Tiny, Always-on, and Fragile

Bias Propagation through Design Choices in On-device Machine Learning Workflows

Hutiri, Wiebke (Toussaint); Ding, Aaron Yi; Kawsar, Fahim; Mathur, Akhil

DOI
10.1145/3591867

Publication date
2023

Document Version
Final published version

Published in
ACM Transactions on Software Engineering and Methodology

Tiny, Always-on, and Fragile: Bias Propagation through Design Choices in On-device Machine Learning Workflows

WIEBKE (TOUSSAINT) HUTIRI and AARON YI DING, Delft University of Technology, The Netherlands
FAHIM KAWSAR and AKHIL MATHUR, Nokia Bell Labs, UK

Billions of distributed, heterogeneous, and resource constrained IoT devices deploy on-device machine learning (ML) for private, fast, and offline inference on personal data. On-device ML is highly context dependent and sensitive to user, usage, hardware, and environment attributes. This sensitivity and the propensity toward bias in ML makes it important to study bias in on-device settings. Our study is one of the first investigations of bias in this emerging domain and lays important foundations for building fairer on-device ML. We apply a software engineering lens, investigating the propagation of bias through design choices in on-device ML workflows. We first identify *reliability bias* as a source of unfairness and propose a measure to quantify it. We then conduct empirical experiments for a keyword spotting task to show how complex and interacting technical design choices amplify and propagate *reliability bias*. Our results validate that design choices made during model training, like the sample rate and input feature type, and choices made to optimize models, like light-weight architectures, the pruning learning rate, and pruning sparsity, can result in disparate predictive performance across male and female groups. Based on our findings, we suggest low effort strategies for engineers to mitigate bias in on-device ML.

CCS Concepts: • **Software and its engineering → Software creation and management**; • **Computing methodologies → Machine learning**; • **Hardware → Analysis and design of emerging devices and systems**; • **General and reference → Reliability;**

Additional Key Words and Phrases: Bias, on-device machine learning, embedded machine learning, design choices, fairness, audio keyword spotting, personal data

ACM Reference format:
Wiebke (Toussaint) Hutiri, Aaron Yi Ding, Fahim Kawsar, and Akhil Mathur. 2023. Tiny, Always-on, and Fragile: Bias Propagation through Design Choices in On-device Machine Learning Workflows. *ACM Trans. Softw. Eng. Methodol.* 32, 6, Article 155 (September 2023), 37 pages.
https://doi.org/10.1145/3591867

1 INTRODUCTION

Rising concerns about digital privacy and personal data protection [43] are motivating a shift in data processing and **machine learning (ML)** from cloud servers to end devices [10]. On-device

The work was partially done while the Wiebke (Toussaint) Hutiri was an intern at Nokia Bell Labs.
This research is partially supported by SPATIAL project that has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement No. 101021808.
Authors' addresses: W. (Toussaint) Hutiri and A. Y. Ding, Delft University of Technology, Jaffalaan 5, 2628 BX, Delft, The Netherlands; emails: w.toussaint@tudelft.nl, Aaron.Ding@tudelft.nl; F. Kawsar and A. Mathur, Nokia Bell Labs, P.O. Box 1212, Cambridge, UK; emails: fahim.kawsar@nokia-bell-labs.com, akhil.mathur@nokia-bell-labs.com.

1049-331X/2023/09-ART155 $15.00
https://doi.org/10.1145/3591867

ML is an emerging computing paradigm that makes this shift possible [3]. In contrast to ML on centralized cloud-servers, on-device ML processes data directly on the device that collected them. This has important gains for privacy: If the data never leaves the device, then the potential for unsolicited use or abuse by third parties is greatly reduced. Additionally, by eliminating data transfer during inference, on-device ML enables instantaneous, continuous and offline data processing, making it possible to operate devices in an always-on mode.

From earphones to embedded cameras, billions of tiny devices across the globe deploy on-device ML for inference on personal data. However, a growing body of research shows that ML systems are prone to bias [40, 46], and can lead to unfair predictions that favour or are prejudiced against particular groups of people. Bias in ML is concerning, as it can result in decisions that inflict harm on people, oftentimes vulnerable groups or minority populations [4]. Uber's fatal self-driving car crash in 2018 [20] acts as a stark reminder of the consequences of unchecked bias. The crash, which killed a pedestrian, was attributed to software design decisions, which resulted in a series of misclassifications of the crash victim by Uber's ML system [32]. In the case of on-device ML, bias affects system reliability, that is to say the ability of on-device ML to "deliver stable and predictable performance in expected [operating] conditions" [11]. Unexpected system failures can result in products and services that inflict harm on users or the public. To our knowledge, no prior studies have considered whether on-device ML systems are equally reliable for all users, that is to say, whether they are biased.

In this article, we study bias in on-device ML. We approach the topic from a software engineering lens, and investigate the emergence of bias in the on-device ML workflow. On-device ML presents a unique development environment. While the cloud offers limitless computing resources, on-device ML needs to account for the inherent hardware constraints of end devices: limited memory, compute and energy resources [14]. Developers aim to retain predictive accuracy while overcoming these constraints with engineering interventions in the on-device ML workflow. Engineering interventions demand that developers make design choices during product development.

In previous work, we investigated the impact of pre-processing parameter design choices in a keyword spotting task [59]. The study found that pre-processing parameters have a statistically significant impact not only on accuracy but also on bias. The effect is more pronounced for lightweight neural network architectures and at lower sample rates, making this a relevant insight for the development of on-device ML. Informed by that work, we hypothesize that interdependent engineering design choices and unpredictable operating contexts can result in unexpected performance disparities between user groups in on-device ML applications. This article builds on our previous study in three ways: (1) We develop a decision map of on-device ML to consider design choices in the development workflow systematically. (2) We expand our keyword spotting experiments on pre-processing parameters to more datasets and languages. (3) We conduct experiments to study bias due to design choices made during model compression.

Our article is the first study of bias in on-device ML workflows, and makes the following contributions:

(1) We present a decision map to help developers identify design choices in the on-device ML workflow (Section 3).
(2) We identify and quantify metrics to evaluate *reliability bias* (Section 4).
(3) We empirically show that design choices made during model training (Section 6.1) and model optimization (Section 6.2) can amplify and propagate disparate performance across user groups, and thus *reliability bias*.
(4) We suggest strategies for mitigating *reliability bias* without compromising accuracy (Section 7).

The article starts with an overview of background knowledge and related work in Section 2. We then present an overview of on-device ML and design choices arising during on-device ML development in Section 3. In Section 4, we define and quantify *reliability bias*. In Section 5, we introduce an empirical study of an audio keyword spotting task. We present our experimental results in Section 6, and propose strategies for mitigating *reliability bias* in Section 7. Finally, we discuss the implications of our work for the development of fairer on-device ML in Section 8 and conclude in Section 9.

2 BACKGROUND AND RELATED WORK

Bias in ML software engineering is a new area of research. In this section, we thus present interdisciplinary background literature on fairness and bias in ML. We define the concepts that we use in this work, and discuss bias measures and their limitations. We then highlight perspectives on bias in ML development from software engineering and statistical learning, with a focus on the impact of design choices.

2.1 Bias and Fairness in Machine Learning

2.1.1 Concepts and Definitions. Fairness in decision-making systems is considered as the "absence of any prejudice or favouritism toward an individual or a group based on their inherent or acquired characteristics" [40]. Biases in a system can render it unfair, and result in different individuals or groups of individuals being treated differently. If individuals belong to a protected class (e.g., based on their nationality, gender, socio-economic status or age) and they experience differential treatment that disadvantages them based on their membership in that class, then this is considered discrimination and can carry legal consequences [12]. An example of discrimination is denying an individual a home loan based on their gender. Bias can (but does not necessarily) lead to discrimination. We consider systems to be fairer and more inclusive if they are less biased. Practically, building ML systems that have no bias is difficult and maybe even impossible. However, quantifying and reducing bias are attainable and important steps toward building more inclusive and fairer ML systems.

2.1.2 Bias Measures. To measure bias, researchers in ML have quantified fairness measures that operationalize fairness definitions. Fairness definitions are categorized as measuring individual or group fairness [40]. Individual fairness measures require that similar people are treated similarly, while group fairness measures require that different groups are treated similarly. Verma and Rubin [63] broadly categorise fairness measures into statistical (parity) measures, similarity-based measures and causal reasoning. Most statistical measures rely on metrics that calculate various ratios from error rates and prediction outcomes. A bias evaluation then establishes if metrics are equal for members of protected and unprotected groups. Equalised odds [19], for example, is a fairness measure that establishes if protected and unprotected groups have the same false-negative and false-positive error rates.

