Automated Generation of Transformations to Mitigate Sensor Hardware Migration in ADS

Meriel von Stein[1], Hongning Wang[1], and Sebastian Elbaum[1]

***Abstract*—Autonomous driving systems (ADSs) rely on massive amounts of sensed data to train their underlying deep neural networks (DNNs). Common sensor hardware migrations can render an existing DNN-dependent pipeline inadequate. This necessitates the development of bespoke transformations to adapt new sensor data to the old trained network, or the retraining of a new network with new sensor data. These solutions are expensive, often performed reactively to sensor hardware migration, and rely only on empirical reconstruction and validation metrics which lack knowledge of the features important to the trained DNN. To address these challenges, we propose PreFixer, a technique that can systematically generate transformations for many types of sensor hardware migration during the ADS development lifecycle. PreFixer collects small datasets using colocated new and old sensors, and then uses that data and the output of the original trained DNN to train an augmented encoder to learn a transformation that maps new sensor data to old sensor data. The trained encoder can then be deployed as a preprocessor to the original trained DNN. Our study shows that, for a common set of camera sensor hardware migrations, PreFixer can match or improve the performance of the best-performing specialized baseline technique in terms of distance travelled safely with 10% of the training dataset, and take at most half of the training time of a new network.**

***Index Terms*—Autonomous Vehicle Navigation, Deep Learning for Visual Perception, Vision-Based Navigation**

I. Introduction

AUTONOMOUS Driving Systems (ADSs) are becoming more advanced and ubiquitous, enabled by increasingly sophisticated deep neural networks (DNNs). Producing these DNNs is expensive, requiring massive amounts of data, labelling, training, and iterations of refinement [1], [2], [3].

System evolution, however, can make even the most powerful ADS network inadequate. In particular, sensor hardware migrations where one sensor is replaced by another to reduce cost or increase sensing quality occur frequently in practice. These migrations are problematic because they may render data different from that employed in the ADS development, potentially affecting the system performance and reliability[4], and mitigating their downstream effects on DNN predictions can be costly [4].

Manuscript received: December 21, 2023; Revised February 15, 2024; Accepted April 30, 2024.

This paper was recommended for publication by Editor Ashis Banerjee upon evaluation of the Associate Editor and Reviewers' comments. This work was supported in part by NSF Award #2312487 and AFOSR Award #FA9550-21-1-0164.

[1]Meriel von Stein, Hongning Wang, and Sebastian Elbaum are with the School of Engineering, Computer Science Department, University of Virginia, USA. {meriel, hw5x, selbaum}@virginia.edu

Digital Object Identifier (DOI): see top of this page.

(a) Original Camera Image (b) Camera Shallow Depth

(c) Camera Resolution Decrease (d) Camera Fisheye Lens

Fig. 1: Image transformations that result from camera migrations and the corresponding change in steering prediction.

Table I details a diverse but representative sample of those sensor migrations, the resulting feature changes in the sensor reading, and the image transformation needed to approximate the original sensor reading. Some migrations aim to improve sensing quality, such as the migration from rolling to global shutter, whereas some are cost saving measures, such as the migration to a camera with lower resolution. Now consider the sample images in Fig. 1 produced by four cameras. Let's assume an ADS neural network is trained to consume images from the original camera to steer the vehicle. If the original camera is replaced with another with different capabilities even in a well-defined driving scenario, then that network is likely to produce unexpected steering. The superimposed arrows in Figures 1b-d show dramatic steering angle differences between the images of the migrations (blue arrows) and the image from the original camera in Figure 1a (red arrows).

To mitigate the impact of sensor migrations, previous approaches typically employ three strategies [12], [13]. First, developers can design bespoke transformations that map the new sensor output distribution back to the old sensor one to reuse the trained DNN. However, custom transformations are highly dependent on domain knowledge and occasionally infeasible [14]. Second, developers can fine tune existing DNNs with a smaller dataset collected using the new sensor. However, fine tuning may not be a sufficient intervention if the migration gives rise to significant changes to the features of the sensor reading, and it can be inconsistent and difficult to apply [15], [16]. Third, developers can train a new DNN on a new dataset collected from the new sensor. However, collecting a comprehensive dataset and retraining is time-consuming,

Systems	Camera hardware migration	Feature change	Image transformation
Level 5 / Woven Planet	100m to 60m stereo depth accuracy[5]	Depth of field	Depth estimation [6]
Waymo	Pinhole camera to fisheye lens camera [7]	Fisheye	Radial distortion [8]
Waymo	Rolling shutter to global shutter [7]	Removed artifacting	Inpainting
Argo v1.0 to 2.0 (ring cameras)	(1920 x 1200) to (2048 x 1550) inc. resol. [9], [10]	Resolution increase	Interpolation/Superresolution
Argo v1.0 to 2.0 (front stereo cameras)	(2056 x 2464) to (2048 x 1550) dec. resol. [9], [10]	Resolution decrease	Downsampling
Argo v1.0 to 2.0	Landscape to portrait orientation [9], [10]	Orientation	Mirror edge/Black padding
Tesla AutoPilot 2 to 2.5/3.0	RCCC filter to RCCB.8 filter [11]	Filter color shift	Computational filter
Tesla AutoPilot 1.0 to 2.0	Black and white to RGB color [11]	Dimensionality inc.	Inpainting/Retrain

TABLE I: Common hardware migrations and resulting image transformations. See Figure 1 for image examples captured through a subset of these cameras.

expensive, and may require external validation [17].

This state of the art leaves a clear need for a systematic, low-cost, highly flexible technique that can handle significant sensor hardware migrations in the ADS pipeline. Our key insight to address this challenge is that generating a mapping between sensor distributions that are image-to-image is an easier problem than retraining a DNN to map images to a control signal prediction and thus requires fewer resources.

