TimelyNet: Adaptive Neural Architecture for Autonomous Driving with Dynamic Deadline

JIALE CHEN, Nanyang Technological University, Singapore
DUC VAN LE*, Nanyang Technological University, Singapore
YUANCHUN LI, Tsinghua University, China
YUNXIN LIU, Tsinghua University, China
RUI TAN, Nanyang Technological University, Singapore

To maintain driving safety, the execution of neural network-based autonomous driving pipelines must meet the dynamic deadlines in response to the changing environment and vehicle's velocity. To this end, this paper proposes a real-time neural architecture adaptation approach, called TimelyNet, which uses a *supernet* to replace the most compute-intensive neural network module in an existing end-to-end autonomous driving pipeline. From the supernet, TimelyNet samples *subnets* with varying inference latency levels to meet the dynamic deadlines during run-time driving without fine-tuning. Specifically, TimelyNet employs a one-shot prediction method that jointly uses a lookup table and an invertible neural network to periodically determine the optimal hyperparameters of a subnet to meet its execution deadline while achieving the highest possible accuracy. The lookup table stores multiple subnet architectures with different latencies, while the invertible neural network models the distribution of the optimal subnet architecture given the latency. Extensive evaluation based on hardware-in-the-loop CARLA simulations shows that TimelyNet-integrated driving pipelines achieve the best driving safety, characterized by the lowest wrong-lane driving rate and zero collisions, compared with several baselines, including the state-of-the-art driving pipelines.

CCS Concepts: • **Computer systems organization → Embedded and cyber-physical systems**; **Real-time systems**.

Additional Key Words and Phrases: Autonomous driving, supernet, dynamic deadline, invertible neural network, neural architecture adaptation

ACM Reference Format:
Jiale Chen, Duc Van Le, Yuanchun Li, Yunxin Liu, and Rui Tan. 2025. TimelyNet: Adaptive Neural Architecture for Autonomous Driving with Dynamic Deadline. *ACM Trans. Embedd. Comput. Syst.* xx, xx, Article 67 (September 2025), 23 pages. https://doi.org/XXXXXXX.XXXXXXX

* Duc Van Le is now with University of New South Wales, Australia.

This research/project is supported by the National Research Foundation, Singapore under its AI Singapore Programme (AISG Award No: AISG4-GC-2023-006-1B).
This article appears as part of the ESWEEK-TECS special issue and was presented in the International Conference on Embedded Software (EMSOFT), 2025.

Authors' Contact Information: Jiale Chen, Nanyang Technological University, Singapore, jiale004@e.ntu.edu.sg; Duc Van Le, Nanyang Technological University, Singapore, duc.le1@unsw.edu.au; Yuanchun Li, Tsinghua University, China, liyuanchun@air.tsinghua.edu.cn; Yunxin Liu, Tsinghua University, China, liuyunxin@air.tsinghua.edu.cn; Rui Tan, Nanyang Technological University, Singapore, tanrui@ntu.edu.sg.

1 Introduction

Deep neural networks (DNNs) are widely used for perception, prediction, and planning in autonomous driving [13, 14, 20, 23]. To ensure driving safety, DNN model inference should be completed before a certain deadline in each driving control period [17]. Ideally, the required deadline can vary with the vehicle's driving speed and acceleration across control periods [28]. For instance, when the vehicle runs at a higher speed, more frequent control actions should be made to adapt to the fast-changing environment. As a result, a shorter latency of the DNN inference for deciding the control action is needed. Current autonomous driving systems tend to conservatively select lightweight DNN models that can always meet dynamic deadlines under all possible driving conditions. However, the lightweight models may lead to low inference accuracy. Thus, identifying and using the best DNN model under each possible driving condition is desirable for maintaining the driving performance and safety of autonomous vehicles [8, 24, 31]. This paper aims to design an approach that can be integrated into existing autonomous driving pipelines to perform online generation of DNN model variants with the highest accuracy levels and the corresponding latencies matching the dynamic driving conditions.

Existing DNN architecture adaptation approaches can be divided into the following two categories. The first category maintains a pool of various model candidates, among which a suitable model is selected at runtime. For example, the study in [8] stores multiple DNN-based object detection models with distinct accuracy and latency levels and switches among them at runtime based on the specified deadline. However, this approach is not memory-efficient and thus not suitable for the memory-constrained compute hardware platforms found on autonomous vehicles. The second category aims to search for a suitable subnet from an architecture-changeable DNN [3, 11, 25]. This approach is more memory-efficient because it does not need to store many DNN models. However, the search process is time-consuming. For example, in [11], evaluating 200 architecture candidates with a server-class hardware accelerator takes about one minute. Although trained latency and accuracy predictors can speed up evaluation [3], the search process remains slow when confronting the millisecond-level latency requirements of autonomous driving systems, as analyzed for the experiments conducted in this paper. If the search is performed for each inference, the remaining time available for running the selected model is greatly restricted or even diminishes.

This paper presents TimelyNet, a real-time neural architecture adaptation approach that can be integrated into existing autonomous driving pipelines to meet dynamic latency requirements. The main idea is to use a dynamic DNN model called *supernet* [10] to replace the most time-consuming module in existing autonomous driving pipelines. Specifically, the supernet is an over-parameterized DNN, from which we can sample a *subnet* consisting of a subset of the supernet's parameters to perform the same function, with possibly reduced accuracy and latency. This approach has two major advantages. First, it has the ability to generate numerous distinct subnets with diverse accuracy and latency levels which are correlated with the subnet architecture. Thus, the accuracy and latency resolutions achieved by the cohort of all possible subnets can be high. Second, TimelyNet is memory-efficient compared with the model switching approach, as we only need to store the supernet. To ensure that the supernet can generate subnets with diverse accuracy and latency levels, we train the supernet and the autonomous driving pipeline jointly in the offline training stage, so that the subnet can be sampled from the supernet at runtime without fine-tuning. To use the supernet in real-time applications like autonomous driving, it is essential to develop an efficient method to determine the subnet architecture. The success of TimelyNet depends on the ability to quickly identify the subnet that achieves the highest accuracy while meeting the dynamic latency requirement for each inference. However, this is challenging due to the vast search space.

To ensure efficient search over the vast search space, we employ a one-shot learning-based optimal architecture prediction approach instead of using iterative heuristic search methods. Specifically, we construct a lookup table that stores the optimal subnet architectures for a number of latencies with a regular interval and train a generative machine learning model to predict the optimal subnet architecture given an arbitrary latency requirement. The lookup table is constructed by profiling a set of subsets randomly sampled from the supernet, where the profiling includes measuring the latency and control quality of a sampled subnet. However, due to the limited sample size with respect to the large search space, predictions from the lookup table may be inaccurate. To address this, we jointly use the lookup table and an invertible neural network (INN), a generative model that learns the relationship between latency and subnet architecture. The INN can learn the distribution of subnet architecture given the latency through two tasks: (1) predicting latency from the subnet architecture and (2) generating the subnet architecture based on the latency requirement. At runtime, the subnet with fewer parameters, among the two generated by the INN and the lookup table, is selected to increase the likelihood of meeting the latency requirement. Due to the simple structure of the INN and the lookup table, the search process can be completed within milliseconds.

In this paper, we implement TimelyNet on top of an existing end-to-end autonomous driving pipeline, which is Trajectory-guided Control Prediction (TCP) [26]. To evaluate the TimelyNet-integrated TCP pipeline, we conduct hardware-in-the-loop experiments in the CARLA simulator. Specifically, we evaluate the driving performance and safety of TimelyNet by controlling the vehicle to follow the predefined route in the CARLA simulator. Results show that TimelyNet follows the predefined trajectory with the lowest mean absolute error (MAE) under all driving speeds among all the considered methods. Moreover, it only introduces a small execution time overhead of 2.5 ms in searching for the optimal subnet in each control period, which is short compared with the autonomous driving pipeline's inference latency of over 41.1 ms. More importantly, TimelyNet can achieve the lowest wrong-lane driving rate of 4% and zero collisions during the entire simulation, delivering the highest level of safety among all evaluated methods. To demonstrate the generalizability of TimelyNet, we also integrate it with Interfuser [19], which is a multi-modal autonomous driving pipeline, and run the system on Jetson AGX Orin embedded GPU platform.

Our main contributions can be summarized as follows:

- We design TimelyNet, a real-time neural architecture adaptation framework that jointly uses a latency lookup table and an INN to efficiently determine the optimal subnet architectures. TimelyNet generates subnets that meet the dynamic latency requirements within 2.5 ms, which is short compared with the autonomous driving pipeline's inference latency of over 41.1 ms.
- TimelyNet is integrated into both uni-modal and multi-modal end-to-end autonomous driving pipelines. By replacing their image encoders with a supernet and jointly training the supernet and the autonomous driving pipelines, TimelyNet enables the pipelines to adapt to dynamic latency requirements at runtime without fine-tuning.
- We conduct hardware-in-the-loop experiments in CARLA to evaluate the performance of TimelyNet. The evaluation results demonstrate that TimelyNet can improve the driving performance and safety of the TCP and Interfuser pipelines, achieving the lowest trajectory MAE and zero collisions.

The rest of the paper is organized as follows. In §2, we review the related work on autonomous driving pipelines and the online DNN architecture adaptation approach. In §3, we present the preliminaries of this work. In §4, we present the motivation and problem statement of TimelyNet. In §5, we present the design of TimelyNet. In §6, we evaluate the driving performance of the TCP and Intefuser pipelines equipped with TimelyNet. In §7, we discuss the limitations of TimelyNet and future work. In §8, we conclude the paper.