Fairness measures can be further categorized as bias preserving and bias transforming, based on the measure's treatment of historic biases [65]. Parity-based fairness measures require equal error rates between groups of people. They are considered bias preserving as they propagate historic bias, for example, through data labelling decisions, which can replicate a biased worldview. Wachter et al. [65] argue that to support the objective of substantive equality in European non-discrimination law, fairness measures should be bias transforming. However, if labels can be exactly known, no historic bias exists, known performance disparities are legally justified, or where systems are designed to replicate social bias, for example, for the purpose of debugging, then bias preserving fairness measures are sufficient. In many on-device ML applications, such as

wake-word detection, keyword spotting, object detection and speaker verification, data labels are exactly known and undisputed. We consider parity-based measures as useful measures of bias in this context. In Section 4, we introduce a parity-based bias measure that we use to quantify performance disparities between user groups in on-device ML.

2.2 Bias in the Machine Learning Workflow

2.2.1 Evidence of Bias in Decision-making Systems. The algorithmic fairness literature has focused predominantly on studying bias in ML systems for classification tasks, with a particular view toward the proliferation of decision-making systems that increasingly dominate public life [40]. Many studies have revealed evidence of bias in ML applications, ranging from natural language processing [5] and gender classification [6] to face recognition [49] and automatic speech recognition [31, 58]. As on-device ML is used for similar tasks, and leverages algorithmic approaches and data processing techniques from ML, it is necessary to investigate bias in on-device ML.

2.2.2 Bias as a Concern for ML System Quality. Recent work in software engineering has highlighted the need to model quality aspects of ML systems in detail [54]. Bias has been identified as a new concern affecting ML software quality that should be considered as a non-functional requirement during development [25]. In requirements engineering and quality modeling, bias considerations are allocated to data-related aspects [54, 64]. However, from the perspective of statistical learning problems, bias can come from the training data, the predictive model and the evaluation mechanism [41]. The engineering and design nature of on-device ML requires an expanded view on bias to what is currently offered by the software engineering and statistical learning perspectives individually. First, the bias of a component cannot be considered in isolation but must be considered within the evolving and dynamic system in which it is incorporated. Second, bias is not only a data concern. In reality, bias can emerge at different stages in the ML workflow and create reinforcing feedback loops [53]. This article expands current perspectives on bias in software engineering by studying how design decisions in the ML workflow influence bias.

2.2.3 Bias Propagation through Design Choices and System Composition. Design choices play an important role in mitigating or propagating bias in the ML workflow. Mehrabi et al. [40] consider bias mitigation measures at different stages of the ML workflow: during pre-processing, in-processing and post-processing. This perspective aligns well with studies that consider the effects of design choices on ML bias in software engineering. For example, Hort and Sarro [26] show how choosing thresholds for categorical data, a pre-processing decision, can impact both the degree of bias, and which groups are favoured. The Fair-SMOTE algorithm, however, is a pre-processing intervention that removes biased data labels and balances the training data distribution based on sensitive attributes and class labels [8]. In on-device ML settings, trained ML models also undergo multiple post-processing steps to overcome resource constraints for on-device deployment and distribution shifts due to context heterogeneity. Some of these post-processing steps, like domain adaptation [55] and model compression [24], have been found to be biased.

Holstein et al. [23] have observed that developers can feel a sense of unease at the societal impacts that their technical choices have, while Toussaint et al. [60] have shown that early collaboration between clinical stakeholders and AI developers is important to guide design decisions to support social objectives within the public health sector. Dobbe et al. [16] examine the impact of design choices on safety in AI systems for socio-technical decision-making in high-stakes social domains. Rather than looking at specific low-level technical choices in the ML workflow, they consider situations in which technical choices that promote different values are difficult to compare. They argue that developers ought to adopt diagnostic practices to proactively anticipate these

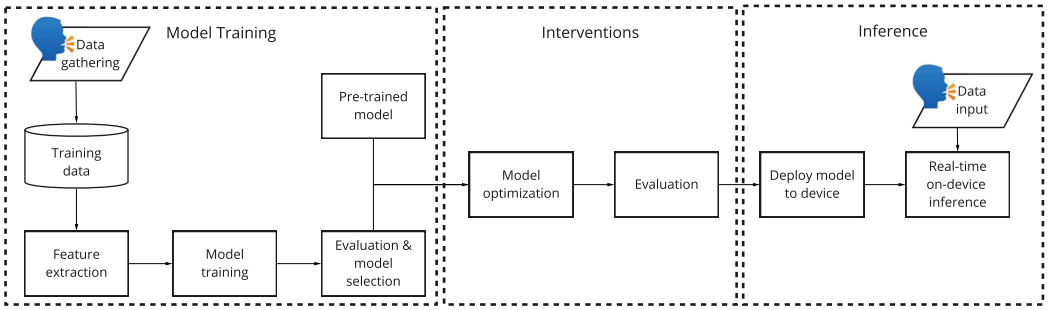

Fig. 1. Data processing pipeline for on-device machine learning development.

choices and resolve them through feedback with stakeholders. Neglecting to do this, the authors further argue, gives rise to socio-technical gaps, where the technical functions do not satisfy the social requirements of AI systems. Drawing on these perspectives, this article considers pre- and post-processing design decisions that arise in the inherently constrained on-device ML context, and examines the extent to which a relatively comprehensive set of design choices can support the social requirement for inclusive on-device ML.

3 ON-DEVICE MACHINE LEARNING SYSTEMS

Having laid the foundations for bias in ML, and the importance of design choices in propagating or mitigating bias in ML development, we turn our focus to on-device ML. Heterogeneous devices, diverse users, and unknown usage environments make the performance of on-device ML highly context dependent. During development, engineers are faced with a large number of decisions to choose interventions that overcome hardware constraints and meet operational demands. Collectively, constraints and context-dependency make on-device ML development a complex engineering undertaking that requires mastery of hardware, software engineering, and data processing techniques, alongside an in-depth understanding of the application context. In this section, we provide an overview of the data processing workflow for on-device ML systems, and highlight the various constraints, intervention strategies and design choices that an engineer encounters while designing on-device ML systems in practice.

3.1 Data Processing Workflow for On-device ML

The key processing steps during on-device ML development are model training, interventions, and inference. A typical data processing pipeline for on-device ML, as shown in Figure 1, consists of familiar ML processing steps for model training, evaluation, selection and inference. Key differences between on-device ML and cloud-based ML development arise due to the low compute, memory and power resources of end devices [14]. To enable on-device inference, interventions are needed to optimize a trained model and its data processing pipeline for on-device deployment. These aspects are described in greater detail below.

3.1.1 Training. The dominant approach for developing on-device ML is to delegate resource-intensive model training to the cloud, and to deploy trained and optimized models to devices [14]. The approach for training models is similar to typical ML pipelines: input data is gathered and undergoes a number of pre-processing operations to extract features from it. Thereafter, ML models are trained, evaluated and selected after optimizing a loss function on the data. Pre-trained models can also be downloaded and used if training data or training compute resources are not available.

3.1.2 Interventions. The key differences between on-device ML and cloud-based ML development arise due to the low compute, memory and power resources of end devices [14]. To enable on-device deployment of the trained model, various *interventions* are needed to optimize the model and its data processing pipeline. Common interventions include techniques such as model pruning, model quantization, or input scaling; all of which are aimed at optimizing device-specific performance metrics such as response time or latency [2], memory consumption [17], or energy expenditure [68] with minimal impact on the model's accuracy. We elaborate on these intervention approaches in Section 3.2.

3.1.3 Inference. Once deployed, the trained and optimized model is used to make real-time, on-device predictions. On-device inference performance is determined by the model training process, from data collection to model selection, and the real-time sensor data input, but also by deployment constraints and interventions applied to the model.

3.2 Design Choices in On-device ML Engineering

Building on the on-device data processing workflow that we described, we now discuss the key design choices that an engineer has to make in this workflow. We first explain the constraints of on-device ML that necessitate these design choices, and thereafter discuss the various interventions that can be taken to satisfy these constraints. We also highlight how these interventions could impact the accuracy and bias of on-device ML models.

