ConvertNet: Training-Time Model Scaling for In-Vehicle Continuous Learning

Qinglong Zhang, Rui Han, Chi Harold Liu, *Fellow, IEEE*, Guoren Wang, *Senior Member, IEEE*, and Lydia Y. Chen, *Senior Member, IEEE*

***Abstract*—Vehicles are transforming from transportation means to mobile intelligent terminals, powered by artificial intelligence (AI) technologies such as large language (LLM) and vision large models (VLM). To maintain high learning accuracy in all situations, in-vehicle AI models encounter the acute need to continuously learn and dynamically retrain AI models on the fly. The volatile resource demands of such stochastic retraining jobs and inference jobs of higher priority however can not be accommodated simultaneously by existing training systems, which rely on pre-generated compressed models of fixed architecture. In this paper, we propose ConvertNet, a novel continuous learning system that effectively scales up/down a compressed model through training-time neuron memorizer (TNM) - a max heap of neurons in tree structure. Based on TNM, ConvertNet estimates the model's resource-accuracy trade-off per neuron, thus efficiently utilizing the limited available resources to scale up and (re)train the model to improve its accuracy. Evaluated on six representative in-vehicle scenarios, comparative experiments against eleven state-of-the-art techniques show that ConvertNet achieves as much as 22.33% improvement in learning accuracy, and reduces energy consumption by 3.65x.**

***Index Terms*—Continuous learning, in-vehicle training, training-time scaling, transformer.**

I. Introduction

VEHICLES are increasingly powered by artificial intelligence (AI) applications [1], [2], [3], such as object detection [4], [5], video understanding [6], speech recognition [7], and task-oriented dialog [8]. Through these AI models, the vehicle is able to dialogue with humans (e.g. drivers or passengers), analyze visual information from the surrounding, and is promised to support autonomous driving services [9], [10]. The *original models* are usually compressed and then deployed on the vehicles due to stringent latency requirements and limited resources [11], [12], [13], [14], [15]. In a mobile and open environment, maintaining compressed models' high accuracy requires **continuously adopting and retraining models** when encountering input distribution/domain drifts [16], [17], [18]. For example, in Fig. 1(a)'s autonomous driving application, the deployed compressed large language model (LLM) needs to continuously learn from the road condition or it is personalized for different road layouts and capacities [19].

The key to achieving the full promise of in-vehicle AI applications is leveraging the original model's learning capacity to improve retraining accuracy. This is non trivial in practice for two reasons. First, the straightforward approach is to retrain the original model, but the training cost may exceed the tight resource budget on in-vehicle devices. In Fig. 1(a)'s autonomous driving application [20], retraining the original model with batch size is 16 requires 23.1 GB memory, but a commodity device usually has less than 16 GB available memory for this retraining. Second, existing on-device retraining techniques pre-generate a compressed model to match the available resources [15], [21] or the current input distribution [22]. However, they keep the model's **network architecture fixed** during training. In reality, runtime available resources for retraining jobs may *frequently change* due to dynamic events such as starting new jobs or closing existing ones, or changed resource requirements from inference jobs [15], [23], as shown in the top of Fig. 1(b). This requires on-device retraining techniques to make **training-time** resource-accuracy trade-offs in a timely fashion according to current available resources. In conclusion, most existing retraining techniques are based on fixed compressed models at training time, which give rise to two limitations.

First, *fixed network architectures lower accuracy improvement to resource consumption*. Whenever available resources change during retraining, existing retraining techniques keep compressed models fixed and respond in three ways: adjusting training iteration [23], [24], [25], [26], batch size [15], or freezing weights [27]. However, the compressed models suffer from limited learning capacity, as *the quantization model theory of neural scaling* [28] verifies that scaling up network architectures brings larger accuracy improvement per resource consumption. Hence, a **compressed model scaling and retraining** approach should invest available resources on the original model's parts (e.g. layers and neurons in Fig. 1(a)) bringing the largest accuracy benefit. In Fig. 1(b)'s example resource increase and decrease scenarios, compressed model scaling and retraining approach improves model accuracies by 16.8% and 14.1%, respectively. Developing such a scaling method is nontrivial

Received 9 August 2025; revised 16 December 2025; accepted 10 February 2026. Date of publication 23 February 2026; date of current version 5 June 2026. This work was supported in part by the National Natural Science Foundation of China under Grant 62272046, Grant 62132019, and Grant 61872337, in part by the high-quality development project of the Ministry of Industry and Information Technology under Grant CEIEC-20240, in part by the Science and Technology Innovation Project of Beijing Institute of Technology under Grant 2023CX01017, and in part by the Open Project of Key Laboratory of the Ministry of Education under Grant 2023PY002. Recommended for acceptance by S. Ucar. *(Corresponding author: Rui Han.)*

Qinglong Zhang, Rui Han, Chi Harold Liu, and Guoren Wang are with the Beijing Institute of Technology, Beijing 100811, China (e-mail: 3120211050@bit.edu.cn; hanrui@bit.edu.cn; chiliu@bit.edu.cn; wanggr@bit.edu.cn).

Lydia Y. Chen is with the University of Neuchatel and TU Delft, 2600 AA Delft, Netherlands (e-mail: lydiaychen@ieee.org).

Digital Object Identifier 10.1109/TMC.2026.3667390

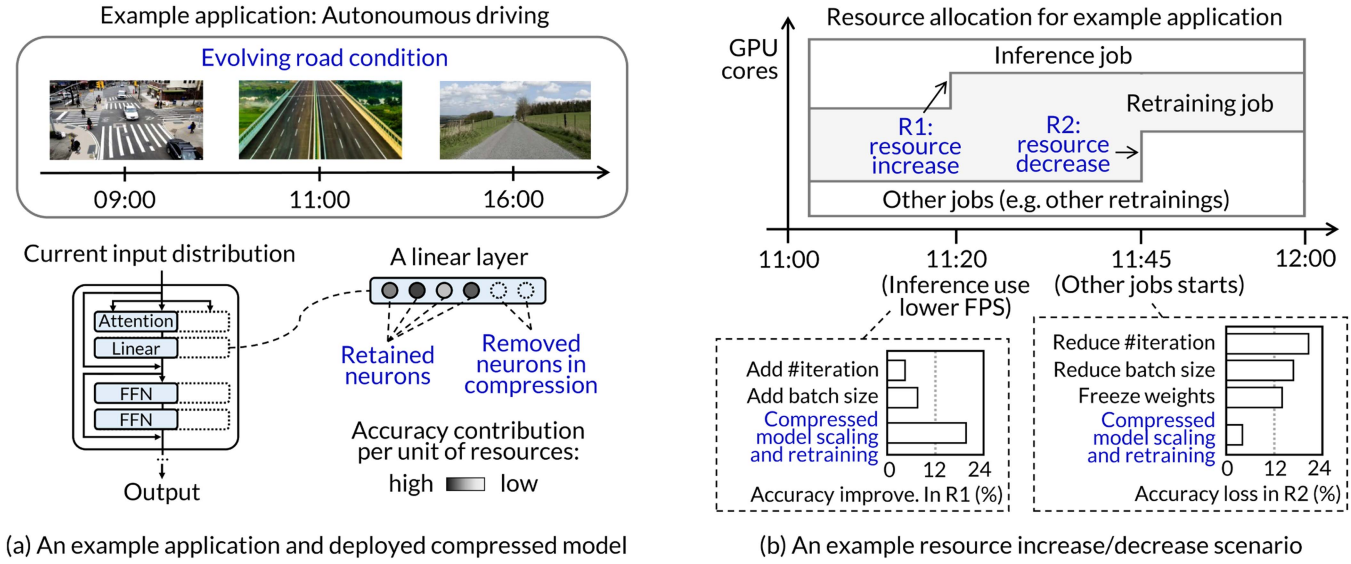

Fig. 1. Motivation scenario of in-vehicle continuous learning and compressed model scaling and retraining.

since, in practice, when scaling up a network, the newly added model parts need to be trained from scratch using extra resources [29]; in scaling down, directly pruning model weights significantly deteriorates accuracy [30]. The **first challenge**, therefore, is how to solve both problems when scaling up or down a model during its retraining.

Second, *fixed network architectures hinder fine-grained allocation of each unit of resource to increase accuracy.* To make resource-accuracy trade-offs during training, existing retraining techniques either adjust training hyperparameters and equally allocate resources to all model parts [15], [23], [24], [25], [26], or first allocate resources to CNN tensors/filters with the highest accuracy contribution but ignore their resource consumptions [27], [30]. To maximize the accuracy improvement per unit of resources, the **second challenge** requires us to comprehensively estimate different model parts' resource consumptions and their impacts on accuracy.

Our solution: We present ConvertNet, a compressed model scaling and retraining approach for in-vehicle continuous learning. The key idea of ConvertNet is introducing a **training-time neuron memorizer** (TNM), a max heap of neurons that stores *knowledge* (model weights that can extract useful features from input data) and allows the compressed model to quickly search relevant knowledge for scaling. Specifically, to scale up the compressed models, ConvertNet pops neurons from the top of TNM, adds them into the compressed model, and trains them based on their stored knowledge. To scale down, ConvertNet freezes neurons in the compressed model and adds frozen neurons into TNM, storing the lost knowledge to reduce accuracy loss. Moreover, given a neuron in TNM or the compressed model, ConvertNet supports online estimating its accuracy and resource on the current input distribution. The estimation enables optimal neuron-grained scaling in one retraining job and scheduling across multiple jobs, thus maximizing accuracy improvement per unit of allocated resources. In particular, the contributions of this paper are as follows:

- *Compressed model scaling and retraining via TNM:* TNM leverages the original model's large learning capacity, and dynamically transfers it to the compressed model according to available resources *during retraining*. In contrast, existing techniques only support such transfer once at pre-training [27] or before each retraining job [21], [22].
- *Resource-accuracy trade-offs at the neuron granularity:* ConvertNet designs an accuracy predictor and a FLOPs calculator to estimate neurons' accuracy contributions and resource consumptions. Its model scaling algorithm then iteratively assigns resources to neurons bringing the largest accuracy improvement per unit of resource for both one retraining job or multiple ones.
- *Implementation and evaluation:* We implement ConvertNet on NVIDIA DriveOS, a dedicated in-vehicle operating system, and conduct extensive evaluation against 11 state-of-the-art retraining techniques on 6 representative in-vehicle applications. We release the source code of ConvertNet in https://github.com/LINC-BIT/ConvertNet.

Summary of experimental results: We extensively compare ConvertNet against state-of-the-art (SOTA) techniques, using in-vehicle AI applications. These applications are developed based on six prevalent transformers (e.g. ViT [31], GLIP [32], VideoMAE [6], Conformer [7], LLaMA [8], and Transfuser [20]) and 12 datasets commonly used in intelligent vehicle community. The results show: (i) ConvertNet is able to provide neuron-grained and optimized resource-accuracy trade-off in model retraining. Compared with eleven retraining techniques, it increases accuracy by 22.33% with the same resource budget, and reduces memory footprint, retraining time, and energy consumption by 1.63x, 2.69x, 3.65x for the same learning task; (ii) Our scheduling approach

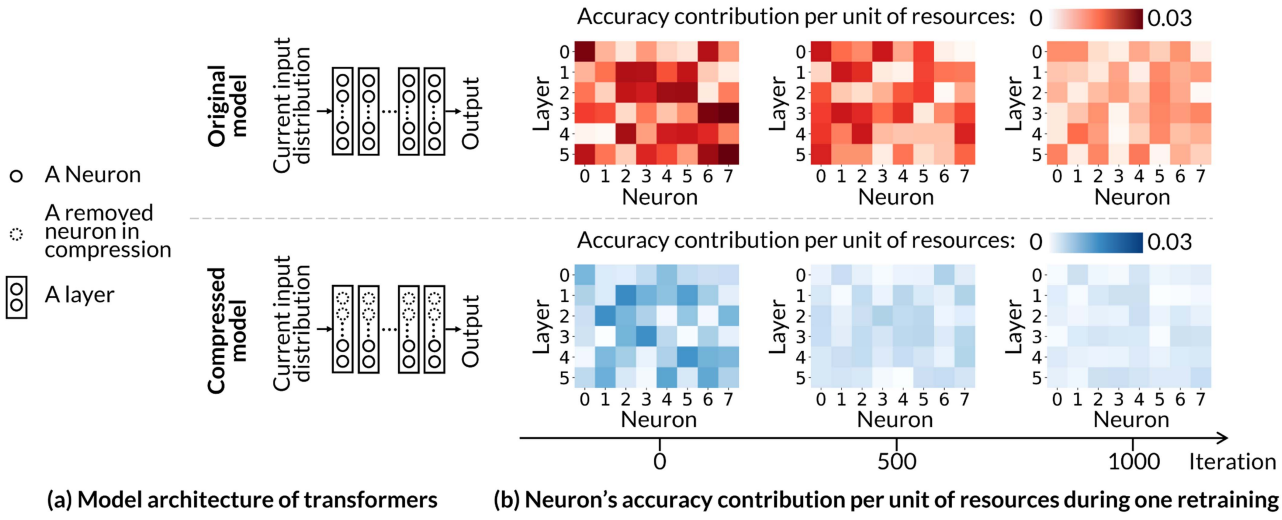

Fig. 2. Understanding neurons in original and compressed models during retraining. For convenient comparison between two groups of heatmaps, the two color bars have the same range.

outperforms nine on-device schedulers by achieving 10.61% accuracy improvement in five multi-application scenarios; and (iii) we fully test the design choices of `ConvertNet` and discuss its limitations (e.g. maximum supported model sizes) when applied in resource-constrained vehicles.

