeuOS using a set of popular DNN models on two different platforms: an NVIDIA Jetson TX2 SoC (with architecture designed for low overhead embedded systems), and an NVIDIA AGX Xavier SoC (with architecture designed for complex autonomous systems such as self-driving cars).
- **Multi-DNN scenarios.** We ensure that our system can trade-off and balance multiple DNNs in all conditions by testing NeuOS under three cohort sizes: a small 1-process, a medium 2-4 process, and a large 6-8 process.

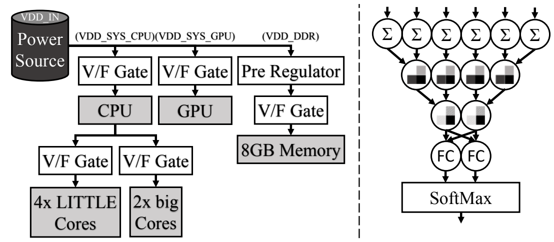
[3]The latest version of NeuOS can be found at https://github.com/Soroosh129/NeuOS.

- **Latency predictability.** We extensively compare NeuOS to six state-of-the-art solutions in literature, and find that NeuOS rarely misses deadlines under all evaluated scenarios, and can improve runtime latency on average by 68% (between 8% and 96% depending on DNN complexity) on TX2, by 40% on average (between 12% and 89%) on AGX, and by 54% overall.
- **Versatility.** NeuOS can be easily adapted to the following three constraint scenarios:
  - **Balanced energy and accuracy.** Without any system constraints given, NeuOS is proved to be energy efficient while sacrificing an affordable degree of accuracy, improving energy consumption on average by 68% on TX2, by 40% on average on AGX, while incurring an accuracy loss of 21% on average (between 19% and 42%).
  - ***Min energy.*** When energy is constrained to be minimal, NeuOS is able to sacrifice accuracy a small amount (at most 23%) but further improve energy consumption by 11% over the general unrestricted case, while meeting the latency requirement.
  - ***Max accuracy.*** When accuracy is given as a constraint, NeuOS is able to improve accuracy by 10% on average compared to balanced case, but also sacrifices energy by only a small amount, increasing by 23% on average.

2 Background

DVFS space in autonomous systems. The trade-off between latency and power consumption is usually achieved via adjustments to frequency and/or voltages of hardware components. A software and hardware technique typical of modern systems is DVFS. Through DVFS, system software such as the operating system or hardware solutions can dynamically adjust voltage and frequency. To understand this technique better, consider Fig. 2(a), showing the components of a Jetson TX2, which contains a Parker SoC with a big.LITTLE architecture with 2 NVIDIA Denver big cores and 4 ARM Cortex A53 LITTLE cores. The Parker SoC also contains a 256-core Pascal-architecture GPU. The TX2 module also contains 8 GB of shared memory (the Jetson AGX Xavier also used in Sec. 5 has a more advanced Xavier SoC with 8 NVIDIA "Carmel" cores, a 512 Volta-architecture GPU, and 16GB of shared memory). Each component includes a voltage/frequency (V/F) gate that can be adjusted via software. The value for frequency and voltage for each component forms a unique topple, called a DVFS configuration throughout this paper.

DNN and its approximation techniques. Fig. 2(b) depicts a simplified version of a Deep Neural Network (DNN). Neurons are the basic building blocks of DNNs. Depending on the layer neurons belong to, they perform various different operations. A DNN may contain multiple layers of different types, such as the convolutional and the normalization layers, which are connected via their inputs and outputs.



(a) Jetson TX2 V/F Structure (b) An example DNN

Figure 2: DVFS configuration space and DNN structure.

DNNs by nature are approximation functions [36]. DNNs are trained on a specific training set. After training, accuracy is measured by using a test data set, set aside from the training set and measuring the accuracy (e.g., top-5 error rate–comparing the top 5 guesses against the ground truth). The accuracy of the overall DNN can be adjusted by manipulating the layer parameters.

A rich set of DNN approximation techniques have been proposed in the literature and adopted in the industry [17, 58, 24, 29, 43, 56, 7, 18]. Such techniques aim at reducing the computation and storage overhead for executing DNN workloads. An example technique to provide approximation for convolutional layers is Lowrank [45], which performs a lowrank decomposition of the convolution filters. In our implementation, dynamic accuracy adjustment or "hot swapping" layers will refer to applying the lowrank decomposition to the upcoming layers before their execution. Note that applying such approximation adjustments on the fly is possible because the generated pair of layers have the exact combined input and output dimensions. Moreover, this adjustment is only possible for future layers at each layer boundary.

Measuring Accuracy. The approximation on the fly will affect the final accuracy. Due to the dynamic nature of this adjustment, the exact value of accuracy measurement using traditional methodology is impractical. Most related work thus incorporate an alternative scoring method [3], where the system will deduce the accuracy score accordingly if certain approximation techniques are to be applied to the next layer. In our method, we assume a perfect score for the original DNN, and switching to the lowrank approximation of any layer will reduce the score by a set amount. For example, running AlexNet in its entirety will result in a score of 100. If we swap a convolutional layer with a lowrank version of that layer, the overall accuracy will be affected by some amount (e.g., 1 in our method), thus yielding a lower score (e.g., 99 under the scoring method). Therefore, the score is always relative to the original DNN configuration and not related to the absolute value of accuracy on a particular dataset. This method of keeping relative accuracy is still invaluable to maximizing accuracy in a dynamic runtime environment but cannot be used to calculate the exact accuracy loss.

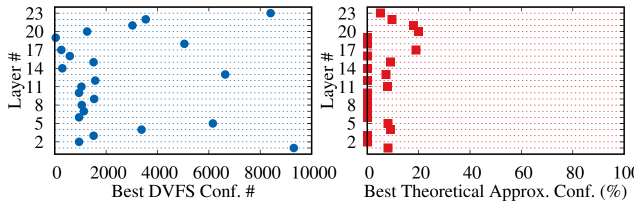


Figure 3: Calculated best system level DVFS configuration and best application level theoretical approximation configuration for AlexNet on Jetson TX2 in order to meet a 12ms deadline (0 means no approximation).

3 Motivation

In this section, we lay out several motivational case studies to understand the challenges that exist for DNNs, and gain insights on why existing approaches (or naively extended ones) may fail under our problem context.

3.1 Balancing in two-dimensional Space

The trade-off to meet a specified latency target while maximizing accuracy is done in a 2-dimensional space by choosing an approximation configuration for the application. Similarly, the 2-dimensional trade-off between energy and latency is done by changing an optimal DVFS configuration. Traditional control-theory based solutions treat the entire application as a black-box, and decide on what DVFS or approximation configuration should be chosen every few iterations of that specific application [21, 20]. However, treating DNNs as a blackbox does not yield the most efficient results. Fig. 3 left hand shows the best DVFS configuration for each layer of AlexNet among all possible DVFS configurations for a Jetson TX2 in terms of energy consumption. The y-axis is the layer number for AlexNet, and the x-axis is the DVFS configuration index, partially sorted based on frequency and activated core counts. The dots show the configuration that has the absolute minimum energy consumption. As is evident, each layer has a different optimal DVFS configuration. More interestingly, we observe a non-linearity where sometimes faster DVFS configurations have lower energy consumption. This is due to the massive parallelism of GPUs, where increasing the frequency by 2x for example can yield a 10-fold improvement in performance, which outweighs the momentary increase in energy consumption. Fig. 3 right hand shows the best theoretical approximation configurations required for each layer of AlexNet in order to meet a 12ms deadline[4]. As is evident in the figure, each layer requires a different approximation configuration for optimal results.

Thus, the DNN must somehow become transparent to the system, conveying layer-by-layer information in order to make the correct decisions. This can make the decision space in the 2-dimensional space at least an order of magnitude harder (e.g., AlexNet has 23 layers) since every layer must be considered for each execution of the DNN application.

[4]Please see Sec. 4.2 for more details on how this is calculated.

