2024 International Conference on Assured Autonomy (ICAA) | 979-8-3315-2101-1/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICAA64256.2024.00018

GPU Partitioning & Neural Architecture Sizing for Safety-Driven Sensing in Autonomous Systems

Shengjie Xu*, Clara Hobbs*, Yukai Song†, Bineet Ghosh‡, Tingan Zhu*, Sharmin Aktar*, Lei Yang§, Yi Sheng§, Weiwen Jiang§, Jingtong Hu†, Parasara Sridhar Duggirala*, Samarjit Chakraborty*
*The University of North Carolina at Chapel Hill †University of Pittsburgh
‡The University of Alabama §George Mason University

***Abstract*—Neural networks are now routinely used for perception processing in autonomous systems. Often, these neural networks are used to estimate the *state* of the system, such as distance and velocity of the car in front, that is used in downstream control tasks. While significant advances in neural architecture search and sizing have been made towards improving inference quality, they do not take into account the effect of these improvements in the performance of the *overall system*. In this paper, we examine a setup where multiple neural networks for estimating various state components of the same system share the same graphics processing unit (GPU) — a limited computational resource. We address the problem of optimal resource allocation for each neural network, e.g., how to suitably size these networks, while improving the overall performance — specifically, safety — of the system. In particular, we distinguish between optimizing the performance of individual neural networks, versus optimizing the system-level performance or safety. Our main technical contribution is a set of techniques for neural architecture sizing with the goal of optimizing overall system safety for a given GPU capacity. Our evaluation on two different benchmarks shows that we can explore the architecture space with 10x to 100x improvements in running time.**

***Index Terms*—GPU partitioning, optimal neural architecture sizing, autonomous system, uncertainty, reachability, sensitivity**

I. Introduction

The progress in algorithms, hardware, and software frameworks for deep learning has led to the widespread deployment of neural networks (NNs) in various machine learning tasks, especially in the context of autonomous systems. For example, in an autonomous car, the data from various cameras, lidar, radar, and ultrasonic sensors are partially or fully processed using multiple NNs. Due to its mission criticality, autonomous driving systems usually require high accuracy from NNs. While advances in deep learning have steadily improved the state-of-the-art in NN accuracy, a large portion of this is achieved through an even greater increase in training data size and neural network parameters, significantly increasing computational demands. While hardware performance has constantly improved and various techniques have been developed to compress NNs to realize reasonable accuracy with less computational resources (*e.g.* [1]–[3]), the model sizes and accuracy requirements keep outgrowing the hardware performance improvement. Another challenge is that compute platforms in autonomous systems are usually constrained by space, energy, and cost. Therefore, designers have to work with limited computational resources and must allocate these resources among multiple different NNs with different functionalities and accuracy requirements.

While research efforts like [4] demonstrate the feasibility of partitioning GPU resources to meet specific accuracy requirements for individual neural networks (NNs), they often overlook the need for optimizing NNs as *parts of a larger system*. In the broader context of autonomous systems, which consist of sensing, decision, and actuation stages, the overall system performance depends on accurate system state estimation by NNs in the sensing stage and the sensitivity of the control system's performance (including stability and safety) to state estimation errors (NN accuracy) is not uniform.

Therefore, optimizing each neural network in isolation, or even considering their average accuracy, might not lead to the most effective overall system performance. To illustrate this, consider an agent equipped with p sensors, each processed by a different NN offering different trade-offs between accuracy and resource requirements. Given that state estimation serves as the foundation for feedback control, an inaccurate estimate could lead to a deviation from the desired path, potentially violating the performance and safety requirements of the system. Addressing this requires more than just optimizing individual NN accuracy. State components with high sensitivity might require greater accuracy in state estimation to avoid significant behavioral alterations in the system. Achieving a globally optimal solution requires allocating resources judiciously, prioritizing crucial components while ensuring adequate resources for others.

Yet, manually tuning performance by evaluating every potential resource allocation is often impractical, given the complex dynamics of autonomous systems and the increasing number of NNs. This complexity calls for a systematic approach to resource allocation that can dynamically adapt to the evolving needs of the system.

Contributions of this paper: In this paper, we present a design automation framework for resource allocation of neural networks in autonomous systems. Our primary observation is that resource allocation for a neural network should depend on its contribution to the overall system performance. Given the dynamics of an autonomous system, a list of neural networks that can be used for state estimation, and a resource constraint, our framework efficiently allocates the resources for state estimation that optimize the overall system's control performance. This is achieved by selecting neural networks that minimize the deviation in the closed loop behavior of the system from the ideal behavior, while taking into account the uncertainty in state estimation by the neural networks. To the best of our knowledge, this is the first work to explicitly relate the control performance of autonomous systems with the resource allocation of neural networks. We also associate the problem of NN sizing to system safety, where the safety of the system is connected to the size of the reachable state space of the closed-loop system. With smaller NNs, the uncertainty of their inference is higher, which translates to a larger reachable state space. The main contributions of this work include:

- a constant time sensitivity analysis heuristic that, for a given resource constraint, produces one solution that is reasonably close to the Pareto front of optimal solutions,
- a dynamic programming heuristic that captures virtually all solutions on the Pareto front and in the average case uses much less time than the exhaustive search, and
- a fast iterative heuristic that captures solutions extremely close to the Pareto front and runs in polynomial time

All our methods scale well even when the exhaustive search becomes infeasible. These three heuristics offer various optimality and speed tradeoffs and are suitable for different scenarios. Our evaluation shows that they achieve near-optimal results while improving the search time by $10\times$ to $100\times$ when compared to the baseline.

II. Related Work

Given that embedded systems are cost- and energy-sensitive, considerable efforts have been made to lower the memory (*e.g.* by using smaller networks) and computational requirements of neural networks running on them (*e.g.* by employing early exit strategies). For example, EfficientNet [1] represents a class of neural networks designed for scalability and efficiency. Its unique approach to scaling network dimensions allows for improved accuracy and lower parameter count, adaptable to different platform constraints. This makes EfficientNet ideal for various applications, where a model is selected and trained according to the specific needs of the target platform.

In contrast, [3], [5] focused on scaling down pre-trained models efficiently to retain reasonable performance compared to their explicitly trained counterparts while eliminating the need for retraining, reducing the cost of adapting similar networks to a diverse set of platforms. While these works focused on achieving high accuracy while reducing resource usage, they did not study the impact of the accuracy-resource tradeoffs in the context of control performance.

Our work is also inspired by recent trends in GPU partitioning, with [4], [6]–[9] being some recent examples in this area. The work in [4] holds particular promise, as it enables the partitioning of GPUs at the fine granularity of individual stream processors. This corresponds well with the optimal neural architecture sizing problem presented in this work, where a finite budget of resources (*e.g.*, GPU) is distributed among different neural network tasks, with the goal of optimizing system-level performance/safety. Finally, this work is related to cyber-physical systems safety [10], security [11], [12] and reliability [13] verification [14], and in particular to the testing [15] and verification of NN-enabled systems.