We present PreFixer, the first technique to mitigate the impact of sensor hardware migration in ADS with a high rate of success, low cost to implement, and minimal disruption to the ADS pipeline. To accomplish a successful transformation with respect to a driving DNN, we prioritize the features encoded by that DNN to translate between the feature representations of the new and old sensor data distributions. Unlike previous work [18], [6], [19], our technique makes no assumptions on the unconditioned distribution of features of new data or the similarity of old data to new. PreFixer employs two key components: a simultaneous deployment of old and new sensors to produce a collocated sensor dataset, and an augmented encoder of a generative model trained on that dataset. To effectively train the encoder, we rely on a specialized loss function that accounts for the output of the downstream DNN to ensure a feature selection that is meaningful to both image reconstruction and prediction accuracy. This process can learn an image transformation with a small amount of data and preserves prediction accuracy of the original driving network by optimizing for it in the encoder training loss calculation. We show that PreFixer handles many common camera sensor migrations, including those for which heuristics do not exist. Our contributions are:

- an implementation and exploration of state-of-the-art techniques to mitigate the impact of sensor migrations, specifically camera sensors in commercial ADS;
- our technique, PreFixer, which provides a low-cost, effective, and highly generalizable remedy to this problem that is hardware independent; and
- an artifact with pretrained weights available via https://github.com/MissMeriel/PreFixer

II. Background

This section provides background on encoders and mitigating image transformations.

An encoder [20], [21] is a type of neural network that learns efficient embeddings of unlabeled data. The set of embeddings is often referred to as a dictionary. Variational auto-encoders (VAEs) [22], [23] use an encoder, dictionary, and a decoder

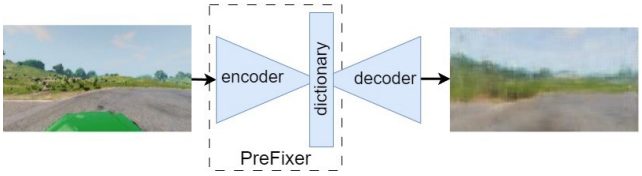

Fig. 2: Output from PreFixer VQVAE for input Figure 1d to approximately reconstruct the image in Figure 1a.

to reconstruct the original data using its lower-dimensional embedding. VAEs can suffer from gradient collapse and have difficulty with non-categorical data. VQVAEs [24] were designed to overcome such deficiencies by using an autoregressive prior and discretizing the embedding dictionary, which is a more natural representation modality for languages, image features, and planning. VQVAEs are well-suited to encode images correlated with a continuous output distribution from the original DNN. A discrete dictionary also circumvents "posterior collapse" where the decoder overpowers the latent encoding and leads to poor reconstructions. Figure 2 illustrates how encoders are used to generate reconstructions. An input image is passed through the encoder to generate a dictionary embedding, then the decoder uses the embedding to generate the reconstruction. As we discuss later, PreFixer manipulates the encoder architecture and embedding dimensions.

Using custom image transformations to approximate a feature space mapping has been explored in the computer vision community [19]. However, producing mappings between sensor changes is a relatively new problem for engineers of cyber-physical systems [25], [18]. Current research [6] shows that engineers can introduce transformations to "clean" images, but cannot always translate between them. For example, novel view synthesis [26] can support the translation between images due to differences in camera extrinsics by interpolating between perspectives, however feature deformations due to changes in camera intrinsics prevent this translation. Image transformations such as depth of field, camera motion, view/placement changes, and noise, each require a unique mitigation techniques such as deblurring GANs specific to the type of motion blur, infill techniques, or denoising CNNs [27]. Confounding factors, such as changes to multiple aspects of the hardware (e.g. changes to both resolution color shift) can further complicate this process. Existing work on feature space mapping to balance the needs of the system and the hardware with which an image is collected [28] are designed to handle only one type of change at a time. Moreover, they are evaluated on image similarity metrics rather than their prediction fidelity by another neural network. Instead, our technique seeks to

maximize prediction fidelity.

Image transformation methods do not account either for downstream system effects, so developers often use other techniques to mitigate sensor changes. Some take shortcuts to retrain the original DNN such as initializing training with pretrained weights, or warm-starting [29], which can offer improvements compared to random initialization of weights. Ash et al. mention this practice consistently hurts generalization, while deceptively having little effect on training accuracy. Completely retraining a DNN on a new dataset is costly [30], [31], [32], [33], so warm-starting and fine-tuning are viable options for some applications.

Our work also relates to efforts to ameliorate the effects of adversarial attacks. Guo [34] and Naseer [35] take a similar approach to "fixing" DNN inputs. Although their overall goal is to defend against robustness attacks, they use image transformations on DNN inputs to reduce perturbation strength and mitigate consequences to the system. Guo et al. do this through types of image reconstruction (variance minimization and image patching) and Naseer et al. do this through feature distortion during training. However, it is unclear how effectively these techniques can adapt to regression networks and how they will generalize to features not seen during training. Moreover, these techniques are geared towards "targeted noise" attacks like Carlini-Wagner, rather than hardware-derived feature deformations.

Our technique is architecturally similar to network chaining [36], [37], in which predictions from some networks become features for other downstream networks. This setup is used in commercial autonomous driving systems [38]. Similarly, we use the VQVAE as a preprocessor for a navigation DNN, applying transformations to important features that have subtle but impactful influence on the DNN's prediction.