2 Related Work

2.1 Autonomous Driving Pipelines

Conventional autonomous driving pipelines can be divided into two categories: *modular* and *end-to-end*. A modular pipeline consists of multiple models, each of which is individually developed as a stand-alone model for perception, planning, and control [9, 22]. However, modular pipelines are susceptible to error propagation and often require significant overhead for training and managing their models [5]. To address these drawbacks, end-to-end autonomous driving pipelines have emerged to combine perception, planning, and control into a single model that can be jointly trained. The end-to-end pipelines directly take raw sensor data as inputs to generate driving control outputs [6, 15, 19, 26]. State-of-the-art end-to-end autonomous driving pipelines utilize various neural networks (NNs) as encoders to extract features and generate inputs for subsequent modules to predict waypoints or control commands. For example, TCP [26] employs a CNN-based encoder to extract features from a single RGB image. The extracted features are then fed into two modules to predict the waypoints and control actions. Moreover, Interfuser [19] utilizes two encoders based on ResNet to extract features from RGB images and LiDAR point cloud data. These features are then fused using a transformer decoder to accurately predict future waypoints. Differently, UniAD [15] utilizes a single transformer-based encoder to extract the features from multi-view images. However, the neural networks in both modular and end-to-end autonomous driving pipelines are designed for fixed latency and lack adaptability to dynamic environments in real-time autonomous driving systems. To address this limitation, we propose TimelyNet, an approach that can be incorporated into existing autonomous driving pipelines to produce subnets with varying latency and accuracy levels at runtime, to respond to changing driving conditions. In this paper, we focus on the integration of TimelyNet into end-to-end autonomous driving pipelines.

2.2 DNN Architecture Adaptation

Existing studies of DNN architecture adaptation mainly focus on selecting the optimal model from a set of pre-trained models at runtime or searching for the optimal model architecture from an architecture-changeable DNN. For example, in [8], the authors prepare multiple object detection models with different accuracy and latency levels and switch among them at runtime based on the latency requirements. However, this approach can only switch among a limited number of models due to the high memory usage required to store multiple models. Differently, the authors in [3, 11, 25] train a dynamic DNN with a changeable architecture and formulate the search for a suitable architecture instance as an optimization problem. In [25], a heuristic on-device search algorithm is proposed to search for the optimal model architecture given the latency budget. It requires at least 117.6 seconds to find the optimal architecture because of the large overhead of evaluating the model performance during the search process. Moreover, the authors in [3] use latency and accuracy predictors to accelerate the search process by avoiding the evaluation of the model performance. However, the search time is still more than 200 ms. Current autonomous driving pipelines require a latency of less than 100 ms to ensure real-time performance [16], rendering this approach unsuitable for per-inference adaptation in the autonomous driving context. To address this issue, TimelyNet employs a one-shot optimal architecture prediction approach that jointly uses a lookup table and an INN to generate the subnet hyperparameters, instead of solving the optimization problem using heuristic algorithms. As a result, it can adapt in real time to a wide range of dynamic latency requirements in autonomous driving systems.

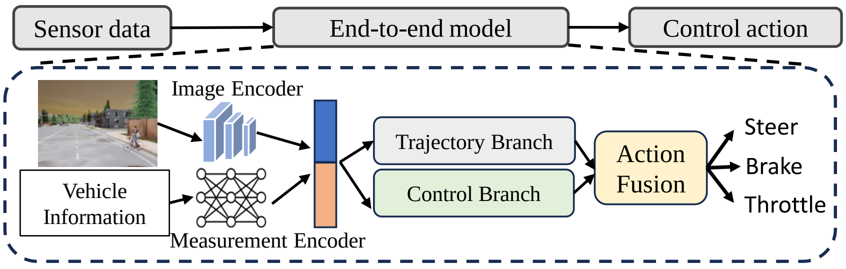


Fig. 1. End-to-end TCP Autonomous Driving Pipeline.

3 Preliminaries

In this section, we present the background about the end-to-end TCP autonomous driving pipeline, supernet, and INN model.

3.1 End-to-end TCP Autonomous Driving Pipeline

In this paper, we focus on the end-to-end driving pipelines available in closed-loop simulations, which can provide real-time dynamic driving information required by TimelyNet. We present the design and implementation of TimelyNet for the TCP pipeline [26], which ranks the third in the CARLA leaderboard 1.0 [4]. It achieves a 1.0% lower driving score but has fewer collisions compared with the second-place pipeline, named Interfuser [19]. The first-place pipeline is not open-source. The main objective of TCP is to safely and efficiently control an autonomous vehicle to reach a target destination by following a predefined path provided by the global planner. As shown in Fig. 1, TCP consists of a CNN-based image encoder, which takes a driving scene image as input to generate the image feature map. Meanwhile, a multilayer perceptron (MLP)-based measurement encoder takes the current movement speed and navigation information as inputs to produce the measurement feature. Then, the image and measurement features are concatenated and fed into the trajectory and control branches for generating a driving control action, denoted by $a = (t, b, s)$, where $t \in [0, 1]$, $b \in [0, 1]$ and $s \in [-1, 1]$ are the control signals for throttle, brake, and steering, respectively. The detailed design of the TCP pipeline can be found in [26]. In this paper, TimelyNet is applied to adapt the run-time neural architecture of the CNN-based image encoder, because it is the most compute-intensive component in the TCP pipeline. Such an adaptation ability allows the inference of the autonomous driving pipelines to meet the dynamic deadlines while maximizing the control quality.

3.2 Supernet

Supernet [3] is an over-parameterized and dynamic NN from which subnets with distinct structures can be sampled to provide a wide range of inference latency and accuracy levels. The sampled subnets can complete the same task as the supernet, but with different inference latency and accuracy. This offers great flexibility to meet dynamic latency and accuracy requirements in real autonomous driving scenarios. Compared with other dynamic models, a key advantage of the supernet is its high memory efficiency. Specifically, a supernet can generate numerous subnets with a small memory footprint because the subnets can inherit weight parameters from the supernet. For example, in [3], the supernet can generate more than 10^{19} subnets with latency levels ranging from 150ms to 350ms, while requiring only 7.7M parameters to store all of them. In TimelyNet, we utilize a supernet to replace the CNN-based image encoder, which is the most compute-intensive NN in the TCP pipeline. At runtime, subnets are sampled from the supernet to provide suitable

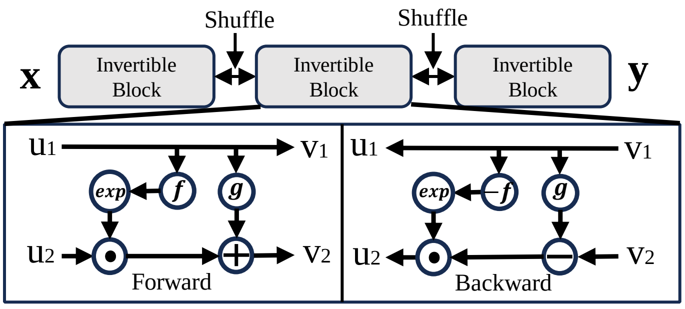


Fig. 2. Illustration of an INN structure.

image encoders, ensuring that the execution of the end-to-end TCP pipeline can meet the dynamic latency requirements.

3.3 Invertible Neural Network

INN [2, 27] is a type of flow-based generative model that can be used to map its data input, denoted by x, to a latent variable following a known prior distribution, denoted by y. The output y can then be used to generate samples of x. Fig. 2 presents a typical architecture of the INN model that consists of multiple invertible blocks, each of which includes the forward and backward processes. Specifically, the forward process can be expressed as:

$$v_1 = u_1, \qquad v_2 = u_2 \odot \exp(f(u_1)) + g(u_1), \tag{1}$$

where u_1 and u_2 are the first and second halves of the input x, respectively; v_1 and v_2 are the first and second halves of the output y, respectively; f and g are the scale and translation functions which are often implemented by NNs, respectively; and exp denotes the exponential function. Meanwhile, the backward process can be expressed as:

$$u_1 = v_1, \qquad u_2 = (v_2 - g(v_1)) \odot \exp(-f(v_1)). \tag{2}$$

The functions f and g are shared by both the forward and backward processes, which is the key to enabling the invertibility of the INN. The output y of an invertible block is shuffled and split into two parts, which are used as u_1 and u_2 inputs of the next blocks. The training process of INN aims to train the forward process to maximize the likelihood of the data in the dataset, which can be calculated using the value of the latent variable y and the known prior distribution. After training, an instance of y can be sampled from its prior distribution and used as inputs of the backward process to generate a corresponding instance of x. One advantage of using INN in TimelyNet is that it can model the distribution of the subnet hyperparameters given the latency requirements, which can be extended to generate subnet hyperparameters given the latency requirements that are not in the training set.

4 Motivation and Problem Statement

In this section, we present the motivation for the dynamic desired latency and then formulate an optimization problem with the objective of finding a suitable NN architecture given the desired latency. Then, we discuss the challenges in solving the formulated problem in real time and overview our proposed approach.

4.1 Motivation

The control process of end-to-end autonomous driving is divided into multiple control periods. The duration of the control period i is denoted by L_i. The beginning time instant of a control period i is called a *time step*, denoted by t_i. Starting from t_i, the end-to-end autonomous driving pipeline is executed within the control period i to generate a new control action to be applied for the next control period. The execution is supposed to be finished before the deadline of $t_i + L_i$. In other words, the model execution time should be less than L_i. Our motivational experiments presented below show the intricate interplays among the setting of L_i, vehicle speed, model size, and the driving performance. The results suggest that adapting the setting of L_i according to vehicle speed and the corresponding adaptation of the model offer new opportunities for improving driving performance.

The experiments are conducted based on the TCP pipeline, which use ResNet [12] as the image encoder to extract features from the input image. We choose ResNet-34 and ResNet-152 as the image encoders for the TCP pipeline, denoted by TCP-Res34 and TCP-Res152, to achieve different trade-offs between accuracy and latency. TCP-Res152 has more convolutional layers and can provide more accurate control actions than TCP-Res34, but it has a longer inference latency. We use trajectory mean absolute error (MAE) as a metric to measure the driving performance, which is the average distance between the vehicle's actual position and the nearest position in the reference path. The average is taken over all control periods. Lower trajectory MAE indicates better driving performance, as the vehicle follows the predefined path more precisely. Note that, in the experiments, the vehicle accelerates from 0 m/s to a specified maximum speed $v_{\max}$, and then maintains at the maximum speed for the rest of the journey.

In the first experiment, we measure the trajectory MAE of TCP-Res152 under different settings of the maximum speed $v_{\max}$ and different settings of the control period L_i. In order to set different settings for L_i, we conduct the experiments in the synchronous mode, where the duration of each control period can be set manually, even when the same model is used in all control periods. With this capability, we can investigate the sole impact of the control period setting on the driving performance. The duration L_i is selected from three options of 50 ms, 100 ms, and 150 ms. The speed $v_{\max}$ is chosen from three options of 5 m/s, 10 m/s, and 15 m/s. Fig. 3 (a) shows the average and standard deviation of the trajectory MAE values over 10 runs under each setting of L_i and $v_{\max}$. The trajectory MAE generally increases with the L_i and $v_{\max}$. When a small L_i setting is adopted, the model is executed more frequently and each execution needs to be faster, representing higher computational overhead. From the results shown in Fig. 3, to balance the computational overhead and the path tracking performance, a good strategy is to adopt a larger setting for L_i when the speed is low and a smaller setting for L_i when the speed is high.