II. Background and Motivation

A. In-Vehicle AI Applications

In recent years, technical giants like NVIDIA, Huawei, and Bosch release their own automotive operating systems to facilitate AI application development [33]. For example, NVIDIA DriveOS [34] is embedded with essential runtime libraries (e.g. CUDA) for AI applications and it is compatible with AUTOSAR Adaptive software standard [35]. Typically, modern in-vehicle intelligent applications can be divided into three categories.

Driver-assist applications aim to enhance drivers' safety and comfort in the cockpit domain. For example, the traffic sign recognition application [36], [37] identifies traffic signs ahead and notifies the driver. The driver attention monitoring application [38], [39] analyzes the driver's behaviors such as distraction and gives necessary warnings.

Human-vehicle cooperation applications improve human-related personalized experiences. For example, the speech recognition application [7], [40] converts spoken commands from drivers and passengers into texts. The task-oriented dialogue application [8], [41] interprets these texts to control the vehicle equipments (e.g. opening the window).

Autonomous driving applications are designed to perceive the surrounding environment and predict planning actions (e.g. control signals for vehicles) in the intelligent driving domain [3], [20], [42].

B. Understanding Neurons in In-Vehicle Continuous Learning

Since 2023, transformer-based models (also called foundation models) are widely used in in-vehicle AI applications. For example, in the driver-assist applications, VLMs such as ViT [31], GLIP [32], and VideoMAE [6] support accurate traffic object detection and tracking. In human-vehicle cooperation applications, LLMs such as Conformer [7], GPT [43], and LLaMA [8] can parse the human voice as specific controlling commands. In addition, VLMs (e.g. Transfuser [20]) and LLMs (e.g. DriveGPT [44]) reveal the potential of transformers in end-to-end autonomous driving.

Neurons in continuous learning: Continuous learning techniques incrementally retrain the deployed model using new incoming samples. Recent studies focus on optimizing the retraining algorithm [45], [46], [47], hyperparameters [15], [24], or resource allocation [15], [23], [26] upon fixed compressed models. At a finer granularity, these models are composed of neurons, as shown in Fig. 2(a). Conceptually, a **neuron** is the basic computational unit of extracting features and each neuron has its unique contribution to model accuracy. In Fig. 2(b)'s example, this contribution is calculated by the FBS module [48]. In Fig. 2, we take the autonomous driving application (Transfuser [20] model and CARLA dataset [42]) as an example, and retrain the original and the compressed models (as shown in Fig. 2(a)) on an in-vehicle computing platform (NVIDIA AGX Orin). We randomly select several neurons from both models, and visualize their contributions during retraining (Fig. 2(b)).

We have two key observations: First, the neurons in the original model (the upper part of Fig. 2(b)) consistently exhibit *much higher contributions* (2.4x higher) than those in the compressed model. This observation motivates us to *exploit the original model's learning capacity to scale the compressed model to improve its accuracy*. Second, different neurons in a model have different contributions, and these contributions vary over training iterations. For instance, neuron 1 in layer 2 has 2.7x higher contribution than neuron 2 in the same layer at iteration 0. However, at iteration 1000, the former neuron's contribution is 1.2x lower. This motivates us to *online monitor all neurons' contributions and optimally allocate resources to them on the fly*.

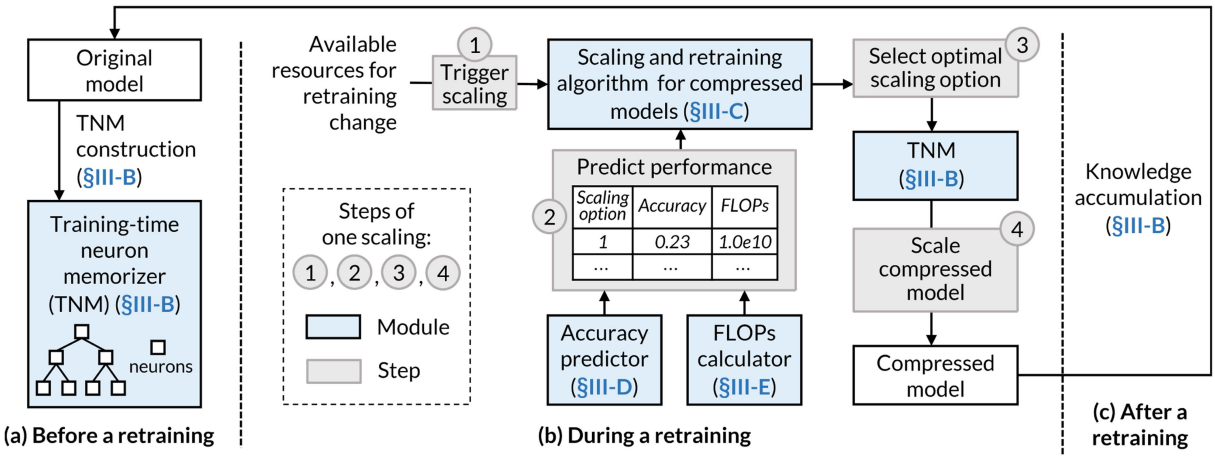

Fig. 3. ConvertNet's modules and running steps.

III. Design of ConvertNet

A. Overview

ConvertNet is designed to answer two research questions: (1) how to support compressed model scaling and retraining (Section III-B); and (2) how to make the optimal resource-accuracy trade-off at the neuron granularity during retraining (Section III-C, Section III-D, and Section III-E).

Fig. 3 illustrates ConvertNet's architecture with four modules. The key module is *training-time neuron memorizer (TNM)*, a data structure initialized when deployment and reconstructed before retraining of compressed models, as shown in Fig. 3(a). It stores neurons in a max heap (an ordered complete binary tree) and enables scaling of compressed models by exchanging key neurons. During each retraining, the other three modules select the optimal scaling option that maximizes the retraining accuracy under available resources, and apply it on TNM to perform scaling. As shown in Fig. 3(b), each scaling includes four steps. At step 1, ConvertNet triggers a scaling upon the available resources change. At step 2, the *accuracy predictor* and the *FLOPs calculator* modules take each candidate scaling option as input and output its retraining accuracy and FLOPs. At step 3, the *scaling algorithm* searches for the optimal scaling option under memory constraints. Following the searched option, ConvertNet scales the compressed model based on TNM at step 4.

Note that retraining is *continuous* in our scenario. Hence, as shown in Fig. 3(c), ConvertNet accumulates the newly learned knowledge into the original model after each retraining, and transfers it to the TNM of next retraining for reusing. Moreover, ConvertNet supports multi-application scenarios with multiple concurrent retraining jobs (Section III-F). ConvertNet constructs a global TNM for all jobs, and extends the scaling algorithm to maximize the overall retraining accuracy of all models.

B. Training-Time Neuron Memorizer (TNM)

TNM is designed as a max heap to store the original model's rich knowledge. Nodes in TNM store neurons and they are ordered by their accuracy contributions per unit of resources. This data structure enables three key operations for training-time model scaling: (1) *scaling up* adds the neuron with the highest contribution (from the heap's top) to the compressed model; (2) *scaling down* freezes a neuron in the compressed model and storing its knowledge into a node in the heap; (3) *knowledge accumulation* retains the compressed model's learned knowledge into the original model by neuron-wise tensor calculations.

TNM construction: The construction begins with an empty max heap, and iteratively adds the original model's neurons into the heap using four steps. Step 1 either extracts one neuron from a linear layer or a feed forward layer, or extracts one neuron from an attention layer's Q/K/V simultaneously. This is because neurons in Q/K/V focus on the same features and contribute equally to accuracy. Step 2 calculates the **accuracy contribution per resource consumption** of the extracted neuron(s) on the current input distribution, denoted as ΔA:

$$\Delta A = \frac{A^{after} - A^{before}}{F^{after} - F^{before}} \tag{1}$$

where A^{before}/F^{before} and A^{after}/F^{after} represent the compressed model's accuracy / FLOPs before and after adding the extracted neuron(s) (Section III-D and Section III-E). Step 3 creates a leaf node in the heap to store the extracted neuron(s). Step 4 repeatedly swaps the new node with its parent node, until every node has a larger ΔA than its child nodes. The construction is completed when all neurons in the original model are stored in TNM.

Fig. 4 demonstrates an example: five neurons in a linear layer are stored individually in nodes 1-5, respectively; 15 neurons in an attention layer are stored in nodes 6-10, where each node stores 3 neurons from Q, K, and V, respectively.

One scaling up operation: The operation contains two steps. Step 1 extracts a node from the top of the TNM, such as node 4 in Fig. 5(a). Step 2 adds the node's stored neuron(s) into the compressed model's corresponding layer. In Fig. 5's example, the neurons in node 4 are from the original model's first layer, and hence added to the compressed model's first layer. These neurons already contain knowledge that is highly relevant to the current distribution and do not require training from scratch.

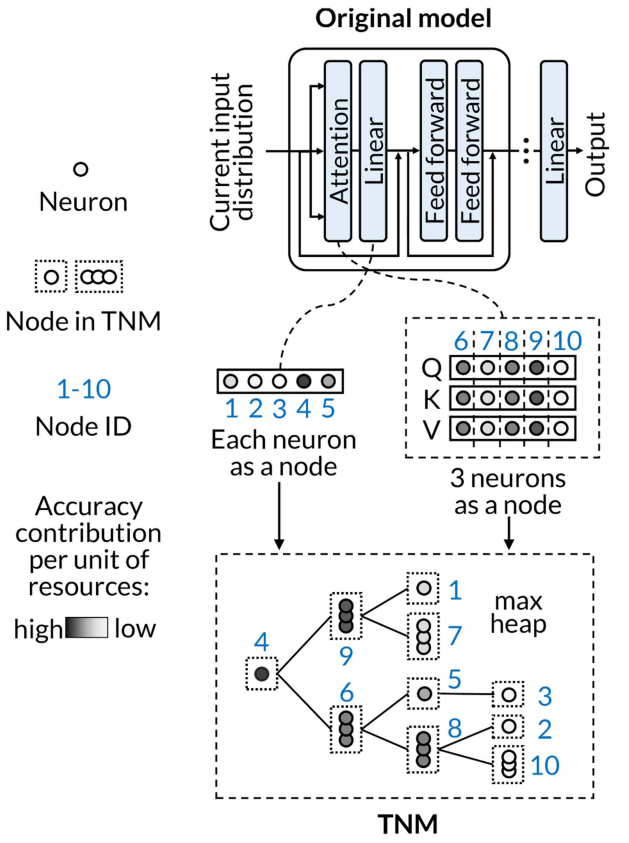

Fig. 4. An example of constructing TNM.

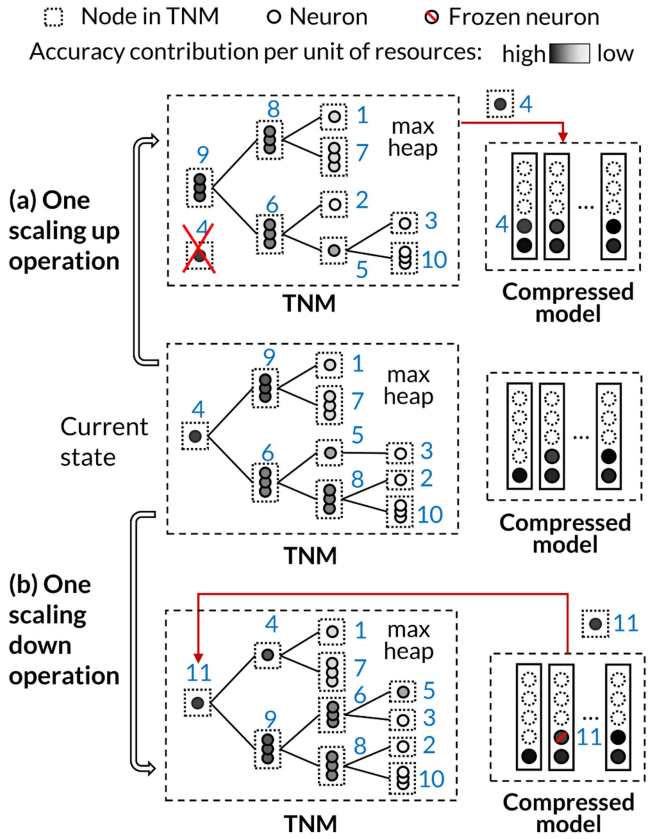

Fig. 5. Examples of one scaling up/down operation.

One scaling down operation: This operation includes three steps. Step 1 calculates ΔA for each neuron in the compressed model using (1). Step 2 freezes the neuron with the lowest ΔA (e.g. layer 2's neuron 2in Fig. 5(b)). Finally, step 3 creates a new node (e.g. node 11 in Fig. 5(b)) in the top of TNM to store the neuron. Such an operation preserves the compressed model's learning capacity for two reasons. First, the stored neuron will be returned to the compressed model in future scaling up. Second, the operation avoids pruning neurons, which may significantly degrade model accuracy [49].