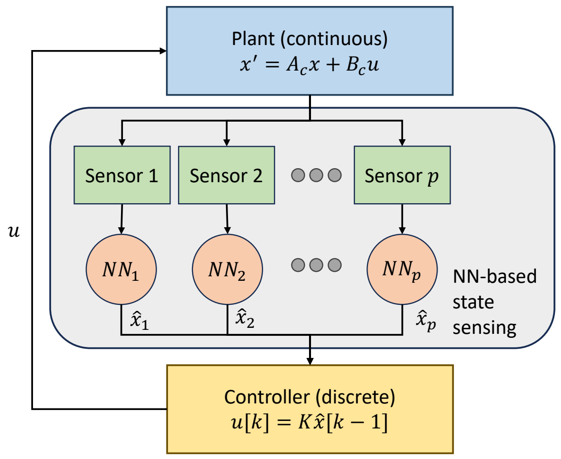


Fig. 1. Control system with neural network-based state sensing.

III. Preliminaries

A. Control System Models

This section introduces the basics of feedback control systems and the specific models studied in this paper. An overview of the system under study is outlined in Figure 1. Control systems are dynamic in nature and are often described using differential equations. One common representation of control systems is the state-space model, where the *state* of the system is represented by a state vector $x(t) \in \mathbb{R}^p$, where elements in the state vector x_i are state components, and the *input* to the system by $u(t) \in \mathbb{R}^q$. Using these notations, the state-space model of a continuous, linear time-invariant (LTI) system is given by

$$\dot{x}(t) = A_c x(t) + B_c u(t), \tag{1}$$

where $A_c \in \mathbb{R}^{p\times p}$, and $B_c \in \mathbb{R}^{p\times q}$. Equation (1) shows that the rate of change of the system state $\dot{x}(t)$ depends both on the current state $x(t)$ and the control input $u(t)$. While many real-world control systems are non-linear, they are often modeled as linearized systems around their fixed operation points and controlled with linear controllers in practice.

To enable feedback control, the control input $u[k]$ is computed by a periodic real-time task running on a processor. Computing $u[k]$ requires discretizing the continuous state-space model with a constant sampling period h. Assuming periodic sampling, *i.e.*, $t_{k+1} - t_k = h$, matrices A and B can be derived from Equation (1) such that:

$$x(t_{k+1}) = Ax(t_k) + Bu(t_k),$$

where $A = e^{A_c h}$ and $B = \int_0^h e^{A_c s} B_c ds$ For simplicity, we denote $x(t_k)$ as $x[k]$ to obtain the discrete state-space model:

$$x[k+1] = Ax[k] + Bu[k]\ . \tag{2}$$

Using the discrete model, the control input $u[k]$ can be computed by:

$$u[k] = Kx[k-1] \;, \tag{3}$$

where $K \in \mathbb{R}^{q\times p}$ is the *feedback gain*. We follow the logical execution time (LET) paradigm, where the real-time task computing control inputs has a deadline equal to its sampling period. That is, the system state sampled at time step $k-1$ is used to compute the control input u for time step k.

B. Reachable Sets under Uncertainty

Given system dynamics and an initial *set* of states $X[0] \subseteq \mathbb{R}^p$, it is possible to compute the reachable sets of the system over a finite time horizon H from Equations (2) and (3). These reachable sets represent the possible states of the system at various time instances given that it was initially in $X[0]$.

However, Equations (2) and (3) model the system evolution only when the system state is known exactly. In this paper, since the state estimation is performed by NNs, we assign an uncertainty with each state component. More specifically, we assume that the estimate of every state component $\hat{x}_i[k] \in [x_i[k]-e_i, x_i[k]+e_i]$ where $e \in (\mathbb{R}^+)^p$ is a bounded vector. This uncertainty can be expressed as set operations over Zonotopes: $x \oplus E$, where $\oplus$ is the Minkowski sum operator, and E is the uncertainty bounding Zonotope. To model the evolution of the system in presence of uncertainties, we replace the closed loop control inputs in Equations (2) and (3) with set operations as follows:

$$\begin{aligned} x[k+1] &\in Ax[k] \oplus BK\hat{x}[k] \\ &= Ax[k] \oplus BK(x[k] \oplus E) \\ &= (A+BK)x[k] \oplus BKE. \end{aligned} \tag{4}$$

The reachable sets for closed loop system with uncertainty in state estimation can be computed using Equation (4).

Given the state-space model $x[k+1] = Ax[k] + Bu[k]$, feedback gain K, uncertainty bounding Zonotope E, initial set $X[0]$, and time horizon H, we compute the reachable set and denote the maximum diameter of reachable set across time as $\mathrm{diam}(A, B, K, E, X[0], H)$. This diameter is a measure of the overall system performance in presence of uncertainties; larger diameter of reachable set implies that estimation uncertainties can result in larger deviation of trajectory and smaller diameter implies that the system is robust to estimation uncertainties.

C. Neural Network Accuracy-Cost Tradeoff

Empirical studies have consistently shown that increasing the size of neural networks tends to enhance their accuracy in various machine-learning tasks, but this improvement comes with a price. For example, consider the relationship between the size and classification accuracy of 8 EfficientNet [1] provided in Figure 2. EfficientNet_B0 is a small baseline network that has been optimized through neural architecture search. Building upon it, EfficientNet B1-B7 are scaled up uniformly while keeping a similar neural architecture. A diminishing returns of accuracy as a function of network size can be clearly

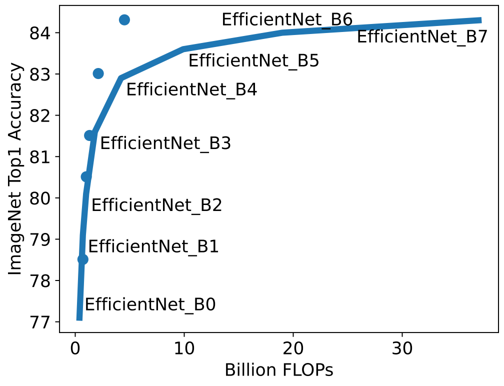


Fig. 2. Diminishing returns of neural network accuracy vs. size from different EfficientNet [1] configurations

seen from the figure. This highlights a number of challenges in neural network design:

1) **Cost Implications of Large Neural Networks:** Large neural networks are resource-intensive, requiring significant computational power [16], memory [17], [18], and energy [19] for training, which can be prohibitive in mobile or embedded systems.
2) **Diminishing Returns on Accuracy:** Although larger networks generally yield higher accuracy, the improvement rate diminishes as the network size increases [20], [21], indicating inefficiencies in resource utilization without proportional performance gains.
3) **Challenges in Network Scaling:** Scaling neural networks involves increasing the depth (adding more layers), the width (adding more channels per layer), or the input resolution (increasing the number of pixels in input images) [22], [23]. It remains unclear how to scale up neural networks to achieve the best efficiency and accuracy [1].
4) **Neural Architecture Search (NAS):** Traditionally, neural network architectures are manually designed, a process that is both time-consuming and requires extensive expertise. This manual approach is becoming increasingly untenable with the rise of diverse hardware platforms each requiring optimized solutions. Neural Architecture Search (NAS) offers a promising solution by automating the design of neural networks using techniques like reinforcement learning [24]–[26], evolutionary algorithms [27], or gradient-based methods [28]–[30] to find architectures that balance computational efficiency and accuracy to find architectures that balance computational efficiency and accuracy.