III. Problem Definition

An ADS $\mathcal{S}$ is equipped with sensor c and a navigation DNN $\mathcal{M}$. At each timestep t, c produces x_c, defined by feature space X_c, which is consumed by $\mathcal{M}$ to produce control signal ψ. Given a migration from c to a new sensor c' defined by feature space $X_{c'}$ (non-congruent with X_c of c), the problem is to find a transformation $T_{c':c}$ that maps an $x_{c'}$ to an x_c to produce an reconstruction image $\hat{x}$, such that:
(1) $T_{c':c}(X_{c'}) = \hat{X} \approx X_c$: the transformation of the feature space of the new sensor must consistently approximate the feature space of the original sensor.
(2) $\text{argmin}_{T_{c':c}} f(x)\mathcal{M}(T_{c':c}(X_{c'}) = \hat{X}) \approx \mathcal{M}(X_c)$: the transformation approximately reproduces the mapping from image to DNN output of the original sensor component such that it minimizes prediction error.

IV. Technique

PreFixer aims to learn a transformation $T_{c':c}$ to mitigate the impact of a sensor migration by preserving the prediction accuracy of the navigation network that consumes those images.

Figure 3 provides an overview of the phases of PreFixer: data collection setup, encoder configuration, and training and deployment of the encoder. Data collection uses collocated original and new sensors, c and c', to create a dataset of conjugate images $(x_c, x_{c'})$. The configuration space modifies the augmented encoder to suit the transformation $T_{c':c}$. Then, the encoder is trained on the dataset using a loss function that leverages knowledge of $\mathcal{M}$. Upon deployment, $\mathcal{M}$ is equipped with sensor c' and the trained encoder as a preprocessing module of $\mathcal{M}$. We next discuss how this architecture fulfils the problem requirement and describe each stage in detail.

A. Technique Requirements and Fulfillment

PreFixer aims to (R1) support a diverse range of camera hardware migrations commonly occurring in practice (see Table I); (R2) be resource-efficient in that it is easier, faster, and cheaper to apply than existing techniques such as designing custom image transformations or retraining a DNN; and (R3) provide a set of configuration principles that require minimal setup, parameter tweaking, and domain knowledge.

1) R1: To support a range of camera migrations, we tailor a VQVAE encoder to each transformation to avoid gradient collapse and leverage the output distribution of the original DNN. The encoder is responsible for the one-hot mapping of the image input to the embedding space vectors, whereas the decoder is responsible for producing the reconstruction $\hat{x}$. The encoder is flexible enough to support variations in input image size as well as feature deformation according to the configuration of the convolutional layers. The encoder therefore ensures that all images can be encoded to the same low-dimensional space, reducing parameter tweaking once a sufficient embedding dimension has been found. Manipulating the encoder thus also connects to the resource efficiency requirement R2 and minimal configuration requirement R3.

2) R2: To satisfy R2, we have taken several steps to ensure PreFixer is resource-efficient. First, we augment the VQVAE training loss function to optimize for the reconstruction of features important to $\mathcal{M}$, improving prediction accuracy on $\hat{x}$. Our use of predictions from $\mathcal{M}$ is rooted in recent results suggesting that a well-trained image processing network will help with the training of another [29], [36], [16]. Thus prediction loss serves as a regularization term to better differentiate encoding centroids for otherwise visually similar images [39]. A steering DNN is a good application of this because VAEs enforce normality and steering predictions are a skewed normal distribution. Second, using a VQVAE enables the use of a smaller dataset as it has fewer trainable parameters than $\mathcal{M}$, and is thus less prone to overfitting with a smaller dataset. Backpropogation during training is faster for the smaller VQVAE as well. Finally, by manipulating only the encoder, the decoder can be reused for expedited learning of future transformations, because all transformed images have an onto mapping to a single image in the original camera configuration and thus occupy a similar place in the embedding space. By manipulating only the encoder, we have narrowed the configuration space such that the default VQVAE configuration of layers and embeddings works well for a set of common transformations.

3) R3: To satisfy R3, we design PreFixer to need minimal investment in setup, parameter exploration, or domain

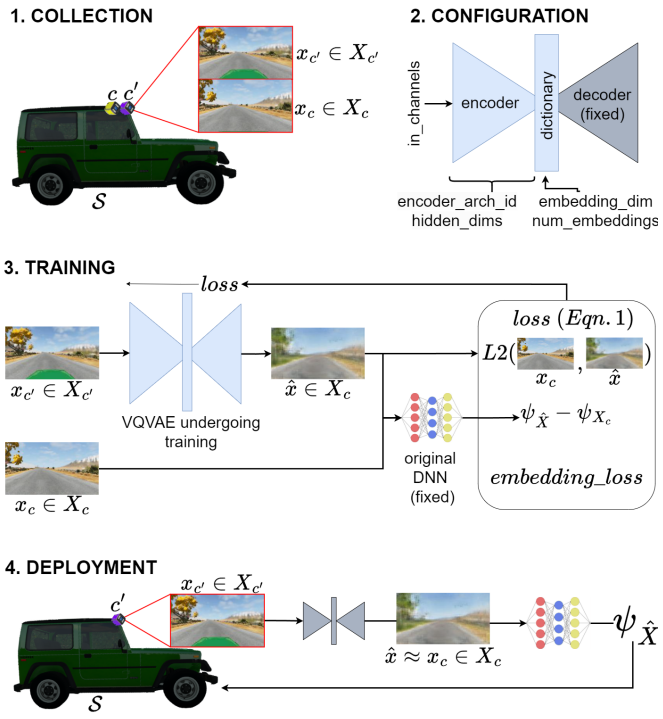

Fig. 3: Overview of PreFixer.

knowledge. The VQVAE offers a single automated mechanism to learn transformation regardless of type, eliminating the need for developer time to craft bespoke transformations or domain knowledge beyond the dimensions of the images. Its unsupervised learning approach automates the mapping between feature spaces of the new and old sensors in such a way that features critical to the predictions of the downstream DNN are automatically prioritized in reconstruction.

We offer in the study and tool repo a set of encoder architectures, several of which work well for mappings between images of identical size and two that work for mapping between differing image sizes. Our augmented VQVAE encoder can be configured according to image size and transformation complexity. This flexibility allows the encoder to map between a wide set of transformed images.