Knowledge accumulation operation. This operation conducts three steps for each neuron in the compressed model. Step 1 extracts the neuron's learned knowledge, where a neuron θ^c's knowledge $\Delta\theta^c$ is defined as the difference of its weights before and after retraining. Step 2 finds the neuron's corresponding layer in the original model, and denotes this layer's contained neurons as $\{\theta_1^{ori}, \theta_2^{ori}, \cdots\}$. Finally, step 3 accumulates knowledge $\Delta\theta^c$ into the original model by applying the following equation: $\theta_i^{ori} \leftarrow \theta_i^{ori} + \Gamma_i \cdot \Delta\theta^c$, where Γ_i is the similarity between θ^c and θ_i^{ori} $(0 < \Gamma_i < 1)$.

Note that Γ_i is designed to apply regularization on θ_i^{ori}, which promotes positive knowledge transfer and mitigates catastrophic forgetting [50]. First, a large value of Γ_i means the weights of θ_i^{ori} and θ^c are similar and hence θ_i^{ori} can store more of θ^c's learned knowledge without conflict. This is based on the observation that two similar weights have a similar updating direction and magnitude in the retraining loss space [51]. In contrast, two dissimilar weights have conflict updating directions that cause knowledge forgetting. In our approach, a small value of Γ_i reduces the updating magnitude of θ_i^{ori} and prevents it from forgetting previously stored knowledge.

Overhead analysis: TNM is designed for low computational overhead and memory footprint. First, TNM construction only involves memory copy operations with negligible overhead and calculations of ΔA (which is also lightweight, see Section II-I-D). Second, `ConvertNet` partitions the whole heap into several sub-heaps, and sequentially loads them into memory for scaling up/down operations. By doing so, its memory overhead is reduced multiple times while keeping the *logarithmic* time complexity of scaling up/down operations. Third, knowledge accumulation involves only basic tensor multiplication and addition calculations and can be completed quickly. Taking ViT-Huge [31] as an example, TNM construction takes 1 s, a scaling up/down operation takes 80 MB memory and 100 ms, and knowledge accumulation takes less than 3 s in typical in-vehicle devices.

C. Scaling and Retraining Algorithm for Compressed Models

The scaling algorithm searches for the optimal scaling option that maximizes the retraining accuracy under the current available resources. A scaling relies on multiple TNM's scaling up/down operations, and hence a **scaling option** represents how many times these operations are performed. The basic idea is to greedily repeat these operations once resources change, until the model size matches the available resources.

Algorithm 1: Scaling and retraining algorithm for compressed models.

```
1  for each retraining iteration do
2      ΔR = monitor_resources_change();
3      if ΔR > 0 then
4          while ΔR > 0 do
5              Perform one operation of scaling up;
               /* Consume one unit of resources r      */
6              ΔR− = r;
7      else if ΔR < 0 then
8          while ΔR < 0 do
9              Perform one operation of scaling down;
               /* Release one unit of resources r      */
10             ΔR+ = r;
11     Retrain for an iteration;
```

As shown in Algorithm 1, the algorithm monitors the resource change ΔR before each retraining iteration starts (line 2). If available resources increase ($\Delta R > 0$, line 3), the algorithm repeats scaling up operations and allocating one unit of resources r (lines 5-6), until the additional resources are exhausted (line 4). On the other hand, if available resources decrease ($\Delta R < 0$, line 7), the algorithm repeats scaling down operations and releasing one unit of resources r (lines 9-10), until the total released resources are sufficient (line 8). After scaling, ConvertNet continues the current retraining iteration on the scaled compressed model.

Design rationale: This scaling algorithm is accurate and efficient for two reasons.

- *Neuron-grained scaling:* The involved scaling operations make neuron-grained resource-accuracy trade-offs because they are aware of the accuracy contribution per resource consumption of all neurons.
- *Pruned searching space:* Algorithm 1 contains $\Delta R/r$ scaling up/down operations. When TNM has Q nodes, each scaling operation traverses $\log(Q)$ nodes on average. That is, the overall time complexity is $O(\Delta R/r \cdot \log(Q))$, more efficient than brute-force search with time complexity $O(\Delta R/r \cdot Q!)$. With pruned search space, the algorithm only needs to enumerate less than 0.01% of all possible options. Hence the search can be completed within 200ms on commodity vehicles.

D. Accuracy Predictor

The predictor estimates the retraining accuracy (A^{before} and A^{after} in (1)) to calculate the accuracy contribution ($|A^{after} - A^{before}|$) for all neurons. Specifically, the predictor develops a scaling law function f that takes five factors affecting retraining accuracy as inputs, and outputs the predicted accuracy:

$$A = f\left(N^{all}, N^{frozen}, R, \beta, \gamma\right) \tag{2}$$

where N^{all} and N^{frozen} represent the set of all neurons and frozen neurons in the compressed model. R represents the available resources. β and γ are the coefficients representing the characteristics of the current input distribution.

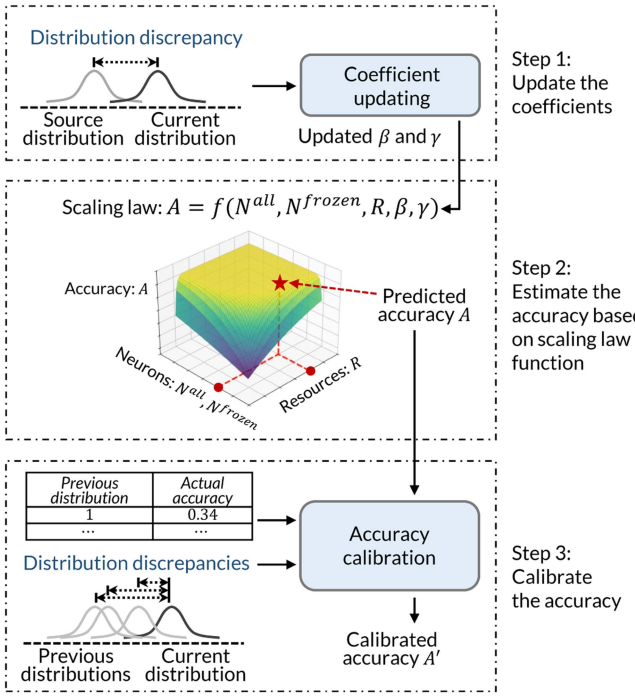

Fig. 6. Estimation process of the accuracy predictor.

Given a scaling option, the predictor has three steps, as shown in Fig. 6. Step 1 updates coefficients β, γ of the scaling law function A according to the current input distribution. Step 2 uses this function to estimate retraining accuracy under the scaling option. Finally, step 3 calibrates the estimated accuracy using the accuracy curves of previous retraining. The prediction is reliable because the predictor utilizes both the *new* distribution's characteristics (in steps 1 and 2) and the *past/historical* recorded accuracies (in step 3).

Step 1: Coefficient updating: To represent the current input distribution, the predictor updates coefficients β and γ in the scaling law function ((2)). This updating is based on the *domain adaptation theory* [52], which proves that the model accuracy on the current distribution is negatively correlated with its two characteristics. That is, (1) the discrepancy between the source distribution used in pre-training and the current distribution used in retraining and (2) the retraining loss on the current distribution. However, both characteristics are sensitive to the high-dimensional input distribution and model parameters, hence it is difficult to use them in the mathematical equation. To address this issue, our predictor employs a tiny three-layer neural network to convert them into two low-dimensional coefficients β and γ, where β and γ will be mapped to the accuracy A in (2).

The tiny neural network is constructed offline. Specifically, ConvertNet emulates different retraining scenarios to collect training samples. Each sample is generated using three steps. The first step creates a simulated new distribution by applying data augmentation techniques to the source distribution [53]. The second step randomly scales the model and retrains it using the simulated distribution. The third step collects relevant information to construct the sample's inputs and output.

Step 2: Scaling law based estimation: Traditionally, scaling law is designed for server-side pre-training, and its function constructs the mapping relationship between a model's training accuracy and three factors (its model size, the amount of training data, and the number of training iterations) [54]. In `ConvertNet`, scaling law is designed for *in-vehicle model retraining*, and its function ((2)) has five factors affecting model accuracy. These factors can be divided into three categories:

- *Model scalable factors:* the set of neurons N^{all} and frozen neurons N^{frozen} imply two scalable factors of the model: (i) the number of neurons in its i-th layer w_i^{all}; (ii) the number of frozen neurons in its i-th layer w_i^{frozen}.
- *Resource factor in retraining:* the available resources R, which corresponds to the number of retraining iterations.
- *Input-related factors:* two updated coefficients β and γ, which are two vectors representing the impact of the current input distribution on accuracy A.

Taking these factors (w_i^{all}, w_i^{frozen}, R, β, and γ) as input, the scaling law function's mathematical expression is defined as follows:

$$A = \prod_{i=1}^{L} \beta_i (w_i^{all})^{\gamma_i} \times \prod_{i=1}^{L} \beta_{i+L} \left(w_i^{frozen}\right)^{\gamma_{i+L}} \times \beta_{1+2L} (R)^{\gamma_{1+2L}}$$

where β_i and γ_i represent the i-th element in the vector β and γ and L is the number of scalable layers. This definition is based on the *quantization model theory of neural scaling* [28]), which proves that both the number of model parameters (i.e. w_i^{all} and w_i^{frozen}) and the available resources R hold a power-law relationship to the model accuracy A.

Step 3: Accuracy calibration: The predicted accuracy in the scaling law is based on the profiled information collected offline and the characteristics of the current input distribution. The predictor hence further calibrates this accuracy using the input distributions and the actual measured accuracies of historical retraining. Specifically, the predictor first calculates the accuracy on the current distribution as a weighted average of previous accuracies, where the weights are the closenesses (reciprocal of discrepancies) of the current distribution to previous distributions. It then averages the calculated accuracy with the scaling law's outputted accuracy to generate the final estimated accuracy.

E. FLOPs Calculator

The calculator computes the compressed model's FLOPs (F^{before} and F^{after} in (1)) to calculate the resource consumption ($|F^{after} - F^{before}|$) for all neurons. It first calculates FLOPs of forward, backward, and weight updating on each layer, and summarizes them as FLOPs of the whole model. Let F^f, F^b, and F^w be the FLOPs of performing one forward pass, backward pass, and weight updating on one individual neuron, respectively.

- *Forward pass:* the compressed model takes a batch of training data as input and sequentially uses model layers from top to bottom to extract features. The forward FLOPs of the i-th layer are proportional to the number of neurons w_i^{all} and are calculated as $F^f \cdot w_i^{all}$.
- *Backward pass:* the compressed model computes the loss value and backpropagates the loss from bottom to top to calculate gradients. The backward FLOPs of the i-th layer are 0 if all neurons in this layer and all preceding layers are frozen; otherwise, the backward FLOPs jump from 0 to $F^b \cdot w_i^{all}$. Therefore, we observe that the optimal scaling down option typically freezes entire layers from top to bottom, rather than individual layers [27] or individual neurons across different layers.
- *Weight updating:* the compressed model updates the weights of all unfrozen neurons. The weight updating FLOPs of the i-th layer are proportional to the number of unfrozen neurons $w_i^{all} - w_i^{frozen}$ and are calculated as $F^w \cdot (w_i^{all} - w_i^{frozen})$.

F. Extension to Multi-Application Scheduling

`ConvertNet` conducts two extensions to maximize the average accuracy of all running retraining jobs.

- *Global TNM:* `ConvertNet` uses neurons of all jobs' original models to construct a global TNM. In the operation of scaling up, `ConvertNet` extracts neurons from the top of the global TNM, and adds them to the corresponding application's compressed model. In the operation of scaling down, `ConvertNet` freezes the neuron with the lowest ΔA *among all jobs*, and adds it into the global TNM.
- *Scheduling algorithm:* `ConvertNet` adds an resource allocation operation in Algorithm 1: (i) after the operation of scaling up (line 5), the algorithm allocates one unit of resources r to the application that is scaled up; (ii) after the operation of scaling down (line 9), the algorithm releases one unit of resources r from the application that is scaled down.

The scheduling is designed for both low memory footprint and computational overhead. First, similar to TNM partition in the single-application scenario (Section III-B), the global TNM is partitioned into sub-heaps and sequentially loaded into memory for scaling operations. Second, the scheduling's search space has a similar size to the scaling algorithm (Algorithm 1). Specifically, the scheduling has a time complexity $O(\Delta R/r \cdot Q \cdot J)$, where J is the number of running jobs. J is typically less than 5in commodity vehicles, and hence the scheduling can be completed within 1 s.

G. Discussion of Scaling Granularity

In this section, we discuss why `ConvertNet` chooses the neuron granularity based on two observations, and discuss the feasibility of applying *hybrid* scaling granularity.