This work focuses on the deployment of multiple neural networks in a resource-constrained setting, making the tradeoff between accuracy and efficiency the central consideration. Since the NNs are used in estimating the state components, we use accuracy as a proxy for the uncertainty associated with the estimation. To quantify the trade-offs between network size and performance under practical constraints, we define:

- ϵ as the uncertainty of a neural network's estimation,
- c as the cost associated with the neural network.

IV. Neural Architecture Sizing Methods

This section introduces the optimal neural architecture sizing problem, the assumptions made in this study, and the methods proposed to approach the problem.

We make the following assumptions: 1) We are given a p-dimensional linear system with feedback control, described by the state-space model $x[k+1] = Ax[k] + Bu[k]$, and a linear controller K, described by $u[k] = K\hat{x}[k]$. 2) $\hat{x}[k]$ is an estimation of $x[k]$, and each component $\hat{x}_i$ is estimated using a separate neural network. 3) A pool of n neural networks with varying cost-uncertainty tradeoffs are available and can be used to estimate *any* of the p system state; multiple instances of the same neural network may be used to estimate different state components.

It is important to note that the assumption (3) is purely for the ease of constructing the problem statement. In practice, it is completely reasonable that different state variables require different neural networks for estimation – *i.e.*, a unique set of n_i neural networks is available for each state component x_i – and our methods would still be applicable.

We formally define the optimal neural architecture sizing problem based on the above assumptions:

Problem 1. *Given a p-dimensional linear system described by matrices A and B, feedback gain K, initial set $X[0]$, and time horizon H, and n neural networks described by their costs $c_1, \cdots, c_n$ and uncertainties $\epsilon_1, \cdots, \epsilon_n$, find an optimal selection of neural networks $J = j_1, \cdots, j_p$, where $j_i \in \{1, \cdots, n\}$, such that the maximum diameter of the reachable sets $\operatorname{diam}(A, B, K, E, X[0], H)$ is minimized and the cumulative cost of all neural networks stays within a budget C. i.e., $\sum_{i=1}^{p} c_{j_i} \leq C$.*

Essentially, the system designer is presented with a collection of neural networks with varying cost and uncertainty tradeoffs, Where the j-th neural network is associated with a cost c_j and an uncertainty ϵ_j. This is similar in concept to the different configurations of EfficientNet [1]. The job of the designer is to assign one neural network for each state component to optimize the overall system performance, while ensuring the total cost of state estimation is below a budget C. This cost can be used to model constraints on computational resources, memory usage, price, or energy consumption.

We now present a naive approach — exhaustive search — for discovering the optimal allocation and three heuristics for solving the optimal neural architecture sizing problem.

A. Exhaustive Search: The Naive Approach

One way to find the optimal allocation of neural networks is by enumerating every possible allocation of neural networks that satisfies the budget constraints. The maximum diameters of the reachable sets for each allocation will then be calculated. Formally, an allocation of neural networks $J = j_1, \cdots, j_p \in \{1, \cdots, n\}^p$ has the associated uncertainty values of $\epsilon_{j_1}, \cdots, \epsilon_{j_p}$. The uncertainty bounding zonotope E is then defined as

$$E = \left\{ x \in \mathbb{R}^p : x = \mathbf{0} + \sum_{i=1}^{p} \xi_i g_i,\ \xi_i \in [-1, 1]\ \forall i \right\}, \quad (5)$$

where $g_i = (0, \cdots, \epsilon_{j_i}, \cdots, 0) \in \mathbb{R}^p$ is a vector with all zero values except for the i-th element, which is ϵ_{j_i}. The reachable sets $X[k]$ for the time horizon $k = 1, \cdots, H$ can be then calculated by iterating Equation (4), starting from the initial set $X[0]$. Once the reachable sets are calculated, the diameter of reachable sets can be calculated by $\operatorname{diam}(X[k]) = \max_{x,y \in X[k]} d(x, y)$ for any metric – such as the Euclidean distance – between two points x and y. The maximum diameter of the reachable sets is therefore defined as

$$\operatorname{diam}(A, B, K, E, X[0], H) = \max_{k \in [0,H]} \operatorname{diam}(X[k]). \quad (6)$$

Note that the only variable in Equation (6), the error bound zonotope E, is calculated from the selection of neural networks J. For convenience, we define the maximum diameter for a specific selection J as

$$\operatorname{diam}(J) = \operatorname{diam}(A, B, K, E(J), X[0], H) \quad (7)$$

where $E(J)$ follows Equation (5), and the cost for a specific selection J as $\operatorname{cost}(J) = \sum_{i=1}^{p} c_{j_i}$. Finally, the allocation with the smallest diameter $\operatorname{diam}(J)$ and a cost $\operatorname{cost}(J) < C$ is determined to be optimal. This approach is intractable as the possible number of allocations is $O(n^p)$, exponential in the number of states p. Whenever possible, we use this naive search as the baseline for comparing other algorithms.

B. Sensitivity Analysis-Based Heuristic

Our primary insight for accelerating the search for optimal assignment is to quantify the effect of neural network uncertainty on the overall system performance. More specifically, we introduce the notion of sensitivity of a given state component x_i as the multiplicative effect of the uncertainty in x_i on the system trajectory. Simply put, state components with a greater sensitivity value would perturb the trajectory to a greater degree than state components with lesser sensitivity. Given the sensitivity values for each state component, the heuristic then considers the overall budget C and the uncertainty bound-to-cost (ϵ_i, c_i) tradeoffs of candidate neural networks to 1) "guess" an optimal selection of neural networks that satisfies the budget constraint, and 2) perform a *single* reachability analysis to calculate the diameters of the reachable sets using this combination.

1) Computing the Sensitivity of State Components: An existing work [31] have explored a way to compute sensitivity of *system dynamics* using perturbation theory. More precisely, given a system dynamics matrix A, and an index (i, j), [31] computes the effect of introducing an ϵ uncertainty at $A[i, j]$ on the system trajectory and in turn the system's safety [32]. As illustrated in Figure 3, the reachable set as a result of uncertainties introduced at different indices, namely U_1 and U_2

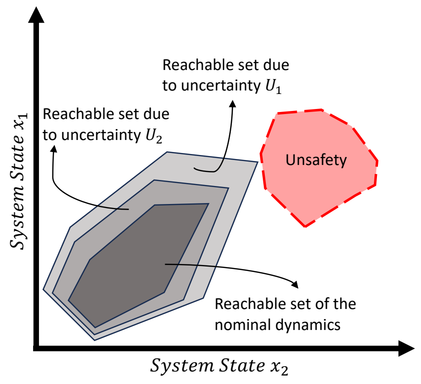

Reachable set due to uncertainty U_1
Reachable set due to uncertainty U_2
Unsafety
Reachable set of the nominal dynamics
System State x_1
System State x_2

Fig. 3. Impact of uncertainties on a system's reachable set.

can differ significantly. Clearly, the uncertainty U_1 is resulting in an increased volume of reachable set when compared to U_2. Therefore, we say that the system exhibits a higher sensitivity to U_1 than to U_2. In the case where the reachable set resulted from U_2 is not strictly contained in the one from U_1, we consider the system more sensitive to U_2 if its reachable set has a larger diameter as defined in Equation (6). Furthermore, the sensitivity value (due to a given uncertainty) is a quantitative measure of the stretch in the reachable set compared to the reachable set of the nominal dynamics. It is worth noting that the method presented in [31] can compute such sensitivity values without computing reachable sets, thus making the technique computationally very efficient and suitable for our purpose.