In the second experiment, we compare the trajectory MAE of TCP-Res34 and TCP-Res152 under different speeds. We use the asynchronous mode to conduct this experiment, where the duration of each control period L_i is determined by the inference latency of TCP-Res34 and TCP-Res152, which are 45.4 ms and 104.1 ms, respectively. The speed $v_{\max}$ is chosen from two options of 5 m/s and 15 m/s. Fig. 3 (b) shows the mean and standard deviation of trajectory MAE of TCP-Res34 and TCP-Res152 over 10 runs. The trajectory MAE of TCP-Res152 is lower and higher than that of TCP-Res34 when the speed setting is 5 m/s and 15 m/s, respectively. The results indicate that faster, but less accurate, models are more suitable for high-speed driving scenarios, while slower, but more accurate, models are more suitable for low-speed driving scenarios. These two experiments motivate us to adapt the model latency and accuracy in response to the vehicle's movement speed to achieve better driving performance. To this end, we propose to incorporate the supernet into

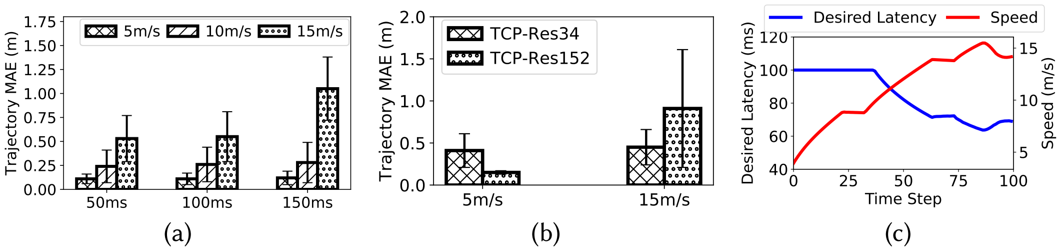


Fig. 3. (a) Trajectory MAE of TCP-Res152 under different control periods and maximum speeds. (b)Trajectory MAE of TCP-Res152 and TCP-Res34 under different maximum speeds. (c) Desired latency vs. movement speed.

existing autonomous driving pipelines to generate subnets with different latency and accuracy levels at runtime.

4.2 Dynamic Desired Latency

Based on the motivational experiment results, we propose to adapt the deadline of the i^{th} control period and the corresponding desired latency L_i based on the autonomous vehicle's movement speed and acceleration. In particular, we adopt a model, in which the vehicle issues a new control action for every distance d_c traveled. As a result, the desired latency L_i is dynamically adjusted based on the real-time movement speeds and accelerations of the autonomous vehicle. Specifically, the desired latency L_i is determined such that the autonomous driving pipeline is executed for every movement distance d_c. Formally, we define $L_i = \min(L_d, L_u)$, where L_d is the autonomous vehicle's movement time to travel a distance of d_c and L_u is a predefined upper limit of L_i. Let v_i and λ_i denote the autonomous vehicle's movement speed and acceleration at the time step t_i, respectively. Therefore, the solution for L_d is given by $L_d = \frac{-v_i+\sqrt{v_i^2+2\lambda_i d_c}}{\lambda_i}$ if $\lambda_i \neq 0$; and $L_d = \dfrac{d_c}{v_i}$ if $\lambda_i = 0$. Fig. 3 (c) shows the trace of L_i over 100 control periods for an acceleration process from 0 m/s to 15 m/s, for the settings of distance $d_c = 1$ m and $L_u = 100$ ms. From Fig. 3 (c), the L_i decreases as the movement speed v_i increases. It shows that shorter execution latency is required when the autonomous vehicle moves at a higher speed, which is consistent with our observation in the second motivational experiment presented in §4.1.

4.3 Problem Formulation

The neural architecture adaptation problem, denoted by Opt, is solved at every time step t_i to adapt the execution latency of the TCP pipeline to the dynamic desired latency L_i. Specifically, Opt aims to find an optimal subnet of the supernet, with the objective of maximizing the autonomous vehicle's movement control quality subject to the latency constraints. Let $S = \{s_1, \ldots, s_{|S|}\}$ denote the set of possible subnets that can be sampled from the original supernet. In particular, the supernet is the largest subnet that can be sampled, denoted by $s_{sup} \in S$. Then, Opt is formulated as follows:

$$\underset{s_k \in S}{\text{Maximize}}\ Q(s_k), \text{ s.t. } L(s_k) \leq L_i, \tag{3}$$

where $Q(s_k)$ and $L(s_k)$ are the control quality and execution latency of the TCP pipeline with the subnet s_k. We assume the movement control action generated by the TCP pipeline with the supernet s_{sup} is the best action that leads to the least discrepancy between the autonomous vehicle's actual positions and reference positions. Let a_{sup} and a_k denote the control actions generated by the TCP pipeline with the supernet s_{sup} and subnet s_k, respectively. Recall that $a = (t, b, s)$ is a tuple of the control actions, including the throttle, brake, and steering. Then, the control quality $Q(s_k)$ is

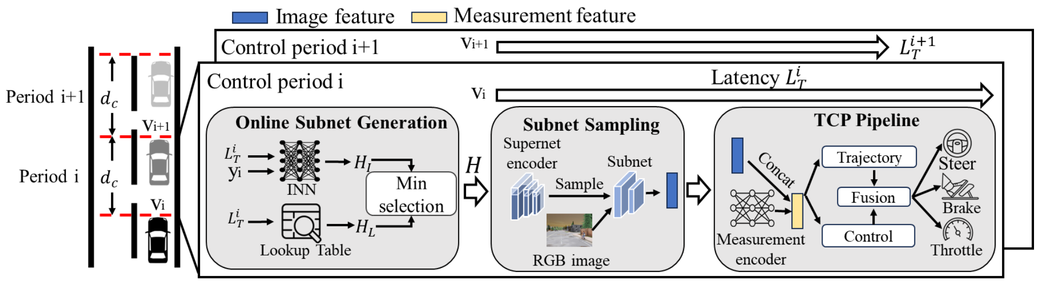


Fig. 4. Design overview of TimelyNet.

calculated as: $Q(s_k) = \sum_{i=0}^{2}(1 - \frac{|a_k[i]-a_{sup}[i]|}{\Delta a[i]})/3$, where Δa is the maximum action difference for each element in a.

4.4 Challenges and Approach

Solving Opt faces the following practical challenge. The search process to solve Opt may use an excessive amount of time, although the control quality $Q(s_k)$ and latency $L(s_k)$ for any candidate subnet s_k can be measured with a validation dataset. For instance, with a supernet that is capable of generating subnets with execution latencies in [41ms, 83ms], the search space is 9×10^{10}. The solving latency of Opt using a genetic algorithm (GA)-based optimization solver is about 199.3 ms. However, in autonomous driving, the latency of the driving pipeline should be less than 100 ms to ensure real-time performance [16]. As a result, the time required to solve Opt is unacceptable.

A potential approach to solve Opt in real time is to build a lookup table consisting of multiple entries, each of which stores an optimal subnet that can maximize the control quality while satisfying a certain desired latency. At runtime, the lookup table is queried to find the optimal subnet corresponding to the desired latency L_i. However, the lookup table has the following limitations. First, a lookup table can only store a limited number of subnets with discrete latency levels. Thus, it may not provide the solution for all possible desired latencies, especially in dynamic driving scenarios. Second, it is impractical to profile all possible subnets to identify the globally optimal subnet for each latency level. For instance, our experiments show that the process of measuring the control quality and latency of a subnet takes about 8 seconds. As such, the profiling process for all 9×10^{10} subnets of a supernet can take about 22,831 years. In this paper, we can build a lookup table based on a dataset consisting of the profiling results for some subnets, which can be obtained in a reasonable amount of time. As a result, the built lookup table may not always provide a subnet to generate the best control action for the dynamic desired latency. To further improve the control quality, we additionally train an INN to learn the relationship between the subnet hyperparameters and the execution latency of the subnets. Then, the learned INN is used to generate a subnet for a certain latency requirement level with better control quality that is not included in the dataset used to build the lookup table. Unlike the lookup table, the INN can provide the subnet solutions for the continuous latency requirement. At runtime, given a specific desired latency L_i, either the lookup table output or the INN output is used as the final subnet solution. TimelyNet chooses the smaller one between the two options, because our evaluation shows that this policy leads to better driving performance.

5 Design of TimelyNet

In this section, we present the design overview of TimelyNet, the training process of the TCP pipeline with the supernet, and the subnet generation process.

5.1 Design Overview

Given an existing end-to-end autonomous driving pipeline, our objective is to replace the most time-consuming module with a supernet module, thereby generating subnets with varying latency. To achieve the neural network architecture adaptation, we collect a dataset of the architecture hyperparameters, latency, and control quality of different subnets sampled from the supernet. Using this dataset, we can construct a lookup table and train an INN to predict the optimal architecture hyperparameters of the subnet based on the dynamic latency requirements. Between the outputs of the INN and the lookup table, TimelyNet chooses the one with fewer parameters to perform inference, so that the used subnet can have a higher chance to meet the desired latency.

In this paper, we implement TimelyNet on top of the end-to-end TCP pipeline [26]. TimelyNet replaces the CNN-based image encoder in the TCP pipeline with a supernet. Then, the TCP pipeline with the supernet is trained using an imitation learning approach. The training process will be presented in §5.2. At runtime, given a desired latency L_i at each time step t_i, TimelyNet runs the following three steps to yield a movement control action for autonomous driving, as illustrated in Fig. 4. First, the lookup table and INN model are executed to generate two sets of hyperparameters of the subnet, denoted by H_L and H_I, respectively. The design of the lookup table and INN model will be detailed in §5.3. Then, between H_L and H_I, the one with fewer weights is selected as the final choice, denoted by H. Second, the subnet with the hyperparameter set H is sampled from the supernet. Finally, the obtained subnet is integrated into the TCP pipeline as an image encoder to extract the features of the autonomous vehicle's current input image. The TCP pipeline with the subnet is executed to generate an action for controlling the throttle, brake, and steering of the autonomous vehicle.