Observation 1: The coarser granularity leads to lower accuracy improvement per unit of resources: Coarse-grained scaling first allocates resources to the model component (e.g. a tensor, a layer, or a block) with the highest average accuracy contribution. However, this component usually contains a proportion of low-contribution model parameters. Taking ViT-B/16 as an example, we observe that the proportion increases as the granularity becomes coarser: 13.59% at the tensor granularity, 49.02% at the layer granularity, and 75.68% at the block granularity.

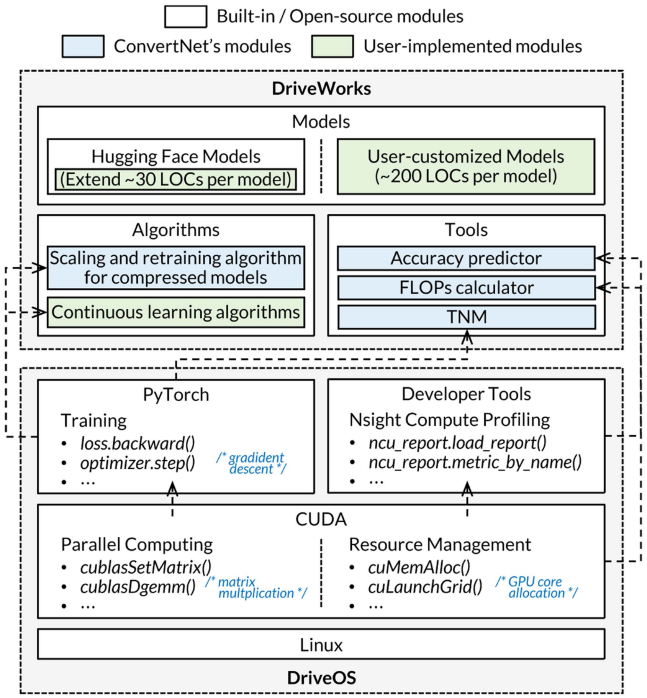

Fig. 7. ConvertNet's implementation in NVIDIA DriveOS [34].

This is because the coarser granularity results in more resources being allocated to low-contribution parameters, and lowers the accuracy improvement per unit of resources.

Observation 2: The finer (parameter) granularity leads to high overheads and low applicability in vehicles: First, the parameter-grained scaling expands the search space of scaling options by a factor of p (p is the number of parameters within one neuron). Taking LLaMA as an example ($4e3 \leq p \leq 2e5$), the finer granularity increases the accuracy prediction time from 200ms to 1.1 hours. Second, its implementation relies on specific hardware and computational libraries [55], which are usually unavailable on in-vehicle devices.

Feasibility of hybrid granularity: ConvertNet can support hybrid-grained scaling to achieve higher scaling efficiency. For instance, when the available resource is considerably reduced, ConvertNet can first perform layer-grained scaling (through a TNM that stores layers), thereby quickly releasing substantial resources. Once the remaining resource to be released is less than one layer's resource consumption, ConvertNet switches to neuron-grained scaling to further refine the accuracy-resource trade-off.

IV. System Implementation

We implement ConvertNet in Python 3.8.5 with 3 k LOCs, and integrate it into NVIDIA DriveOS [34], an operating system dedicated to autonomous driving and AI-powered cockpit experiences. NVIDIA DriveOS is gradually being adopted in the latest autonomous vehicles, including those from manufacturers such as Tesla, Toyota, and BYD. As shown in Fig. 7, ConvertNet builds three components upon NVIDIA DriveOS's APIs, and allows developers to integrate new models and learning algorithms.

Integration of new models and learning algorithms: Integrating a model into ConvertNet requires implementing three groups of APIs: (1) inference on input samples, (2) testing accuracy on a given dataset, and (3) manipulating the model structure (e.g. modifying/replacing a specific network layer/neuron). ConvertNet supports two types of transformers.

- *Hugging Face models [56]:* ConvertNet implements the aforementioned APIs using Hugging Face's built-in APIs. For example, ConvertNet uses AutoModel.forward() for inference on input samples, and uses the package evaluate for testing accuracy. Developers only need to add around 30 LOCs to integrate a Hugging Face model into ConvertNet.
- *User-customized models:* ConvertNet develops an abstract class ModelManager to decouple the internal modules and integrated models of ConvertNet. To integrate a user-customized model, developers need to write around 200 LOCs to fully implement three groups of APIs in ModelManager.

Moreover, the new learning algorithm can also be integrated into ConvertNet, such as the unsupervised domain adaptation techniques [57] and supervised continual learning techniques [58]. The integration requires implementing two APIs using 50 to 200 LOCs: (1) fetching training samples from the current input distribution, and (2) performing forward operation and calculating loss function.

V. Evaluation

Testbeds: We choose four mainstream in-vehicle GPU devices [59] with different architectures: (1) NVIDIA Xavier NX with 16 GB memory and 384-core Volta GPU; (2) NVIDIA AGX Xavier with 32 GB memory and 512-core Volta GPU; (3) NVIDIA AGX Orin with 32 GB memory and 1792-core Ampere GPU; and (4) NVIDIA AGX Orin with 64 GB memory and 2048-core Ampere GPU. They are equipped with Ubuntu 18.04.5, CUDA 10.2, and Python 3.8.5.

Workloads: We choose six AI applications which target the most important continuous learning tasks on vehicles, and construct six distinct workloads based on six Hugging Face models. The details are listed in Table I.

- *Traffic sign recognition* takes an image as input and recognizes the category of its containing traffic sign (e.g. stop, pedestrian crossing, and turn left).
- *Traffic object detection* takes a video frame as input and identifies the category and location of its containing traffic objects (e.g. cars and pedestrians).
- *Driver inattention monitoring* takes a surveillance video clip as input and recognizes the category of the driver inattention behavior (e.g. using phones and feeling sleepy).
- *Speech recognition* takes a voice clip as input and recognizes the corresponding text.
- *Task-oriented dialog* takes a textual question as input and outputs the corresponding answer.

TABLE I
SUMMARY OF APPLICATIONS AND WORKLOADS

Application	Input distribution in pre-training	Input distribution in retraining	Original model	#Model parameters
Traffic sign recognition	SYNSIGNS [36]	GTSRB [37] with corruptions [60]	ViT-B/16 [31]	0.09B
Traffic object detection	MSCOCO [61]	GTA5 [5] CityScpaes [4]	GLIP-T [32]	1.8B
Driver inattention monitoring	NoIR [39]	Drive&Act [38]	VideoMAE-B [6]	0.10B
Speech Recognition	LibriSpeech [40]	LibriSpeech with different speakers/volumes/ speeds/noises [40]	Conformer-L [7]	1.2B
Task-oriented Dialog	MultiWoz-MultiDomains [41]	MultiWoz-Taxi MultiWoz-Hotel [41]	LLaMA [8]	3B
Autonomous Driving	CARLA Town01 [42]	CARLA Town02-07 [42]	Transfuser [8]	0.20B

- *Autonomous driving* takes a video frame and LiDAR signal as input and outputs the values of the vehicle's steer, throttle, and brake.

Accuracy metrics: The six workloads listed above use top-1 classification accuracy, mAP@0.5 (Mean Average Precision over Intersection over Union (IoU) threshold 0.5), top-1 classification accuracy, WER (Word Error Rate), BLEU (Bilingual Evaluation Understudy), and IS (Infraction Score) as the accuracy metrics, respectively.

A. Resource-Accuracy Trade-Offs in In-Vehicle Continuous Learning

In this evaluation, we implement and compare `ConvertNet` with two types of eleven retraining techniques on each workload.

- *Static compression*: it keeps the network architecture fixed during retraining, including four types of techniques: (1) *PEFT* (Parameter-Efficient Fine-Tuning) adds a few extra parameters to each model layer and then only retrains them (LoRA (Low-Rank Adaptation) [62], DoRA (Decomposed Low-Rank Adaptation) [63], and VeRA (Vector-based Low-Rank Adaptation)) [64]; (2) *Compressed model* techniques pre-compress the original model on the source distribution and then retrain it on the current distribution (JIT (Just-In-Time Adaptation) [45], AMS (Adaptive Model Streaming) [46], and EdgeMA (Edge Model Adaptation) [47]); (3) *Model zoo* techniques maintains a zoo of historical models, and selects the best model to retrain (RECL (Responsive Edge Continual Learning) [26]); (4) *Before-retraining scaling* techniques scale the model before retraining begins based on current input distribution and available resources, but maintain the model fixed throughout the retraining (ElasticDNN [21] and EdgeTA (Edge Transformer Adaptation) [22]).
- *Adaptive retraining*: *ElasticTrainer* [27] and *PruneTrain* [30] scale down the model upon a decrease in available resources, but keep the model fixed upon an increase in available resources.

Pre-training original model: For each workload, we pre-train the original model using the source distribution offline. We conduct the pre-training on an RTX 8000 graphics card. The pre-training time of baseline techniques is 3.77 to 48.12 hours, and the time of `ConvertNet` is 8.53 to 83.12 hours, because it needs extra time to construct scalable layers and neuron indexes. The pre-training is executed once.

Online retraining: We use EdgeVisionBench [65] to construct on-device continuous learning scenarios. This benchmark takes each workload as input, constructs 30 new distributions accordingly, and searches hyperparameters for each evaluated technique. We set the retraining window to 600 s for all workloads and techniques. During each retraining, we randomly trigger three changes of available resources. We evaluate the traffic sign recognition workload on NVIDIA Xavier NX, traffic object detection and driver inattention monitoring workloads on NVIDIA AGX Xavier, speech recognition workload on NVIDIA AGX Orin (32 GB), and task-oriented dialog workload on NVIDIA AGX Orin (64 GB). The evaluation considers two settings:

Comparison under the same resource budget: Fig. 8 demonstrates the retraining accuracies of all techniques over evolving distributions. Specifically, Fig. 8(a) demonstrates detailed accuracy curves over all 30 distributions, and other subfigures summarize accuracy curves by box plots. We have two observations from the results. First, `ConvertNet` consistently achieves the highest accuracy in all workloads, because its TNM enables scaling of compressed models by fully utilizing the original model's large learning capacity. As shown in Fig. 8(a), `ConvertNet` promptly improves accuracy when more resources become available, as the scaled model receives additional knowledge from TNM. Conversely, when resources are reduced, `ConvertNet` preserves accuracy by storing lost knowledge into TNM. Second, `ConvertNet` tends to bring higher accuracy improvement after seeing several distributions (e.g. Fig. 8(a)). This is because its TNM accumulates the newly learned knowledge in the original model, and this knowledge is transferred to the compressed model in future retraining. In contrast, baseline techniques have low accuracies for two reasons: (1) the static compression techniques hinder fine-grained allocation of resources, and (2) the existing adaptive retraining techniques fail to store the lost knowledge during scaling down. For example, PruneTrain directly discards pruned neurons and thus has the lowest accuracy. Overall, `ConvertNet` improves the retraining accuracy by 19.37% over static compression techniques, 25.23% over adaptive retraining techniques, and 22.33% over all baselines.

Discussion of training windows: Following the setting of the previous evaluation, we vary the retraining windows from 600 s to 0 s (no retraining) and test their impact on model accuracy. As shown in Fig. 9, `ConvertNet` still consistently achieves the highest accuracy, even without any retraining. This proves the efficacy of `ConvertNet`'s TNM in selecting the most accuracy-related neurons in each scaling. In summary, `ConvertNet` brings an average of 18.85% accuracy improvement with tighter resource budgets.

Comparison under the same model accuracy: In this evaluation, both `ConvertNet` and baselines perform the same training until convergence. Table II lists the evaluation results. We can see that compared to all baseline techniques, `ConvertNet` saves memory footprint, training latency, retraining

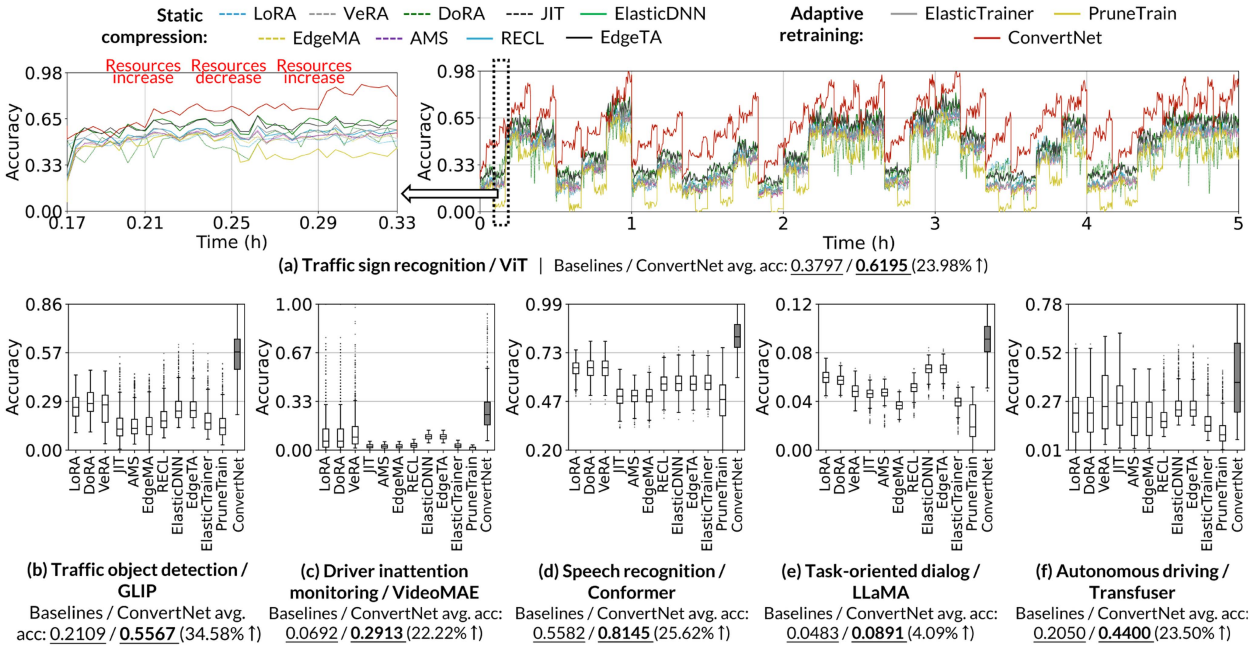

Fig. 8. Accuracy comparison over the evolving distributions under the same resource budget.