At its core, [31] highlights a link between the reachable set and the maximum singular value (SV) of the dynamics. The one step reachable set for the system $x[t+1] = Ax[t]$ is a linear transformation on a given initial set Θ, i.e., $A \cdot \Theta$. Given a singular value decomposition (SVD) of $A = M\Sigma N^*$, where M and N^* are unitary matrices and Σ is a diagonal matrix with the singular values, we examine the geometric interpretation of the linear transformation $(A \cdot \Theta)$. The transformation $A \cdot \Theta$ can be interpreted as an ordered sequence of three distinct linear transformations, with N^* being the initial rotation, followed by Σ's scaling, followed by a rotation M. Because both M and N^* are unitary matrices, these transformations do not lead to any "stretching" (increase in relative distances among the elements) of the set in any direction. However, the set undergoes stretching in the second step as a result of the transformation by Σ, with the most significant stretching occurring due to the maximum singular value.

It is worth noting that the maximum singular value of the dynamics matrix A is responsible for causing the maximum stretching of the reachable set. Using this observation, any set of perturbations to A (*i.e.*, uncertainties) that causes the most increase in the maximum singular value of A, will cause the most stretching in the reachable set. For instance, in our previous example, when A is faced with a set of uncertainties U_1 than U_2, the increase in the maximum singular value due to U_1 is more than that of U_2. Therefore, U_1 causes more stretching in the reachable set than U_2, as illustrated in Figure 3. Given a perturbation of ϵ at an index $[i, j]$, the algorithm in [31] computes the multiplicative factor q such that the maximum singular value increases by $q \cdot \epsilon$. This multiplicative factor is called the sensitivity of the index $[i, j]$.

We adapt the method proposed in [31] to work with uncertainties in *state component estimation* x—instead of the dynamics A—by using augmented system dynamics. Particularly, we leverage the fact that $u[t] = K\hat{x}[t] = K\lambda x[t]$, where

$$\lambda = \begin{bmatrix} 1+\delta_1 & 0 & \cdots & 0 \\ 0 & 1+\delta_2 & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \cdots & 0 & 1+\delta_p \end{bmatrix}$$

represents the uncertainty in estimating $x[t]$. The augmented dynamics can then be described by

$$\begin{aligned} x[t+1] &= Ax[t] + Bu[t] \\ &= Ax[t] + BK\lambda x[t] \\ &= (A + BK\lambda)x[t]. \end{aligned}$$

We call $\Phi' = (A + BK\lambda)$ as the perturbed dynamics, analogous to introducing perturbations in the system dynamics A in [31]. To compute the sensitivity value of a particular state component x_i, we set $\delta_j = 0$ for all $j \neq i$, and observe that with $\Phi = A + BK$ and $Y_i = (BK)_{*,i}$, the perturbed dynamics become $\Phi' = \Phi + \delta_i Y_i$. This is nothing but introducing perturbations in the dynamics A at specific indices determined by BK. We then use the method in [31] to compute the sensitivity value S_i for x_i.

2) Heuristic Leveraging Sensitivity Values: Given the sensitivity values S_i for all states components, our heuristic uses integer programming to select neural networks for each state. The integer programming optimization problem can be formulated as follows:

$$\min_{j_1,\cdots,j_p} \quad \sum_{i=1}^{p} \epsilon_{j_i} \cdot S_i \tag{8a}$$

$$\text{subject to} \quad \sum_{i=1}^{p} c_{j_i} \leq C \tag{8b}$$

$$j_i \in \{1, \cdots, n\} \; \forall i. \tag{8c}$$

Intuitively, we try to minimize the sum of products of the sensitivity value and uncertainty bound for each state component, while being subject to the system budget C. Notice that the optimization problem attempts to allocate more resources to the highly sensitive state components while satisfying the cost constraints. It is possible to formulate several such optimization problems with quadratic functions in the minimization, however, they might require more resources for solving the optimization problem and might not scale well with increase in number of neural networks or dimensionality of the system. Once we obtain such a selection $J = j_1, \cdots, j_p$, we compute the reachable set for the neural network allocation and compute its maximum diameter.

One of the primary advantages of using this heuristic over others is its scalability. The sensitivity values of the states can be computed very efficiently in polynomial time. Furthermore, the sensitivity values have to be computed only once.

Algorithm 1: Dynamic programming heuristic

```
Data: $A$, $B$, $K$, $X_0$, $H$, $c_1, \cdots, c_n$, $\epsilon_1, \cdots, \epsilon_n$
Result: List of $\langle J, \mathrm{diam}(J), \mathrm{cost}(J)\rangle$
$J \leftarrow$ vector of $p$ ones;
Add $J$ to current solutions;
Add $J$ to tried solutions;
repeat
    foreach $J \in$ current solutions do
        Add $\langle J, \mathrm{diam}(J), \mathrm{cost}(J)\rangle$ to results;
        foreach $i \in 1 \ldots p$ do
            $J' \leftarrow J$ incremented in position $i$;
            if $J' \in$ tried solutions then
                continue;
            Add $J$ to next solutions;
            Add $J$ to tried solutions;
    foreach $J \in$ next solutions do
        Prune $J$ if dominated by other solutions (Equation (9))
    current solutions $\leftarrow$ next solutions;
    next solutions $\leftarrow \emptyset$;
until current solutions $= \emptyset$;
return results
```

C. Dynamic Programming-Based Heuristic

Our next heuristic for assigning resources for neural network state sensing is a dynamic programming-based heuristic. It is important to note that, despite the use of dynamic programming, this method does not guarantee optimality. This is because the behavior of control systems cannot be simply predicted by neural network uncertainty alone.