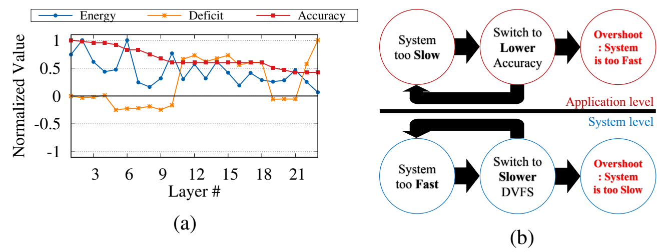


Figure 4: Negative feedback loop between an application-level solution and a system-level solution.

Observation 1: Layer-level trade-off makes the problem an order of magnitude harder than ordinary blackbox techniques.

3.2 Balancing in three-dimensional Space

Balancing energy/latency and accuracy/latency in isolation can be naive, and lead to unnecessary consumption of energy or reduced accuracy. Fig. 4a shows a similar experiment to Sec. 3.1, but both the system and application (Alexnet) are employed at the same time without any coordination. The goal of both solutions is to reach a 20ms deadline (by using latency deficit, LAG, as a guide (Sec. 4.2)). In the case of AlexNet, the system-level DVFS adjustment can be enough to meet the desired deadline. In an ideal scenario, only energy is adjusted slightly until AlexNet is not behind schedule. However, as is evident in the figure, normalized energy consumption and accuracy for each layer are both decreased continuously and dramatically. This is due to an unwanted negative loop, where a negative deficit (indicating that the system is behind schedule) has resulted in the application-level solution switching to a lower approximation configuration. Because these configurations are discrete, as we shall discuss in Sec. 4.2, the deficit will overshoot (at around layer 10) and becomes positive (meaning the system is ahead of schedule). The system-level solution would see this deficit as a headroom to reduce energy consumption, and in the case of Fig. 4a, has turned the positive deficit into a small negative at around layer 18. This cycle (as depicted in Fig. 4b) is repeated until the minimum approximation configuration is reached. This result is extremely undesirable in accuracy-sensitive applications such as autonomous driving (but can be okay for energy sensitive applications such as remote sensing). Thus, a feasible solution would be for the system and application to communicate, and make decisions based on given constraints for an application based on given constraints. This communication should be done at the granularity of layers, which makes the problem extra hard.

Observation 2: Trade-off in a 3-dimensional latency, energy, and accuracy optimization space is a significant challenge due to both system constraints as well as lacking harmony between application-level and system-level solutions.

3.3 Balancing for Multi-DNN Scenarios

To the best of our knowledge, no existing approach deals with multiple DNN instances in a coordinated manner.

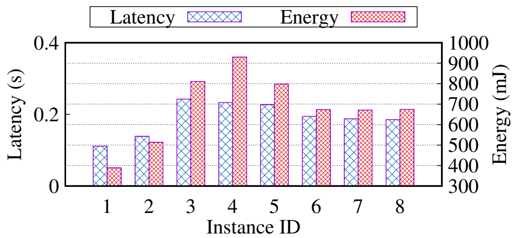


Figure 5: Energy consumption and execution time of running 8 instances of Resnet-50 on a Jetson TX2 under PredJoule.

Straightforwardly extending single-tasking latency/energy trade-off approaches, such as PredJoule [5], to multi-tasking scenarios would only result in decision-making that is local and greedy, based on locally measured variables. To showcase why coordination in this additional dimension is a key issue, examine Fig. 5, which shows the latency and energy consumption for running 8 DNN instances together averaged over 20 iterations under PredJoule on a Jetson TX2. We chose PredJoule because in our experiments, it outperformed all other existing solutions on exploring the 2D tradeoff between latency and energy for DNNs. The left (right) y-axis in Fig. 5 depicts the latency (energy consumption) in seconds (miliJoules) for each instance. As is evident in the figure, the DVFS management is greedy, resulting in instances 1 and 2 having relatively good latency and energy consumption. This greediness has pushed the rest of the DNN instances into unacceptable latency range (which is above 150ms for ResNet-50) because the chosen DVFS configuration at each layer boundary has been mostly beneficial only to the current layers of DNN instance 1 and 2. Moreover, the distribution of timing and energy consumption is not even across all instances because of the same reason. This disparity is the result of an uncoordinated system solution that chooses DVFS configurations greedily based on local variables.

Observation 3: In addition to the 2D and 3D complexities of solving the latency/accuracy/energy trade-off, a complete system solution must also accommodate for Multi-DNN scenarios, which are inherently more complicated to model and predict than single-DNN scenarios. The case studies also imply that naive extensions on existing single-DNN 2D solutions may fail in multi-DNN cases because they make greedy decisions based on local variables without coordination towards being globally optimal.

4 System Design

4.1 NeuOS Overview

To optimize the three-dimensional tradeoff space at the layer granularity, two basic research questions need to be answered first: 1) *how* to define and track the values of the three performance constraints in the system, and 2) *what* target should be imposed for optimizing each constraint.

For the first research question, we define a value of LAG (defined in Sec. 4.2, as a measurement of how far behind the DNN is compared to an ideal schedule that meets the relative deadline D), which tracks the progress of DNN execution at layer boundaries, P for energy consumption (in mJ) for each layer, and a variable X to reflect accuracy. We choose to track LAG at runtime instead of using an end-to-end optimization because it is more practical due to two reasons: 1) in a multi-DNN scenario, predicting the overlap between different DNN instances (and thus coordinating an optimal solution) cannot be done offline without making unrealistic assumptions, such as synchronized release times, and, 2) LAG is especially useful in a real system since it can account for outside interference, such as interference by other processes in the system, whereas an end-to-end optimization framework could miss the latency target. Moreover, as we shall discuss in Sec. 4.2, the value of P can be inferred by LAG in our design as these two variables fundamentally depend on the runtime DVFS configuration. Thus, the essential variables to track the status of a DNN execution can be simplified to $\{LAG, X\}$. Since we are dealing with a multi-DNN scenario, each DNN instance will have its own set of these variables. To know the collective status of the system, each DNN instance will put its variables in a shared queue.

In order to answer the second question regarding what optimization targets should be imposed on the system, we focus on the following three typical scenarios (expanded on in Sec. 4.3) that entail different performance constraints:

- **Min Energy** (M_P) is when NeuOS is deployed on an embedded system with a critically small energy envelope. Thus, the system should minimize energy without sacrificing too much accuracy. This scenario is motivated by applications seen in extremely power-limited systems such as drones, robotics, and a massive set of internet-of-thing devices.
- **Max Accuracy** (M_A) is when NeuOS is deployed on a system that has limited energy but accuracy is of utmost importance. Thus, the system should try to maximize accuracy without losing too much energy. This scenario is motivated by CPS-related applications such as autonomous driving.
- **Balanced Energy and Accuracy** (S) describes a more general, flexible scenario when the system is limited by both energy consumption and accuracy requirements, but no priority is given to either. Thus, the system should try to balance energy consumption and accuracy.

With the given scenarios and the values of $\{LAG, X\}$ at hand, we can answer the two key research questions presented in our motivation: 1) how to coordinate in a multi-DNN scenario such that the overall system is balanced and can meet the performance constraints, and, 2) how to efficiently tradeoff between latency, energy, and accuracy given the complexity of the problem space and how to prevent the negative feedback loop discussed in Sec. 3.2?

Design overview. Fig. 6 shows the overall design of NeuOS

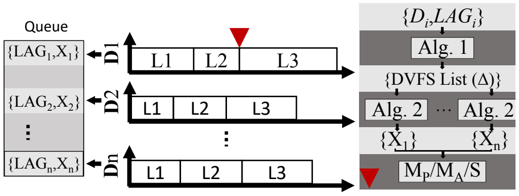


Figure 6: Design Overview

around $\{LAG, X\}$. The left side depicts the shared queue among multiple DNN instances. In the middle, a simple example of n concurrently running DNN instances each with three layers is shown. NeuOS makes runtime decisions on DVFS and DNN approximation configuration adjustments at layer boundaries, i.e., whenever a layer of a DNN instance completes. This is beneficial not only because applying approximation on-the-fly is possible only at layer boundaries, but in terms of overhead as well (as proved by our evaluation).