3.2.1 Deployment Constraints. On-device ML development needs to take into account the limited memory, compute and energy resources of the end devices [14]. The available *storage and runtime memory* on a device limits the size of the ML models that can be deployed on it. The execution speed of inferences on the device is directly tied to the available *compute* resources. Moreover, the amount of computations required by a model has a direct relation to its energy consumption; given that many end devices are battery powered with limited *energy* resources, it becomes imperative that ML models operate within a reasonable energy budget. In addition to these resource constraints, on-device ML also has to deal with *variations in the hardware and software stacks* of heterogeneous user devices [3]. For instance, prior research [38] has shown that different sensor-enabled devices can produce data at different sampling rates owing to their underlying sensor technology and real-time system state. Such variations can impact the quality of sensor data that is fed to the ML model, which in turn can impact its prediction performance.

3.2.2 Interventions. Research in on-device ML is largely concerned with overcoming these constraints and satisfying hardware-based performance metrics while achieving acceptable predictive performance [14]. Prior works have developed interventions to overcome *memory* and *compute* limitations, such as weight quantization [17] and pruning [35]. Other approaches such as input filtering and early exit [27], partial execution and model partitioning [13] allow for dynamic and conditional computation of the ML model depending on the available system resources. Another commonly used alternative to satisfy resource constraints is to design light-weight architectures that reduce the model footprint [7, 68]. Finally, solutions have been proposed to make ML models robust to different resolutions of the input data [42], which is a key to dealing with sampling rate variations in end devices. Common to all these interventions is that they trade-off a model's resource efficiency with its prediction performance. For example, model pruning or the use of light-weight neural architectures can result in a model with smaller memory footprint and faster inference speed; however, it comes at the expense of a slight accuracy degradation [7, 35, 68].

3.2.3 Design Choices. To build on-device ML, software engineers need to navigate deployment constraints and interventions alongside ML training and deployment. This is technically

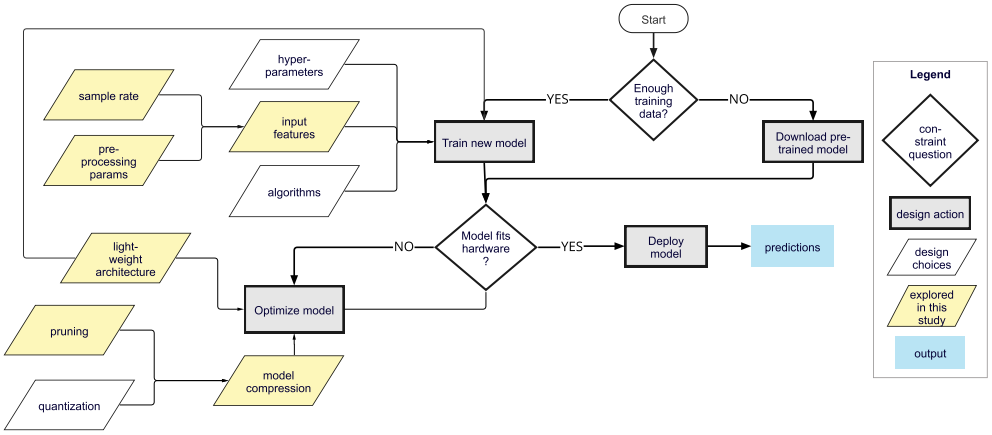

Fig. 2. Decision map of design choices during on-device ML engineering. Yellow chart elements are design choices studied in this article.

challenging and charges engineers with the responsibility to take design actions and make design choices at each development step. Importantly, as on-device deployment constraints require interventions in the development process, design choices like the choice of model architecture, sample rate, input features, and model compression techniques affect predictive and hardware performance, as well as bias [59]. Even though some design choices can be optimized through automated experimentation, iterating through all possible values requires extensive computing resources and time. This increases the cost of training. In countries and contexts where the low cost of on-device ML is a key enabler and driver for technology adoption [29] increased training costs reduce technology access. Moreover, each design choice can introduce bias into the system. If time or compute are limited, then engineers may need to limit the extent of their experimentation and only focus on a small set of choices.

We visualize some of the key design choices as a decision map in Figure 2. The availability of training data is a logical starting point during development, as it determines whether a new model can be trained, or if a pre-trained model must be downloaded. Once an engineer commits to the design action of training a new model, they are confronted with design choices to select an algorithm, hyper-parameters, input features, pre-processing parameters and a data sample rate. After training or downloading the model, the engineer needs to determine if it fits within the memory, compute and power budget. If it does, then they can deploy the model to make predictions. If the model does not fit within the hardware budget, then the engineer must take design actions to optimize the model and reduce its resource requirements. This can be done through interventions like training a more light-weight architecture or compressing the model. These choices present further sub-choices, for example model compression can be done with pruning, quantization, or both. In comparison to quantization, pruning involves more hyper-parameters and thus requires more design choices. Each design choice modifies the model and has the potential of introducing bias in its predictions.

4 BIAS IN ON-DEVICE ML

On-device ML systems are deployed on billions of personal and low-resource devices that continuously capture and monitor individuals or groups of people and the environment. In such **cyber-physical systems (CPS)** of distributed devices, ML functions mechanistically and is constructed

from and activated by personal data collected with sensors. ML models can be seen as technical components of CPS, which along with other hardware and software components affect the system's functionality. Adopting a well-established definition[1] of reliability for trustworthy CPS [11], we posit that to be reliable and trustworthy, on-device ML should "deliver stable and predictable performance in expected [operating] conditions." Instead, if ML components fail unexpectedly, then devices become unreliable and users can be inconvenienced or even harmed.

Thus, we define an on-device ML model as biased if it causes devices to have disparate performance across user groups based on their personal or sensitive attributes. This can lead to devices that unexpectedly fail for some users, even if they deliver stable and predictable performance for others. If unexpected failures systematically affect particular users, then these users are subjected disproportionately to harms that result from device failure. Biased on-device ML components thus lead to *reliability bias* of a device, which then becomes a source of unfairness in the CPS. While a number of factors can lead to biased ML components, the focus of this work is to study how design choices during on-device ML development (see Figure 2) impact the model's accuracy for different demographic user groups, and thus propagate reliability bias.

4.1 Quantifying Reliability Bias

We consider an on-device ML model a reliable device component for a group if the group's predictive performance equals the model's overall predictive performance across all groups. If a model performs better or worse than average for a group, then we consider it to be biased, showing favour for or prejudice against that group. Both favouritism and prejudice increase *reliability bias*. We want to operationalize *reliability bias* with a measure that captures these definitions and penalizes favouritism and prejudice equally. Additionally, the measure should be able to score models as being more or less biased, and should consider positive and negative prediction outcomes. Given these requirements, we first define bias of a model with respect to a group i ($i = 1 \ldots N$) as

$$bias_i = ln\left(\frac{performance_i}{performance_{overall}}\right), \tag{1}$$

where $performance_i$ is a metric computed for data samples belonging to the ith group, and $performance_{overall}$ is computed for all samples in the test set. $bias_i$ is 0 when a model is unbiased toward group i, negative when it performs worse than average and positive when it performs better than average for the group. The magnitude of the measure is equal for a performance ratio and it's inverse, as $ln(x) = -ln(\frac{1}{x})$. This has intuitive appeal that supports the interpretability of our measure: $bias_i$ is equal in magnitude but has opposing signs for groups that perform half as good and twice as good as average. Given the group bias scores, *reliability bias* is the sum of absolute score values across all groups:

$$reliability\ bias = \sum_{i=1}^{N} |bias_i|. \tag{2}$$

In this article, we assume that all groups are equally important. The *reliability bias* measure is thus unweighted and does not take group size into consideration. *Reliability bias* has a lower bound of 0, and an infinite upper limit. Lower scores are preferred and signify that the performance across all groups is similar to the overall performance. We now turn toward an empirical audio **keyword spotting (KWS)** study to show how design choices in the on-device ML workflow propagate *reliability bias*.

[1]This definition has been adopted in the Cyper-Physical Systems Framework proposed by the National Institute of Standards and Technology in the United States.

5 A STUDY ON BIAS IN ON-DEVICE AUDIO KEYWORD SPOTTING

In the remainder of the article, we examine the propagation of bias through design choices in on-device audio KWS. KWS systems, which activate voice-based interactions with digital services [50] on smart speakers and smart phones, are a prominent use case of on-device ML [2]. Voice-based service activation can be particularly beneficial for increasing access to digital services for individuals who suffer from restricted vision, mobility and movement, and for emergency response, home and elderly care. Many commercial products now exist that provide voice-activated urgent response with on-device KWS (e.g., "call help") [47, 62]. Users place confidence in these products to support them in moments of crisis and provision them with access to critical care services. Despite the evident societal promise of on-device KWS, human speech signals exhibit variability based on social and physiological attributes of the speaker [18]. This makes it essential to ensure that systems work reliably for all users irrespective of their personal attributes such as age and gender. A starting point for ensuring inclusive on-device audio KWS is to evaluate inference performance for speaker groups with different attributes. In this study, we consider groups based on gender.