Assuming that the neural networks j_i are sorted in increasing order of cost, we first assign the lowest-cost network to each state variable, *i.e.*, $J \leftarrow 1, 1, \cdots, 1$, and run a reachability algorithm to determine the resulting maximum diameter E. Following this, we take an iterative approach to explore neural network options for each state. Taking the solutions from the previous iteration, we increase the resources allocated to each state variable. For example, the allocation $[2, 3, 3]$ is followed by $[3, 3, 3]$, $[2, 4, 3]$, and $[2, 3, 4]$. The maximum diameter of the reachable sets is computed for each of these allocations. We then prune all solutions that are dominated and add the new solutions to our suboptimal set. A solution J_1 is considered dominated by another solution J_2 if J_1 both has a higher cost and results in a larger diameter. I.e., J_1 is dominated by J_2 iff

$$\mathrm{cost}(J_1) > \mathrm{cost}(J_2) \wedge \mathrm{diam}(J_1) > \mathrm{diam}(J_2) \qquad (9)$$

We call this dynamic programming heuristic because we explore the solution space by systematically increasing the resources allocated for estimating each state component and aggressively prune the solution space. The iteration terminates once the highest cost allocation ($J = n, n, \cdots, n$) has been explored, or when all solutions are pruned in one iteration. The pseudocode for this heuristic is listed in Algorithm 1.

However, it does so at the expense of exploring a non-negligible number of non-optimal assignments, making it still relatively slow, especially for high-dimensional systems. To alleviate this, we propose a faster, albeit farther from optimal, heuristic in the next section.

Algorithm 2: Fast iterative heuristic

```
Data: $A$, $B$, $K$, $X_0$, $H$, $c_1, \cdots, c_n$, $\epsilon_1, \cdots, \epsilon_n$, order
Result: List of $\langle J, \mathrm{diam}(J), \mathrm{cost}(J)\rangle$
$J \leftarrow$ vector of $p$ ones;
repeat
    Add $\langle J, \mathrm{diam}(J), \mathrm{cost}(J)\rangle$ to results;
    Increment next element of $J$ following order;
until $J >$ vector of $p$ $n$'s;
return results
```

D. Greedy Fast-Iterative Heuristic

Exploring the solution space in a systematic way as done in the dynamic programming heuristic might still explore a large part of solution space. We therefore propose a faster heuristic that explores even fewer solutions and give a quick estimate of the cost and overall performance tradeoff. Instead of increasing the resources allocated for state estimation in a systematic way, this heuristic only explores the solution space that are estimated to be optimal (hence the name, greedy heuristic). This heuristic works by simply increasing the resources for estimating a state component one at a time in a pre-determined order. For instance, if the order is $[1, 3, 2]$ (i.e., state component x_1 is given first priority, then x_3 and then x_2), then we will try the assignments $[1, 1, 1]$, $[2, 1, 1]$, $[2, 1, 2]$, $[2, 2, 2]$, and so on, until the most expensive option is being used for all networks. In this way, the resources allocated for various state components remains almost uniform, but a part of the solution space can be explored very quickly. This tradeoff turns out to be near-optimal in our experiments. The pseudocode for this heuristic is listed in Algorithm 2. While any predetermined order can be used for the heuristic, we make an informed choice and give priority to state components with higher sensitivity as described in Section IV-B. Additionally, because the assignments' cost is monotonically increasing, the heuristic may be easily modified to return only the closest assignment to a maximum budget, skipping reachability analysis for all other assignments. The timing complexity of the greedy fast-iterative search is therefore $O(np)$.

V. Evaluation of the Proposed Methods

We evaluated the three classes of heuristics with two case studies: 1) A case study where the neural network uncertainty and costs are derived from EfficientNet [1], and 2) an evaluation using the Dist-YOLO [33] object detection and distance estimation model with different backbones.

For both case studies, we use a numerical five-dimensional state-space model [34] from the JuliaReach [35] toolbox. We use Julia to implement and evaluate the proposed heuristics. The heuristics are evaluated against the exhaustive search method on an Apple M2 processor at 3.5 GHz in single-threaded mode.

A. Case Study using EfficientNet Accuracy and Cost Values

As a proof of concept, in [36] we proposed to use cost and uncertainty values from EfficientNet [1] for our first case study. We used the uncertainty-cost tradeoff values from EfficientNet [1] because it is an architecture designed to perform

TABLE I
PARAMETERS OF THE EFFICIENTNET CONFIGURATIONS

Model	FLOPs	Accuracy
EfficientNet_B0	0.39G	77.1%
EfficientNet_B1	0.70G	79.1%
EfficientNet_B2	1.0G	80.1%
EfficientNet_B3	1.8G	81.6%
EfficientNet_B4	4.2G	82.9%
EfficientNet_B5	9.9G	83.6%
EfficientNet_B6	19G	84.0%
EfficientNet_B7	37G	84.3%

well across a wide range of configurations and network sizes. Although the networks are evaluated on image classification tasks instead of regression tasks that are more relevant to control systems, we expect well-trained regression models to exhibit similar cost and uncertainty tradeoffs. Furthermore, EfficientNet is already being used as the backbone for a number of regression tasks [33], [37].

The work in [1] presented 8 configurations, EfficientNet B0 to B7, and provided the ImageNet Top-1 accuracy, ImageNet Top-5 accuracy, and FLOPS (billions) for each configuration. These parameters are shown in Table I. We used $(\text{Top-5 accuracy})^{-1} - 1$ as an uncertainty measure, and floating point operations (FLOPs) as a cost measure.

We conducted two experiments using the first 5 and 8 configurations from EfficientNet, respectively. For each experiment, we performed an exhaustive search over the entire solution space, followed by the three proposed heuristics. The number of points explored and the execution time are recorded for both experiments, in Table II. The solutions from the first experiment using 5 EfficientNet configurations are shown in Figure 4. To illustrate a potential safety violation as a result of uncertainty in estimation, we marked solutions as "unsafe" if the third dimension of the system crosses the unsafe region of $x_3 \leq -25$. These are shown as "$\times$" in Figure 4. Additionally, Figure 5 shows the unsafe region along with reachable sets of states, projected on to state dimensions 3 and 4. Green regions indicate reachable sets with no estimation uncertainty. The reachable sets achieved by using the most accurate NNs for all states are shown in yellow, while the reachable sets with least accurate NNs for all states are shown with blue.

While the latter two heuristics search the whole space, the sensitivity analysis heuristic only explores a single solution for any given budget. To provide a more comprehensive evaluation of its performance compared to the other two heuristics, we ran the sensitivity analysis heuristic with 40 budget values, evenly placed between the lowest and highest possible budgets.