B. Detailed Design of PreFixer

Our approach relies on two key components to automatically compute the transformation $T_{c':c}$: a flexible variable encoder architecture and the ability to capture a small dataset with both sensors running concurrently. These components play a role in the collection, configuration, training, and deployment stages of the development lifecycle.

1) Collection: The collocated sensors setup enables the collection of a dataset that captures pairs of readings from c and c' [1]. $\mathcal{S}$ is then instructed to navigate scenarios using $\mathcal{M}$ to produce a dataset with cognate sensor readings from c' for every reading from c, accompanying the navigation signal ψ from $\mathcal{M}$. As postulated earlier under R1 and later shown in Section V-D, a small, diverse dataset will be sufficient to learn a generative model that encodes transformation $T_{c':c}$ because this model has fewer trainable parameters than $\mathcal{M}$, making backpropogation during training faster and rendering a model less prone to overfitting.

2) Configuration: The configuration space affects the encoder and the dictionary. The encoder configuration consists of the input channels for the sensor reading (for example, 3 input channels for an RGB image), the encoder architecture identifier (*encoder_arch_id*), and the size of the feature dimensions in the hidden layers (*hidden_dims*). The dictionary reduces internal representation of an input to the important features in a lower-dimensional space. This dictionary configuration consists of the number of embedding vectors (*num_embeddings*), and the dimensions of each vector (*embedding_dims*). The feature dimensions in the hidden layers are the number of tensor channels and consequently the number of convolutional kernels for each hidden layer in the encoder. The embedding dimensions are the size of each of the embedding vectors that define the latent space of the dictionary, and the number of embeddings are the number of quantized vectors used to map inputs in the latent space. By configuring only the encoder, the decoder can be reused for expedited learning of future transformations, because all transformed images have an onto mapping to a single image in the original camera configuration and thus occupy a similar place in the embedding space. Section V-B further details how the configuration space is parameterized.

3) Training: The training portion of Figure 3 shows training for one input. The c' reading $x_{c'}$ is passed to the VQVAE. The resulting reconstruction $\hat{x} \in X_c$ is then used to calculate the terms of the $loss$ function, defined as:

$$loss = L_1(\psi_{\hat{x}} - \psi_x) + L_2(x_c - \hat{x}) + embedding\_loss \quad (1)$$

The first term of $loss$ is the prediction loss. $\hat{x}$ is passed to $\mathcal{M}$ to produce prediction $\psi_{\hat{x}}$, which is compared to the prediction from the original sensor reading, ψ_x We use L1 to calculate prediction loss because L1 penalizes large and small loss terms proportionally, as a single high error prediction may not be as catastrophic as a series of low error predictions. The second term of $loss$ calculates the pixelwise reconstruction loss as an L2 norm between the reconstruction $\hat{x}$ and the original sensor reading x_c to penalize large shifts in pixel values. The third term, embedding loss, deals with the loss from quantizing embedding vectors such that embedding vectors themselves are defined to minimize $||x_c - \hat{x}|| \forall x \in X_{c'}$. Embedding loss is part of the VQVAE training but it is not unique to PreFixer [24].

Our use of predictions from $\mathcal{M}$ is rooted in recent results that suggest a high-quality image processing network will help with the training of another network for a related task [29], [36]. (See discussion of prediction loss as a regularization term in Section IV-A2). The prediction loss term can be weighted to further prioritize it during training. We found empirically that past a weight of 1.0 the prediction loss on the validation set plateaus, likely due to the proportionality of prediction loss and reconstruction loss, but this is an exploration for future work. VQVAEs (like VAEs) assume that prior and posterior

[1]The sensors should be placed as close as possible on $\mathcal{S}$, and the encoder can manage small perspective shifts. Section V-D shows it is able to handle lossy transformations and low-quality images that require infill.

are Gaussian [39]. Our technique circumvents this by using the pretrained $\mathcal{M}$ which explicitly maps a non-Gaussian image distribution to a skewed Gaussian steering distribution.

4) Deployment: The final section of Figure 3 shows the deployment setup. The new sensor c' sends an image $x_{c'} \in X_{c'}$ to the trained VQVAE which produces a reconstruction $\hat{x} \approx x_c \in X_c$. This reconstruction is then sent to $\mathcal{M}$ to predict ψ, which is then sent to the system $\mathcal{S}$ for actuation.

The costs of PreFixer are: (1) collecting a small dataset using collocated sensors; and (2) training a VQVAE using that dataset, the cost of which is proportional to dataset size. As we shall see in the study, even with these costs, PreFixer gives a higher probability of prediction accuracy, leading to better generalization in new environments and more stable system behavior under deployment, as well as lower training overhead or developer involvement than alternative approaches.

C. Generalization

In this section we discuss two broad challenges to the generalization of PreFixer: (1) the scope of factors related to sensor migrations, specifically sensor parameters and sensor types, and (2) the integration with existing ADS pipelines.

PreFixer focuses on intrinsic changes in sensor parameters because those are the ones primarily affected by common sensor migrations (see Table I) and there exists no single technique to mitigate them. Changes in some extrinsic parameters, such as sensor placement, could be accommodated. However, that would require the reconstruction of unseen image areas, rather than remapping features that are present but deformed.

In terms of sensor types, PreFixer can be adapted to sensors beyond standard cameras. For example, adapting to RGB-D would not require significant changes, as RGB-D cameras would not interfere with each other during dataset collection and the current encoder set handles 4-channel images similarly to 3-channel ones. Adapting to LiDAR, however, would require carefully consideration to sensor placement to minimize interferences so the transformation is learnable by the encoder and replicable when the old sensor is eventually removed. In addition, the encoder *in_channels* and *encoder_arch_id* must be configured to match the LiDAR pointcloud structure. PreFixer could also be used for migrations between sensor types (e.g. RGB to LiDAR) as long as sensor readings can be represented in tensor form and the encoder architecture accommodates the sensor reading dimensions. Exploration of these adaptations is part of future work.