As illustrated in the figure, at the boundary between layers L2 and L3 of the first DNN instance, NeuOS is going through the process of decision-making which contains several steps. The first step is Alg.1, which senses the last known value of LAG for each DNN instance. Alg. 1 decides what DVFS configuration (at system level) is best for each instance in order to meet their deadlines D, outputting a list of potential DVFS configurations (Δ), where each member of the list corresponds to a DNN instance. In the next step, the list of potential DVFS configurations are fed into Alg.2, which predicts what approximation $\{X_i\}$ (at application level) would be required for other DNN instances to meet the deadline if the DVFS configuration for any one of the DNN instances is applied. Thus, Alg. 1 and Alg. 2 in tandem discover all possible courses of action the system can take to meet the deadline. However, at this point, no decision has been made on what DVFS configuration or accuracy configuration should be chosen for the system, because that depends on the given system constraint. This problem is inherently an optimization problem of finding the best possible choice in the propagated configuration space. We present this optimization problem formally in Sec. 4.3, where depending on broad scenarios, a particular setting is chosen for the next period of execution. In the last step of NeuOS, the system chooses one of these possibilities based on the scenario involved.

4.2 Coordinated System- and Application-level Adjustments

In this section, we expand on how runtime LAG is measured, how it relates to energy consumption, how accuracy X is calculated, and how the two developed algorithms take advantage of these two measurements to discover all possible choices the system can make efficiently in order to reduce the LAG to zero and meet the deadline.

LAG. We quantify the relationship between the partial execution time at time t of DNN instance i (e_i) and its relative deadline D_i as a form of LAG [51], denoted by LAG_i. LAG_i is a local variable (that can be updated at layer boundaries) for each DNN instance that keeps track of how far ahead or how far behind the DNN instance is compared to the deadline at time t. LAG_i is calculated as:

$$LAG_i(t, L_i(t)) = \sum_{l \in L_i(t)} (d_l - e_l), \tag{1}$$

in which $L_i(t)$ is the list of the layers of instance i that have completed by time t. For layer $l \in L_i(t)$, d_l and e_l depict the sub-deadline for layer l and the recorded execution time for layer l, respectively. NeuOS keeps track of e_l by measuring the elapsed time between each layer. Moreover, we use the proportional deadline method [38] to devise sub-deadlines for each layer based on D_i, the relative (end-to-end) deadline of DNN instance i, in which the subdeadline d_l for layer l is calculated as:

$$d_l = (e_l / \sum_{x \in L_i} (e_x)) \cdot D_i, \tag{2}$$

where $\sum_{x \in L_i}(e_x)$ denotes the execution time of DNN i. The proportional nature of sub-deadlines means that they only need to be calculated once for the lifetime of a given DNN instance on a platform.

Each DNN instance i would broadcast LAG_i among all instances via the shared queue. Thus, LAG_i would reflect the last known status of DNN instance i up to the last executed layer. We call the collection of LAG from all instances the LAG cohort, and we denote it by Φ. At completion of a DNN instance, a special message is sent to the cohort so that every DNN instance in the system is aware of their exit.

Based on the LAG cohort, the DNN instances can make decisions on accuracy and DVFS. A cohort will be perfect if every LAG within it is 0, or $\forall LAG_i \in \Phi, LAG_i = 0$. This means that all layers have exactly finished by their sub-deadline so far. Thus, the system has reasons to believe that the DNN instances will exactly finish by the deadline and do not require a faster DVFS or an approximation configuration, saving energy and accuracy in the process.

Since LAG indicates how far behind ($LAG < 0$) or ahead ($LAG > 0$) each DNN is, the DVFS and the approximation configuration need to be adjusted to run faster or slower accordingly. However, energy consumption and accuracy constraints must also be considered. We discuss each next.

System-level DVFS adjustment. At system-level, the question is which DVFS configuration is the best given the state of Φ to minimize energy consumption while reducing LAG to zero? The answer would vary between different DNN instances in the cohort, as they exhibit different LAGs. Moreover, different layers react differently to DVFS adjustments.

Alg. 1 is responsible for finding the best DVFS configuration for each DNN instance in the cohort. Alg. 1 takes as input the LAG cohort Φ and a SpeedUp/PowerUp table for the current layer of each DNN instance i. The structure of

Algorithm 1 Δ Calculator.

```
Input: Φ        ▷ Progress Cohort
Input: SpeedUp/PowerUp[]    ▷ The SpeedUp/PowerUp table of DNNs.
Output: Δ

1: function ReturnΔ(Φ)
2:     for LAG_i in Φ do
3:         S_{P_i} ← (D_i + LAG_i) / D_i.
4:         δ_i ← LookUp(SpeedUp/PowerUp[S_{P_i}])
```

Algorithm 2 X_i Calculator.

```
Input: Δ        ▷ Potential DVFS list.
Input: SpeedUp/Accuracy[]   ▷ The SpeedUp/Accuracy table of DNNs.
Input: SpeedUp/PowerUp[]    ▷ The SpeedUp/PowerUp table of DNNs.
Output: X[][]   ▷ The accuracy list for each DNN instance for each δ

1: function ReturnX_i(Δ)
2:     for δ_c in Δ do
3:         for i = 0 to i < |Δ| do
4:             S_{A_i} ← S_{P_i}(δ_c)·(D_i + LAG_i) / D_i
5:             X[c][i] ← LookUp(SpeedUp/Accuracy[S_{A_i}])
```

Table 1: SpeedUp/PowerUp and SpeedUp/Accuracy tables.

(a) SpeedUp/PowerUp for a layer of DNN instance i.

DVFS Configuration(δ)	SpeedUp	PowerUp
1	1x	1x
2	2.1x	2x
3	2.8x	1.5x

(b) SpeedUp/Accuracy.

X	SpeedUp
81%	1x
71%	1.8x
59%	2.5x

the SpeedUp/PowerUp table is depicted in Table 1a. The first column of Table 1a is the index for all the possible DVFS configurations in the system. The second column indicates how fast each DVFS configuration is in the worst case scenario compared to the baseline DVFS configuration (baseline is usually chosen to be the slowest configuration). The third column indicates how much power that DVFS configuration will consume relative to baseline.

Storing relative speedup and powerup values (instead of absolute measurements) is useful for looking up the table. In Alg. 1, given a LAG_i (line 2) and a relative deadline D_i for DNN instance i, the required speedup (denoted as S_i) could be directly calculated as (line 3):

$$S_P = \frac{D_i + LAG_i}{D_i}, \tag{3}$$

in which S_P is the speedup (or slowdown) value calculated as the relationship between the current projected execution time ($D_i + LAG_i$) and the ideal execution time (D_i). Since LAG can be negative or positive, the value of S_P can indicate a slowdown or speedup, where the slowdown is a way to conserve energy, which is the goal of NeuOS. The LookUp procedure (line 4) would then find the closest DVFS configuration that matches the speedup (or slowdown) in relation to the current configuration.

For our Alg. 1 to operate, we prepare a structure such as Table 1a for all DNN instances in a hashed format[5]. The LookUp procedure would then directly find a bucket by using the SpeedUp as an index. The output of Alg. 1 is a set $\Delta = \{\delta_1, \delta_2, ..., \delta_n\}$, in which δ_i is the ideal DVFS configuration for DNN instance i in order to meet the deadline. Imagine we ultimately decide that $\delta_c \in \Delta$ is the best DVFS configuration for the next scheduling period. A very interesting question would be that, what is the effect of applying δ_c on other DNN instances $i \neq c$? The speedup of δ_c for other DNN instances can be calculated by using δ_c as the lookup key in their corresponding SpeedUp/PowerUp table. But what if this speedup does not reduce LAG_i to zero? To solve this problem, we next present the algorithm that calculates the application-level approximation required to reduce LAG_i to zero given a DVFS configuration $\delta_c \in \Delta$.