Discussions: In this section, we compare the results from the three heuristics and provide discussions on the distinct traits displayed by each of them. The first observation is that out of the 50 points on the Pareto front, the dynamic programming heuristic explored was able to capture nearly all of them (49) with its 268 points (18%); the fast iterative heuristic captured 7 points on the Pareto front with its 21 points (33%), but all points are very close to the Pareto front

TABLE II
NUMBER OF POINTS AND RUNTIME COMPARISON OF THE ALLOCATION HEURISTICS

	5 Configurations		8 Configurations	
Method	Points	Time	Points	Time
Exh. Search	3125	411 s	32768	4685 s
Sens. Anal. (total)	40	5.8 s	40	6.5 s
Sens. Anal. (avg.)	1	0.14 s	1	0.14 s
Dyn. Prog.	268	41.4 s	414	64.5 s
Fast Iter.	21	3.5 s	36	6.0 s

TABLE III
NUMBER OF POINTS COMPARISON FOR VARIOUS VALUES OF p (# OF STATES) AND n (# OF NNS)

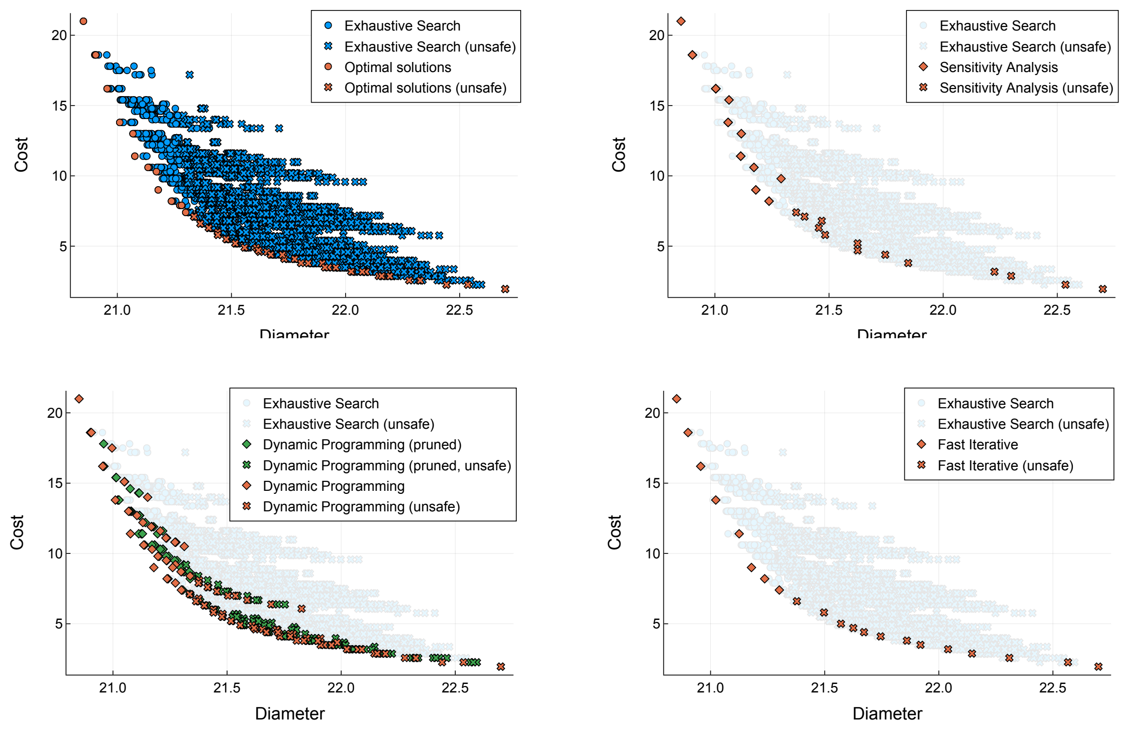
Method	p=5, n=5	p=5, n=20	p=20, n=5	p=10, n=8
Exh. Search	3125	3.2×10^6	9.5×10^{13}	1.1×10^9
Dyn. Prog. (est.)	268	1200	1013	888
Fast Iter.	21	96	81	71

as seen in Figure 4d; the sensitivity analysis heuristic captured 11 Pareto-optimal points with its 40 points (28%), but are on average worse than the fast iterative method, as shown in Figure 4b.

The second observation is that different methods require vastly different times to obtain these results. As shown in Table II, the dynamic programming and fast iterative heuristics achieved $10\times$ and $118\times$ speedups in the first experiment, and $72\times$ and $780\times$ in the second experiment. These speedups directly correlate to the number of points explored by each method.

To verify the scalability of our methods beyond the case study, we calculated the number of points that would be required with various values of p (number of states) and n (number of neural network configurations). The results are shown in Table III. Notably, for a practical setup of $p = 10, n = 8$, the exhaustive search is already exploring 1.1×10^9 solutions. If each solution takes $1\,\text{s}$ to compute, the total execution time will exceed 34 years. In contrast, the dynamic programming heuristic is estimated to take less than $15\,\text{min}$, the fast iterative heuristic less than one-and-half minutes, and the sensitivity analysis heuristic takes just one second (if a budget is given). Note that since the number of points explored by the dynamic programming heuristic depends on the concrete values of diameters, it cannot be precisely calculated without actually running the heuristic. Instead, we took the average ratio between the number of points explored by the dynamic programming heuristic and the number of points explored by the fast iterative heuristic in Table II (12.5) and multiplied it with the values of fast iterative in Table III. It is clear that the exhaustive search is not feasible for larger values of p and n, while the proposed heuristics scale much better. The sensitivity analysis heuristic is not included in Table III since it only requires exploring a single point for any given budget, regardless of the number of states or neural networks.

Depending on the use case, we see that the three methods might be desirable in different scenarios. The dynamic programming heuristic and fast iterative heuristic are more



(c) Dynamic programming heuristic

(d) Fast iterative heuristic

Fig. 4. Results of the exhaustive search and the three allocation heuristics, using cost and uncertainty values derived from Table I.

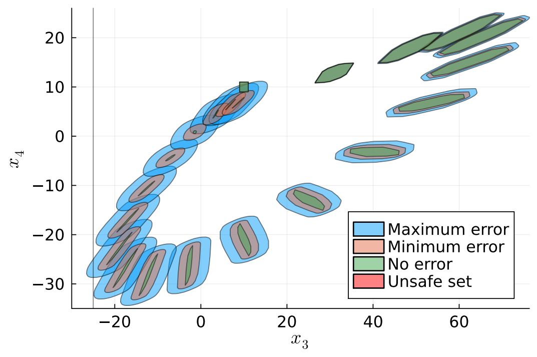


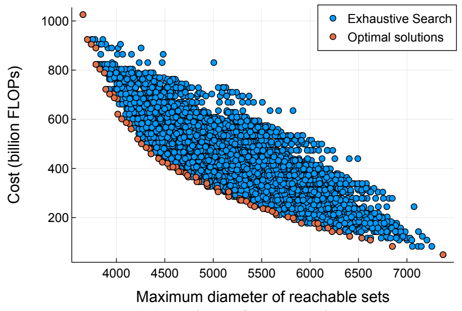
Fig. 5. Reachable sets of the system states for the first 20 steps with perfect state sensing, min., and max. uncertainty among all NNs considered, projected into dimensions 3 and 4, up to time 20.

suitable when a concrete budget has not been set, and a search over the whole space might provide valuable insight into the cost-performance tradeoff of the system. In this scenario, the two methods differ in the granularity of the results: the dynamic programming approach finds more optimal points but is slower, and the fast iterative approach leaves some gaps but is more efficient. The sensitivity analysis approach is more suitable when a budget has already been set, or when the design space is too huge for using other methods.