5.2 Offline Training of TCP with Supernet

Table 1 shows the latency of TCP-Res152 and TCP-Res34 models, respectively. The inference latency of the image encoder accounts for 91.7% and 78.6% of the total latency of TCP-Res152 and TCP-Res34, respectively. The results indicate that the image encoder is the most time-consuming module in the TCP pipeline. Thus, we replace the original image encoder with a supernet-based image encoder to generate subnets with varying latency. Fig. 5 presents the structure of the supernet. The supernet includes multiple CNN units, each of which consists of a number of convolutional blocks, denoted by $d_{\text{sup},i}$. Each convolutional block in the unit i has a specific number of input channels, denoted by $w_{\text{sup},i}$. A subnet with hyperparameter H sampled from the supernet is characterized by its elastic depth and width parameters, denoted by $d_{\text{sub},i}$ and $w_{\text{sub},i}$, respectively. In other words, $H = \{d_{\text{sub},i}, w_{\text{sub},i} | i = 1, ..., N_{\text{sup}}\}$, where N_{sup} is the number of CNN units in the supernet. By selecting $d_{\text{sub},i} \in [0, d_{\text{sup},i}]$ and $w_{\text{sub},i} \in [1, w_{\text{sup},i}]$, we can generate multiple subnets with various latency and control quality levels. If the $d_{\text{sub},i} = 0$, the unit i is not included in the subnet.

Given the required execution latency range, a supernet with certain values of N_{sup}, $d_{\text{sup},i}$, and $w_{\text{sup},i}$ is integrated into the TCP pipeline. Then, we adopt an imitation learning-based approach to train the TCP pipeline with the supernet as an end-to-end autonomous driving pipeline. We utilize a driving expert to generate a training dataset and train the TCP pipeline with the supernet using the dataset to emulate the expert's driving behaviors. Specifically, the expert driver uses a deep reinforcement learning (DRL) agent to learn a movement policy through predefined routes from extra information such as the bird-eye-view segmentation images provided by the simulator. In real-world autonomous vehicles, the expert driver could also be a human. Then, the trained expert driver is used to create a ground-truth dataset, denoted by $D = \{s, a^*\}$, where a^* is the movement control action generated by the expert driver at the system state s. The state s encompasses the sensor data, including the captured RGB image, movement speed, navigation command, and target

Table 1. Latency of TCP pipeline with ResNet-152 and ResNet-34 as image encoders.

	TCP-Res152		TCP-Res34	
Latency (ms)	Total	Encoder	Total	Encoder
	104.1	95.4	45.4	35.7

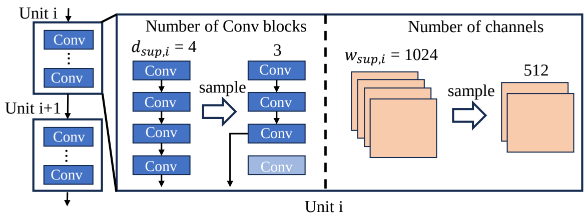


Fig. 5. Structure of supernet-based image encoder.

destination of the autonomous vehicle. Finally, the dataset $D = \{s, a^*\}$ is used to train the TCP pipeline, in which the a^* is considered as the ground-truth action for the state s. The detailed training process of the TCP pipeline can be found in [26].

The above training process drives the TCP pipeline with the supernet to learn to generate optimal control actions under dynamic driving scenarios. Note that, the TCP pipeline mainly uses the subnets of the trained supernet to generate actions at runtime. Thus, it is desirable to retrain the TCP pipeline with the subnet to recover the accuracy loss due to the subnet sampling. However, compute-intensive retraining is infeasible at runtime since it violates the timeliness requirement. To address this issue, we retrain the TCP pipeline with a certain set of subnets in the offline training phase. First, we randomly select the values of $d_{\text{sub},i}$ and $w_{\text{sub},i}$ to create a subnet. Then, we retrain the TCP pipeline with the subnet using the training dataset $D = \{s, a^*\}$. These two steps are repeated for a certain number of randomly selected subnets, denoted by K.

5.3 Subnet Generation

In what follows, we first present our approach to collect a training dataset used to construct the lookup table as well as to train the INN model. Then, we show the *min selection* method that yields either the lookup table result or the INN result as the final subnet, with which the execution of the TCP pipeline can be completed within the latency constraint while maximizing the autonomous vehicle's movement control quality.

5.3.1 Collection of Training Dataset. Let $L_{\min}$ and $L_{\max}$ denote the minimum and maximum latency requirement levels, respectively. The range of $[L_{\min}, L_{\max}]$ is divided into multiple levels, by an interval of ΔL. As discussed in §4.3, it is impractical to profile all possible subnets of the supernet and use the profiling results to find an optimal subnet for each latency level $L_k \in [L_{\min}, L_{\max}]$. Thus, we begin with randomly sampling a certain number of subnets, denoted by $S_{tr} = \{s_1, \ldots, s_{|S_{tr}|}\}$. Each subnet s_i is characterized by a set of hyperparameters, denoted by H_i. We measure the execution latency $L(s_i)$ and control quality $Q(s_i)$ of every subnet $s_i \in S_{tr}$. As a result, we obtain a dataset, denoted by $D_{tr} = \{(H_i, L(s_i), Q(s_i)) | i = 1, \ldots, |S_{tr}|\}$. Then, we check if the dataset D_{tr} includes at least 20 samples with the latency $L(s_i) \in [L_k, L_k + 1ms]$ for all $L_k \in [L_{\min}, L_{\max}]$. Finally, we profile the latency and quality control of additional subnets and include them in the dataset D_{tr} until there are at least 20 samples within any latency interval of 1 ms. Such a process ensures that the dataset D_{tr} fully covers the latency levels in $[L_{\min}, L_{\max}]$.

5.3.2 Construction of Lookup Table. We use the above dataset D_{tr} to construct a lookup table with multiple entries, each of which stores a subnet s_k that can meet the latency requirement L_k. Specifically, s_k is chosen as the subnet with the highest control quality $Q(s_k)$ among the subnets in the D_{tr} whose latency is less than or equal to L_k. As a result, the lookup table is constructed as $T = \{(L_k, H_k) | k = 1, \ldots, |T|\}$, where H_k is the hyperparameter of subnet s_k. Due to the size limit of the dataset D_{tr} and the discrete nature of the lookup table, the H_k may not be the optimal hyperparameter for the desired latency L_i calculated at runtime.

5.3.3 Training of INN. To overcome the limitations of the lookup table, we also train an INN to model the relationship between latency and hyperparameters. We use the dataset D_{tr} to train the INN. We only keep the subnets with control quality $Q(s_i) \geq Q_{min}$, where Q_{min} is a predefined threshold, to ensure that the INN can generate subnets with acceptable control quality. Let f_I denote the forward process of the INN, which can be formulated as: $[L(s_k), y] = f_I(H_k)$, where y is the hidden variable that follows a Gaussian distribution with zero mean and unit variance, denoted by $p(y)$. The value of y can be used to calculate the likelihood of the hyperparameters H_k given the latency $L(s_k)$. Accordingly, the backward process can be formulated as: $H_k = f_I^{-1}([L(s_k), y])$. During the training of the INN, we train the forward process which takes the H_k as input to predict the latency $L(s_k)$ and maximize the likelihood of the hyperparameters H_k. The hyperparameters H_k are encoded as one-hot vectors. The $[L(s_k), y]$ has the same dimension as the one-hot vectors. The loss function consists of two parts as follows.

$$\mathcal{L}_{\text{Pred}} = \|L_{pred} - L(s_k)\|_2, \quad \mathcal{L}_{\text{MLE}} = \underset{\theta}{\text{argmax}} \log p(H_k|L(s_k)), \\ \mathcal{L} = \mathcal{L}_{\text{Pred}} + \mathcal{L}_{\text{MLE}}, \tag{4}$$

where the prediction loss of $\mathcal{L}_{\text{Pred}}$ is used to minimize the difference between the predicted latency of L_{pred} and the measured latency of $L(s_k)$. Meanwhile, the maximum-likelihood estimation loss of $\mathcal{L}_{\text{MLE}}$ is used to learn the mapping between the distribution $p(y)$ of the hidden variable y and the distribution $p(H_k|L(s_k))$. As we know the distribution of $p(y)$, we can calculate the log-likelihood $\log p(H_k|L(s_k))$ according to the change of variables theorem, which can be calculated as $\sum_H \log p(y) + \log\|\det\left(\frac{\partial f_I(H)}{\partial H}\right)\|$, where det is the determinant of the Jacobian matrix $\frac{\partial f_I(H)}{\partial H}$ of the INN model. Using the dataset D_{tr}, we train the INN model to minimize the loss function $\mathcal{L}$.

5.3.4 Subnet Generation with Min Selection. At runtime, the built lookup table and trained INN are used to generate the subnets that can meet the desired latency L_i at each time step t_i. Then, we select from the outputs of the INN and the lookup table to generate the hyperparameters of the subnet. The detailed procedure is as follows. First, we calculate the desired latency L_i of the current control period i given the information of v_i and λ_i. Let Δt denote the time required to query the INN and lookup table. The inference latency of the TCP pipeline is $L_T^i = L_i - \Delta t$. Next, we query the lookup table to find a subnet with the hyperparameter H_L that can meet the latency requirement of L_T^i. In the case that L_T^i does not fall within the latency levels stored in the lookup table, we take a subnet within the lookup table with a latency closest to L_T^i as the subnet solution. Then, we use the concatenation of L_T^i and y_i as inputs of the backward process of the INN to generate an additional subnet with the hyperparameters H_I. Note the y_i is set to zero, i.e., the mean value of the Gaussian distribution $p(y)$. With the two sets of hyperparameters H_L and H_I, we can obtain the total numbers of convolutional blocks and channels of the corresponding subnets. Therefore, we can estimate the numbers of weights of the generated subnets and select the one with fewer weights as the final hyperparameters H of the subnet, such that the selected subnet has a higher probability of meeting the latency requirement.

6 Evaluation

In this section, we present the hardware information, implementation details, and the evaluation of TimelyNet.