Regarding the generalization to ADS pipelines, PreFixer assumes an end-to-end network with a Gaussian output distribution to better leverage the assumptions of VQVAEs. PreFixer can thus generalize well to network for different tasks, such as predicting driver trust, or networks that produce multiple outputs, such as a network that predicts both steering and throttle, as long as there is a Gaussian signal to guide training of the model.

Lastly, PreFixer can be applied in pipelines with sufficient slack to tolerate its training costs and runtime latency. The usage of our technique may not be feasible for systems in which the onboard processing power is limited or where software throughput is too high to accommodate the added latency (see Section V-D2).

V. Study

Our study aims to answer the following research question:

RQ1) *How effective is PreFixer compared to other techniques in supporting common camera migrations?* To answer this question, we explore 4 image transformations related to camera migrations observed in real-world systems. We then test PreFixer on 10 validation road segments not seen in training.

RQ2) *How much data does PreFixer need to successfully learn each image transformation?* To answer this question, we compare performance across 4 dataset sizes for all 4 image transformations.

A. Setup

1) System Under Test: Our study uses the BeamNG high-fidelity driving simulator [40]. For the autonomous vehicle under test, we equip the prepackaged "hopper" vehicle with an onboard camera at the top edge of the windshield. The camera has a 50° field of view angled upwards 5° relative to the vehicle pitch and collects images at 15 Hz to mimic established self-driving setups [41]. For vehicle control, we re-implemented the DAVE2 architecture [42], which consumes camera images to steer a vehicle. We trained DAVE2 with 145,521 images collected over multiple prebuilt BeamNG driving environments. The trained network final loss was 0.012 MSE between predicted and ground truth steering angles. A PID regulates throttle. This system is able to traverse all road segments in this study with the original camera configuration.

2) Validation Road Segments: We test all techniques in simulation on 10 unseen 100-meter road segments. These road segments were chosen for their difficulty for the DAVE2 network to navigate them, the variety of road topologies (straight, winding, left and right turns), and the variety of features in the driving environment (mountains, city driving, country roads). Each technique was run 25 times per segment to account for built-in randomness in the simulator.

3) Image transformations from real-world camera hardware migrations: Table I samples the broad spectrum of image transformations that occur in real-world autonomous systems. These examples motivate the transformations we chose for this study. Rows 1 and 5 are lossy, where the new camera hardware is of a reduced quality. Row 4 is lossless because the new camera hardware is better (e.g. increased resolution). Row 2 is additive and includes a feature deformation, where the new sensor is qualitatively different in a way that captures more information (e.g. pinhole to fisheye camera).

For the related image transformations explored in this study, the fisheye transformation increases the field of view from 50° to 75°. The depth of field transformation decreases the depth in focus from 1000m to 100m. Depth and fisheye maintain the image size of 108×192. The resolution increase transformation increases original image dimensions from 108×192 pixels to 270×480 pixels. The resolution decrease transformation adjusts the original dimensions to 54×96 pixels.

B. PreFixer Configuration

To accommodate the image transformations in Table I, we configure the VQVAE architecture primarily through the *encoder_arch_id* metaparameter. Each convolutional layer is parameterized by padding p, kernel size k, and stride s. Padding adds a border of zeroed pixels p wide before convolving. Kernel size determines the number of elements in the tensor that are used to compute the element in the output tensor of the layer. For each feature channel there is one kernel of size $k \times k$. Stride steps the kernel by s pixels across the input tensor before the kernel is applied again.

For all transformations, we adjust the configuration to reverse the original VQVAE architecture [43], starting with a smaller kernel and stride relative to the size of the image to preserve image features that may be combined, weighted, or discarded in later convolutions. If needed for the transformation, we use padding in the outermost convolutional layers to fit the dimensions of the target image and maximize use of the original image pixels. We then tailored *encoder_arch_id* further for each transformation as follows.

The resolution decrease camera migration (Figure 1c) requires a transformation that handles blurring and pixelation of features to successfully map back to the original (larger) image. This requires a smaller convolutional kernel relative to image size in order to average between multiple pixels and possibly some padding in earlier layers to leverage what information is available in the image tensor. Kernel and stride are small relative to the input image size throughout the encoder layers to ensure that the feature space is not over-reduced and the embedding vector size is maintained to reconstruct the original image dimensions. For the resolution increase transformation (mapping from larger to smaller images), the kernel and stride are large relative to the size of the image in the first layer and decrease on the subsequent convolutional layers. This ensures that reconstruction image size is decreased and features are not overreduced by subsequent layers.

The "pinhole to fisheye" camera migration causes features towards the center to appear smaller and features towards the edges to appear curved. The different distortion of features across the image requires a 1×1 kernel and 1×1 stride in the first layer to group pixels by features (the number of which is defined by *hidden_dims*) before true convolutions are applied. After pixel selection, a relatively small kernel and stride of identical size are applied in subsequent layers to retain efficient processing while capturing all values in the tensor output by the previous layer. This same *encoder_arch_id* is appropriate for the "decreased depth of field" image transformation, which also has distortions appearing in different areas of the image.

After we completed the configuration for each transformation, for each one of them, we trained 4 encoders using 4 datasets with the following sizes: 5K, 10K, 25K, and 50K, with each dataset being a subset of the next.

C. Baselines

We now describe existing techniques to handle camera sensor hardware migrations as baselines to assess PreFixer.