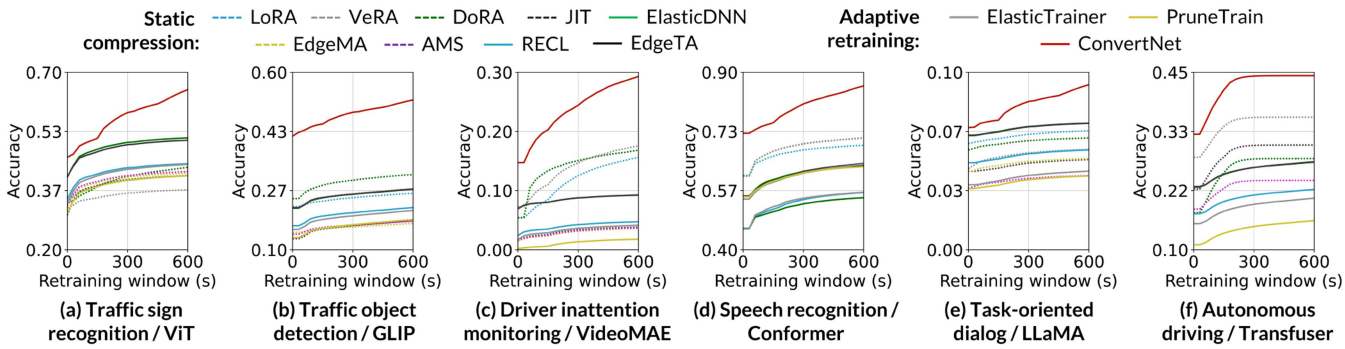

Fig. 9. Comparison of learning accuracy under different retraining windows.

time, and energy consumption by 1.63x, 1.81x, 2.69x, and 3.65x, respectively, because its training converges the fastest.

ConvertNet's response time under resource fluctuations: To respond to a resource fluctuation, ConvertNet conducts up to five operations to scale the model: *TNM generation*, *accuracy prediction*, *FLOPs calculation*, *scaling option selection*, and *scaling*. In this evaluation, we test six workloads and list these five operations' wall-clock time, as shown in Fig. 11. We can see that in all workloads, these operations take less than 12 s (2% of retraining window) to complete. Moreover, subsequent scaling only requires the last two operations, which take less than 3 s to complete. The result indicates that ConvertNet responds promptly to resource fluctuations.

Breakdown of ConvertNet's extra memory overhead: ConvertNet's extra memory footprint comes from three modules: TNM, the accuracy predictor, and the FLOPs calculator. Our evaluation results on all workloads show: (1) TNM contributes 99.7% to 99.8% of the memory footprint among the three modules. This is because TNM stores the original model's neurons, while the other two modules only involve a tiny prediction network (<0.5 MB) and basic mathematical operations. (2) TNM's memory footprint remains low (ranging from 80 MB to 83 MB, only occupying 2% of the overall retraining memory footprint). This is because ConvertNet partitions the whole TNM into small sub-heaps for sequential loading, that is, only one small sub-heap is loaded in the memory at a time.

B. Multi-Application Scheduling

In this evaluation, we run multiple workloads concurrently on NVIDIA AGX Orin (64 GB), and compare the average accuracies of ConvertNet with two types of nine on-device schedulers.

- *Static compression*: it keeps the model fixed and allocates resources by adjusting the number of training iterations (EOMU (Edge-assisted On-device Model Update) [24], AdaEvo (Adaptive Evolution of Models) [25], AdaInf (Adaptive Inference) [23], and RECL [26]), batch size

TABLE II
COMPARISON OF RETRAINING OVERHEADS UNDER THE SAME MODEL ACCURACY. "(A)" TO "(F)" REPRESENT SIX WORKLOADS, RESPECTIVELY

	Memory footprint (GB)						Retraining time (minute)						Energy consumption (J)					
	(a)	(b)	(c)	(d)	(e)	(f)	(a)	(b)	(c)	(d)	(e)	(f)	(a)	(b)	(c)	(d)	(e)	(f)
LoRA	8.2	14.8	13.2	30.7	30.7	13.2	3.7	27.3	13.8	15.1	24.2	7.2	2726	22908	11062	13206	21231	5273
DoRA	10.6	19.7	14.3	32.6	31.7	14.1	4.6	36.3	17.8	15.9	25.1	7.9	3453	30450	14194	13935	21961	5766
VeRA	8.0	14.8	13.2	30.7	30.7	13.2	3.8	27.3	13.8	15.1	24.2	7.2	2800	22908	11062	13206	21231	5387
JIT	5.8	13.0	10.0	13.0	19.8	7.8	1.9	18.0	13.3	7.5	13.4	3.9	879	9428	6664	4108	7342	1638
AMS	5.9	13.0	10.1	13.4	20.2	7.9	2.0	17.9	13.5	7.4	13.5	4.0	929	9385	6747	4071	7387	1709
EdgeMA	5.9	13.0	10.1	13.0	19.8	7.8	1.9	18.1	13.4	7.5	13.4	3.9	899	9472	6706	4108	7342	1644
RECL	5.9	13.0	10.1	13.0	19.8	7.8	2.9	18.9	14.4	8.5	14.4	4.5	1369	9913	7210	4660	7893	2390
ElasticDNN	5.7	12.9	9.7	12.9	19.8	7.6	1.7	17.3	13.4	7.4	12.8	3.4	792	9061	6697	4049	7004	1866
EdgeTA	5.7	12.9	9.7	12.9	19.8	7.6	1.7	17.3	13.4	7.4	12.8	3.4	792	9061	6697	4049	7004	1866
ElasticTrainer	5.5	12.7	9.6	12.5	18.7	7.1	1.9	17.9	13.9	7.8	13.3	4.2	885	9376	6947	4268	7277	2305
PruneTrain	5.6	12.6	9.5	12.5	18.1	7.1	2.9	20.6	16.0	9.4	15.1	7.8	1351	10790	7996	5143	8262	4282
`ConvertNet`	**4.5**	**10.5**	**8.7**	**10.5**	**13.3**	**5.4**	**1.2**	**14.5**	**2.3**	**5.6**	**11.6**	**2.4**	**561**	**7595**	**1126**	**3040**	**6338**	**989**

	Training latency (second per iteration)					
	(a)	(b)	(c)	(d)	(e)	(f)
LoRA	0.44	3.28	1.66	1.81	2.91	0.77
DoRA	0.56	4.36	2.13	1.91	3.01	0.97
VeRA	0.45	3.28	1.66	1.81	2.91	0.78
JIT	0.23	2.16	1.60	0.90	1.61	0.40
AMS	0.24	2.15	1.62	0.89	1.62	0.46
EdgeMA	0.23	2.17	1.61	0.90	1.61	0.44
RECL	0.35	2.27	1.73	1.02	1.73	0.61
ElasticDNN	0.22	2.11	1.56	0.76	1.44	0.41
EdgeTA	0.22	2.11	1.56	0.76	1.44	0.38
ElasticTrainer	0.23	2.38	1.79	1.31	2.01	0.40
PruneTrain	0.21	1.99	1.43	1.23	1.41	0.37
`ConvertNet`	**0.14**	**1.74**	**0.92**	**0.67**	**1.39**	**0.25**

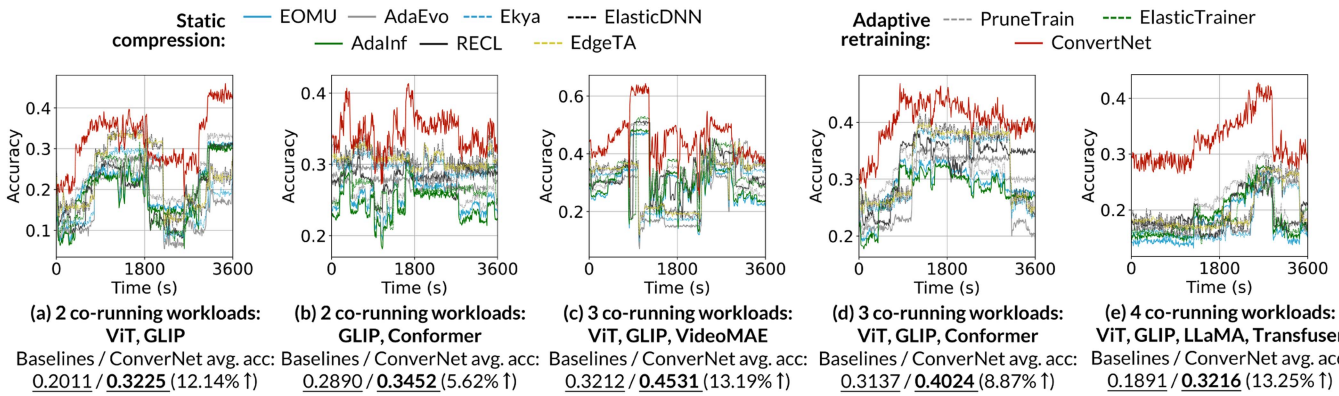

Fig. 10. Comparison of average accuracy across all retraining jobs in multi-application scenarios.

(Ekya [15]), or model size and GPU cycle allocation (ElasticDNN [21] and EdgeTA [22]).

- *Adaptive retraining*: ElasticTrainer [27] and PruneTrain [30] uniformly scale down all jobs' models when available resources decrease.

Comparison of accuracy: We first develop a benchmark that co-runs a mix of retraining jobs. In an evaluation whose duration is one hour, the benchmark picks 10 random time points, and launches a randomly selected retraining job at each of these points. The retraining window for each job is 600 seconds. We create five different mixes of retraining jobs, and report the average accuracy across all jobs in Fig. 10.

We can see `ConvertNet` consistently brings the highest overall accuracy for three reasons: (1) `ConvertNet`'s global TNM enables scaling of compressed models in multiple concurrent applications; (2) `ConvertNet`'s scheduling algorithm concentrates resources on the neurons bringing highest accuracy improvement among all applications; (3) `ConvertNet` can solve the best schedule in less than one second, allowing it to quickly respond to dynamic events. In contrast, baseline

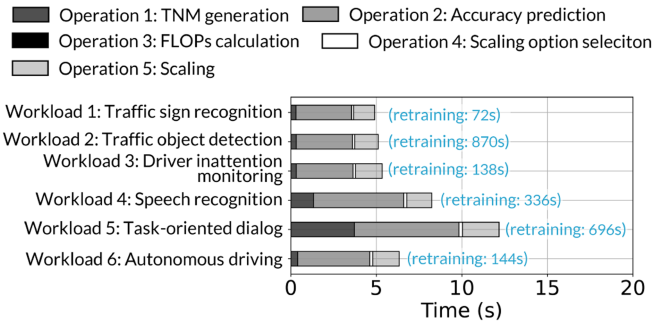

Fig. 11. Breakdown of ConvertNet's response time under resource fluctuations.

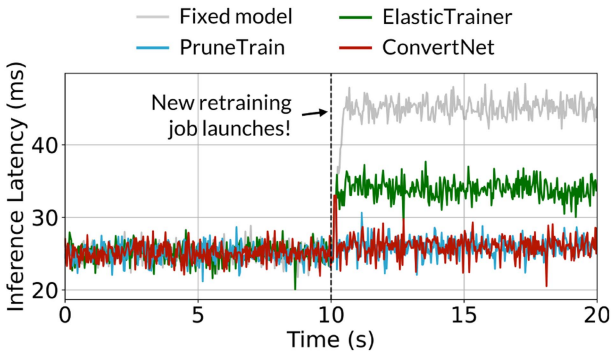

Fig. 12. Comparison of stability to resource contention.