6.1 Evaluation Settings

6.1.1 Hardware Platform. We use two computers to implement and evaluate the performance of TimelyNet. They run TimelyNet and CARLA simulator, respectively, and are connected through an Ethernet network to transmit the sensor data and control actions of the autonomous vehicles. The CARLA simulator runs on a computer with an Intel Xeon Silver 2.2 GHz CPU and one NVIDIA Quadro RTX 4000 GPU with 8 GB memory. TimelyNet runs on a computer with an Intel Xeon CPU with 3.3 GHz CPU and one NVIDIA Quadro RTX 8000 GPU with 48 GB memory. The computation capability of the NVIDIA Quadro RTX 8000 GPU is 16.3 tera floating-point operations per second (TFLOPS), which is equivalent to commercial autonomous vehicles computing platforms, such as Tesla FSD with 18.45 TFLOPS [21]. We also implement TimelyNet on Jetson AGX Orin embedded GPU with 64 GB memory and 5.3 TFLOPS computation capability.

6.1.2 Implementation and Settings of TimelyNet. We implement TimelyNet using PyTorch 1.8. We replace the image encoder of the TCP pipeline with a supernet encoder consisting of 5 convolutional units. The $d_{\text{sup},i}$ is set to 2, meaning that we have three choices for the number of convolutional blocks in each unit. The $w_{\text{sup},i}$ of 5 units are set to 64, 256, 512, 1024, and 2048, respectively, which follows the design of ResNet50. We adopt imitation learning to train the TCP pipeline with a supernet by emulating the driving behavior from another driving expert, called Roach [30]. Specifically, Roach is trained with bird's-eye view (BEV) images and meta-information from the CARLA simulator using deep reinforcement learning. We use Roach as the driving expert to collect the dataset using the predefined routes in Town 04 and Town 06 of the CARLA simulator [26]. The total number of routes is 242, which includes 182 and 60 routes in Town 04 and Town 06, respectively. As a result, the training dataset has 96,369 steps and 6,138 validation steps. Each step consists of the sensor data, the control action, and the navigation information of the autonomous vehicle controlled by Roach. The TCP pipeline with the supernet is trained over 120 epochs with a batch size of 16 and the Adam optimizer with a learning rate of 0.0001. Then, we use $K = 50$ additional epochs to train the TCP pipeline with the randomly selected subnets. To evaluate the driving performance of TimelyNet, we define a test route in Town 05 of the CARLA simulator, which does not appear in the training dataset. The total length of the routes is 500 meters. The maximum speed and acceleration of the autonomous vehicle are set to 15 m/s and 7 m/s^2, as the driving scenarios of the test routes are urban driving. We compare the performance of TimelyNet under different d_c settings of 0.5, 1.0, 1.5, 2.0, and 2.5 meters. The default setting for d_c is set to 1.0 meters, as it has the best performance among all the settings. The upper bound of the desired latency is set to 100 ms, which is the fastest reaction time of the human driver [16]. The lower bound of the desired latency is 47 ms, calculated based on the maximum speed and acceleration of the autonomous vehicle. The autonomous vehicle is controlled by TimelyNet to follow the predefined trajectory from the global path planner. The test route is repeated 10 times to evaluate the driving performance of TimelyNet.

The INN model is implemented using the FrEIA library [1]. It consists of 4 invertible units, each of which consists of one input layer, one hidden layer with 512 units, and one output layer. To build the profiling dataset, we randomly sample 10,000 different subnets and measure their latency and control quality using 100 image samples. We ensure the number of subnets within a latency interval is higher than 20 to ensure that the INN model can learn the distribution of the latency and architecture of the subnets. The hyperparameters H are converted to a one-hot vector with a length of 72. The hidden variable y is a vector with a length of 70. Each value of y follows a Gaussian distribution with a mean of zero and a variance of one. 90% of the samples are used for training and validation, while the remaining samples are used for testing. The INN model is trained for 200 epochs with a batch size of 16 and the Adam optimizer with a learning rate of 0.0001.

Table 2. Execution Latency Breakdown.

	Supernet		Smallest Subnet	
	Total	Encoder	Total	Encoder
Latency (ms)	84.3	75.2	41.1	33.9

Table 3. Solution Searching Latency.

	DNN	INN	Lookup	Min&Max selection
Latency (ms)	3.2	2.2	0.3	2.5

6.2 Evaluation of TimelyNet and Subnet Searching Baselines

6.2.1 Baselines. We evaluate the subnet searching performance of the following five methods:

(1) ***Min selection***, denoted as Min, is our proposed method described in §5.3.4.

(2) ***Max selection***, denoted as Max, is a variant of the min selection but selects the subnet with more parameters to maximize the control quality.

(3) ***DNN*** is the method that uses a DNN to predict the subnet hyperparameters given the latency requirement at runtime.

(4) ***INN*** is the method only uses an INN to predict the subnet hyperparameters at runtime.

(5) ***Lookup table*** is the method only uses the subnet solution obtained from the lookup table.

6.2.2 Evaluation Metrics. We evaluate the latency of TimelyNet and use the following four metrics to evaluate the performance of the subnet searching methods.

(1) ***Searching latency*** is the time to search the optimal subnet.

(2) ***Miss rate*** is calculated as N_o/N_{total}, where N_{total} is the total number of control periods and N_o is the number of control periods in which the execution of the driving models exceeds the desired latency.

(3) ***Average control quality*** is calculated as $Q = \sum_{n=1}^{N_{total}} Q_c^n/N_{total}$, where Q_c^n is the control quality of each generated subnet. The control quality of the subnet that exceeds the desired latency is set to zero.

(4) ***Trajectory MAE*** is the average distance between the vehicle's actual position and the nearest position in the predefined path, where the average is taken over all control periods. Specifically, it is calculated as $\frac{1}{N_{total}} \sum_{n=1}^{N_{total}} |x_{t_i} - x'_{t_i}| + |y_{t_i} - y'_{t_i}|$, where (x_{t_i}, y_{t_i}) is the actual position of the vehicle and (x'_{t_i}, y'_{t_i}) is the nearest position in the predefined path at time step t_i. Lower trajectory MAE indicates better driving performance, because the vehicle can follow the predefined path more accurately.

6.2.3 Evaluation Results. In this subsection, we present the latency results and the evaluation results of the subnet searching performance of TimelyNet and other baseline methods.

Execution Latency of TimelyNet: Table 2 shows the execution latency results of the TimelyNet using the supernet image encoder and the smallest subnet image encoder. The total latency range of TimelyNet is from 41.1 ms to 84.3 ms. The latency of the supernet image encoder is from 33.9 ms to 75.2 ms, which accounts for 89% and 82% of the total latency, respectively. The latency range of TimelyNet can cover the possible model latency in all driving scenarios. As stated in §6.1.2, the latency ranges from 47 ms to 100 ms. Thus, the execution latency of TimelyNet can meet the dynamic desired latency in our driving scenarios. In addition, when the supernet's latency is smaller than the desired latency, TimelyNet can use the supernet to generate the action, for the best control quality.

Searching Latency: Table 3 shows the mean latency of searching a subnet using TimelyNet and other searching baseline methods. We execute each method 100 times and calculate the mean latency. The searching latency of all the methods is negligible compared with the inference latency of TimelyNet. Specifically, the lookup table achieves the smallest mean latency of 0.3 ms. The INN and DNN methods achieve a mean latency of 2.2 ms and 3.2 ms, respectively. Min selection and max selection, which switches between the INN and lookup table, achieve the same latency of 2.5

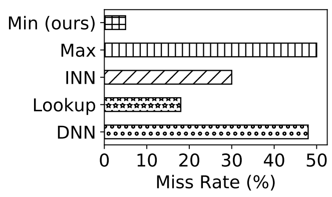


Fig. 6. Miss rate.

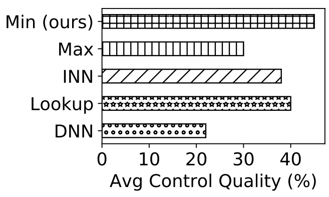


Fig. 7. Average Control Quality.

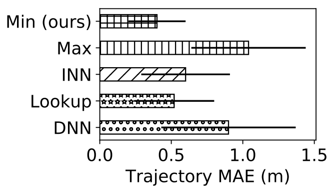


Fig. 8. Trajectory MAE.

ms. The latency of the selection methods is less than 6% of the latency of TimelyNet, ensuring that the searching process only uses a small amount of available inference time. We also test the latency of solving the optimization problem using the Genetic Algorithm (GA) in [3], which is 54.08 ms. Compared with solving the optimization problem, the INN and lookup table methods achieve around 80 times speedup.

Subnet Searching Performance: Fig. 6 and Fig. 7 illustrate the miss rate and average control quality of different subnet searching methods. The desired latency is calculated based on the speed and acceleration of the ego vehicle at each time step. The inference latency of TimelyNet and the control quality of the generated subnet are recorded at each time step. By comparing the recorded inference time with the desired latency, we can calculate the miss rate. In Fig. 6, min selection achieves the lowest miss rate of 4%, while INN and lookup table achieve a higher miss rate of 30% and 18%, respectively. The INN and lookup table cannot consistently find the suitable subnet across all desired latency ranges, primarily due to the limited number of subnets in the training dataset. By taking the subnet with fewer parameters from INN and the lookup table, min selection can select the subnet with lower latency and have a higher chance of meeting the desired latency. Different from min selection, max selection selects the subnet with more parameters and higher latency, resulting in a higher miss rate than min selection. The DNN method has a high miss rate because it cannot learn the mapping between the architecture and latency well. In Fig. 7, min selection achieves the highest average control quality of 45%, which improves the average control quality by 7% and 5%, compared with INN and lookup table, respectively. Although max selection attempts to achieve higher control quality by selecting the larger subnet, the high miss rate leads to a lower average control quality of 30%, however. Lastly, the DNN method achieves the lowest average control quality of 22%. The high miss rate and low control quality of DNN indicate that it cannot learn the mapping between the architecture and latency well.

Driving Performance: Fig. 8 shows the trajectory MAE of different subnet searching methods. Similarly, min selection achieves the lowest trajectory MAE of 0.41 meters, meaning that autonomous vehicles can follow the predefined path with the lowest error. It improves the trajectory MAE by 0.2 meters and 0.12 meters compared with INN and the lookup table, respectively. The improvement over the lookup table indicates that the lookup table cannot select the optimal subnet under all desired latencies. Jointly using INN can help mitigate the limitation of the lookup table, because INN can find better subnets under some desired latencies. The max selection method has a higher trajectory MAE of 1.04 meters because it attempts to select the subnet with higher latency, which exceeds the desired latency and leads to a higher trajectory MAE. The DNN method has the highest trajectory MAE of 0.9 meters. In summary, the min selection method can better adapt to the desired latency and achieve better control quality in terms of trajectory MAE than the other methods.