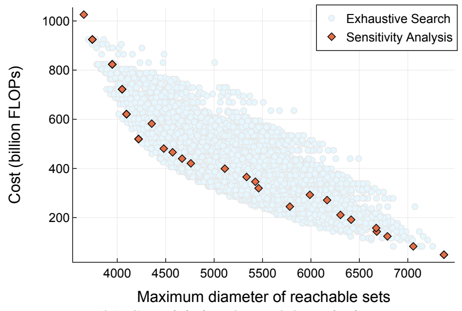
B. Case Study with Dist-YOLO

In our second case study, we aim to verify the results obtained from EfficientNet uncertainty and cost values by evaluating our method with realistic neural networks. Towards this, we trained the Dist-YOLO [33] object detection and distance estimation model on the KITTI dataset [38], namely the KITTI 3D Object Detection Evaluation 2017 dataset. It contains 7481 images with ground truth distance labels, of which, 6699 images are used for training and 782 images for testing. We used six different backbones configurations such as MobileNet [2], EfficientNet [1], and Xception [39]. For each configuration, we trained and evaluated Dist-YOLO model on the KITTI dataset, reporting the absolute relative error (ARE) in distance estimation and floating point operations (FLOPs) required by each configuration. They are outlined in Table IV and in Figure 7. We use the absolute relative error (ARE) as the uncertainty values ϵ and FLOPs as the cost values c.

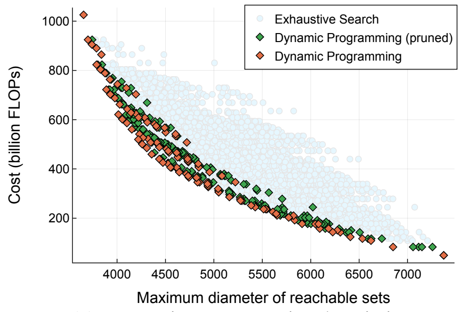
Figure 6 highlights the exhaustive search and heuristic results using cost and uncertainty values from the Dist-YOLO models. Given the relatively high error in estimation, the maximum diameters in this case study are higher, potentially indicating that the system reaches a divergent state. This has also made it less meaningful to distinguish between a safe and unsafe maximum diameter. Nonetheless, we observe from the exhaustive search results in Figure 6a that the general shape of the maximum diameter vs. cost tradeoff surfaces remains similar to the previous case study. We evaluate the heuristics against the exhaustive search and report the results in Figures 6b to 6d. Out of the three heuristics, the dynamic programming-based heuristic still performs very well, capturing nearly all points on the Pareto front. In contrast,



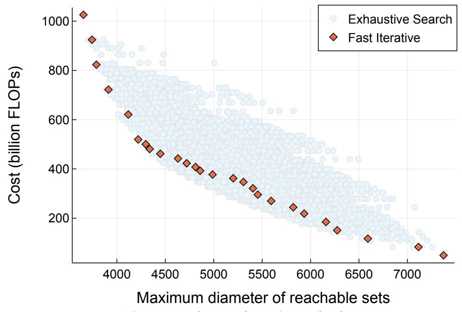
(a) Exhaustive search



(b) Sensitivity-based heuristics



(c) Dynamic programming heuristic



(d) Fast iterative heuristic

Fig. 6. Results of the exhaustive search and the three allocation heuristics, using cost and uncertainty values from Dist-YOLO [33] in Table IV.

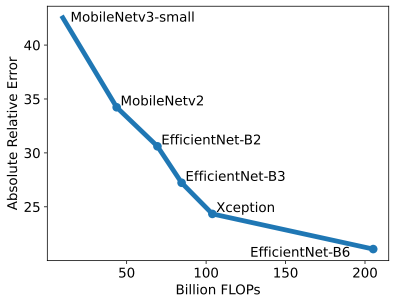


Fig. 7. Absolute relative error vs. size from different Dist-YOLO [33] configurations utilizing various backbones.

TABLE IV
DIST-YOLO FLOPS VS. ABSOLUTE RELATIVE ERRORS (TRUCK AND CAR) WITH DIFFERENT SIZED BACKBONES

Backbone	FLOPs	Abs. Rel. Error
MobileNetv3_small	9.833G	42.53%
MobileNetv2	43.714G	34.23%
EfficientNet_B2	69.371G	30.61%
EfficientNet_B3	84.574G	27.23%
Xception	103.935G	24.34%
EfficientNet_B6	205.171G	21.08%

the greedy fast iterative heuristic produces solutions that are close but not quite on the Pareto front. We hypothesize that this is due to the uncertainty vs. cost tradeoff in our trained Dist-YOLO setup, as shown in Figure 7, is not as optimized as the EfficientNet configurations, whereas the greedy fast iterative heuristic assumes diminishing returns in increased neural network sizes. Finally, the sensitivity analysis-based heuristic also performed worse than in the previous case study.

VI. CONCLUDING REMARKS

In this work, we associated the control performance of autonomous systems to the optimal neural architecture sizing problem. Our key observation is that optimizing the accuracy of neural networks in isolation does not necessarily lead to better control performance. Instead, it is important to consider the dynamics of the system, and resource allocation for neural networks should reflect their contribution to the overall system performance. We presented three heuristics with varying optimality and speed tradeoffs and evaluated them on a numerical case study against solutions obtained from exhaustive searches. We found that they were able to identify close-to-optimal solutions while drastically reducing the design space exploration time.

This work also points to the possibility of co-designing (a) neural architectures for perception processing, and (b) controllers that use the output of such neural networks. There is a large volume of literature on the co-design [40], [41] of controllers and communication schedules for a variety of protocols [42]–[44] that impact the sensor-to-actuator delays experienced by the controllers [45]. Scheduling control tasks [46] and communication protocols together with neural architectures and controllers will be an interesting extension to the existing control/architecture co-design literature.

While our methods effectively minimize the diameters of reachable sets, there is an avenue for future exploration in considering alternative metrics for evaluating the performance or safety of autonomous systems, such as liveness properties. In addition, our focus has been on the accuracy and cost tradeoffs of neural networks in an offline context. Yet, there are many other properties of neural networks that are relevant in autonomous system design. For instance, neural network early exit policy, an online strategy that trades accuracy and inference latency, remains unexplored in our framework. Extending our framework to encompass these richer settings will be a fruitful future direction. Finally, this work belongs to the less explored area of context-dependent neural network design, where the goal is no longer to design networks with the best inference, but rather the best system-level performance.

Acknowledgments: This work was partially supported by the NSF grant CPS-2038960.

REFERENCES

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

[2] A. G. Howard *et al.*, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," Apr. 2017. arXiv:1704.04861.

[3] T.-J. Yang *et al.*, "NetAdapt: Platform-Aware Neural Network Adaptation for Mobile Applications," 2018. CoRR: arXiv:1804.03230v2.

[4] J. Bakita and J. H. Anderson, "Hardware Compute Partitioning on NVIDIA GPUs," in *29th RTAS*, 2023.

[5] H. Cai *et al.*, "Once-for-All: Train One Network and Specialize it for Efficient Deployment," Apr. 2020. CoRR, arXiv:1908.09791.

[6] "NVIDIA H100 Tensor Core GPU Architecture Overview." https://resources.nvidia.com/en-us-tensor-core/gtc22-whitepaper-hopper.