Application-level accuracy adjustment. Alg. 2 portrays the procedures to calculate the required approximation for the upcoming layers of all DNN instances based on a DVFS configuration. If the instance i is behind the ideal schedule by LAG_i, with a relative deadline of D_i, and if the chosen DVFS configuration is δ_c, the remaining required speedup can be calculated as follows (line 4):

$$S_{A_i}(\delta_c) = \frac{S_{P_i}(\delta_c) \cdot (D_i + LAG_i)}{D_i}, \tag{4}$$

in which $S_{A_i}(\delta_c)$ is the required speedup (or slowdown) via approximation for DNN instance i when DVFS configuration δ_c is chosen, and $S_{P_i}(\delta_c) \cdot (D_i + LAG_i)$ is the new projected execution time of DNN instance i. The value of S_{A_c}, the speedup from accuracy for the chosen DVFS configuration, should always be zero or less than zero since by definition, δ_c is the ideal DVFS configuration for c and requires no additional speedup from approximation.

The value of S_{A_i} is then used as a lookup key to a new table, called the SpeedUp/Accuracy table, depicted in Table 1b. Table 1b stores the relative worst case execution times for each layer's approximation configuration. We index each row by X, which is the value of the total accuracy of that configuration[6]. Note that the exact value of X has no effect in the algorithm and what matters is the relative order in Table 1b (i.e., the lower we go down the table, the lower the relative accuracy). The output of Alg.2 is the row index in the SpeedUp/Accuracy table sufficient to meet the deadline for all DNN instances except c. We denote this index for layer k of DVFS configuration i as X_i^k. This value is then broadcasted in the accuracy cohort and indicates the application-level

[5] Our hashing is custom, and hashes the relationship between SpeedUp and PowerUp. This method relies on partially sorting the DVFS configuration space. You can find the latest hashing code at https://git.io/Jfogq

[6] Each row could be indexed by any measure. However, indexing with X has benefits in overhead reduction for the LookUp procedure in Alg. 2 because it can be more easily hashed.

configuration chosen for the next immediate layer of the corresponding DNN instance.

The remaining question is that which δ_c should be chosen. We answer this question next.

4.3 Constraints and Coordination

The combination of Alg. 1 and Alg. 2 produces a list of potential DVFS configurations Δ, and for each DVFS configuration in Δ, a corresponding list of required approximations for all DNN instances in the cohort if that DVFS configuration were to be applied. Such a scenario can be visualized as a decision tree. The remaining question of our design would be which path to go down to in order to have a perfect *LAG* cohort. As discussed in Sec. 3.2, the requirements on energy and accuracy can vary depending on specific scenarios. We present the following three approaches based on the three scenarios defined in Sec. 4.1, i.e., minimum energy (M_P), maximum accuracy (M_A), and balanced energy and accuracy (S).

Min Energy. This approach aims at minimizing power usage at the cost of accuracy. To choose the best DVFS configuration in the DVFS candidate set Δ, we should look at the corresponding $S_{P_i}(\delta_c), \delta_c \in \Delta$ values in the SpeedUp/PowerUp table and choose the δ_c that has the smallest PowerUp value for that corresponding DNN instance, namely:

$$\delta_c = \{\delta_i \in \Delta \mid PowerUp_i(\delta_i) \leq PowerUp_i(\delta_x), \forall \delta_x \in \Delta\}, \tag{5}$$

in which $PowerUp_i(\delta_i)$ is extracted from the SpeedUp/PowerUp table of DNN instance i. Note that in our experience, the values of PowerUp can be non-linear in relation to SpeedUp, and hence, a comprehensive search as noted above is required. Then, using Alg. 2, the accuracy cohort can be calculated and broadcasted based on the projected new execution times. Even though this approach has the best power consumption, it will not have the best accuracy since many processes will most likely not meet the deadline without significant loss of accuracy, since the speedup from DVFS alone will likely not make up for the vast majority of the progress values in the cohort.

Max Accuracy. In this method, our system chooses the DVFS configuration δ_c in such a way that:

$$\delta_c = \{\delta_i \in \Delta \mid \sum(S_{Aj}(\delta_i)) \leq \forall \sum(S_{Aj}(\delta_x \in \Delta)), \\ \forall \text{ DNN instance j in cohort}\}, \tag{6}$$

in which $\sum S_{Aj}(\delta_i)$ is the sum of all the required speedups from approximation (S_{Ai}) for configuration δ_i, and $\leq \forall \sum(S_{Aj}(\delta_x \in \Delta))$ is indicating that the sum of approximation-induced speedup for the chosen δ_c should be less than or equal any other sum of approximation values for other $\delta_x \in \Delta$ (this indirectly ensures minimized accuracy loss).

Statistical Approach for Balanced Energy and Accuracy. To achieve balanced energy and accuracy, we propose a statistical approach that checks the state of Δ and the projected accuracy cohort in statistical terms to make a decision. The calculation of S_{P_i} and S_{A_i} (which depends on S_{P_i}) resemble the form of Bivariate Regression Analysis (BRA) [57], in which:

$$S_{A_i} = S_{P_i} \cdot \frac{D_i + LAG_i}{D_i} + 0, \tag{7}$$

in which, $\frac{D_i+LAG_i}{D_i}$ is called the influence of S_{P_i} on the required approximation. To measure this influence, we first calculate

$$I = \sum_{i=0}^{i=n-1} \frac{D_i + LAG_i}{D_i}, \tag{8}$$

in which I is the collective influence of LAG on approximation. If the value of I is high, it means that the accuracy can be more adversely affected by a low value of DVFS-induced speedup(S_{P_i}). Similarly, a low value of I means that the accuracy can remain minimal even with a low value for DVFS-induced speedup. We simplify our decision making by dividing the *LAG* cohort Φ into three groups based on how big or small the value of *LAG* is. The boundary for the intervals is calculated using:

$$Boundary = \frac{\max\{\Phi\} - \min\{\Phi\}}{3}. \tag{9}$$

The three groups $G1[0 \ldots Boundary]$, $G2[Boundary \ldots 2 \cdot Boundary]$, and $G3[2 \cdot Boundary \ldots 3 \cdot Boundary]$ are then formed, and the ultimate DVFS configuration is chosen as:

$$\delta_c = \begin{cases} median(G1) & if(I < t) \\ median(G2) & if(t < I < 1+t) \\ median(G3) & if(I > 1+t) \end{cases}, \tag{10}$$

in which, t is a threshold for I, set to the standard deviation σ of the set I. However, t can be chosen by the system designer to indicate a requirement on power consumption and accuracy. A small value for t will push the system towards faster DVFS configurations and vice versa.

Discussion on choosing modes and safety. We would like to conclude our design by a discussion on which modes to choose and the safety concern it might entail. Our stand from a system perspective is to design a flexible system architecture that can adapt to various external needs. Where absolute mission-critical applications are concerned, we offer Max Accuracy. Nonetheless, depending on the safety requirement, our Balanced approach might be good enough with the proper threshold t even for applications such as self-driving vehicles. However, choosing a mode dynamically at runtime or statically for a particular system offline has more to do with the certification standards (which are in their preliminary stages for self-driving vehicles) as well as the requirement on maximum reaction time and accuracy. Thus, we believe the decision should be relegated to an external policy controller [54, 31, 6, 34, 52]

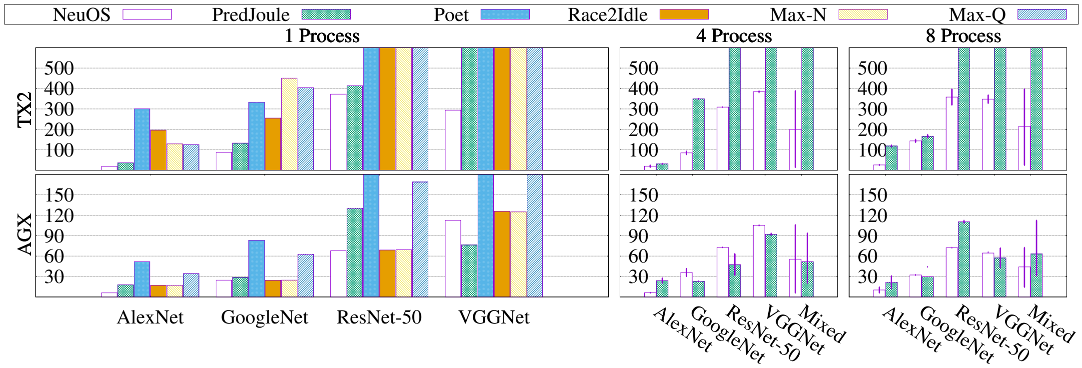


Figure 7: Energy under various methods (in mJ) for 1, 4, and 8 instances of 4 DNN models.