6.3 Performance of TimelyNet on TCP Pipeline

In this section, we present the details of the comparison between TimelyNet on TCP pipeline with other driving baselines.

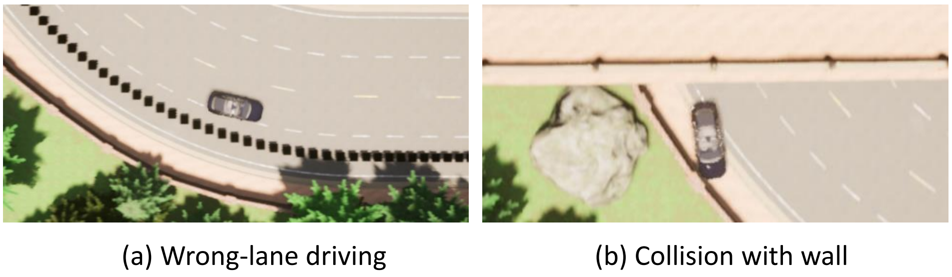


Fig. 9. Examples of wrong-lane driving and collision with walls. Black dots in (a) represent the predefined path.

6.3.1 Baselines: The main target of TimelyNet is to improve existing autonomous driving pipelines under dynamic deadlines, and it is implemented on top of the TCP pipeline. We compare TimelyNet with several variants of the original TCP pipeline and the model switch method. The compared methods are described as follows:

(1) **TimelyNet-Max** is the TCP pipeline that uses the supernet as the image encoder. Thus, it uses the maximum number of parameters.

(2) **TimelyNet-Min** is the TCP pipeline that uses the subnet with the minimum number of parameters as the image encoder.

(3) **TCP-Res152** is the TCP pipeline that uses ResNet152 as the image encoder.

(4) **TCP-Res101** is the TCP pipeline that uses ResNet101 as the image encoder.

(5) **TCP-Res50** is the TCP pipeline that uses ResNet50 as the image encoder.

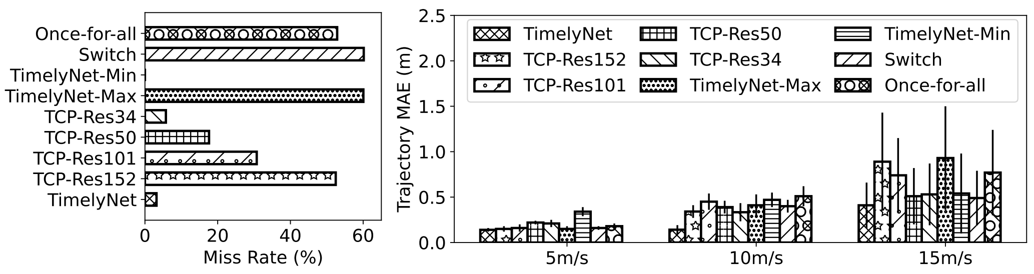
(6) **TCP-Res34** is the TCP pipeline that uses ResNet34 as the image encoder.

(7) **Model switch** is a method in [8] that stores multiple models with the same function but different latency-accuracy trade-offs and switches among these models at runtime to adapt to the end-to-end dynamic deadline. To ensure fast switching, the models are stored in memory to avoid the lengthy processing of loading from disk. Similar to [8], we store five variants of the TCP pipeline with different image encoders. The five image encoders are selected from the subnet dataset to train the INN and the lookup table. Specifically, we store the model with a latency ranging from 40 ms to 80 ms with an interval of 10 ms. For each latency, we select the subnet architecture in the lookup table and store the corresponding model in memory.

(8) **Once-for-all** is the method in [3] that uses an iterative evolution algorithm to search for the subnet architecture that maximizes the model accuracy given the latency constraints. In this method, a latency predictor and an accuracy predictor are trained to predict the latency and model accuracy of the sampled subnet, which can avoid the measurement of the latency and accuracy on a specific dataset. The latency predictor is a lookup table that stores the latency of each layer in the supernet. The accuracy predictor is an MLP model that predicts the accuracy of the subnet. The MLP model has 4 hidden layers with 400 units in each layer. The total searching time of the once-for-all method is 54.08 ms.

6.3.2 Evaluation Metrics. To evaluate the driving performance, we use miss rate and trajectory MAE. To evaluate the driving safety, we measure the wrong-lane driving rate and the number of collisions, which are the common infractions in the CARLA simulator. We use driving score as the overall metric to represent driving safety. Finally, we also measure the memory usage of all the methods. The newly introduced metrics are defined as follows:

(1) ***Wrong-lane driving rate***, denoted as R, is the percentage of the route that the autonomous vehicle deviates from the predefined path over 0.8 meters to the total length of the route. The



(a) Miss rate. (b) Trajectory MAE.

Fig. 10. Miss rate and trajectory MAE of TimelyNet and baselines on TCP.

autonomous vehicle is supposed to follow the predefined route in the center of the lane. If the vehicle deviates from the predefined path over 0.8 meters, it will cross the edge of the lane and be considered as wrong-lane driving.

(2) ***Number of collisions***, denoted as N_{col}, is the number of collisions that the autonomous vehicle makes.

(3) ***Driving score***, denoted as S, evaluates the driving safety of the autonomous driving pipeline in CARLA. It considers all the infractions that the autonomous vehicle makes during the simulation, including wrong-lane driving and collisions. It is calculated as $(1 - R) \times \prod_{k=0}^{N_{col}} P_k$, where P_k is the penalty factor for collisions, which is set to 0.5, as defined in the CARLA Leaderboard challenges. The driving score S ranges from 0 to 100%. Higher scores indicate better driving safety.

(4) ***Memory usage*** is the memory that each baseline method uses to store the models.

6.3.3 Evaluation Results. In this subsection, we show the driving performance, driving safety, and memeory usage of TimelyNet with other driving baselines in the CARLA simulator.

Driving Performance: Fig. 10 shows the miss rate and the trajectory MAE of TimelyNet and the eight baselines under different maximum speeds: 5 m/s, 10 m/s, and 15 m/s. From Figs. 10 (a) and (b), TimelyNet achieves a miss rate of 2% and the lowest trajectory MAE of 0.02 meters, 0.14 meters, and 0.41 meters under maximum speeds of 5 m/s, 10 m/s, and 15 m/s, respectively. The low miss rate and trajectory MAE indicate that adapting the subnet architecture to different vehicle speeds and accelerations helps the autonomous vehicle follow the predefined path more accurately. The improvement of TimelyNet over the baselines is more significant at higher maximum speeds. TimelyNet-Min achieves zero miss rate, as it always selects the subnet with the minimum number of parameters to satisfy the desired latency. However, it has a higher trajectory MAE than TimelyNet, because the minimum subnet cannot achieve the best driving performance. In contrast, TimelyNet-Max has higher miss rates due to larger inference latency, leading to a higher trajectory MAE. To show different trade-offs using different backbones, we also compare TimelyNet with the TCP pipeline using different variants of ResNet. The miss rate decreases from TCP-Res152 to TCP-Res34, because ResNet34 has the lowest inference latency among all the ResNet variants. However, the trajectory MAE of TCP-Res50 is the lowest among all the ResNet variants, because it has a better balance between inference latency and control quality. We also compare TimelyNet with the model switch method and the once-for-all method, which are two methods that can adapt to the dynamic deadline. The miss rate of the once-for-all method is 52.8%, which is high due to the longer searching time of the evolution algorithm. The model switch method has a miss rate of 20%, which is lower than the once-for-all method but higher than TimelyNet. The trajectory MAE of the model switch method is 0.58 meters, which is higher than that of TimelyNet. The limited improvement of the

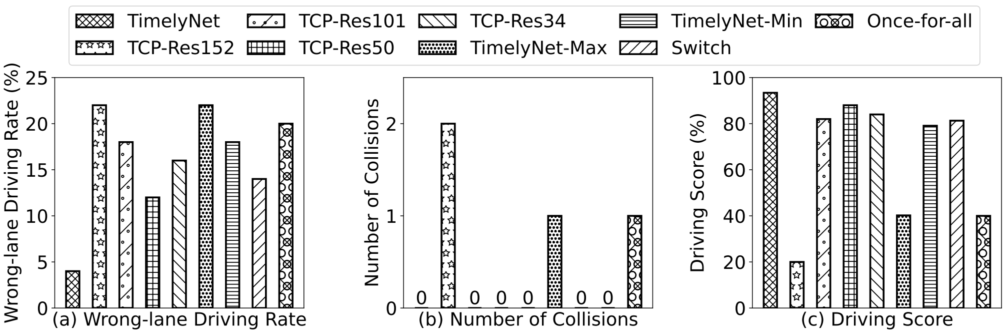


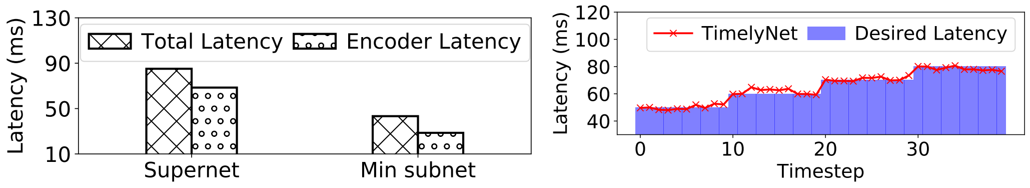
Fig. 11. Infractions and driving scores of TimelyNet and baseline methods on TCP

Table 4. Memory usage of TimelyNet and the baselines on TCP

	TimelyNet			Timely-Net-Min	Timely-Net-Max	Switch	Once for all		TCP-Res			
Memory usage (MB)	super-net	Lookup-table	INN				super-net	predi-ctor	152	101	50	34
	608.1	0.2	1.8	340.0	608.1	1904.4	608.1	2.8	747.9	595.3	367.0	287.0

model switch method is due to the limited number of models stored in memory, which cannot cover all the desired latency ranges. The results demonstrate that TimelyNet can achieve better driving performance by adapting the subnet architecture to different vehicle speeds and accelerations. The improvement over the TCP pipeline using ResNet variants demonstrates that TimelyNet can improve the driving performance of the TCP pipeline by incorporating a dynamic supernet image encoder. Furthermore, the improvement over the model switch method and the once-for-all method demonstrates that the key to achieving better driving performance is the large subnet space and the fast searching time of TimelyNet.