1) Bespoke Transformation Only (BTO): This technique employs a custom and domain-specific inverse transformation that maps the new sensor feature space back to the old sensor feature space before being sent to the DNN trained for the original camera hardware. In practice, $T_{c':c}$ is designed and tuned over time by the developer using domain knowledge and validation results. *BTO* avoids new data collection, labelling, and training, but required bespoke transforms. In our study we selected existing transformations as follows: for the fisheye transformation we used a discorpy backward radial distortion [44], for the depth transformation we employed a deblurring GAN [45], and for resolution increase and decrease we use the PIL resize with bilinear interpolation [46].

2) Transform and Retrain (TR): This technique retrains the DAVE2 architecture using a transformed version of the dataset where existing images are mapped to the feature space of the new sensor. $T_{c:c'}$ is developed using a similar pipeline to the previous baseline, but the transformations are inversed to map from the feature space of the old camera to the new one. The fisheye transformation is accomplished via a discorpy forward radial distortion, the depth inverse transformation applies a computational blur transformation and a depth estimator, and the resolution increase and decrease transformations are again accomplished with a PIL resize with bilinear interpolation. Overall, this baseline eliminates the need for new data collection and labelling, but again requires the engineering of a high-fidelity transformation between the feature spaces of the two cameras and training the network.

3) Fine-tuning (FT): This baseline uses warmstarting by keeping the weights of the trained DAVE2 network and fine-tuning those weights using new sensor data [29], [47]. For transformations involving changes to input size, the network's last convolutional layer was adjusted. The new datasets used and their size is discussed further in Section V-D.

D. Results

1) RQ1: Effectiveness Compared to Baseline Techniques: Figure 4 shows the average distances travelled across ten 100m tracks for all techniques on the 4 migrations. This shows the effectiveness of the techniques to navigate a range of environments, as well as the comparative difficulty of each transformation mapping. An ideal technique would enable the vehicle to still navigate for 100m across all tracks and camera migrations. For RQ1, we compare *BTO*, *TR*, and *FT* and PreFixer trained with the 50K dataset.

On average, PreFixer is more successful than all other techniques, with all transformations achieving between 87.2 and 93.0m travelled. Even when compared with the best alternative technique, *BTO*, PreFixer can equal or surpass the performance for all transformations. Out of the 4 techniques, *BTO* is the second-most effective and has the lowest variance across transformations, with an average distance of 86.9m across all transformations and all transformations between 84.4m and 90.0m travelled. *FT* is third-most effective, with all transformations falling within 71.8m and 82.5m distance travelled. *TR* is the least effective on average – 50m for fisheye, 64m for depth, 57m for resolution decrease, and 37m for

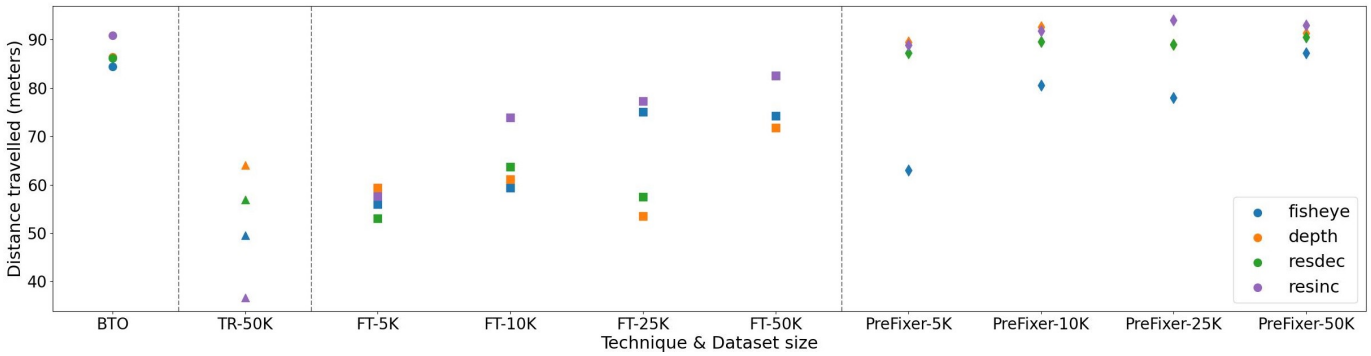

Fig. 4: Effectiveness of each technique with varying dataset sizes from 5K to 50K samples.

resolution increase. This is likely due to the DNN being trained on an approximation of the feature space of the new camera, which does not map to the same predictions.

Fisheye is the most difficult transformation for all techniques, and resolution increase the easiest. This seems intuitive, as fisheye deforms features unevenly across the image. Conversely, resolution increase presents an easier task of averaging pixels to shrink a larger image without any deformation.

2) RQ2: PreFixer Resource-Performance Tradeoffs: The right half of Figure 4 provides several insights into data efficiency for *FT* and PreFixer across the 4 training set sizes. As expected, PreFixer encoder models trained with less data (5K and 10K) under-perform those with more (25K and 50K), but the disparity is lower than anticipated. Fisheye has the highest difference between 5K and 50K (24.2m), but all other transformations had a difference below 5m. Conversely, *FT* shows high susceptibility to dataset size, with an average 17.7m jump in distance travelled between 5K and 50K dataset size across all transformations. The minimum distance travelled for *FT* is 53.1m and no transformation exceeds 82.5m.

Overall, **PreFixer can learn an effective transformation with $\frac{1}{3}$ or less data than it takes to retrain a network.** PreFixer improves upon the best baseline by about 5% in the average case for a problem that is an inevitable part of the ADS development lifecycle. It also significantly improves upon the resource efficiency of other techniques by precluding the need for comprehensive new datasets and copious amounts of developer time. We also find that **even when compared with the best alternative technique, *BTO*, PreFixer can equal or surpass the performance for three of the four transformations with just a 5K dataset.** Our technique can improve upon the best-performing baseline for 3 out of 4 common image transformations using only a 5K dataset. Given a 50K dataset, PreFixer outperforms all baselines by a margin of 3m or more. PreFixer also outperforms *TR* and *FT* for all transformations when comparing identical dataset sizes.