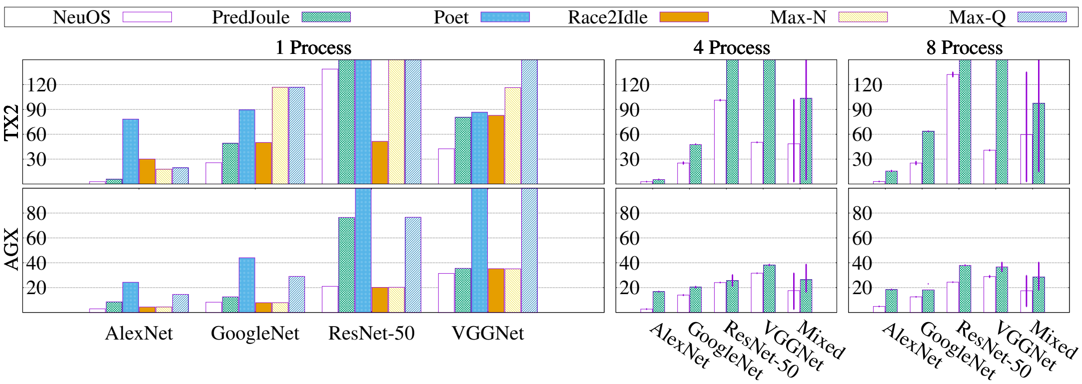


Figure 8: Latency under various methods (in ms) for 1, 4, and 8 instances of 4 DNN models.

5 Evaluation

In this section, we test our full implementation on top of Caffe [25] with an extensive set of evaluations.

5.1 Experimental Setup

In this section we lay out our experimental setup, which includes two embedded platforms and four popular DNN models. We compare NeuOS to 6 existing approaches.

Testbeds. We have chosen two different NVIDIA platforms imposing different architectural features (since deployed autonomous systems solutions, particularly for autonomous driving and robotics, seem to gravitate towards NVIDIA hardware as of writing this paper [26, 30]) to showcase the cross-platform nature of our design when it comes to hardware. We use NVIDIA Jetson TX2, with 6 big.LITTLE ARM-based cores and a 256-core Pascal based GPU with 11759 unique DVFS configurations, and the NVIDIA Jetson AGX Xavier, the latest powerful platform for robotics and autonomous vehicles with an 8-core NVIDIA Carmel CPU and a 512-core Volta-based GPU with 51967 unique DVFS configurations.

DNN models. Having a diversified portfolio of DNN models can showcase that NeuOS is future proof in the fast-moving field of neural networks. To that end, we use AlexNet [32], ResNet [19], GoogleNet [2], and VGGNet [49] in our experiments. Our method dynamically applies a lowrank version of a convolutional layer whenever approximation is necessary by keeping both version of the layer in memory for fast switching. The deadline for each DNN instance is based on their worst-case execution time (WCET) on each platform, and is set to 10ms, 30ms, 150ms, and 40ms respectively for Jetson TX2 and 5ms, 10ms, 25ms, 30ms respectively for AGX Xavier. Note that ResNet is much slower on Jetson TX2 due to the older JetPack software.

Small Cohort, Medium Cohort, Large Cohort sizes. We test NeuOS under three different cohort size classes to test for adaptability and balance: 1 process for small, 2 to 4 processes for medium, and 6 to 8 processes for large. Each of these cohort sizes have their own unique challenges. We measure average timing, energy consumption, and accuracy for these scenarios and provide a measure of balancing where applicable. For medium and large cohorts, we include a mixed scenario, where different DNN models are executed, which represents systems that use different DNNs (for example for voice and image recognition). For the medium cohort, one instance of each DNN model and for the large cohort, two instances of each model are initiated.

Adaptability to different system scenarios. As discussed in Sec. 4.1, we consider three different scenarios with different limits on latency, energy and accuracy: minimum energy, maximum accuracy, and balanced energy and accuracy.

Compared Solutions. We implement and compare six state-

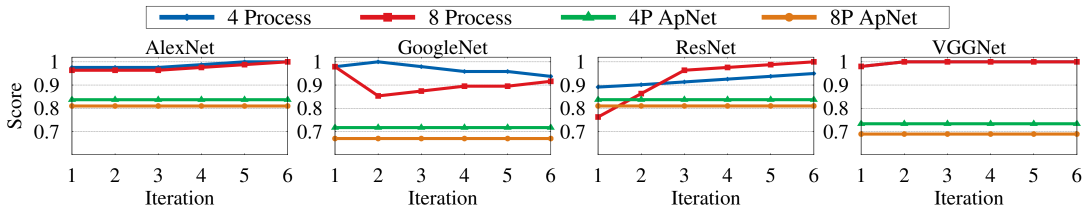


Figure 9: Average accuracy (y-axis as a fraction of 1) of the cohort over iteration of execution for 4 different DNN models with 4 and 8 instances compared to ApNet (x-axis is the iteration number).

of-the art solutions from the literature, including DNN-specific and DNN-agnostic ones, software-based DVFS and hardware-based DVFS, and application-level and system-level solutions. We present a short detail for each as follows.

PredJoule [5] is a system-level solution tailored towards DNN by employing a layer-based DVFS adjustment solution for optimization latency and energy. *Poet* [23] is a system-level control-theory based software solution that balances energy and timing in a best-effort manner via adjusting DVFS. We choose to compare against Poet instead of its extended approaches including JouleGuard [21] and CoAdapt [20], as they employ essentially the same set of control theory-based techniques as Poet. *ApNet* [3] is an application-level solution based on DNNs that can theoretically provide a per-layer approximation requirement offline to meet deadlines. *Race2Idle* [28] is the classic "run it as fast as you can" philosophy, which is always interesting to compare to. *Max-N* [10, 11] is a reactive hardware DVFS that maximizes frequency and sacrifices energy in the name of speed, in NVIDIA embedded hardware. *Max-Q* [10] is a hardware DVFS on Jetson TX2 that dynamically adjusts DVFS on the fly to conserve energy. However, this feature has been removed from the Xavier platform [11], and is replaced by low level power caps, such as 10W, 15W, and 30W. We use the 15w cap instead of Max-Q on Xavier.

5.2 Overall Effectiveness

In this section, we measure the efficacy of NeuOS on the two evaluated platforms under the balanced scenario. Since our design is concerned with timing predictability, energy consumption, and DNN accuracy, we measure all three constraints and compare against state-of-the-art literature under each platform and each scenario.

5.2.1 Small Cohort

Energy. The left column of Fig. 7 depicts our measurements in terms of average energy consumption compared to a GPU-enabled Poet, Max-Q, Max-N, PredJoule, and Race2Idle using AlexNet, GoogleNet, ResNet-50 and VGGNet as the base DNN model and using lowrank as the approximation method. As is evident in the figure, NeuOS is able to save energy considerably compared to all other methods on Jetson TX2 on all DNN models, with improvements of 68% on average for Jetson TX2 and 46% on average for AGX Xavier. This saving is due to the fact that in some cases accuracy is minimally traded off for the benefit of energy and timing.

On the Jetson AGX Xavier, NeuOS has better energy consumption compared to all other approaches on every DNN model except compared to PredJoule for VGGNet. As we shall see for timing, PredJoule misses the deadline of 30 for VGGNet, and NeuOS has decided to sacrifice energy to meet timing.

Latency. Fig. 8 shows the average execution time for NeuOS compared to the 5 methods and using 4 DNN models. NeuOS outperforms all other approaches, improving on average execution time by 68% on Jetson TX2 and by 40% on AGX Xavier. It is also interesting to note that AGX Xavier is much faster than Jetson TX2, by 70% on average.