Infractions and Driving Safety: In particular, we evaluate the infractions and driving safety of TimelyNet and the baseline methods at the maximum speed of 15 m/s, because the trajectory MAE of all methods is the highest at this speed. A large trajectory MAE may lead to two types of infractions: wrong-lane driving and collisions with walls. Fig. 9 shows examples of wrong-lane driving and collisions with walls. If the autonomous vehicle deviates from the predefined path by more than 0.8 meters, it will cross the boundary of the lane and be considered as wrong-lane driving. Moreover, if the autonomous vehicle deviates to the right, it may collide with the wall on the right side of the road. Fig. 11 shows the infractions and driving safety results of TimelyNet and the baseline methods. In Fig. 11 (a), TimelyNet achieves the lowest wrong-lane driving rate of 4.1% because of the lowest trajectory MAE, while TimelyNet-Max and TCP-Res152 have the highest wrong-lane driving rates of 21%. Moreover, in Fig. 11 (b), only TimelyNet-Max and TCP-Res152 have collisions, while TimelyNet and the other methods do not have any collisions. Specifically, TimelyNet-Max and once-for-all have one collision, while TCP-Res152 has two collisions. This is because they have the highest trajectory MAE at the maximum speed of 15 m/s, leading to collisions with the wall on the right side of the road. In Fig. 11 (c), with the lowest wrong-lane driving rate and zero collisions, TimelyNet achieves the highest driving score of 95.9%. TCP-Res152 has the lowest driving score of 19.65% due to the high wrong-lane driving rate and two collisions. Although model switch, TimelyNet-Min, and TCP-Res34 do not have any collisions, they have higher wrong-lane driving rates than TimelyNet, leading to lower driving scores. The results demonstrate that TimelyNet can achieve safer driving by adapting the subnet architecture to different vehicle speeds and accelerations to reduce the trajectory MAE.



(a) Execution Latency of Interfuser with TimelyNet. (b) Response to Desired Latency.

Fig. 12. Execution Latency and Latency Response of Interfuser with TimelyNet.

Memory Usage: We also conduct a comparison of memory usage between TimelyNet and baseline methods, as shown in Table 4. TimelyNet requires 608.1 MB, 1.8 MB, and 0.2 MB to store the supernet, INN, and lookup table, respectively. The INN and lookup table only require a small amount of memory, which is negligible compared with the supernet. TimelyNet-Max requires 608.1 MB to store the driving model with the supernet, while TimelyNet-Min needs only 340 MB to store the smallest subnet. The model switch method consumes 1904.4 MB to store five models with varying latencies, which is three times the memory usage of TimelyNet. With such significant memory consumption, the model switch method only achieves limited improvement in driving performance compared with TimelyNet. In addition, the once-for-all method consumes 608.1 MB to store the supernet and 2.8 MB to store the latency and accuracy predictors, which is similar to TimelyNet. Meanwhile, the TCP pipeline with different ResNet backbones consumes 747.9 MB, 595.3 MB, 367.0 MB, and 287.0 MB to store the ResNet152, ResNet101, ResNet50, and ResNet34 models, respectively. Compared with TCP-Res152, TimelyNet can enhance driving performance while reducing memory usage. However, compared with TCP-Res101, TCP-Res50, and TCP-Res34, TimelyNet consumes more memory, but it can achieve better driving performance. Although modern autonomous driving hardware offers larger memory capacities, simultaneous execution of multiple tasks, like driving OS, often results in memory constraints for autonomous driving pipelines. Therefore, integrating TimelyNet into existing autonomous driving pipelines to enhance driving performance while maintaining minimal memory overhead is essential.

6.4 Performance of TimelyNet with Interfuser

In this section, we evaluate the performance of TimelyNet with Interfuser, one of the state-of-the-art multi-modal end-to-end autonomous driving pipelines.

Experiment Setup: Interfuser ranks the second in the CARLA Leaderboard challenges for closed-loop evaluation, while the first-place pipeline is not open source. Interfuser combines multi-view RGB images and LiDAR data to generate the control actions of the autonomous vehicle. Specifically, Interfuser also uses ResNet-based encoders to extract features from the RGB images of 4 cameras and LiDAR data. The ResNet-based encoders are shared across all cameras, while the LiDAR data is processed by a separate encoder. To implement TimelyNet on Interfuser, we replace the ResNet-based image encoders with the supernet, as the image encoder has a larger size and more parameters than the LiDAR encoder in the original Interfuser. Then, we use a similar method to train the INN and the lookup table as in the TCP pipeline. We compare the performance of TimelyNet on Interfuser with similar baselines, including TimelyNet-Max, TimelyNet-Min, the model switch method, and Interfuser with different ResNet-based encoders.

Execution Latency and Desired Latency Response: Fig. 12 (a) shows the execution latency of the Interfuser pipeline with TimelyNet. The whole pipeline has a latency range of 43.3 ms to 85.1 ms, with the encoder contributing 33.9 ms to 75.2 ms latency. The results demonstrate that TimelyNet can be integrated into other end-to-end autonomous driving pipelines to achieve a

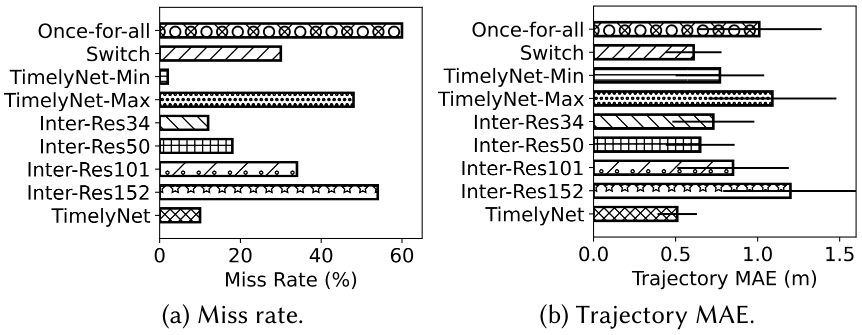


Fig. 13. Miss rate and trajectory MAE of TimelyNet and baselines on Interfuser.

diverse latency range. Moreover, Fig. 12 (b) shows the response of the Interfuser pipeline with TimelyNet to different desired latency. We set the desired latency to 50 ms, 60 ms, 70 ms, and 80 ms, respectively. For each desired latency, we record the inference latency of the Interfuser with the generated subnet for 10 time steps. The results show that the inference latency can be adjusted precisely according to the desired latency. The overall results demonstrate the generalizability of TimelyNet to existing end-to-end autonomous driving pipelines.

Driving Performance: We evaluate the miss rate and trajectory MAE of TimelyNet and the baselines on Interfuser, as shown in Fig. 13. We measure the miss rate and trajectory MAE under the maximum speed of 15 m/s, because the trajectory MAE of all methods is the highest at this speed. Similar to the results on the TCP pipeline, TimelyNet achieves a low miss rate of 5.2% and the lowest trajectory MAE of 0.51 meters. Among the baselines with variants of ResNet, Interfuser-Res50 achieves the lowest trajectory MAE of 0.58 meters. For the model switch method, it can achieve a lower trajectory MAE of 0.54 meters than other baselines, but higher than TimelyNet. The once-for-all method has the highest miss rate and a trajectory MAE of 1.02 meters. The iterative searching process makes it unsuitable for real-time applications. These results demonstrate the generalizability of TimelyNet to multi-modal end-to-end autonomous driving pipelines.

6.5 TimelyNet on Jetson AGX Orin

We also evaluate the performance of TimelyNet on the Jetson AGX Orin, a widely used embedded device for autonomous driving applications. We design a new supernet with a smaller number of parameters to ensure the latency range of TimelyNet on the embedded device is within 50 ms to 100 ms. The TCP pipeline with the new supernet consumes 125.6 MB of memory. Table 5 shows the execution latency of the TCP pipeline with the new supernet and the smallest subnet on Orin. The new TCP pipeline achieves a latency range from 35.4 ms to 75.3 ms, with the encoder contributing 28.9 ms to 67.4 ms latency. Similarly, we collect the subnet dataset on Orin and train the INN and the lookup table. As CARLA is not supported on Orin, we collect a data trace in CARLA using the TCP pipeline on the computers introduced in §6.1.1 with the new supernet and replay the data trace on Orin. Specifically, every 50 ms, we record a data frame, which includes sensor data, vehicle speed and acceleration, and the control action generated from the new TCP pipeline with the supernet. We replay the data trace on Orin and measure the miss rate and control quality using the data trace. Table 6 shows the miss rate and control quality of TimelyNet on Orin. TimelyNet achieves a miss rate of 6.3% and the highest control quality of 47.4% on Orin. TimelyNet-Max has the highest miss rate of 78.3%, which results in a lower control quality of 11.7%. TimelyNet-Min achieves a 0% miss rate, but it has the lowest control quality of 8.7%. The results demonstrate that

Table 5. Execution Latency Breakdown on Orin.

	Supernet		Smallest Subnet	
Latency (ms)	Total	Encoder	Total	Encoder
	75.3	67.4	35.4	28.9

Table 6. Performance of TimelyNet on Orin.

	TimelyNet	TimelyNet-Max	TimelyNet-Min
Miss Rate (%)	6.3	78.3	0
Control Quality (%)	47.4	11.7	8.7

TimelyNet can also improve driving performance on the embedded device by adapting the subnet architecture to different latency requirements.

7 Limitations and Discussions

While TimelyNet demonstrates promising results in adapting neural architectures for autonomous driving, there are two limitations: the offline profiling overhead and the generalizability of TimelyNet. First, the offline profiling process requires significant time to train the supernet and build the profiling dataset. Training an efficient supernet is a complex task that requires a large amount of data and computational resources [7]. In our experiments, it takes more than a day to train the supernet and subnets. Moreover, as mentioned in §4.4, profiling one subnet takes about 8 seconds, and building the profiling dataset of 10,000 subnets takes about 22 hours. Second, once deployed on a specific platform, TimelyNet may not generalize well to other platforms or hardware configurations. It requires retraining the supernet and rebuilding the latency lookup table to adapt to the new platform. Although the adaptation process can be done offline, it still requires extra time and resources. Considering these limitations, a natural way to adopt TimelyNet is that the car manufacturer pre-profiles the supernet on a representative dataset before deploying it to the autonomous driving system. If the sensor setup or hardware platform is changed, the car manufacturer can retrain the supernet, rebuild the latency lookup table, and update the TimelyNet on the vehicle through an over-the-air update.