[7] S. Jain, I. Baek, S. Wang, and R. Rajkumar, "Fractional GPUs: Software-Based Compute and Memory Bandwidth Reservation for GPUs," in *IEEE RTAS*, 2019.

[8] B. Wu *et al.*, "Enabling and Exploiting Flexible Task Assignment on GPU through SM-Centric Program Transformations," in *29th ACM on International Conference on Supercomputing*.

[9] T. Yandrofski, J. Chen, N. Otterness, J. H. Anderson, and F. D. Smith, "Making Powerful Enemies on NVIDIA GPUs," in *IEEE RTSS*, 2022.

[10] J. Oetjens *et al.*, "Safety evaluation of automotive electronics using virtual prototypes: State of the art and research challenges," in *The 51st Annual Design Automation Conference (DAC)*, 2014.

[11] P. Mundhenk *et al.*, "Security in automotive networks: Lightweight authentication and authorization," *ACM Trans. Design Autom. Electr. Syst.*, vol. 22, no. 2, pp. 25:1–25:27, 2017.

[12] P. Mundhenk *et al.*, "Lightweight authentication for secure automotive networks," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2015.

[13] G. Georgakos *et al.*, "Reliability challenges for electric vehicles: from devices to architecture and systems software," in *50th Annual Design Automation Conference (DAC)*, 2013.

[14] P. Kumar *et al.*, "A hybrid approach to cyber-physical systems verification," in *49th Annual Design Automation Conference (DAC)*, 2012.

[15] G. Tibba *et al.*, "Testing automotive embedded systems under X-in-the-loop setups," in *35th International Conference on Computer-Aided Design (ICCAD)*, 2016.

[16] S. Han *et al.*, "Eie: Efficient inference engine on compressed deep neural network," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 243–254, 2016.

[17] S. Shivapakash, H. Jain, O. Hellwich, and F. Gerfers, "A power efficiency enhancements of a multi-bit accelerator for memory prohibitive deep neural networks," *IEEE Open Journal of Circuits and Systems*, vol. 2, pp. 156–169, 2021.

[18] W. Meng, Z. Gu, M. Zhang, and Z. Wu, "Two-bit networks for deep learning on resource-constrained embedded devices," *arXiv preprint arXiv:1701.00485*, 2017.

[19] R. V. W. Putra and M. Shafique, "Fspinn: An optimization framework for memory-efficient and energy-efficient spiking neural networks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 11, pp. 3601–3613, 2020.

[20] S. Krishnan, Y. Xiao, and R. A. Saurous, "Neumann optimizer: A practical optimization algorithm for deep neural networks," *arXiv preprint arXiv:1712.03298*, 2017.

[21] Y. N. Dauphin and Y. Bengio, "Big neural networks waste capacity," *arXiv preprint arXiv:1301.3583*, 2013.

[22] I. Bello *et al.*, "Revisiting resnets: Improved training and scaling strategies," *Advances in Neural Information Processing Systems*, vol. 34, pp. 22614–22627, 2021.

[23] T. Yang *et al.*, "A closer look at network resolution for efficient network design," *CoRR*, vol. abs/1909.12978, 2019.

[24] Y. Jaafra, J. L. Laurent, A. Deruyver, and M. S. Naceur, "Reinforcement learning for neural architecture search: A review," *Image and Vision Computing*, vol. 89, pp. 57–66, 2019.

[25] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *arXiv preprint arXiv:1611.01578*, 2016.

[26] B. Lyu, S. Wen, K. Shi, and T. Huang, "Multiobjective reinforcement learning-based neural architecture search for efficient portrait parsing," *IEEE Transactions on Cybernetics*, vol. 53, no. 2, pp. 1158–1169, 2021.

[27] Y. Liu, Y. Sun, B. Xue, M. Zhang, G. G. Yen, and K. C. Tan, "A survey on evolutionary neural architecture search," *IEEE transactions on neural networks and learning systems*, vol. 34, no. 2, pp. 550–570, 2021.

[28] H. Cai, L. Zhu, and S. Han, "Proxylessnas: Direct neural architecture search on target task and hardware," *arXiv preprint arXiv:1812.00332*, 2018.

[29] X. Chen, L. Xie, J. Wu, and Q. Tian, "Progressive differentiable architecture search: Bridging the depth gap between search and evaluation," in *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 1294–1303, 2019.

[30] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," *arXiv preprint arXiv:1806.09055*, 2018.

[31] B. Ghosh and P. S. Duggirala, "Robustness of Safety for Linear Dynamical Systems: Symbolic and Numerical Approaches." 10.48550/arXiv.2109.07632, 2021.

[32] B. Ghosh and P. S. Duggirala, "Robust reachable set: Accounting for uncertainties in linear dynamical systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 18, no. 5s, p. 97, 2019.

[33] M. Vajgl *et al.*, "Dist-YOLO: Fast Object Detection with Distance Estimation," *Applied Sciences*, vol. 12, no. 3, 2022.

[34] "Five dim system - ReachabilityModels." https://juliareach.github.io/ReachabilityModels.jl/dev/models/five_dim_sys/.

[35] S. Bogomolov *et al.*, "JuliaReach: a toolbox for set-based reachability," in *22nd ACM International Conference on Hybrid Systems: Computation and Control*, 2019.

[36] S. Xu *et al.*, "Poster abstract: Neural architecture sizing for autonomous systems," in *15th ACM/IEEE Intenational Conference on Cyber-Physical Systems*, 2024.

[37] S. F. Bhat, I. Alhashim, and P. Wonka, "Adabins: Depth estimation using adaptive bins," *CoRR*, vol. abs/2011.14141, 2020.

[38] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[39] F. Chollet, "Xception: Deep Learning with Depthwise Separable Convolutions," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[40] R. Schneider *et al.*, "Constraint-driven synthesis and tool-support for Flexray-based automotive control systems," in *9th CODES+ISSS)*, 2011.

[41] D. Roy *et al.*, "Semantics-preserving cosynthesis of cyber-physical systems," *Proc. IEEE*, vol. 106, no. 1, pp. 171–200, 2018.

[42] L. Zhang *et al.*, "Task- and network-level schedule co-synthesis of Ethernet-based time-triggered systems," in *19th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2014.

[43] D. Goswami, R. Schneider, and S. Chakraborty, "Re-engineering cyber-physical control applications for hybrid communication protocols," in *Design, Automation and Test in Europe (DATE)*, 2011.

[44] D. Roy *et al.*, "Multi-objective co-optimization of flexray-based distributed control systems," in *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2016.

[45] D. Goswami, R. Schneider, and S. Chakraborty, "Co-design of cyber-physical systems via controllers with flexible delay constraints," in *16th Asia South Pacific Design Automation Conference (ASP-DAC)*, 2011.

[46] S. Chakraborty and L. Thiele, "A new task model for streaming applications and its schedulability analysis," in *Design, Automation and Test in Europe Conference and Exposition (DATE)*, 2005.