Tail Latency. Through response time measurements, we find that NeuOS rarely misses the deadline (3.25% of the time). Moreover, the variance is low with the 99th percentile execution time for AlexNet, GoogleNet, ResNet-50, and VGGNet as 9.2 ms, 48 ms, 130.3 ms and 39.1 ms for TX2 and 5.0 ms, 12.0 ms, 26.1 ms and 36.2 ms for AGX respectively.

Accuracy. We also measure the accuracy loss of NeuOS compared to ground truth and compared to ApNet (Fig. 9 omits small cohort for clarity). ApNet is the only DNN-specific application level solution we are aware of. NeuOS has an approximation score of 0.94% on average (out of 1), which is better than ApNet by 21%.

5.2.2 Medium and Large Cohorts

In order to save space, we only compare PredJoule for the 4-process medium and 8-process large cohort sizes. In our testings, PredJoule already vastly outperforms other methodologies, and thus is a good comparison to NeuOS.

Energy. As is evident in the second and third columns of Fig. 7, NeuOS can almost always outperform PredJoule in terms of energy consumption on Jetson TX2 improving 70% on average. However, rather interestingly, NeuOS performs worse in terms of energy compared to PredJoule for GoogleNet, ResNet, and VGGNet on AGX Xavier. This is due to the fact that PredJoule again misses the deadlines on AGX Xavier, and NeuOS has sacrificed a negligible amount of energy (1.5% on average) in order to meet the deadline.

Latency. NeuOS always outperforms state-of the art, improving by 53% on average for Jetson TX2, and by 32% on average for AGX. This is due to the fact that NeuOS is able to leverage a small amount of accuracy and energy loss (in the case of AGX Xavier) for better timing and energy characteristics.

Tail Latency. We find that deadline miss ratio is about the

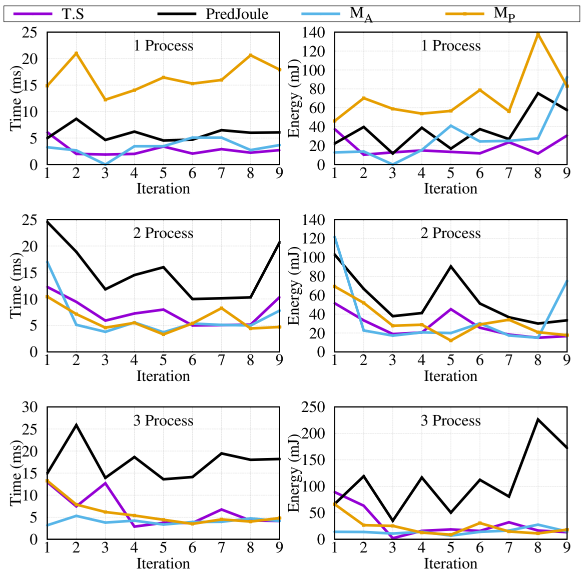


Figure 10: Performance of NeuOS under three scenarios compared to PredJoule on Jetson TX2.

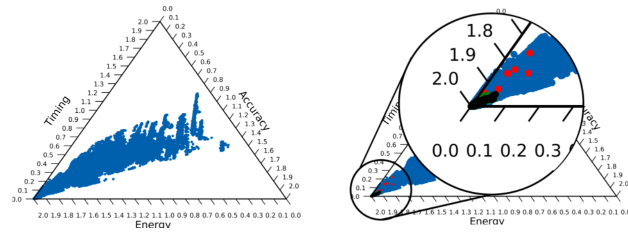
same as the small cohort. Moreover, the variance is similarly low with the 99th percentile execution time for AlexNet, GoogleNet, ResNet-50, and VGGNet as 10.4 ms, 39.2 ms, 101.7 ms and 69 ms for TX2 and 11 ms, 12.5, 26.3 and 35.9 ms for AGX respectively for the medium cohort and 13.6 ms, 40.8 ms, 190 ms and 72 ms for TX2 and 10.7 ms , 54 ms , 62 ms and 36.1 ms for AGX respectively for the large cohort.

Accuracy. Fig. 9 shows the average accuracy of the cohort over 6 iterations on AGX Xavier. As is evident in the figure, NeuOS generally improves upon accuracy as the system progresses because the optimization in Sec. 4.3 is able to find better DVFS configurations. When compared to the efficient approximation-aware solution APnet, NeuOS is able to achieve noticeably better accuracy in all scenarios.

Balance. A very important measure discussed in Sec. 3.3 is how balanced the system solution is when faced with multiple processes. To measure how balanced NeuOS is compared to PredJoule in the 4-process and 8-process scenarios, we include min-max bars in Fig. 7 and Fig. 8 to showcase the discrepancy between minimum and maximum timing/energy. As is evident in the figure, the discrepancy is negligible compared to the total energy consumption and execution time (up to 79 mJ and 4 ms). Thus, NeuOS maintains balance in the cohort. This is due to the coordinated cohorts and a uniform non-greedy decision making approach introduced in our design.

5.3 Detailed Examination on Tradeoff

In this section, we focus on the fact that system designers might require certain constraints that limits the ability of NeuOS in a certain dimension. To this end, we test our platform under three different scenarios: Maximum Accuracy (M_A), Minimum Power (M_P), and Balanced ($T.S$).



(a) The entire configuration space for all DVFS and accuracy combinations for Jetson TX2.

(b) Chosen configurations in the triangle space.

5.3.1 Energy and Latency.

We compare against PredJoule and measure average timing for the cohort in ms and average energy in mJ in Fig. 10 for 1-process (small), 2-process (medium), and 3-process (large) scenarios on AlexNet over 9 iterations on the Jetson TX2. PredJoule is shown as a black line. The deadline in this scenario is set to 25 to show the interesting characteristics of each method.

Balanced. As is evident in the figure, our statistical balanced approach outperforms PredJoule over all iterations. Notably in the case of medium and large cohorts, PredJoule has a particularly bad start in terms of timing and has higher fluctuation due to the greedy nature of DVFS selection.

Min Energy. Interestingly, M_P performs very bad (still meets the deadline) for the small cohort both in terms of timing and energy consumption. This is due to the algorithms discussed in Sec. 4.3. The system has switched to a very slow DVFS configuration to save energy. However, because of the non-linearity inherent in very slow DVFS configurations for GPUs [5], this has resulted in a very bad energy consumption as well. However, the coordination starts to pay off for medium and large cohorts. This is because a coordinated multi-process cohort needs faster DVFS configurations and thus, the circumstances push M_P out of the slow and power inefficient DVFS configuration subset. The greediness of PredJoule is inherent for medium and large cohorts in the form of very large fluctuations throughout the iterations.

Max Accuracy. M_A should improve accuracy while sacrificing energy and timing. We shall discuss the accuracy decisions shortly. However, the timing for M_A is worse than balanced energy by a negligible amount. The same is true for energy consumption (23% on average). This highlights a big design decision of the balanced scenario overall. Even for the balanced general approach, sacrificing accuracy is done very conservatively as was discussed earlier. Thus, the slight push toward perfect accuracy does not introduce very large overheads. However, as was discussed earlier guaranteeing a tight deadline (such as 10ms for AlexNet) requires some approximation if energy consumption is a consideration.

5.3.2 Energy-Accuracy Tradeoff.

For accuracy and to show where the variations of NeuOS jump in terms of system and application configurations, we bring back the triangle of Fig. 1, but with real DVFS and

Table 2: Average execution time overhead of NeuOS compared to other approaches on AlexNet (ms).

	1 Process	4 Process	8 Process
NeuOS	0.145	0.571	0.738
PredJoule	0.772	0.929	1.597
ApNet	0	3.27	5.85
Poet	151.03	604.12	1208.27

accuracy configurations with the selected configurations of M_P, M_A, and $T.S$ highlighted in Fig. 11b. As is evident in the triangle, the deadline limits the possible configurations to the bottom left corner. However, within that limitation, M_P (in red) has chosen configurations that are lower on energy consumption toward the upper right. On the other hand M_A (in green) has chosen configurations that are not as good in terms of energy consumption, but are better in terms of accuracy toward bottom left. Finally, $T.S$, colored black, is similar to M_A because of the high emphasis on accuracy in our design.