Future work can reduce the overhead of the offline profiling process and improve the generalizability of TimelyNet. For instance, the search space of the supernet can be reduced by applying techniques such as pruning the unimportant paths in the supernet [18, 29]. Batch normalization calibration [7] can be employed to improve the performance of the sampled subnets. Another interesting direction is to extend TimelyNet to address modular designs and transformers. TimelyNet can be applied to each of the models in the modular design to generate subnets with diverse latency and control quality at runtime. The assignment of the time budget to the modules will need fine considerations. Transformer models have been used in autonomous driving pipelines for perception, prediction, and planning tasks [6, 15]. Due to the complex structure, their inference times are usually longer than those of CNN models. Designing supernets for transformers is still an open but interesting question.

8 Conclusion

In this paper, we present TimelyNet, a real-time neural architecture adaptation approach for autonomous driving pipelines. TimelyNet integrates a dynamic DNN model called supernet into an existing autonomous driving pipeline named TCP to meet dynamic deadlines. By sampling subnets from the supernet, TimelyNet aims to maximize control quality while meeting the latency requirements. To efficiently determine the subnet architecture on a per-inference basis, we employ a one-shot optimal architecture prediction approach. TimelyNet jointly uses a lookup table and an INN to predict the subnet architecture based on the specified latency requirement. Hardware-in-the-loop experiments have been conducted to evaluate TimelyNet and five driving baselines including variants of the TCP pipeline in the CARLA simulator. The results show that TimelyNet can follow the predefined trajectory with the lowest errors under all driving speeds among all the baseline methods. In addition, TimelyNet can achieve the lowest wrong-lane driving rate and zero collisions, delivering safe driving during the entire simulation. More importantly, it only introduces a small overhead of 2.5 ms to search for the optimal subnet in each control period, around 80x faster than solving the optimization problem using heuristic search methods. The results demonstrate that integrating TimelyNet into the TCP pipeline can achieve better driving performance with only a small overhead of subnet searching time. We also evaluate TimelyNet on Jetson AGX Orin and Interfuser to show its extensibility.

References

[1] Lynton Ardizzone, Till Bungert, Felix Draxler, Ullrich Köthe, Jakob Kruse, Robert Schmier, and Peter Sorrenson. 2018-2022. *Framework for Easily Invertible Architectures (FrEIA).* https://github.com/vislearn/FrEIA.

[2] Lynton Ardizzone, Carsten Lüth, Jakob Kruse, Carsten Rother, and Ullrich Köthe. 2019. Guided image generation with conditional invertible neural networks. *arXiv preprint arXiv:1907.02392* (2019).

[3] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. 2020. Once for All: Train One Network and Specialize it for Efficient Deployment. In *International Conference on Learning Representations (ICLR).*

[4] CARLA. 2023. CARLA Autonomous driving Leaderboard. https://leaderboard.carla.org/.

[5] Pranav Singh Chib and Pravendra Singh. 2023. Recent advancements in end-to-end autonomous driving using deep learning: A survey. *IEEE Transactions on Intelligent Vehicles* 9, 1 (2023), 103–118.

[6] Kashyap Chitta, Aditya Prakash, Bernhard Jaeger, Zehao Yu, Katrin Renz, and Andreas Geiger. 2022. Transfuser: Imitation with transformer-based sensor fusion for autonomous driving. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 45, 11 (2022), 12878–12895.

[7] Xiangxiang Chu, Shun Lu, Xudong Li, and Bo Zhang. 2023. Mixpath: A unified approach for one-shot neural architecture search. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV).* 5972–5981.

[8] Ionel Gog, Sukrit Kalra, Peter Schafhalter, Joseph E Gonzalez, and Ion Stoica. 2022. D3: a dynamic deadline-driven approach for building autonomous vehicles. In *Proceedings of the Seventeenth European Conference on Computer Systems (EuroSys).*

[9] Ionel Gog, Sukrit Kalra, Peter Schafhalter, Matthew A Wright, Joseph E Gonzalez, and Ion Stoica. 2021. Pylot: A modular platform for exploring latency-accuracy tradeoffs in autonomous vehicles. In *2021 IEEE International Conference on Robotics and Automation (ICRA).* IEEE, 8806–8813.

[10] Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. 2020. Single path one-shot neural architecture search with uniform sampling. In *European conference on computer vision (ECCV).* Springer, 544–560.

[11] Rui Han, Qinglong Zhang, Chi Harold Liu, Guoren Wang, Jian Tang, and Lydia Y Chen. 2021. Legodnn: block-grained scaling of deep neural networks for mobile vision. In *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking (MobiCom).* 406–419.

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

[13] Zhijian He, Bohuan Xue, Xiangcheng Hu, Zhaoyan Shen, Xiangyue Zeng, and Ming Liu. 2024. Robust embedded autonomous driving positioning system fusing LiDAR and inertial sensors. *ACM Transactions on Embedded Computing Systems* 23, 1 (2024), 1–26.

[14] Pengfei Hu, Yuhang Qian, Tianyue Zheng, Ang Li, Zhe Chen, Yue Gao, Xiuzhen Cheng, and Jun Luo. 2025. t-READi: Transformer-Powered Robust and Efficient Multimodal Inference for Autonomous Driving. *IEEE Transactions on Mobile Computing* 24, 1 (2025), 135–149.

[15] Yihan Hu, Jiazhi Yang, Li Chen, Keyu Li, Chonghao Sima, Xizhou Zhu, Siqi Chai, Senyao Du, Tianwei Lin, Wenhai Wang, et al. 2023. Planning-oriented autonomous driving. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR).* 17853–17862.

[16] Shih-Chieh Lin, Yunqi Zhang, Chang-Hong Hsu, Matt Skach, Md E Haque, Lingjia Tang, and Jason Mars. 2018. The architectural implications of autonomous driving: Constraints and acceleration. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS).* 751–766.

[17] Arnav Malawade, Mohanad Odema, Sebastien Lajeunesse-DeGroot, and Mohammad Abdullah Al Faruque. 2021. SAGE: A split-architecture methodology for efficient end-to-end autonomous vehicle control. *ACM Transactions on Embedded Computing Systems* 20, 5s (2021), 1–22.

[18] Yuiko Sakuma, Masato Ishii, and Takuya Narihira. 2023. DetOFA: Efficient Training of Once-for-All Networks for Object Detection using Path Filter. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV).* 1333–1342.

[19] Hao Shao, Letian Wang, Ruobing Chen, Hongsheng Li, and Yu Liu. 2023. Safety-enhanced autonomous driving using interpretable sensor fusion transformer. In *Conference on Robot Learning (CORL)*. PMLR, 726–737.

[20] Wei Sun and Kannan Srinivasan. 2022. On the feasibility of securing vehicle-pavement interaction. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 6, 1 (2022), 1–24.

[21] Emil Talpes, Debjit Das Sarma, Ganesh Venkataramanan, Peter Bannon, Bill McGee, Benjamin Floering, Ankit Jalote, Christopher Hsiong, Sahil Arora, Atchyuth Gorti, and Gagandeep S. Sachdev. 2020. Compute Solution for Tesla's Full Self-Driving Computer. *IEEE Micro* 40, 2 (2020), 25–35.

[22] Baidu Apollo team. 2017. Apollo: Open Source Autonomous Driving. https://github.com/ApolloAuto/apollo.

[23] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. 2021. Scaled-YOLOv4: Scaling Cross Stage Partial Network. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 13029–13038.

[24] Ruiqi Wang, Hanyang Liu, Jiaming Qiu, Moran Xu, Roch Guérin, and Chenyang Lu. 2023. Progressive neural compression for adaptive image offloading under timing constraints. In *2023 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 118–130.

[25] Hao Wen, Yuanchun Li, Zunshuai Zhang, Shiqi Jiang, Xiaozhou Ye, Ye Ouyang, Yaqin Zhang, and Yunxin Liu. 2023. AdaptiveNet: Post-deployment Neural Architecture Adaptation for Diverse Edge Environments. In *Proceedings of the 29th Annual International Conference on Mobile Computing and Networking (MobiCom)*. 1–17.

[26] Penghao Wu, Xiaosong Jia, Li Chen, Junchi Yan, Hongyang Li, and Yu Qiao. 2022. Trajectory-guided Control Prediction for End-to-end Autonomous Driving: A Simple yet Strong Baseline. In *Neural Information Processing Systems (NeurIPS)*.

[27] Mingqing Xiao, Shuxin Zheng, Chang Liu, Yaolong Wang, Di He, Guolin Ke, Jiang Bian, Zhouchen Lin, and Tie-Yan Liu. 2020. Invertible image rescaling. In *European conference on computer vision (ECCV)*. Springer, 126–144.

[28] Saehanseul Yi, Tae-Wook Kim, Jong-Chan Kim, and Nikil Dutt. 2023. EASYR: E nergy-Efficient A daptive Sy stem R econfiguration for Dynamic Deadlines in Autonomous Driving on Multicore Processors. *ACM Transactions on Embedded Computing Systems* 22, 3 (2023), 1–29.

[29] Shan You, Tao Huang, Mingmin Yang, Fei Wang, Chen Qian, and Changshui Zhang. 2020. Greedynas: Towards fast one-shot nas with greedy supernet. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 1999–2008.

[30] Zhejun Zhang, Alexander Liniger, Dengxin Dai, Fisher Yu, and Luc Van Gool. 2021. End-to-End Urban Driving by Imitating a Reinforcement Learning Coach. In *Proceedings of the IEEE/CVF international conference on computer vision (ICCV)*.

[31] Qi Zhu, Wenchao Li, Hyoseung Kim, Yecheng Xiang, Kacper Wardega, Zhilu Wang, Yixuan Wang, Hengyi Liang, Chao Huang, Jiameng Fan, et al. 2020. Know the unknowns: Addressing disturbances and uncertainties in autonomous systems. In *Proceedings of the 39th International Conference on Computer-Aided Design (ICCAD)*. 1–9.