Related to resource-performance tradeoffs are training cost and execution latency. Table II shows the wall clock time to train the original DNN with the full dataset compared to each PreFixer VQVAE with 5K and 50K datasets, the smallest and largest datasets we studied. As anticipated, training DAVE2 is more expensive than any PreFixer 50K encoder by a factor of at least 2, and on average by a factor of 4. This indicates that PreFixer is more affordable than retraining a DAVE2 network, even without the cost of collecting a new dataset.

	Training time	**Inference Wall Time**	**Inference FLOPs**
DAVE2 (145K)	7596 min	0.001352 sec	1.5 million
Fisheye (5K, 50K)	158, 919 min	0.007148 sec	1 billion
Depth (5K, 50K)	157, 1058 min	0.007342 sec	1 billion
ResDec (5K, 50K)	134, 811 min	0.007464 sec	848 million
ResInc (5K, 50K)	225, 4340 min	0.006794 sec	1.15 billion

TABLE II: Training and inferencing cost

Table II also shows inference latency in terms of mean wall time and total floating point operations (FLOPs) per prediction. The original DAVE2 architecture takes on average 0.0013s to run on an NVIDIA Quadro RTX 6000, 128Gb RAM, and Intel Xeon W-2225 CPU @ 4.10GHz. The non-optimized PreFixer takes 0.0071s to run on the same hardware, and one simply quantized [48] takes approximately 0.0031 seconds, halving the inference time. That said, we note that DAVE2 is a simplified architecture of common real-world end-to-end driving DNNs [49], [38] and more complex DNNs would require more inference time.

VI. Conclusions and Future Work

In this paper we present PreFixer , the first unsupervised low-data transformation technique applicable to a wide range of common image transformations resulting from sensor hardware migrations. Our technique successfully learns transformations with high prediction accuracy even from small datasets by leveraging the knowledge of the previously trained base network. Our findings open many avenues for future work. We aim to extend our study with migrations that require transformations of higher complexity and their combinations and other factors such as shifts in sensor locations. We will further explore different training datasets to consider how balancing will affect the reconstruction of rare features like those from extreme lighting conditions or unusual driving environments. We will also investigate the application of PreFixer to other sensor types such as LiDAR to generalize from images to point clouds. Lastly, we will investigate alternative labeling techniques to reduce the reliance on sensor co-location, for instance when the old sensor is unavailable or when collocated sensors produce interference.

References

[1] A. Tampuu *et al.*, "A survey of end-to-end driving: Architectures and training methods," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 4, pp. 1364–1384, 2022.

[2] X. Zhang, Y. Cai, and Z. Yang, "A study on testing autonomous driving systems," in *IEEE International Conference on Software Quality, Reliability and Security Companion*, 2020, pp. 241–244.

[3] Waymo, "The waymo driver's training regimen: How structured testing prepares our self-driving technology for the real world," https://waymo.com/blog/2020/09/the-waymo-drivers-training-regime.html, 2020.

[4] A. Luckow *et al.*, "Deep learning in the automotive industry: Applications and tools," in *IEEE International Conference on Big Data*, 2016, pp. 3759–3768.

[5] L. Platinsky *et al.*, "Quantity over quality: Training an av motion planner with large scale commodity vision data," 2022. [Online]. Available: https://arxiv.org/abs/2203.01681

[6] O. F. Kar *et al.*, "3d common corruptions and data augmentation," 2022. [Online]. Available: https://arxiv.org/abs/2203.01441

[7] Waymo, "Waymo open dataset," https://waymo.com/open/data/perception/, 2022.

[8] OpenCV, "Camera calibration with opencv," https://docs.opencv.org/2.4/doc/tutorials/calib3d/camera_calibration/camera_calibration.html, 2022.

[9] M.-F. Chang *et al.*, "Argoverse: 3d tracking and forecasting with rich maps," 2019. [Online]. Available: https://arxiv.org/abs/1911.02620

[10] B. Wilson *et al.*, "Argoverse 2: Next generation datasets for self-driving perception and forecasting," in *NeurIPS*, 2021.

[11] TeslaDriver.net, "Tesla ap1 vs ap2 vs ap3 – difference between autopilot versions," https://tesladriver.net/tesla-ap1-vs-ap2-vs-ap3-difference-between-autopilot-versions/, 2022.

[12] M. B. Alatise *et al.*, "A review on challenges of autonomous mobile robot and sensor fusion methods," *IEEE Access*, vol. 8, pp. 39 830–39 846, 2020.

[13] Q. Rao *et al.*, "Deep learning for self-driving cars: Chances and challenges," in *International Workshop on Software Engineering for AI in Autonomous Systems*. Association for Computing Machinery, 2018, p. 35–38. [Online]. Available: https://doi.org/10.1145/3194085.3194087

[14] J. Jam *et al.*, "A comprehensive review of past and present image inpainting methods," *Computer Vision and Image Understanding*, vol. 203, p. 103147, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1077314220301661

[15] C. Tan *et al.*, "A survey on deep transfer learning," *CoRR*, vol. abs/1808.01974, 2018. [Online]. Available: http://arxiv.org/abs/1808.01974

[16] J. Yosinski *et al.*, "Advances in neural information processing systems," vol. 27, 2014. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2014/file/375c71349b295fbe2dcdca9206f20a06-Paper.pdf

[17] P. Koopman *et al.*, "Autonomous vehicle safety: An interdisciplinary challenge," *IEEE Intelligent Transportation Systems Magazine*, vol. 9, no. 1, pp. 90–96, 2017.

[18] A. Carlson *et al.*, "Sensor transfer: Learning optimal sensor effect image augmentation for sim-to-real domain adaptation," *IEEE Robotics and Automation Letters*, vol. 4, no. 3, pp. 2431–2438, 2019.