5.4 Overhead

Execution time: Table 2 shows the overhead of NeuOS compared to Poet, PredJoule, and ApNet using AlexNet as the baseline model on Jetson TX2 (times are in milliseconds). As is evident in Table 2, the overhead for NeuOS is negligible, especially compared to Poet and the overhead is also negligible compared to the overall execution time of AlexNet itself. The reason Poet is so slow is because it has to go through all DVFS configurations in a quadratic way ($O(n^2)$, n is the number of DVFS configurations). Even $O(n)$ would be unacceptable on embedded systems with more than 10000 unique DVFS configurations. This proves that applying our complexity reduction techniques (via hashing) is a must for a practical system solution. Moreover, NeuOS is more efficient than PredJoule and ApNet, especially in 4 Process and 8 Process scenarios.

Memory: As discussed in Sec. 5.1, our implementation keeps both the original and the lowrank approximation of the model in GPU memory for fast switching at layer boundaries. Moreover, NeuOS also holds the per-layer hash tables containing approximation and DVFS configuration information as described in Sec. 4. Table 3 depicts the added overhead in terms of both raw and percentage of the total NeuOS memory usage. As expected, the lowrank approximated version of each model has slightly less overall size compared to the original model. Nonetheless, this technique sacrifices memory overhead to improve latency and energy consumption. A viable alternative left as future work is dynamic approximation on the fly, which trades off latency for lower memory consumption. Moreover, the overhead of the hash tables is negligible compared to the total memory usage. Finally, the last two columns depict the cumulative maximum percentage of total available memory occupied by one instance of NeuOS for each platform (this memory usage also includes the temporary intermediate layer data [39]).

Table 3: Raw and percentage based memory overhead of applying NeuOS for each model.

	(a) Overhead in addition to Caffe			(b) Ratio to total memory	
	Lowrank	Hash Table	Ratio	Jetson TX2	AGX Xavier
AlexNet	226 MB	331 B	49%	10%	4%
GoogleNet	23 MB	2.1 KB	30%	2%	1%
ResNet-50	82 MB	3.2 KB	45%	7%	3%
VGGNet	509 MB	634 B	48%	25%	12.7%

6 Related Work

Trading off latency and power efficiency has been a hot topic in the related fields including real-time embedded systems and mobile computing [40, 8, 41, 53, 59, 16, 37, 42, 47, 1]. Due to the explosion of approximation techniques in different application domains, there has been several recent works [15, 35, 13] seeking to address this problem in a three dimensional space covering accuracy as well. Unfortunately, these works cannot resolve the problem as their applicability is limited in scope in various ways. JouleGuard [21], MeanTime [13], CoAdapt [20] and other similar approaches provide general system or hardware solution for non-DNN applications that claim to explore the three dimensional optimization space.

A very recent set of works including PredJoule [5], and ApNet [3] are able to provide latency predictability (meeting deadlines) for DNN-based workloads, yet they only consider two out of the three dimensions and focus on single-DNN scenarios. As we have discussed in Sec. 3.2, running ApNet and PredJoule at the same time even at different frequencies will result in a negative feedback loop. Thus, a better, more coordinated approach is required.

Such limitations would dramatically reduce the complexity of the optimization space, as the system-level and application-level tradeoffs focusing on a single task can be considered in an independent manner (e.g., pure system-level and application-level optimization under Poet [23] and CoAdapt, respectively). This results in solutions that are inapplicable to any practical autonomous real-time system featuring a multi-DNN environment.

7 Acknowledgment

We would like to thank our shepherd, Amitabha Roy, and the anonymous referees for their invaluable insight and advice into making this paper substantially better. This work is supported by NSF grant CNS CAREER 1750263.

8 Conclusion

This paper presents NeuOS, a comprehensive latency predictable system solution for running multi-DNN workloads in autonomous embedded systems. NeuOS can guarantee latency predictability, while managing energy optimization and dynamic accuracy adjustment based on specific system constraints via smart coordinated system- and application-level decision-making among multiple DNN instances. Extensive evaluation results prove the efficacy and practicality of NeuOS.

References

[1] Jorge Albericio, Patrick Judd, Tayler Hetherington, Tor Aamodt, Natalie Enright Jerger, and Andreas Moshovos. Cnvlutin: Ineffectual-neuron-free deep neural network computing. In *ACM SIGARCH Computer Architecture News*, volume 44, pages 1–13. IEEE Press, 2016.

[2] Mohammadhossein Bateni, Aditya Bhaskara, Silvio Lattanzi, and Vahab Mirrokni. Distributed balanced clustering via mapping coresets. In *Advances in Neural Information Processing Systems*, pages 2591–2599, 2014.

[3] Soroush Bateni and Cong Liu. Apnet: Approximation-aware real-time neural network. In *2018 IEEE Real-Time Systems Symposium (RTSS)*, pages 67–79, Dec 2018.

[4] Soroush Bateni and Cong Liu. Predictable data-driven resource management: an implementation using autoware on autonomous platforms. In *2019 IEEE Real-Time Systems Symposium (RTSS)*, pages 339–352. IEEE, 2019.

[5] Soroush Bateni, Husheng Zhou, Yuankun Zhu, and Cong Liu. Predjoule: A timing-predictable energy optimization framework for deep neural networks. In *2018 IEEE Real-Time Systems Symposium (RTSS)*, pages 107–118, Dec 2018.

[6] Raunak P Bhattacharyya, Derek J Phillips, Changliu Liu, Jayesh K Gupta, Katherine Driggs-Campbell, and Mykel J Kochenderfer. Simulating emergent properties of human driving behavior using multi-agent reward augmented imitation learning. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 789–795. IEEE, 2019.

[7] Wenlin Chen, James Wilson, Stephen Tyree, Kilian Weinberger, and Yixin Chen. Compressing neural networks with the hashing trick. In *International Conference on Machine Learning*, pages 2285–2294, 2015.

[8] Ping Chi, Shuangchen Li, Cong Xu, Tao Zhang, Jishen Zhao, Yongpan Liu, Yu Wang, and Yuan Xie. Prime: a novel processing-in-memory architecture for neural network computation in reram-based main memory. In *ACM SIGARCH Computer Architecture News*, volume 44, pages 27–39. IEEE Press, 2016.

[9] European Commission. Rolling plan for ict standardisation, 2018.

[10] NVIDIA Corp. Power management for jetson agx xavier devices. https://docs.nvidia.com/jetson/l4t/index.html#page/TegraLinuxDriverPackageDevelopmentGuide/power_management_jetson_xavier.html.

[11] NVIDIA Corp. Power management for jetson tx2 series devices. https://docs.nvidia.com/jetson/archives/l4t-archived/l4t-3231/index.html#page/TegraLinuxDriverPackageDevelopmentGuide/power_management_tx2_32.html.

[12] ETSI. Intelligent transport systems (its); vehicular communications; basic set of applications; part 3: Specifications of decentralized environmental notification basic service, Aug 2018.

[13] Anne Farrell and Henry Hoffmann. Meantime: Achieving both minimal energy and timeliness with approximate computing. In *USENIX Annual Technical Conference*, pages 421–435, 2016.

[14] ISO International Organization for Standardization. 26262: 2018. *Road vehicles—Functional safety*.

[15] Seungyeop Han, Haichen Shen, Matthai Philipose, Sharad Agarwal, Alec Wolman, and Arvind Krishnamurthy. Mcdnn: An approximation-based execution framework for deep stream processing under resource constraints. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '16, pages 123–136, New York, NY, USA, 2016. ACM.

[16] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. Eie: efficient inference engine on compressed deep neural network. In *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on*, pages 243–254. IEEE, 2016.

[17] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[18] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pages 1135–1143, 2015.

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[20] Henry Hoffmann. Coadapt: Predictable behavior for accuracy-aware applications running on power-aware systems. In *Real-Time Systems (ECRTS), 2014 26th Euromicro Conference on*, pages 223–232, 2014.