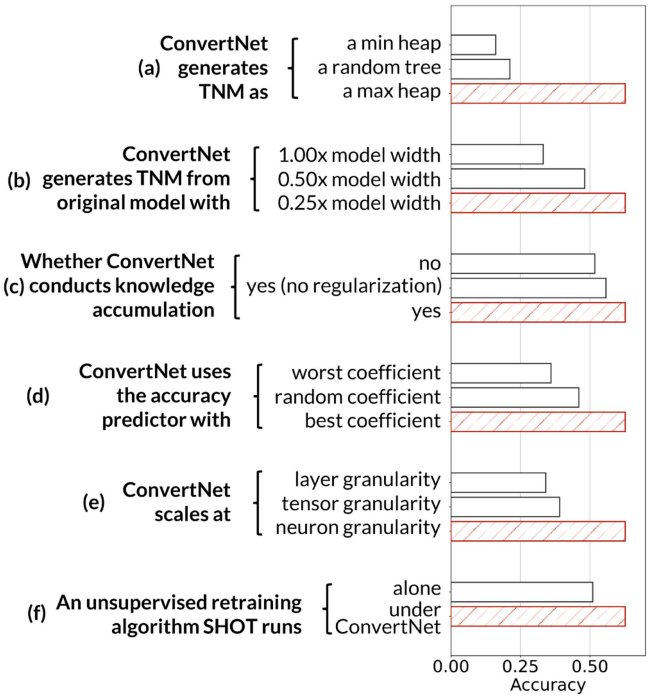

Fig. 13. Model accuracies under different design choices.

schedulers achieve low accuracy for two reasons: the static compression schedulers hinder the fine-grained resource allocation, and the adaptive retraining schedulers uniformly allocate newly available resources among all jobs. In particular, Ekya and AdaEvo improve accuracy slower than other schedulers, because they perform expensive online trial runs. Overall, ConvertNet improves the accuracy by 9.12% than static compression schedulers, 12.31% than adaptive retraining schedulers, and 10.61% than all baseline schedulers.

Comparison of stability to resource contention: In this evaluation, the stability to resource contention is defined as the *latency fluctuation* of high-priority jobs (that is, inference jobs) when a new retraining job is launched. This metric is critical because low stability reduces human–vehicle collaboration efficiency and even compromises driving safety.

To measure stability, we follow Fig. 10(a)'s setting and conduct the following three steps: (1) at 0 s, we launch an inference job (with 20% resources) and a retraining job of ViT (with 80% resources); (2) at 10 s, we launch a new retraining job of GLIP; (3) at 20 s, we visualize the inference job's latency between 0-20 s in Fig. 12. We can see that with available resource fluctuation, the inference latency remains low with ConvertNet but increases suddenly for most baseline schedulers. This is because ConvertNet can promptly release resources by scaling down the model in retraining and hence avoid preempting resources from inference jobs. In contrast, the schedulers based on fixed models do not release resources, and ElasticTrainer can only release much less resources by freezing model parameters.

ConvertNet's extra memory overhead: In Fig. 10's evaluation (including 1 to 4 co-running workloads), ConvertNet's extra memory footprint remains consistently low (ranging from 81 MB to 86 MB). This is because ConvertNet partitions the global TNM into sub-heaps for sequential loading, and the sub-heaps keep small regardless of the number of applications.

C. Design Choice Validation by Ablation

This evaluation tests the traffic sign recognition workload, and reports the retraining accuracy under the same resource budget (Fig. 13(a)–(f)).

TNM construction: Fig. 13(a) shows that constructing TNM as a max heap achieves much higher accuracy (33.31% in average) than a min heap or random tree. This indicates that the neurons with higher accuracy contributions are indeed crucial for scaling of compressed models.

Original model width. Fig. 13(b) shows that generating TNM from the 1.00x width model results in 22.02% higher accuracy compared to those with smaller width. This is because that a larger original model provides more neurons in the TNM, thus improving the upper bound of the compressed model's accuracy.

Knowledge accumulation: Fig. 13(c) shows that accumulating learned knowledge with regularization improves accuracy by 6.92% compared with accumulation without regularization, and by 10.92% compared with no accumulation at all. Fig. 14 further displays the retraining loss curve when ConvertNet encounters the same distribution "MotionBlur" for the first, second, and third time. We can see that the accumulated knowledge indeed accelerates the retraining convergence.

Accuracy prediction on unseen data: Fig. 17 demonstrates the estimation error of ConvertNet's accuracy predictor. Note

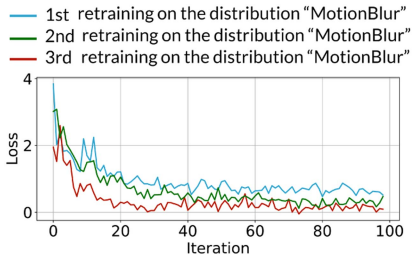

Fig. 14. Loss when ConvertNet retrains on the same distribution multiple times.

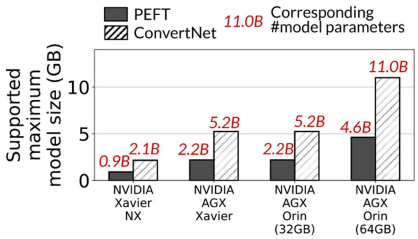

Fig. 15. Maximum model size in PEFT and ConvertNet.

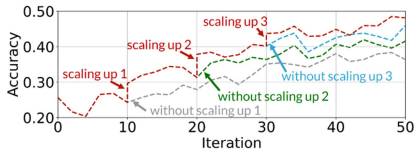

Fig. 16. Accuracy curves with no scaling up (grey line), one scaling up (green line), two times (blue line), and three times (red line).

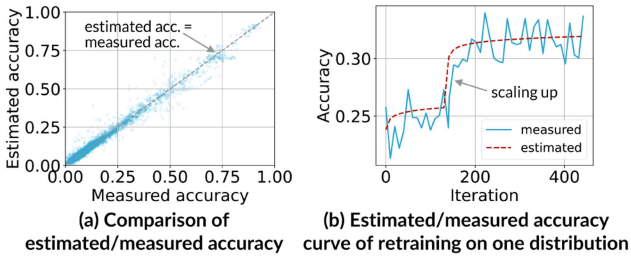

Fig. 17. Precision of ConvertNet's accuracy predictor.

that the error is measured on GTSRB [37], an unseen dataset during offline construction. We can see the predictor can generalize to unseen data and precisely estimate the retraining accuracy: the absolute/relative estimation error is 0.12% and 3.73% in average. These precise estimations improve the retraining accuracy by 10.75% compared to the worst/random predictor, as shown in Fig. 13(d).

Neuron-grained scaling: The ablation study in Fig. 13(e) shows that ConvertNet's neuron-grained scaling improves the accuracy by 24.12% compared to layer/tensor-grained scaling, because it makes resource-accuracy trade-offs at a finer granularity and efficiently utilizes limited resources in the optimal set of neurons.

Effectiveness of scaling: During retraining on the distribution "MotionBlur", we compare the accuracy curves of scaling up the compressed model for 0 (no scaling) to 3 times. The results in Fig. 16 show that each scaling up can immediately improve the retraining accuracy and accelerate the convergence.

Integration with other unsupervised learning algorithms: We choose a classical unsupervised learning algorithm SHOT [57], and the result in Fig. 13(f) shows that ConvertNet improves SHOT's accuracy by 9.62% with its TNM and scaling algorithm.

TABLE III
COMPARISON WITH INFERENCE-TIME SCALING TECHNIQUES

	V-MoE	DejaVu	ConvertNet
Accuracy	0.5123	0.5388	**0.6195**
Memory (GB)	29.7	10.3	**6.1**

D. Discussions and Limitations

Discussions of alternative adaptive retraining techniques: Following Fig. 8's evaluation setting (training ViT on NVIDIA Xavier NX using 600 seconds), we compare ConvertNet with two other adaptive retraining techniques: batch size reduction and quantization [66]. The results show these techniques result in 14.3% lower accuracy than ConvertNet. This is because training large models with small batch size (batch size is reduced by 75%) or low precision (decreased from 32 b to 8 bits) still has slower convergence speed than our approach.

Maximum model size supported by ConvertNet. Using LLaMA as an example, Fig. 15 lists the maximum model sizes supported by PEFT and ConvertNet on various in-vehicle devices. PEFT supports the largest model size among all baseline techniques. ConvertNet can support model size of up to 25 GB (11B model parameters, 2.4x more than PEFT). This is because ConvertNet updates the original model via neuron indexes rather than performing expensive training. Hence its supported maximum model size depends on the device's maximum available memory and the model's maximum compression ratio (e.g. 1.56% for LLaMA).

Discussion of hybrid scaling granularity: Following Fig. 8's evaluation setting (that is, training ViT on NVIDIA Xavier NX using 600 seconds), we compare ConvertNet at the neuron granularity and at the hybrid granularity (layer and neuron), respectively. The result shows that the hybrid granularity indeed accelerates the scaling by 8.91x with similar accuracies.

Discussions of inference-time scaling techniques: Recent mixture-of-experts (MoE) or dynamic routing architectures are typically designed for the *stationary* distribution at the inference phase. These techniques activate a specific subset of experts/neurons for *each input sample*. In contrast, ConvertNet is designed for *evolving* distributions at the retraining phase. Our approach activates a specific subset of neurons for *each new distribution*.

Directly applying MoE or dynamic routing techniques suffers from low accuracy and high overhead in retraining. To demonstrate this, we select a representative MoE (V-MoE [67]) and dynamic routing architecture (DejaVu [68]), test it following Fig. 8's evaluation setting (retraining on NVIDIA Xavier NX with retraining window 600 s), and compare it with ConvertNet in terms of accuracy and memory footprint. Table III's result

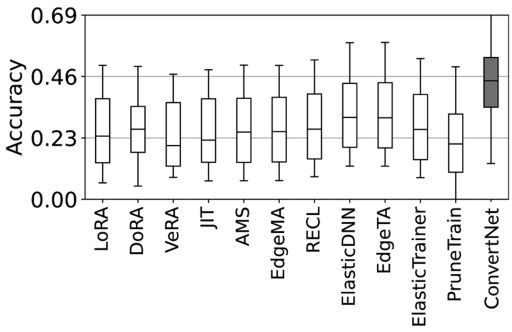

Fig. 18. Accuracy comparison on the non-vehicle workload.

shows: (1) ConvertNet improves the accuracy by 9.40%, because MoE techniques suffer from catastrophic forgetting over evolving distributions. (2) ConvertNet reduces the memory footprint by 3.28x. This is because MoE techniques may activate different experts/neurons at different training iterations, thus maintaining all neurons in the memory during retraining.

The accuracy predictor's sensitivity to hyperparameters: Using the traffic sign recognition workload as an example, we evaluate the predictor's estimation error under different hyperparameters, including learning rates (1e-3 and 1e-4), network widths (64 and 128), and depths (3, 4, and 5). The results show that the relative error consistently stays within 3.01%–5.14%. This indicates that the predictor is insensitive to the hyperparameters, owing to its simple architecture and solid theoretical underpinning.

Generalization to non-vehicle workloads: We evaluate ConvertNet on a new generic/non-vehicle image classification workload: a ViT-B/16 model is pre-trained on the ImageNet dataset [69] and retrained on the ImageNet-C dataset [70]. ImageNet-C includes 15 common distribution shifts such as noise, blur, and weather changes. The retraining runs on NVIDIA Xavier NX for 600 s. The results in Fig. 18 show that ConvertNet improves accuracy by 25.14% over the baselines. This is because ConvertNet operates on neurons, which are universal units in deep neural networks and achieve consistently high accuracy in both vehicle and non-vehicle scenarios.

Integration with other retraining techniques: ConvertNet can scale the model being retrained by various techniques such as meta learning [71], federated learning [72], and distillation [73]. To achieve the above objectives, ConvertNet initializes TNM using the meta model, the model after server-side aggregation, and the teacher model for meta learning, federated learning, and distillation, respectively.

We compare the accuracies of these techniques with and without ConvertNet. Follows Fig. 8's evaluation setting (runs on NVIDIA Xavier NX under the retraining window 600 s), the comparison uses three standard evaluation benchmarks, as listed in Table IV. The results in Table IV demonstrate that ConvertNet can improve the accuracies of these retraining techniques by an average of 7.88%. This is because ConvertNet achieves a better utilization of available resources during retraining.

TABLE IV
INTEGRATION WITH OTHER RETRAINING TECHNIQUES

Workload	1	2	3
Retraining technique	Meta learning (Reptile [71])	Federated learning (FedKNOW [72])	Distillation (LDRLD [73])
Input distribution in pre-training	first 1000 classes in Omniglot [74]	first 10 classes in MiniImageNet [69]	ImageNet [69]
Input distribution in re-training	classes 1000-1500 in Omniglot [74]	classes 10-50 in MiniImageNet [69]	ImageNet-C [70]
Model	ViT-B/16	ViT-B/16	ViT-B/16
Accuracy (run alone)	0.6013	0.7231	0.5498
Accuracy (run with ConvertNet)	**0.6759**	**0.8113**	**0.6233**

TABLE V
SUMMARY OF EXISTING MODEL SCALING TECHNIQUES

	Resource consumption	Scaling time	Network architecture	Proportion of updated model weights
Pre-training	high	hours to days	changeable	all
Fine-tuning	high	seconds to days	changeable	all
Inference	low	seconds	changeable	none
Retraining (baselines)	low	seconds	fixed	partial
Retraining (ConvertNet)	low	seconds	changeable	partial to all

VI. RELATED WORK

Model scaling has been extensively studied in four stages of a transformer-based model's life cycle: pre-training, fine-tuning, inference, and retraining. In this work, model scaling during retraining has two requirements: (i) it should be executed under tight resource budgets and short retraining windows; (ii) it should complete quickly (e.g. using several seconds) to promptly handle each resource fluctuation. As listed in Table V, we summarize existing model scaling techniques in four stages, and discuss their limitations in meeting the above two requirements.