5.1 Overview of Audio Keyword Spotting Task

An audio keyword spotting system as shown in Figure 3 takes a raw speech signal as input and outputs the keyword(s) present in the signal from a set of predefined keywords. Next, we describe the end-to-end training and inference pipeline for a KWS system, while highlighting (**in bold**) the various design choices (refer to Figure 2) available to an ML engineer in this task.

First, a raw speech signal is sampled from the microphone at a predefined **sample rate** (e.g., 8 KHz, 16 KHz) and split into overlapping, short time duration frames using a sliding window approach. This framing operation requires specifying a number of **pre-processing parameters**, which include (i) *Frame length* that defines the duration of each frame, (ii) *frame step* that indicates the step size by which the sliding window is moved, and (iii) *window function*, which helps in reducing spectral leakage in **Discrete Fourier Transform (DFT)**. Thereafter, each frame of the speech signal is transformed into **input features**: first, we apply a DFT to each frame to obtain log-scaled filter bank features known as *log Mel spectrograms*. Optionally, log Mel spectrograms can be decorrelated using a Discrete Cosine Transform to generate **Mel Frequency Cepstral Coefficiencts (*MFCCs*)**. The number of log Mel spectrograms and MFCCs is also a designer-chosen parameter, often tuned empirically. Finally, the frame-level features (log Mel spectrograms or MFCCs) are concatenated across frames and mean-normalized to form a two-dimensional representation of the speech signal, which is used to train a deep neural network classifier, as described in Reference [9]. This process also involves choosing an appropriate **neural network architecture** that satisfies the resource constraints of the deployment device. Optionally, an ML engineer can also choose to optimize the trained neural network by applying various **model compression** techniques such as weight pruning (see Section 3.2)

5.2 Impact of Design Choices and Choice Variables

We now contextualize the design choices in KWS along the lines of the on-device ML workflow presented in Section 3, and explain how prior literature has navigated them.

First, the *sample rate* can be seen as a deployment constraint due to hardware limitations such as microphone capabilities [42]. Prior works [15, 57] have also used sample rate as a tunable parameter to adjust the power consumption on an embedded device during data collection. However, to our knowledge, there has been no study on how the choice of sample rate affects bias of ML models, other than our prior work [59], which is extended by this research.

The choice of audio *pre-processing parameters* is known to have an impact on the performance of a KWS model in an embedded system [42]. Frame length and frame step together determine

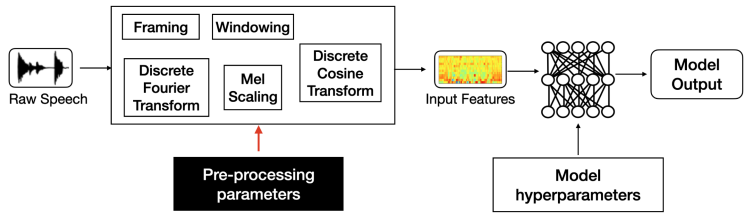

Fig. 3. Audio processing pipeline during training and inference.

the temporal dimension of the 2D features that are fed to a DNN, and the number of log Mel spectrogram or MFCC features determine the length of features in each time segment. Together, these features influence the dimensions of the input data to the model, which in turn impacts the number of computations during inference. This insight was used in a recent work named ePerceptive [42], wherein the authors experimented with different values of *frame step* to achieve a good trade-off between inference accuracy and latency. However, there has been no prior work that has explored potential accuracy-bias trade-offs due to pre-processing parameters.

Unsurprisingly, the *model architecture* plays an important role in the performance of a KWS system. Performing inferences on model architectures with fewer parameters takes less time, but could lead to accuracy degradation. On the contrary, deeper models with a large number of parameters might provide better accuracy, at the expense of higher inference latency. Prior KWS works [1, 9, 21, 22, 61, 69] have experimented with different architectures to achieve a good accuracy-latency trade-off. However none of these studies have evaluated bias in KWS systems due to the choice of model architecture.

Finally, while compressing a trained model for deployment using model pruning, a developer needs to specify a number of parameters such as final sparsity, pruning frequency, schedule, and learning rate. *Final sparsity*, specified in percentage, determines the proportion of weights that will be set to 0 during model pruning. Indeed, a high 'final sparsity' leads to more compressed models, which result in lower storage requirements and reduced inference latency on the device [33]. A *pruning frequency* of "n" indicates that the model should be pruned after every "n" training steps. *Pruning Schedule* can take two values: (i) constant sparsity, which indicates the fixed sparsity level of the model throughout the training, or (ii) polynomial decay, where the pruning sparsity grows rapidly in the beginning from initial_sparsity, but then plateaus slowly to the final sparsity. Finally, *Pruning Learning Rate* controls the step size taken by the model optimizer (e.g., Adam or Stochastic Gradient Descent) during backpropagation. Prior literature on neural network pruning primarily investigates the impact of final sparsity on model accuracy [33] and does not shed light on the impact of other pruning parameters. However, given that these parameters also constitute important design decisions during model optimization, we choose to include them in our experiments. To our knowledge, the effect of pruning parameters on KWS models running on-device has not been studied.

5.3 Experiment Design

Having established that prior KWS literature has not adequately studied the impact of design choices on bias, we set up experiments to investigate design choices related to important design actions for on-device ML: model training and model optimization. Guided by the on-device ML development workflow presented in Section 3 and our prior work [59], we aim to answer the following research questions within the context of an audio KWS task:

(1) How does the choice of architecture affect *reliability bias* and accuracy?
(2) How does the audio sample rate affect *reliability bias* and accuracy?

Table 1. Overview of Design Choice Variables and Values for the Audio Keyword Spotting Study

Design action	Design choice	Choice variable (unit)	Variable values
Train new model	input features | sample rate	sample rate (kHz)	8, 16
Train new model	input features | pre-processing	feature type	log Mel spectrogram, MFCC
Train new model	input features | pre-processing	# Mel filter banks	20, 26, 32, 40, 60, 80
Train new model	input features | pre-processing	# MFCCs	None, 10, 11, 12, 13, 14
Train new model	input features | pre-processing	frame length (ms)	20, 25, 30, 40
Train new model	input features | pre-processing	frame step (% frame length)	40, 50, 60
Train new model	input features | pre-processing	window function	Hamming, Hann
Optimize model	light-weight architecture	model architecture	CNN, low-latency CNN [51]
Optimize model	model compression | pruning	final sparsity (%)	20, 50, 75, 80, 85, 90
Optimize model	model compression | pruning	pruning frequency	10, 100
Optimize model	model compression | pruning	pruning schedule	constant sparsity, polynomial decay
Optimize model	model compression | pruning	pruning learning rate	1e-3, 1e-4, 1e-5

(3) How do pre-processing parameters affect *reliability bias* and accuracy?
(4) How do pruning parameters affect *reliability bias* and accuracy?

The various design choices and choice variables are summarized in Table 1 and explained next.

As discussed earlier, the *neural network architecture* is an important design choice during model training. We experiment with two **convolutional neural network (CNN)** architectures for KWS, originally proposed in Reference [51] and later implemented in the TensorFlow framework. The architecture that we refer to as *CNN* consists of two convolutional layers followed by one dense hidden layer, while the **low-latency CNN (*llCNN*)** consists of one convolution layer followed by two dense hidden layers. The authors in Reference [51] showed that the llCNN architecture, by virtue of having less convolution operations, is more optimized for on-device KWS.

Next, we study choices that affect the input features of the model, namely *sample rate* and *pre-processing parameters*. Audio keyword spotting developer benchmarks often use a 16 kHz audio input [39, 67]. In practice many devices collect data at a lower sample rate of 8 kHz [42] due to hardware constraints. We thus train models with audio data at two sample rates, 16 and 8 kHz, for both architectures.

For studying the impact of pre-processing parameters, we take inspiration from prior KWS literature [1, 9, 21, 22, 61, 69], and experiment with two feature types, log Mel spectrograms and MFCCs. More specifically, we vary the dimensionality of log Mel spectrograms from 20 to 80, and of MFCCs from 10 to 14. We also consider log Mel spectrograms that are used directly as input features, with no MFCCs. Further, we experiment with three temporal pre-processing parameters: frame length (20–40 ms), frame step (40%–60% overlap) and the window type (Hamming/Hann); these values are based on prior on-device KWS works [1, 21, 22, 42, 69].