[19] L. He *et al.*, "Learning depth from single images with deep neural network embedding focal length," *CoRR*, vol. abs/1803.10039, 2018. [Online]. Available: http://arxiv.org/abs/1803.10039

[20] B. A. Olshausen *et al.*, "Emergence of simple-cell receptive field properties by learning a sparse code for natural images," *Nature*, vol. 381, pp. 607–609, 1996. [Online]. Available: https://api.semanticscholar.org/CorpusID:4358477

[21] H. Sharma, "A survey on image encoders and language models for image captioning," *IOP Conference Series: Materials Science and Engineering*, vol. 1116, no. 1, p. 012118, apr 2021. [Online]. Available: https://dx.doi.org/10.1088/1757-899X/1116/1/012118

[22] P. Li *et al.*, "A comprehensive survey on design and application of autoencoder in deep learning," *Applied Soft Computing*, vol. 138, p. 110176, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1568494623001941

[23] D. Bank *et al.*, "Autoencoders," *CoRR*, vol. abs/2003.05991, 2020. [Online]. Available: https://arxiv.org/abs/2003.05991

[24] A. van den Oord *et al.*, "Neural discrete representation learning," *CoRR*, vol. abs/1711.00937, 2017. [Online]. Available: http://arxiv.org/abs/1711.00937

[25] A. Carlson *et al.*, "Modeling camera effects to improve visual learning from synthetic data," in *European Conference on Computer Vision Workshops*, September 2018.

[26] E. R. Chan *et al.*, "Generative novel view synthesis with 3d-aware diffusion models," in *IEEE/CVF International Conference on Computer Vision*, October 2023, pp. 4217–4229.

[27] A. Abdelhamed *et al.*, "A high-quality denoising dataset for smartphone cameras," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 1692–1700.

[28] L. Pan *et al.*, "Bringing a blurry frame alive at high frame-rate with an event camera," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019.

[29] J. T. Ash and R. P. Adams, "On the difficulty of warm-starting neural network training," *CoRR*, vol. abs/1910.08475, 2019. [Online]. Available: http://arxiv.org/abs/1910.08475

[30] K. He *et al.*, "Convolutional neural networks at constrained time cost," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[31] D. Masters *et al.*, "Revisiting small batch training for deep neural networks," *CoRR*, vol. abs/1804.07612, 2018. [Online]. Available: http://arxiv.org/abs/1804.07612

[32] M. Kukar *et al.*, "Cost-sensitive learning with neural networks," in *European Conference on Artificial Intelligence*, 1998. [Online]. Available: https://api.semanticscholar.org/CorpusID:14677235

[33] R. Mahmood *et al.*, "Optimizing data collection for machine learning," *Advances in Neural Information Processing Systems*, vol. 35, pp. 29 915–29 928, 2022.

[34] C. Guo *et al.*, "Countering adversarial images using input transformations," *CoRR*, vol. abs/1711.00117, 2017. [Online]. Available: http://arxiv.org/abs/1711.00117

[35] M. Naseer, S. H. Khan, M. Hayat *et al.*, "A self-supervised approach for adversarial robustness," *CoRR*, vol. abs/2006.04924, 2020. [Online]. Available: https://arxiv.org/abs/2006.04924

[36] J. Read *et al.*, "Classifier chains: A review and perspectives," *CoRR*, vol. abs/1912.13405, 2019. [Online]. Available: http://arxiv.org/abs/1912.13405

[37] K. Zaamout *et al.*, "Improving neural networks classification through chaining," in *International Conference on Artificial Neural Networks and Machine Learning*. Springer-Verlag, 2012, p. 288–295. [Online]. Available: https://doi.org/10.1007/978-3-642-33266-1_36

[38] Z. Peng *et al.*, "A first look at the integration of machine learning models in complex autonomous driving systems: A case study on apollo," in *ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, p. 1240–1250. [Online]. Available: https://doi.org/10.1145/3368089.3417063

[39] B. Dai and D. P. Wipf, "Diagnosing and enhancing VAE models," *CoRR*, vol. abs/1903.05789, 2019. [Online]. Available: http://arxiv.org/abs/1903.05789

[40] BeamNG, "Beamng.drive vehicle simulator," https://www.beamng.com/, 2020.

[41] "Waymo open dataset: An autonomous driving dataset," 2019.

[42] M. Bojarski *et al.*, "End to end learning for self-driving cars," *CoRR*, vol. abs/1604.07316, 2016. [Online]. Available: http://arxiv.org/abs/1604.07316

[43] A. Agrawal and N. Mittal, "Using cnn for facial expression recognition: a study of the effects of kernel size and number of filters on accuracy," vol. 36. Springer, 2020, pp. 405 – 412. [Online]. Available: https://doi.org/10.1007/s00371-019-01630-9

[44] N. T. Vo *et al.*, "Radial lens distortion correction with sub-pixel accuracy for x-ray micro-tomography," *Opt. Express*, vol. 23, no. 25, pp. 32 859–32 868, Dec 2015. [Online]. Available: https://opg.optica.org/oe/abstract.cfm?URI=oe-23-25-32859

[45] J. Lee *et al.*, "Iterative filter adaptive network for single image defocus deblurring," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2021.

[46] J. A. Clark and contributors, "Pillow image library," https://pillow.readthedocs.io/en/stable/index.html, 2023.

[47] H. Liu *et al.*, "Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning," 2022.

[48] R. Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: A whitepaper," *CoRR*, vol. abs/1806.08342, 2018. [Online]. Available: http://arxiv.org/abs/1806.08342

[49] M. von Stein *et al.*, "Finding property violations through network falsification: Challenges, adaptations and lessons learned from openpilot," in *IEEE/ACM International Conference on Automated Software Engineering*. Association for Computing Machinery, 2023. [Online]. Available: https://doi.org/10.1145/3551349.3559500