Model scaling in pre-training: It has been explored mainly through two lines of work. First, Neural Architecture Search (NAS) [75], [76], [77], [78], [79], [80], [81], [82] scales a model to multiple sizes, pre-trains each variant, and searches for the most suitable one for deployment. Second, PruneTrain [30] progressively prunes the model to accelerate pre-training. Both of them are typically conducted on servers and consume substantial resources and prolonged time (several hours [81], [82] to days [75], [76], [77], [78], [79], [80]). In contrast, ConvertNet achieves higher accuracy improvement *per unit of resources* by leveraging the original model's large learning capacity using TNM.

Model scaling in fine-tuning: Given a pre-trained original model, existing techniques typically compress (i.e. scale down)

and fine-tune it for a specific downstream application [15], [83], [84], [85]. Mainstream compression techniques include pruning [83], [86], quantization [66], and knowledge distillation [84], [85]. These techniques require loading the original model into memory, and thus may exceed the memory limit of in-vehicle devices.

Model scaling during inference: Extensive model scaling techniques have been proposed to make resource-accuracy trade-offs during inference [68], [87], [88], [89], [90], [91], [92], [93], [94], [95]. Some of them prepare multiple fixed models of different sizes, and select one of them according to the available resources at run-time [88], [89], [90], [91]. The others construct a model that can dynamically adjust its size to match the resource budget [68], [87], [92], [93], [94], [95]. These techniques use models pre-trained on a stationary input distribution and maintain model weights unchanged during inference. As a result, they may encounter a large accuracy drop in in-vehicle scenarios where the input distribution evolves. In contrast, `ConvertNet` supports updating model weights on evolving distributions, and dynamically select high-contribution neurons/weights for each new distributions.

Adaptive retraining techniques [27] make accuracy-resource trade-off during retraining by freezing/unfreezing model parameters according to available resources. However, they still rely on the fixed network architecture of compressed models, which have lower learning capacity than original models, thus resulting in lower accuracies when encountering new input distributions.

VII. Conclusion

Motivated by frequent resource fluctuations during in-vehicle continuous learning, we develop a compressed model scaling and retraining system, `ConvertNet`, which best scales up/down the model upon the increase/decrease in available resources. The key novel component of `ConvertNet` is a training-time neuron memorizer (TNM) that exchanges rich knowledge with the compressed model to scale it. Based on TNM, an accuracy predictor and FLOPs calculator are proposed to find the optimal scaling option that maximizes accuracy per unit of resources. `ConvertNet` achieves up to 34.58% (22.33% in average) accuracy improvement on six in-vehicle AI scenarios, compared to state-of-the-art baselines in retraining, pruning, architecture searching, and scheduling.

References

[1] Z. Shen et al., "The emerging intelligent vehicles and intelligent vehicle carriers collaborative systems," in *Proc. IEEE Intell. Veh. Symp.*, 2024, pp. 639–644.

[2] S. Chen et al., "VADV2: End-to-end vectorized autonomous driving via probabilistic planning," 2024, *arXiv:2402.13243*.

[3] X. Tian et al., "DriveVLM: The convergence of autonomous driving and large vision-language models," in *Proc. 8th Conf. Robot Learn.*, 2025, vol. 270, pp. 4698–4726.

[4] M. Cordts et al., "The cityscapes dataset for semantic urban scene understanding," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 3213–3223.

[5] S. R. Richter, V. Vineet, S. Roth, and V. Koltun, "Playing for data: Ground truth from computer games," in *Proc. Eur. Conf. Comput. Vis.*, 2016, vol. 9906, pp. 102–118.

[6] Z. Tong, Y. Song, J. Wang, and L. Wang, "VideoMAE: Masked autoencoders are data-efficient learners for self-supervised video pre-training," in *Proc. Adv. Neural Inf. Process. Syst.*, 2022, pp. 10078–10093.

[7] A. Gulati et al., "Conformer: Convolution-augmented transformer for speech recognition," in *Proc. Interspeech'20*, 2020, pp. 5036–5040.

[8] H. Touvron et al., "LLAMA: Open and efficient foundation language models," vol. 10, 2023, *arXiv:2302.13971*.

[9] D. Fu et al., "LimSim++: A closed-loop platform for deploying multimodal LLMs in autonomous driving," in *Proc. IEEE Intell. Veh. Symp.*, 2024, pp. 1084–1090.

[10] M. Aldeen, P. MohajerAnsari, J. Ma, M. Chowdhury, L. Cheng, and M. D. Pesé, "An initial exploration of employing large multimodal models in defending against autonomous vehicles attacks," in *Proc. IEEE Intell. Veh. Symp.*, 2024, pp. 3334–3341.

[11] S. Han, H. Shen, M. Philipose, S. Agarwal, A. Wolman, and A. Krishnamurthy, "MCDNN: An approximation-based execution framework for deep stream processing under resource constraints," in *Proc. 14th Annu. Int. Conf. Mobile Syst., Appl. Serv.*, 2016, pp. 123–136.

[12] D. Kang, J. Emmons, F. Abuzaid, P. Bailis, and M. Zaharia, "No-scope: Optimizing neural network queries over video at scale," *VLDB'17*, vol. 10, no. 11, pp. 1586–1597, 2017.

[13] "Microsoft rocket for live video analytics," 2021. [Online]. Available: https://www.microsoft.com/en-us/research/project/live-video-analytics/

[14] "Custom on-device ML models withlearn2compress," 2018. [Online]. Available: https://research.google/blog/custom-on-device-ml-models-with-learn2compress/

[15] R. Bhardwaj et al., "Ekya: Continuous learning of video analytics models on edge compute servers," in *Proc. 19th USENIX Symp. Networked Syst. Des. Implementation*, 2022, pp. 119–135.

[16] I. Gim and J. Ko, "Memory-efficient DNN training on mobile devices," in *Proc. 20th Annu. Int. Conf. Mobile Syst., Appl. Serv.*, 2022, pp. 464–476.

[17] J. Lin, L. Zhu, W. Chen, W. Wang, C. Gan, and S. Han, "On-device training under 256 KB memory," in *Proc. Adv. Neural Inf. Process. Syst.*, 2022, pp. 22941–22954.

[18] Y. Zhang, T. Gu, and X. Zhang, "MDLdroidLite: A release-and-inhibit control approach to resource-efficient deep neural networks on mobile devices," *IEEE Trans. Mobile Comput.*, vol. 21, no. 10, pp. 3670–3686, Oct. 2022.

[19] S. Khurana, N. Moritz, T. Hori, and J. L. Roux, "Unsupervised domain adaptation for speech recognition via uncertainty driven self-training," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, 2021, pp. 6553–6557.

[20] K. Chitta, A. Prakash, B. Jaeger, Z. Yu, K. Renz, and A. Geiger, "TransFuser: Imitation with transformer-based sensor fusion for autonomous driving," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 45, no. 11, pp. 12878–12895, Nov. 2023.

[21] Q. Zhang, R. Han, C. H. Liu, G. Wang, and L. Y. Chen, "ElasticDNN: On-device neural network remodeling for adapting evolving vision domains at edge," *IEEE Trans. Comput.*, vol. 73, no. 6, pp. 1616–1630, Jun. 2024.

[22] Q. Zhang, R. Han, C. H. Liu, G. Wang, S. Guo, and L. Y. Chen, "EdgeTA: Neuron-grained scaling of foundation models in edge-side retraining," *IEEE Trans. Mobile Comput.*, vol. 24, no. 4, pp. 2690–2707, Apr. 2025.

[23] S. S. Shubha and H. Shen, "AdaInf: Data drift adaptive scheduling for accurate and SLO-guaranteed multiple-model inference serving at edge servers," in *Proc. ACM SIGCOMM 2023 Conf.*, 2023, pp. 473–485.

[24] Y. Kong, P. Yang, and Y. Cheng, "Edge-assisted on-device model update for video analytics in adverse environments," in *Proc. 31st ACM Int. Conf. Multimedia*, 2023, pp. 9051–9060.

[25] L. Wang et al., "AdaEvo: Edge-assisted continuous and timely DNN model evolution for mobile devices," *IEEE Trans. Mobile Comput.*, vol. 24, no. 4, pp. 2485–2503, Apr. 2025.

[26] M. Khani et al., "RECL: Responsive resource-efficient continuous learning for video analytics," in *Proc. 20th USENIX Symp. Netw. Syst. Des. Implementation*, 2023, pp. 917–932.

[27] K. Huang, B. Yang, and W. Gao, "ElasticTrainer: Speeding up on-device training with runtime elastic tensor selection," in *Proc. 21st Annu. Int. Conf. Mobile Syst., Appl. Serv.*, 2023, pp. 56–69.

[28] E. Michaud, Z. Liu, U. Girit, and M. Tegmark, "The quantization model of neural scaling," *in Proc. Adv. Neural Inf. Process. Syst.*, 2023, vol. 36, pp. 28699–28722.

[29] D.-W. Zhou, H.-L. Sun, H.-J. Ye, and D.-C. Zhan, "Expandable subspace ensemble for pre-trained model-based class-incremental learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2024, pp. 23554–23 564.

[30] S. Lym, E. Choukse, S. Zangeneh, W. Wen, S. Sanghavi, and M. Erez, "PruneTrain: Fast neural network training by dynamic sparse model reconfiguration," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2019, pp. 36:1–36:13.

[31] A. Dosovitskiy et al., "An image is worth 16x16 words: Transformers for image recognition at scale," in *Proc. Int. Conf. Learn. Representations*, 2021, pp. 1–12.

[32] L. H. Li et al., "Grounded language-image pre-training," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2022, pp. 10955–10 965.

[33] "Intelligent vehicle e/e architecture research report," ReportLinker, Tech. Rep. ASDR-608868, 2022. [Online]. Available: https://www.asdreports.com/market-research-content-608868/intelligent-vehicle-ee-architecture-research-report

[34] N. Developer, "NVIDIA driveOS," 2024. [Online]. Available: https://developer.nvidia.com/drive/os

[35] H. Heinecke et al., "Automotive open system architecture-an industry-wide initiative to manage the complexity of emerging automotive e/e-architectures," SAE Technical Paper, Tech. Rep. 2004-21-0042, 2004. [Online]. Available: https://saemobilus.sae.org/papers/automotive-open-system-architecture-industrywide-initiative-manage-complexity-emerging-automotive-e-e-architectures-2004-21-0042

[36] B. Moiseev, A. Konev, A. Chigorin, and A. Konushin, "Evaluation of traffic sign recognition methods trained on synthetically generated data," in *Proc. ACIVS'13, Ser. Lecture Notes Comput. Sci.*, 2013, vol. 8192, pp. 576–583.

[37] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel, "The German traffic sign recognition benchmark: A multi-class classification competition," in *Proc. Int. Joint Conf. Neural Netw.*, 2011, pp. 1453–1460.

[38] M. Martin et al., "Drive&Act: A multi-modal dataset for fine-grained driver behavior recognition in autonomous vehicles," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 2801–2810.

[39] M. H. Saad, M. I. Khalil, and H. M. Abbas, "End-to-end driver distraction recognition using novel low lighting support dataset," in *Proc. 15th Int. Conf. Comput. Eng. Syst.*, 2020, pp. 1–6.

[40] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur, "Librispeech: An ASR corpus based on public domain audio books," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, 2015, pp. 5206–5210.

[41] P. Budzianowski et al., "MultiWOZ - A large-scale multi-domain wizard-of-OZ dataset for task-oriented dialogue modelling," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, 2018, pp. 5016–5026.

[42] A. Dosovitskiy, G. Ros, F. Codevilla, A. M. López, and V. Koltun, "CARLA: An open urban driving simulator," in *Proc. Conf. Robot Learn.*, 2017, vol. 78, pp. 1–16.

[43] S. Black, L. Gao, P. Wang, C. Leahy, and S. Biderman, "GPT-Neo: Large scale autoregressive language modeling with Mesh-Tensorflow," Mar. 2021. [Online]. Available: https://zenodo.org/records/5297715

[44] X. Huang et al., "DriveGPT: Scaling autoregressive behavior models for driving," in *Proc. Int. Conf. Mach. Learn.*, 2025, pp. 25908–25921.

[45] R. T. Mullapudi, S. Chen, K. Zhang, D. Ramanan, and K. Fatahalian, "Online model distillation for efficient video inference," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2019, pp. 3573–3582.

[46] M. K. Shirkoohi, P. Hamadanian, A. Nasr-Esfahany, and M. Alizadeh, "Real-time video inference on edge devices via adaptive model streaming," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2021, pp. 4552–4562.

[47] L. Wang et al., "EdgeMA: Model adaptation system for real-time video analytics on edge devices," in *Proc. Int. Conf. Neural Inf. Process.*, 2023, pp. 292–304.