With regards to model optimization, we focus on model compression, in particular *parameter choices during post-training pruning*. Based on prior literature [33, 34], we vary the pruning sparsity from 20% to 90%. For pruning schedule, we experiment with both constant sparsity and polynomial decay as explained in Section 5.2. For learning rate, we choose three values based on prior KWS literature on model training [1, 9, 21, 22, 61, 69]. For pruning frequency, we used two values: 100 (the default frequency in TensorFlow) and a faster option of 10, wherein the pruning operation takes place after every 10 training steps.

5.4 Experiment Setup

5.4.1 Datasets. We trained and evaluated models on the following five spoken keywords datasets spanning four languages—English, German, French, and Kinyarwanda:

Table 2. Audio Keyword Utterance Count (and % of Total Dataset) Across Dataset Splits for MSWC English, MSWC German, MSWC French, and MSWC Kinyarwanda Datasets

dataset split	MSWC English	MSWC German	MSWC French	MSWC Kinyarwanda
female training	79,002 (39%)	34,728 (41%)	31,127 (41%)	20,713 (39%)
male training	79,611 (40%)	34,329 (40%)	30,276 (40%)	21,786 (41%)
female validation	10,496 (5.2%)	4,613 (5.4%)	2,790 (3.7%)	3,580 (6.7%)
male validation	10,238 (5.1%)	3,976 (4.6%)	3,601 (4.8%)	1,801 (3.4%)
female test	10,816 (5.4%)	3,445 (4%)	3,905 (5.2%)	2,511 (4.7%)
male test	10,465 (5.2%)	4,481 (5.2%)	3,945 (%)	3,217 (6%)
total	200,628	85,572	75,644	53,608

Google Speech Commands (*google_sc*) [67] is an English language dataset consisting of 104,541 spoken keywords from 35 keyword classes such as *Yes, No, One, Two, Three*, recorded by volunteer contributors and released at a 16 kHz sample rate. We labeled every utterance as male or female using a crowd-sourced data labelling campaign conducted on Amazon Mechanical Turk. We used the same train, validation and test set splits of 85%, 10%, and 5%, respectively, from the original dataset. Female speakers constituted 30% of the original training data, 32% of the validation and 29% of the test data. During training, we thus ensured that mini-batches have an equal balance of male and female speakers by randomly sampling from the male training set.

Multilingual Spoken Words Corpus Datasets [39]. The **Multilingual Spoken Words Corpus (MSWC)** is a large corpus of spoken words in 50 languages, originally sampled at 48 KHz. Each language partition contains hundreds of hours of audio data with tens of thousands of keyword classes. MSWC has been derived from Mozilla Common Voice[2] by splitting the crowd-sourced, read-speech corpus into individual words. We chose four of the languages with the largest data resources in MSWC to create four keyword spotting datasets in different languages: **MSWC English** (*mswc_en*), **MSWC German** (*mswc_de*), **MSWC French** (*mswc_fr*), and **MSWC Kinyarwanda** (*mswc_rw*). Each of the MSWC datasets was created with data from its language partition, and a consistent approach to select keywords, balance data across male and female speakers, and split the dataset into train, validation and test splits.

For each dataset, we selected keywords from the 35 largest keyword classes to create training datasets that are equivalent to Google Speech Commands. Following the keyword selection strategy of the authors of the MSWC dataset, we only selected keywords with more than three characters. Additionally, if two words started with the same three letters, then we only selected the first occurring word. This resulted in a total of 200,628 keyword utterances for MSWC English, 85,572 keyword utterances for MSWC German, 75,644 keyword utterances for MSWC French, and 53,608 keyword utterances for MSWC Kinyarwanda. The dataset sizes vary based on the language representation in the Mozilla Common Voice corpus.

To ensure gender-balanced datasets, we only used keyword utterances where the gender metadata field was male or female. The gender metadata in Mozilla Common Voice has been provided by data donors and thus corresponds with the self-identified gender of the speaker. For each keyword, we counted the utterances per gender. We included all utterances/keyword of the gender with fewer utterances/keyword, and randomly sampled the same number of utterances/keyword from the gender with more utterances/keyword. We joined the selected data for both genders and all keywords, before splitting the data into train, validation and test sets. To create the dataset splits, we followed the protocol described in Reference [39] as closely as possible while enforcing

[2]https://commonvoice.mozilla.org/.

Table 3. Number of Baseline Models Pruned Per Dataset, Architecture, and Sample Rate

	Google SC	MSWC German	MSWC English	MSWC French	MSWC Kinyarwanda
16 kHz CNN	9	9	9	9	6
16 kHz llCNN	8	9	9	9	7
8 kHz CNN	9	9	9	9	7
8 kHz llCNN	9	9	9	9	6

gender-balance. We first created a list of unique keyword-speaker pairs so that train, validation and test sets are separate. Next, we randomly sampled 80% of keyword-speaker pairs for training. We then randomly sample 10% of keyword-speaker pairs for validation, excluding pairs already in the training set and rounding to the nearest integer. Finally, we allocated the remaining keyword-speaker pairs to the test set. The keyword utterance count and representation for male and female genders across dataset splits are shown in Table 2. During training, we ensured that mini-batches have an equal balance of male and female speakers.

5.4.2 Training Details. Our training setup is implemented in Tensorflow 2.0, and we used a Nvidia V100 GPU to train the models. For each dataset, we iteratively trained models with all combinations of model architectures, sample rates and pre-processing parameters listed in Table 1. This resulted in 3,456 candidate models per dataset, and a total of 17,280 experiments across five datasets. We used the TF HParams API[3] for tuning the training-time learning rate for each model from the following three options: {1e-2, 1e-3, 1e-4}. We used the Adam optimizer for training, with a fixed batch size of 128 samples. Each model was trained for 10 epochs, which was chosen based on empirical evidence that the model performance did not improve beyond 10 epochs.

Thereafter, we used model selection criteria that consider accuracy and bias (discussed in detail in Section 7.1) to select baseline models for model compression. Table 3 lists the number of baseline models selected per dataset for the pruning experiments. For the Google SC and MSWC Kinyarwanda datasets, we could not find models that met all our selection criteria across architectures and sample rates, which is why fewer baseline models were selected for these datasets. We then obtained the compressed version of the baseline models under each combination of pruning parameters listed in Table 1. As with training, we also used 10 epochs for pruning. Our pruning experiments resulted in 72 pruned models for each baseline model, and a total of 12,168 experiments.

5.4.3 Evaluation Protocol. As ground truth labels in audio KWS can be exactly known and are unambiguous, we assume that labels are correct and unbiased. We can thus evaluate bias with the parity-based *reliability bias* measure, which we defined in Equation (2). For our experiments, we compute two evaluation metrics for each model on the held-out test set: (i) reliability bias and (ii) accuracy. For accuracy, we compared five different metrics: Cohen's kappa coefficient, precision, recall, weighted F1 score, and the **Matthews Correlation Coefficient (MCC)**. The trends we observed are consistent across metrics. Thus, we only report accuracy results for the MCC, which is a robust metric for multiclass classification.

6 EMPIRICAL RESULTS AND ANALYSIS

In this section, we present the results of our study and analyze the impact of design choices on reliability bias and accuracy during different stages of the on-device audio KWS workflow. We start

[3]https://www.tensorflow.org/tensorboard/hyperparameter_tuning_with_hparams.

Table 4. Significant Main and Interaction Effects of Model Training Design Choices on **MCC (Accuracy)**

Factorial ANOVA main and interaction effects	**SS**	**df**	**F**	**p(<0.05)**
model architecture	31.9714	1	6.0103E+04	0.0E+00
sample rate	3.4011	1	6.3938E+03	0.0E+00
dataset	160.1572	4	7.5270E+04	0.0E+00
mfccs	17.2272	5	6.4771E+03	0.0E+00
mel filter banks	3.1283	5	1.1762E+03	0.0E+00
frame step	0.1500	2	1.4103E+02	1.8E-61
model architecture * mel filter banks	0.0202	5	7.6075E+00	3.8E-07
dataset * sample rate * mfccs	0.1425	20	1.3391E+01	7.0E-45
dataset * model architecture * mfccs	0.1056	20	9.9288E+00	3.5E-31
Residual	9.100465	17,108	—	—
Model	—	171	2.5277E+03	0.0E+00
R^2**:**	0.9619			
Adjusted R^2**:**	0.9615			

SS = sum of squares, df = degrees of freedom.

with design choices that arise during model training, first analyzing the impact of the architecture and sample rate, then the impact of pre-processing parameters. After that, we consider model optimization design choices that arise during model compression, namely, the impact of pruning hyper-parameters.