[21] Henry Hoffmann. Jouleguard: energy guarantees for approximate applications. In *Proceedings of the 25th Symposium on Operating Systems Principles*, pages 198–214. ACM, 2015.

[22] Sean Hollister. Tesla's new self-driving chip is here, and this is your best look yet., Apr 2019.

[23] C. Imes, D. H. K. Kim, M. Maggio, and H. Hoffmann. Poet: a portable approach to minimizing energy under soft real-time constraints. In *21st IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 75–86, April 2015.

[24] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*, 2014.

[25] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[26] Shinpei Kato, Shota Tokunaga, Yuya Maruyama, Seiya Maeda, Manato Hirabayashi, Yuki Kitsukawa, Abraham Monrroy, Tomohito Ando, Yusuke Fujii, and Takuya Azumi. Autoware on board: Enabling autonomous vehicles with embedded systems. In *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*, pages 287–296. IEEE, 2018.

[27] Jaanus Kaugerand, Johannes Ehala, Leo Mõtus, and Jürgo-Sören Preden. Time-selective data fusion for in-network processing in ad hoc wireless sensor networks. *International Journal of Distributed Sensor Networks*, 14(11), 2018.

[28] D. H. K. Kim, C. Imes, and H. Hoffmann. Racing and pacing to idle: Theoretical and empirical analysis of energy optimization heuristics. In *2015 IEEE 3rd International Conference on Cyber-Physical Systems, Networks, and Applications*, pages 78–85, Aug 2015.

[29] Yong-Deok Kim, Eunhyeok Park, Sungjoo Yoo, Taelim Choi, Lu Yang, and Dongjun Shin. Compression of deep convolutional neural networks for fast and low power mobile applications. *arXiv preprint arXiv:1511.06530*, 2015.

[30] B. Kisacanin. Deep learning for autonomous vehicles. In *2017 IEEE 47th International Symposium on Multiple-Valued Logic (ISMVL)*, pages 142–142, May 2017.

[31] Olga Kouchnarenko and Jean-François Weber. Adapting component-based systems at runtime via policies with temporal patterns. In *International Workshop on Formal Aspects of Component Software*, pages 234–253. Springer, 2013.

[32] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[33] TIMOTHY B. LEE. Tesla's autonomy event: Impressive progress with an unrealistic timeline, Apr 2019.

[34] Jesse Levinson, Jake Askeland, Jan Becker, Jennifer Dolson, David Held, Soeren Kammel, J Zico Kolter, Dirk Langer, Oliver Pink, Vaughan Pratt, et al. Towards fully autonomous driving: Systems and algorithms. In *2011 IEEE Intelligent Vehicles Symposium (IV)*, pages 163–168. IEEE, 2011.

[35] Yongbo Li, Yurong Chen, Tian Lan, and Guru Venkataramani. Mobiqor: Pushing the envelope of mobile edge computing via quality-of-result optimization. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 1261–1270. IEEE, 2017.

[36] Shiyu Liang and R Srikant. Why deep neural networks for function approximation? *arXiv preprint arXiv:1610.04161*, 2016.

[37] Robert LiKamWa, Yunhui Hou, Julian Gao, Mia Polansky, and Lin Zhong. Redeye: analog convnet image sensor architecture for continuous mobile vision. In *ACM SIGARCH Computer Architecture News*, volume 44, pages 255–266. IEEE Press, 2016.

[38] Jane W. S. W. Liu. *Real-Time Systems*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 2000.

[39] Zongqing Lu, Swati Rallapalli, Kevin Chan, and Thomas La Porta. Modeling the resource requirements of convolutional neural networks on mobile devices. In *Proceedings of the 25th ACM international conference on Multimedia*, pages 1663–1671, 2017.

[40] M. Maggio, E. Bini, G. Chasparis, and K. Årzén. A game-theoretic resource manager for rt applications. In *2013 25th Euromicro Conference on Real-Time Systems*, pages 57–66, July 2013.

[41] Nikita Mishra, Huazhe Zhang, John D Lafferty, and Henry Hoffmann. A probabilistic graphical model-based approach for minimizing energy under performance constraints. In *ACM SIGARCH Computer Architecture News*, volume 43, pages 267–281. ACM, 2015.

[42] Brandon Reagen, Paul Whatmough, Robert Adolf, Saketh Rama, Hyunkwang Lee, Sae Kyu Lee, José Miguel Hernández-Lobato, Gu-Yeon Wei, and David Brooks. Minerva: Enabling low-power, highly-accurate deep neural network accelerators. In *ACM SIGARCH Computer Architecture News*, volume 44, pages 267–278. IEEE Press, 2016.

[43] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550*, 2014.

[44] Francisca Rosique, Pedro J Navarro, Carlos Fernández, and Antonio Padilla. A systematic review of perception system and simulators for autonomous vehicles research. *Sensors*, 19(3):648, 2019.

[45] Tara N Sainath, Brian Kingsbury, Vikas Sindhwani, Ebru Arisoy, and Bhuvana Ramabhadran. Low-rank matrix factorization for deep neural network training with high-dimensional output targets. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6655–6659. IEEE, 2013.

[46] Rick Salay, Rodrigo Queiroz, and Krzysztof Czarnecki. An analysis of iso 26262: Using machine learning safely in automotive software. *arXiv preprint arXiv:1709.02435*, 2017.

[47] Ali Shafiee, Anirban Nag, Naveen Muralimanohar, Rajeev Balasubramonian, John Paul Strachan, Miao Hu, R Stanley Williams, and Vivek Srikumar. Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. *ACM SIGARCH Computer Architecture News*, 44(3):14–26, 2016.

[48] Akshay Kumar Shastry. *Functional Safety Assessment in Autonomous Vehicles*. PhD thesis, Virginia Tech, 2018.

[49] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[50] Nikolai Smolyanskiy, Alexey Kamenev, Jeffrey Smith, and Stan Birchfield. Toward low-flying autonomous mav trail navigation using deep neural networks for environmental awareness. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4241–4247. IEEE, 2017.

[51] I. Stoica, H. Abdel-Wahab, K. Jeffay, S. K. Baruah, J. E. Gehrke, and C. G. Plaxton. A proportional share resource allocation algorithm for real-time, time-shared systems. In *17th IEEE Real-Time Systems Symposium*, pages 288–299, Dec 1996.

[52] Chen Tang, Zhuo Xu, and Masayoshi Tomizuka. Disturbance-observer-based tracking controller for neural network driving policy transfer. *IEEE Transactions on Intelligent Transportation Systems*, 2019.

[53] Yaman Umuroglu, Nicholas J Fraser, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees Vissers. Finn: A framework for fast, scalable binarized neural network inference. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 65–74. ACM, 2017.

[54] Jean-François Weber. Tool support for fuzz testing of component-based system adaptation policies. In *International Workshop on Formal Aspects of Component Software*, pages 231–237. Springer, 2016.

[55] Wikipedia contributors. Ternary plot - wikipedia,the free encyclopedia. `https://en.wikipedia.org/wiki/Ternary_plot`, 2019. [Online; accessed 31-May-2019].

[56] Jian Xue, Jinyu Li, Dong Yu, Mike Seltzer, and Yifan Gong. Singular value decomposition based low-footprint speaker adaptation and personalization for deep neural network. In *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, pages 6359–6363. IEEE, 2014.

[57] Taro Yamane. Statistics: An introductory analysis. 1973.

[58] Tien-Ju Yang, Yu-Hsin Chen, and Vivienne Sze. Designing energy-efficient convolutional neural networks using energy-aware pruning. *arXiv preprint*, 2017.

[59] Huazhe Zhang and Henry Hoffmann. Maximizing performance under a power cap: A comparison of hardware, software, and hybrid techniques. *SIGPLAN Not.*, 51(4):545–559, March 2016.

[60] Mengshi Zhang, Yuqun Zhang, Lingming Zhang, Cong Liu, and Sarfraz Khurshid. Deeproad: Gan-based metamorphic autonomous driving system testing. *arXiv preprint arXiv:1802.02295*, 2018.