[48] X. Gao, Y. Zhao, L. Dudziak, R. D. Mullins, and C. Xu, "Dynamic channel pruning: Feature boosting and suppression," in *Proc. Int. Conf. Learn. Representations*, 2019, pp. 1–11.

[49] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," in *Proc. Int. Conf. Learn. Representations*, 2017, pp. 1–12. [Online]. Available: https://openreview.net/forum?id=rJqFGTslg

[50] J. Kirkpatrick et al., "Overcoming catastrophic forgetting in neural networks," *Proc. Nat. Acad. Sci.*, vol. 114, no. 13, pp. 3521–3526, 2017.

[51] X. Ding, G. Ding, Y. Guo, and J. Han, "Centripetal SGD for pruning very deep convolutional networks with complicated structure," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 4943–4953.

[52] S. Ben-David, J. Blitzer, K. Crammer, A. Kulesza, F. Pereira, and J. W. Vaughan, "A theory of learning from different domains," *Mach. Learn.*, vol. 79, no. 1, pp. 151–175, 2010.

[53] W. Deng and L. Zheng, "AutoEval: Are labels always necessary for classifier accuracy evaluation?," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 46, no. 3, pp. 1868–1880, Mar. 2024.

[54] J. Kaplan et al., "Scaling laws for neural language models," 2020, *arXiv:2001.08361*.

[55] H. Cheng, M. Zhang, and J. Q. Shi, "A survey on deep neural network pruning: Taxonomy, comparison, analysis, and recommendations," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 46, no. 12, pp. 10558–10578, Dec. 2024.

[56] T. Wolf et al., "Transformers: State-of-the-art natural language processing," in *Proc. Conf. Empirical Methods Natural Lang. Process.: Syst. Demonstrations*, 2020, pp. 38–45.

[57] J. Liang, D. Hu, and J. Feng, "Do we really need to access the source data? source hypothesis transfer for unsupervised domain adaptation," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 6028–6039.

[58] S. Jung, H. Ahn, S. Cha, and T. Moon, "Continual learning with node-importance based adaptive group sparse regularization," in *Proc. Adv. Neural Inf. Process. Syst.*, 2020, pp. 3647–3658. [Online]. Available: https://proceedings.neurips.cc/paper/2020/hash/258be18e31c8188555c2ff05b4d542c3-Abstract.html

[59] "End-to-end solutions for autonomous vehicles," 2024. [Online]. Available: https://developer.nvidia.com/drive

[60] D. Hendrycks and T. Dietterich, "Benchmarking neural network robustness to common corruptions and perturbations," in *Proc. Int. Conf. Learn. Representations*, 2019, pp. 1–11.

[61] T.-Y. Lin et al., "Microsoft COCO: Common objects in context," in *Proc. Eur. Conf. Comput. Vis.*, 2014, pp. 740–755.

[62] E. J. Hu et al., "LoRA: Low-rank adaptation of large language models," in *Proc. Int. Conf. Learn. Representations*, 2022, pp. 1–13.

[63] S.-Y. Liu et al., "DoRA: Weight-decomposed low-rank adaptation," in *Proc. Int. Conf. Mach. Learn.*, 2024, pp. 32100–32121.

[64] D. J. Kopiczko, T. Blankevoort, and Y. M. Asano, "VeRA: Vector-based random matrix adaptation," in *Proc. Int. Conf. Learn. Representations*, 2024, pp. 1–13.

[65] Q. Zhang, R. Han, C. H. Liu, G. Wang, and L. Y. Chen, "EdgeVisionBench: A benchmark of evolving input domains for vision applications at edge," in *Proc. IEEE 39th Int. Conf. Data Eng.*, 2023, pp. 3643–3646.

[66] T. Dettmers, M. Lewis, Y. Belkada, and L. Zettlemoyer, "GPT3. int8 (): 8-bit matrix multiplication for transformers at scale," in *Proc. Adv. Neural Inf. Process. Syst.*, 2022, vol. 35, pp. 30318–30332.

[67] C. Riquelme et al., "Scaling vision with sparse mixture of experts," in *Proc. Adv. Neural Inf. Process. Syst.*, 2021, vol. 34, pp. 8583–8595.

[68] Z. Liu et al., "Deja vu: Contextual sparsity for efficient LLMs at inference time," in *Proc. Int. Conf. Mach. Learn.*, 2023, vol. 202, pp. 22137–22176.

[69] "Downsampled ImageNet datasets," 2020. [Online]. Available: http://image-net.org/download-images

[70] D. Hendrycks and T. G. Dietterich, "Benchmarking neural network robustness to common corruptions and perturbations," in *Proc. 7th Int. Conf. Learn. Representations*, New Orleans, LA, USA, OpenReview.net, 2019, pp. 1–11. [Online]. Available: https://openreview.net/forum?id=HJz6tiCqYm

[71] A. Nichol, J. Achiam, and J. Schulman, "On first-order meta-learning algorithms," 2018, *arXiv:1803.02999*.

[72] Y. Luopan, R. Han, Q. Zhang, C. H. Liu, G. Wang, and L. Y. Chen, "FedKNOW: Federated continual learning with signature task knowledge integration at edge," in *Proc. IEEE 39th Int. Conf. Data Eng.*, 2023, pp. 341–354.

[73] L. Xu, K. Liu, J. Liu, L. Wang, L. Xu, and J. Cheng, "Local dense logit relations for enhanced knowledge distillation," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2025, pp. 4539–4549.

[74] B. M. Lake, R. Salakhutdinov, and J. B. Tenenbaum, "The Omniglot challenge: A 3-year progress report," *Curr. Opin. Behav. Sci.*, vol. 29, pp. 97–104, 2019.

[75] M. Tan et al., "MnasNet: Platform-aware neural architecture search for mobile," in *Proc. CVPR'19. Comput. Vis. Found. / IEEE*, 2019, pp. 2820–2828. [Online]. Available: http://openaccess.thecvf.com/content_CVPR_2019/html/Tan_MnasNet_Platform-Aware_Neural_Architecture_Search_for_Mobile_CVPR_2019_paper.html

[76] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, "Once-for-all: Train one network and specialize it for efficient deployment," in *Proc. Int. Conf. Learn. Representations*, 2020, pp. 1–15.

[77] P. Dollár, M. Singh, and R. B. Girshick, "Fast and accurate model scaling," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2021, pp. 924–932.

[78] S. Ho, T. V. Vo, S. Ebrahimkhani, and N. Cheung, "Vision transformer neural architecture search for out-of-distribution generalization: Benchmark and insights," in *Proc. Adv. Neural Inf. Process. Syst.*, 2024.

[79] Y. Peng, A. Song, H. M. Fayek, V. Ciesielski, and X. Chang, "SWAP-NAS: Sample-wise activation patterns for ultra-fast NAS," in *Proc. Int. Conf. Learn. Representations*, 2024.
[80] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable architecture search," in *Proc. Int. Conf. Learn. Representations*, 2019.
[81] H. Wen et al., "AdaptiveNet: Post-deployment neural architecture adaptation for diverse edge environments," in *Proc. 29th Annu. Int. Conf. Mobile Comput. Netw.*, 2023, pp. 28:1–28:17.
[82] Z. Zhang et al., "CamoNet: On-device neural network adaptation with zero interaction and unlabeled data for diverse edge environments," *IEEE Trans. Mobile Comput.*, vol. 23, no. 12, pp. 11483–11497, Dec. 2024.
[83] X. Ma, G. Fang, and X. Wang, "LLM-pruner: On the structural pruning of large language models," in *Proc. Adv. Neural Inf. Process. Syst.*, 2023, pp. 21702–21720.
[84] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of BERT: Smaller, faster, cheaper and lighter," 2019, *arXiv:1910.01108*.
[85] X. Jiao et al., "TinyBERT: Distilling BERT for natural language understanding," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, 2020, pp. 4163–4174.
[86] Y. An, X. Zhao, T. Yu, M. Tang, and J. Wang, "Fluctuation-based adaptive structured pruning for large language models," in *Proc. AAAI Conf. Artif. Intell.*, 2024, pp. 10865–10 873.
[87] S. Teerapittayanon, B. McDanel, and H. T. Kung, "BranchyNet: Fast inference via early exiting from deep neural networks," in *Proc. 23rd Int. Conf. Pattern Recognit.*, 2016, pp. 2464–2469.
[88] B. Fang, X. Zeng, and M. Zhang, "NestDNN: Resource-aware multi-tenant on-device deep learning for continuous mobile vision," in *Proc. 24th Annu. Int. Conf. Mobile Comput. Netw.*, 2018, pp. 115–127.
[89] R. Han, Q. Zhang, C. H. Liu, G. Wang, J. Tang, and L. Y. Chen, "LegoDNN: Block-grained scaling of deep neural networks for mobile vision," in *Proc. 27th Annu. Int. Conf. Mobile Comput. Netw.*, 2021, pp. 406–419.
[90] J. Yu, L. Yang, N. Xu, J. Yang, and T. S. Huang, "Slimmable neural networks," in *Proc. Int. Conf. Learn. Representations*, 2019.
[91] J. Yu and T. S. Huang, "Universally slimmable networks and improved training techniques," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2019, pp. 1803–1811.
[92] K. Alizadeh et al., "LLM in a flash: Efficient large language model inference with limited memory," in *Proc. Assoc. Comput. Linguist.*, 2024, pp. 12562–12584.
[93] Y. Song, Z. Mi, H. Xie, and H. Chen, "PowerInfer: Fast large language model serving with a consumer-grade GPU," in *Proc. ACM SIGOPS 30th Symp. Operating Syst. Princ.*, 2024, pp. 590–606.
[94] L. Hou, Z. Huang, L. Shang, X. Jiang, X. Chen, and Q. Liu, "DynaBERT: Dynamic BERT with adaptive width and depth," in *Proc. Adv. Neural Inf. Process. Syst.*, 2020, pp. 9782–9793. [Online]. Available: https://proceedings.neurips.cc/paper/2020/hash/6f5216f8d89b086c18298e043bfe48ed-Abstract.html
[95] L. D. Corro, A. D. Giorno, S. Agarwal, B. Yu, A. Awadallah, and S. Mukherjee, "SkipDecode: Autoregressive skip decoding with batching and caching for efficient LLM inference," 2023, *arXiv:2307.02628*.

Qinglong Zhang is currently working toward the master's degree with the School of Computer Science and Technology, Beijing Institute of Technology. His work focuses on edge intelligence and deep learning applications.

Rui Han received the MSc honor degree with from Tsinghua University, China, in 2010 and the PhD degree from Imperial College London, U.K., in 2014. He is currently an associate professor with the School of Computer Science and Technology, Beijing Institute of Technology, China. He has more than 40 publications in these areas, including papers at MobiCOM, *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions on Computers*, *IEEE Transactions on Knowledge and Data Engineering*, INFOCOM, and ICDCS. His research interests are system optimization for cloud data center workloads (in particular highly parallel services and deep learning applications).

Chi Harold Liu (Fellow, IEEE) received the PhD degree in electronic engineering from Imperial College, U.K. in 2010, and the BEng degree in electronic and information engineering from Tsinghua University, China in 2006. He is currently a full professor, vice dean with the School of Computer Science and Technology, China. Before that, he worked for IBM Research - China and Deutsche Telekom Laboratories, Berlin, Germany, and IBM T. J. Watson Research Center, USA. His current research interests include the Industrial Big Data and Embodied AI. He was the recipient of the IBM First Plateau Invention Achievement Award in 2012, IEEE DataCom Best Paper Award in 2016, ACM SigKDD Best Paper Runner-up Award in 2021, ACM MobiCom Best Community Paper Runner-up Award in 2021, Gold Medal for Invention Performance Award Nuremburg in 2025, First Class Scientific Award and First Class Teaching Award of China Institute of Electronics in 2024 and 2025. He is a fellow of IEEE, IET, British Computer Society and Royal Society of the Arts.

Guoren Wang (Senior Member, IEEE) received the BSc, MSc, and PhD degrees from the Department of Computer Science, Northeastern University, Shenyang, China, in 1988, 1991, and 1996, respectively. He was an assistant president for Northeastern University, China. He is currently a full professor and director with the Institute of Data Science and Knowledge Engineering, the Department of Computer Science and Technology, Beijing Institute of Technology, Beijing, China. He has authored or coauthored more than 150 research papers in top conferences and journals like *IEEE Transactions on Knowledge and Data Engineering*, ICDE, SIGMOD, VLDB. His research interests include XML data management, query processing and optimization, bioinformatics, high dimensional indexing, parallel database systems, and cloud data management.

Lydia Y. Chen (Senior Member, IEEE) received the BA degree from National Taiwan University and the PhD degree from the Pennsylvania State University. She is currently a professor with the University of Neuchatel and Technology University Delft. Prior to joining TU Delft, she was a research staff member with the IBM Zurich Research Lab from 2007 to 2018. She has authored or coauthored more than 80 papers in journals, such as *IEEE Transactions on Distributed Systems*, *IEEE Transactions on Service Computing*, and conference proceedings, such as INFOCOM, Sigmetrics, DSN, and Eurosys. Her research interests center around dependability management, resource allocation and privacy enhancement for large scale data processing systems and services. She was the corecipient of the best paper awards at CCgrid'15 and eEnergy'15.