6.1 Design Choices during Model Training

To analyze the impact of the pre-processing parameters, we performed factorial ANOVA tests that allow for interactions [44] on our balanced study design. This type of statistical test is used to determine the influence of two or more independent variables on one dependent variable, which makes it suitable for our study. We coded deviation (or sum) contrasts and used type 3 sums of squares. The analysis was done in python using the *scipy stastmodels* package and is available as a jupyter notebook on github.[4] Given the large number of possible interactions between our independent variables (i.e., choice variables in Table 1), we designed the first factorial ANOVA model (see Model 1 in the Appendix) to consider a subset of interactions that we deemed important for accuracy and reliability bias of KWS models based on prior visual analysis. We continued to improve the factorial ANOVA models separately for the two dependent variables (MCC (accuracy) and reliability bias) by removing all non-significant interactions, and then including lower-level interactions. The final ANOVA models are included in the Appendix, with Models 2 and 3 capturing variables and interactions of model training design choices on the accuracy score and reliability bias, respectively.

Tables 4 and 5 show statistically significant interaction and main effects of the final factorial ANOVA models. For completeness, we have included main effects even if they already contribute to an interaction. The final factorial ANOVA models are significant (MCC (accuracy): $F(171) = 2527.2$, $p = 0.0$, $R^2_{adj.} = 0.9615$ and reliability bias: $F(92) = 367.95$, $p = 0.0$, $R^2_{adj.} = 0.6615$). For reference, the critical F statistics at p-values less than 0.01 and 0.05 are shown in Table 6. Based on the F statistics, we reject the null hypothesis that neither design choices made during model training, nor their interactions affect KWS model accuracy and reliability bias. The R^2 values indicate that the accuracy ANOVA model ($R^2 = 0.9619$, $R^2_{adjusted} = 0.9615$) better captures the effects than the

[4] https://github.com/akhilmathurs/fair-ondevice-ML.

Table 5. Significant Main and Interaction Effects of Model Training Design Choices on **Reliability Bias**

Factorial ANOVA main and interaction effects	**SS**	**df**	**F**	**p(<0.05)**
model architecture	0.96734	1	469.248554	1.10E-102
sample rate	0.477009	1	231.393075	6.45E-52
dataset	62.4386	4	7.5721E+03	0.0E+00
mel filter banks	0.1225	5	1.1887E+01	1.7E-11
dataset * sample rate	0.7840	4	9.5078E+01	3.9E-80
dataset * mel filter banks	0.3758	20	9.1140E+00	5.1E-28
dataset * model architecture * mfccs	0.9662	20	2.3436E+01	1.8E-85
Residual	35.4366	17,190	—	—
Model	—	92	3.6795E+02	0.0E+00
R^2**:**	0.6633			
Adjusted R^2**:**	0.6615			

SS = sum of squares, df = degrees of freedom.

Table 6. Critical F-values for Determining Significance at $p < 0.01$ and $p < 0.05$ for Different Degrees of Freedom (df)

df	1	2	4	5	8	10	20	40
F_{crit} $(p < 0.01)$	4,052.1807	98.5025	21.1977	16.2582	11.2586	10.0443	8.0960	7.3141
F_{crit} $(p < 0.05)$	161.4476	18.5128	7.7086	6.6079	5.3177	4.9646	4.3512	4.0847

reliability bias ANOVA model ($R^2 = 0.6633$, $R^2_{adjusted} = 0.6615$), in which a portion of variance in the dependent variable remains unaccounted for. Next, we examine the impact of the model architecture and sample rate, and of the pre-processing parameters in greater detail.

6.1.1 Impact of Model Architecture and Sample Rate. The results of the statistical tests in Tables 4 and 5 show that model architecture and sample rate contribute to significant interaction effects that impact accuracy and reliability bias. We now examine how these two metrics are affected by variable values and their interactions.

Figure 4 shows a boxplot of accuracy and reliability bias for CNN and light-weight llCNN architectures trained on 16 and 8 kHz audio data. A higher MCC score implies better prediction performance. The trends in accuracy scores for models trained with different architectures and sample rates are consistent across datasets. CNN and llCNN architectures trained at 8 kHz have a lower median accuracy score (i.e., they are worse) than those trained at 16 kHz, and CNN architectures have higher scores than their light-weight counterparts. While models trained on the mswc_rw dataset still follow this trend, their performance, in general, is considerably worse than that of the other models. Possible reasons for this are that less training data was available for these models, and Kinyarwanda is a different language family than the languages in the other datasets. It is out of the scope of this study to consider bias due to language and accent, which remains an important area for future work.

For reliability bias, we observe that median scores are higher (i.e., worse) for models trained at 8 kHz than those for models trained at 16 kHz. For the google_sc, the mswc_de, and the mswc_fr datasets, models trained at lower sample rates also have a higher **interquartile range (IQR)** in reliability bias scores. The light-weight llCNN architecture tends to have a higher median reliability bias and greater IQR than the CNN architecture, but the effect is not as pronounced as for accuracy. Models trained on the mswc_rw dataset do not follow these trends. While median reliability bias

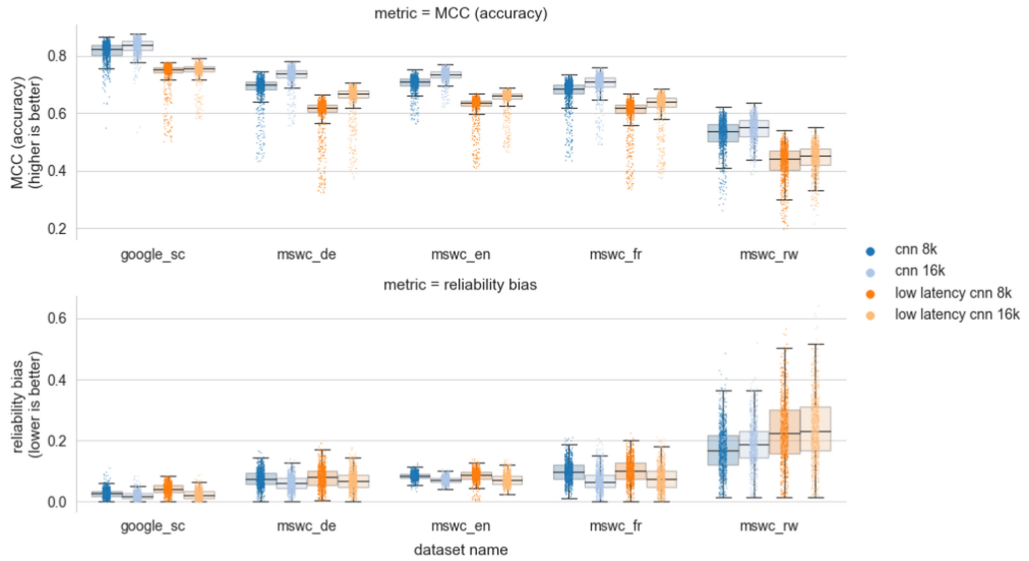

Fig. 4. Experimental results of MCC (accuracy) and *reliability bias* for CNN and low-latency CNN model architectures with 16 and 8 kHz sample rates trained on five different datasets.

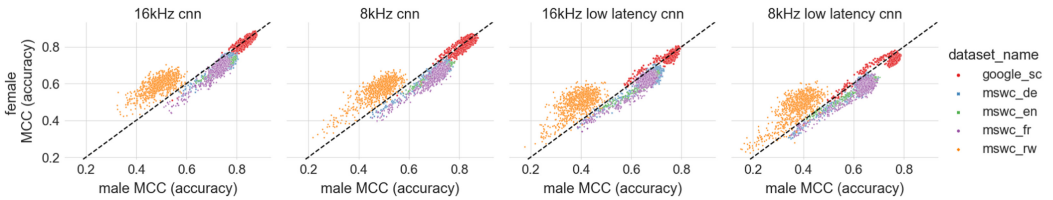

Fig. 5. Disaggregated MCC (accuracy) scores for males (x-axis) and females (y-axis) for a single model trained with a unique combination of pre-processing parameters. On the black diagonal the performance is equal for both subgroups.

of CNN models is lower than that of llCNN models, 8 kHz models are also less biased than 16 kHz models. We anticipate that the deviation between trends observed for the mswc_rw models and the remaining models contributes significantly to the large effect size of the dataset variable that we observe in Tables 4 and 5.

Delving deeper into these findings, we analyze the relationship between male and female MCC scores across architectures and sample rates in Figure 5. Each data point represents the disaggregated male and female accuracy scores of a single model trained with a unique combination of pre-processing parameters. The dotted black diagonal represents equal performance for male and female speakers. Points above the diagonal perform better for females, and points below perform better for males. For the mswc_de, _en, and _fr datasets it is evident that accuracy scores are biased to favour male speakers. For the mswc_rw dataset, models always favour female speakers. For the google_sc dataset the results are more nuanced. Models trained with CNN architectures tend to favour male speakers, whereas models trained with llCNN tend to favor female speakers. This figure shows that the nature of the training data contributes significantly to bias. We also observe that for each dataset there exist models that lie on or very close to the diagonal. These models have a lower reliability bias than the remaining models. We hypothesize that pre-processing parameters

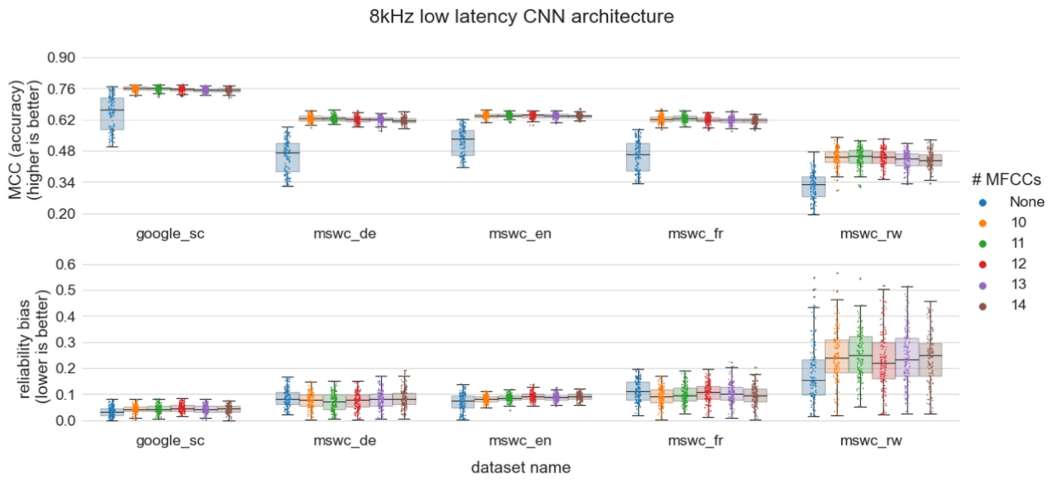

Fig. 6. Effect of MFCC dimensions on accuracy and reliability bias for **8 kHz low-latency CNN** models. Models without MFCC features (blue), i.e., models that directly use log Mel spectrograms as input features, perform considerably worse than those that use MFCC features.

contribute to reliability bias, and thus the distance of experiments from the diagonal. This leads us to the next section, where we analyze the role of pre-processing parameters on accuracy and reliability bias.

Key insights: Model accuracy is lower at lower sample rates and for light-weight architectures. Median and IQR of reliability bias tend to be greater at lower sample rates and for light-weight architectures. The direction of bias is strongly influenced by the training dataset. Overall, male speakers are favoured by models. An exception to this are models trained on the mswc_rw dataset, which have considerably lower accuracy and favour female speakers.

6.1.2 Impact of Pre-processing Parameters. Having studied the effect of the architecture and sample rate, we now turn to pre-processing parameters, the next design choice listed in Table 1. The F statistics and p-values in Tables 4 and 5 indicate that the dimensions of log Mel spectrograms and MFCC features significantly affect accuracy and reliability bias. For accuracy there exist interaction effects between Mel filter banks and architecture, between MFCCs, dataset, and sample rate, and between MFCCs, dataset, and architecture. The latter interaction effect also exists for reliability bias, as well as an interaction effect between Mel filter banks and dataset. Figure 6 visualizes accuracy and reliability bias for the six MFCC and log Mel spectrogram dimensions across all datasets for the 8 kHz low-latency CNN models. As highlighted earlier, the lower sample rate and light-weight architecture result in models that experience greater decline in accuracy and reliability bias. We thus anticipate that the impact of pre-processing parameters is more pronounced for these models.

In Figure 6 the number of MFCC dimensions implies the choice of input feature type. Models with no MFCCs (i.e., # MFCCs = None) use only log Mel spectrograms as input features. It is clear from the figure that the accuracy of models trained with log Mel spectrograms (i.e., the blue boxes) is significantly worse than that of models trained with MFCC input features. For models trained with MFCC features, fewer dimensions (i.e., # MFCCs = 10 or 11) tend to result in a higher median accuracy than more dimensions. However, the impact of this is much smaller than that of using

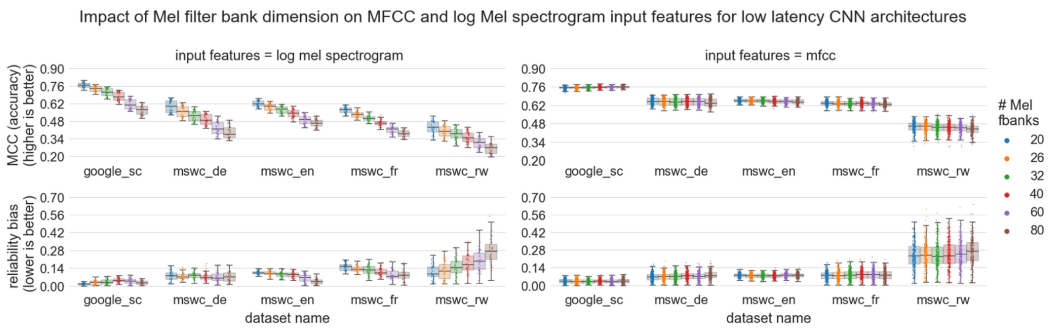

Fig. 7. Effect of log Mel spectrogram dimensions (# Mel fbanks) on accuracy and reliability bias, disaggregated by input feature type for **low-latency CNN** architectures. The number of Mel filter banks clearly impacts models that directly use log Mel spectrograms as input features.

log Mel spectrograms. For reliability bias, we observe mixed results that depend on the training dataset. Google_sc, mswc_en and mswc_rw have a lower median reliability bias when using log Mel spectrograms. However, for the mswc_de and mswc_fr datasets the median reliability bias is lower for models trained with MFCC input features. Figure 13 in the Appendix shows comparable results for 16 kHz CNN models. Here, we still observe that the median accuracy is lower for log Mel spectrogram input features, except for the google_sc dataset. This dataset also has a lower median and smaller IQR of reliability bias scores when using log Mel spectrograms. Overall, the impact of the number of MFCC dimensions and by association the input feature type is less pronounced for CNN models trained at 16 kHz.

Figure 7 visualizes the impact of the number of Mel filter banks on accuracy and reliability bias for low-latency CNN architectures. Figure 14 in the Appendix visualizes results for CNN architectures, which show similar trends. It is clear that when models use log Mel spectrograms directly as input features (left column), the number of Mel filter bank dimensions has a critical impact on accuracy: MCC scores deteriorate rapidly as the number of Mel filter banks increases. The impact on reliability bias is more varied. For the mswc_en and _fr datasets the Mel filter bank dimensions pose a trade-off between accuracy (models with more filter banks are less accurate) and reliability bias (models with more filter banks are less biased). Models trained with the google_sc and mswc_de datasets show no clear trend. Only models trained with the mswc_rw dataset have lower reliability bias for fewer Mel filter banks, thus allowing developers to choose Mel filter bank dimensions that increase accuracy while reducing bias.

When used with MFCCs, log Mel spectrograms serve a purpose of dimensionality reduction. In contrast to log Mel spectrogram input features, MFCC features (right column) are robust to the number of Mel filter banks used across all datasets. Interestingly, when comparing the results of the low-latency CNN models trained with MFCCs in this figure and the CNN models in Figure 14 in the Appendix, the distributions of accuracy scores for the google_sc and mswc_en datasets have a smaller IQR for the light-weight architecture. This suggests that the dimensionality reducing effect of the spectrograms can be particularly advantageous for smaller model architectures. As fewer input dimensions reduce the computational overhead during training and inference, our results present an opportunity for on-device ML developers: MFCCs that use log Mel spectrograms with fewer filter banks (e.g., 20) can improve computational efficiency without compromising accuracy or reliability bias.

To gain an appreciation of how pre-processing parameters affect the performance of KWS models for male and female subgroups, we show the impact of feature type on male and female

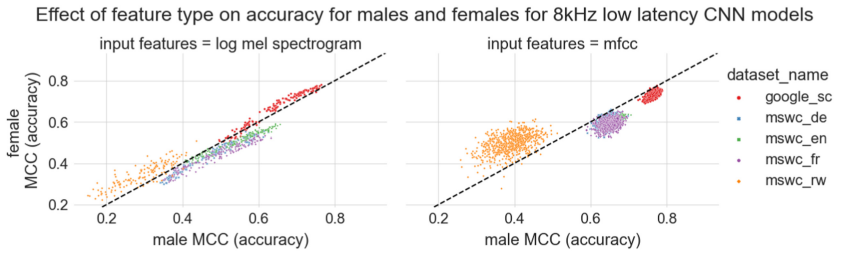

Fig. 8. Accuracy scores for males (x-axis) and females (y-axis) for log Mel spectrogram (left) and MFCC (right) feature types for **8 kHz low-latency CNN** models.

subgroup accuracy in Figure 8. For mswc_de, _en and _fr accuracy is always greater for males, irrespective of the feature type. For mswc_rw the opposite holds true: accuracy is almost always greater for females, irrespective of the feature type. For the google_sc dataset log Mel spectrograms generate models that have higher accuracy for females than for males, while MFCC features generate models with lower accuracy for females than males. For the MSWC datasets MFCC features clearly result in more accurate models for both subgroups while for the google_sc dataset both feature types generate models with similar maximum accuracy. When training on this dataset the choice of feature type can thus be a source of reliability bias.

As highlighted by a recent study in the speaker recognition domain [37], only limited feature extractors have been considered, since the adoption of deep neural networks for speech processing tasks. Some prior studies have noted the limits of log Mel- and MFCC-based features, and have proposed alternatives. For example, per-channel energy normalization features have been proposed for robust keyword spotting [66] and power-normalized cepstral coefficients for robust speech recognition [30]. However, while these studies consider robustness in noisy and far-field environments, they do not consider bias in their analysis of robustness. Based on our findings, we consider further characterisation of the effect of input features on reliability bias across a wider range of feature extractors an important area of future work.

Key insights: Feature type and dimensions impact KWS accuracy and reliability bias. Their effect is further influenced by the training dataset. In general, MFCC type features perform better than log Mel spectrograms. However, they can also increase reliability bias, prejudicing models against females and favouring males. For MFCC features, fewer dimensions (i.e., cepstral coefficients and Mel filter banks) can reduce computational demands with a negligible impact on accuracy and reliability bias.

6.2 Design Choices during Model Optimization

We focused our study of model optimization design choices on model compression and in particular model pruning. Pruning increases model sparsity, which reduces storage, memory and bandwidth requirements when downloading models to devices. We followed the experimental setup described in Section 5.4.2 to prune a subset of the most accurate and least biased models.

To analyze the impact of the pruning hyper-parameters, we performed factorial ANOVA tests to determine the effects of pruning hyper-parameters on change in reliability bias and change in accuracy due to pruning. The factorial ANOVA tests were designed following the same process as described for pre-processing parameters in the previous section. The first factorial ANOVA model (see Model 4 in the Appendix) considers interactions between all the independent variables,

Table 7. Significant Main and Interaction Effects of Pruning Hyper-parameters on **Change in MCC (Accuracy)**

Factorial ANOVA main and interaction effects	**SS**	**df**	**F**	**p(<0.05)**
pruning schedule	6.2660	1	2.9467E+03	0.0E+00
reliability bias baseline model	0.7614	1	3.5805E+02	1.1E-78
dataset	6.4134	4	7.5402E+02	0.0E+00
pruning learning rate	89.6026	2	2.1069E+04	0.0E+00
final sparsity	143.5794	5	1.3504E+04	0.0E+00
dataset * pruning schedule * final sparsity	0.2113	20	4.9694E+00	1.9E-12
sample rate * pruning learning rate * final sparsity	0.2759	10	1.2974E+01	7.2E-23
dataset * pruning learning rate * pruning schedule	0.1467	8	8.6237E+00	8.5E-12
dataset * pruning learning rate * final sparsity	2.7539	40	3.2377E+01	3.7E-232
model architecture * pruning learning rate * pruning schedule * final sparsity	0.2685	10	1.2625E+01	3.6E-22
dataset * model architecture * sample rate * final sparsity	0.1929	20	4.5357E+00	6.2E-11
Residual	25.2897	11,893	—	—
Model	—	274	5.5574E+02	0.0E+00
R^2	0.9276			
Adjusted R^2	0.9259			

SS = sum of squares, df = degrees of freedom.

Table 8. Significant Main and Interaction Effects of Pruning Hyper-parameters on **Change in Reliability Bias**

Factorial ANOVA main and interaction effects	**SS**	**df**	**F**	**p(<0.05)**
model architecture	2.0088	1	2.2874E+02	3.3E-51
pruning learning rate	15.0129	2	8.5475E+02	0.0E+00
final sparsity	23.8760	5	5.4374E+02	0.0E+00
reliability bias baseline model	8.5336	1	9.7171E+02	3.3E-205
dataset	22.9815	4	6.5421E+02	0.0E+00
pruning schedule * final sparsity	1.0450	5	2.3798E+01	6.8E-24
model architecture * final sparsity	2.8381	5	6.4633E+01	8.5E-67
dataset * pruning learning rate * final sparsity	10.8173	40	3.0794E+01	3.2E-220
dataset * model architecture * sample rate * pruning learning rate	1.1197	8	15.937276	1.3E-23
Residual	105.508229	12,014	—	—
Model	—	148	1.1070E+02	0.0E+00
R^2	0.5769			
Adjusted R^2	0.5717			

SS = sum of squares, df = degrees of freedom.

including pruning hyper-parameters, dataset, architecture, sample rate, the baseline model accuracy and baseline model reliability bias. We continued to improve the factorial ANOVA models separately for change in accuracy and change in reliability bias by removing all non-significant interactions, and then including lower-level interactions. The final ANOVA models are included in the Appendix, with Models 5 and 6 capturing variables and interactions of pruning design choices on the change in MCC score and change in reliability bias, respectively.

Tables 7 and 8 show statistically significant interaction and main effects of the final factorial ANOVA models. For completeness, we have included main effects even if they already contribute to an interaction. The final factorial ANOVA models are significant (change in MCC (accuracy): $F(274) = 555.74$, $p = 0.0$, $R^2_{adj.} = 0.9259$ and change in reliability bias: $F(148) = 110.70$, $p = 0.0$, $R^2_{adj.} = 0.5717$). We again point the reader to Table 6 for reference of the critical F statistics at p-values less than 0.01 and 0.05. Based on the F statistics, we reject the null hypothesis that KWS model accuracy and reliability bias are unaffected by pruning hyper-parameters and their interactions during model optimization. As with the statistical analysis of the pre-processing

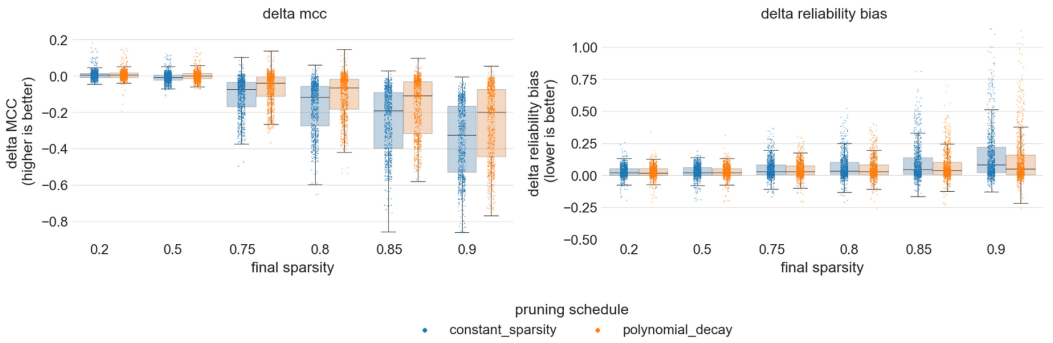

Fig. 9. Interaction effect of final sparsity and pruning schedule on **change in MCC (accuracy)** (left) and **change in reliability bias** (right). The *polynomial decay* (orange) pruning schedule results in higher median change in MCC scores and lower median change in reliability bias. Polynomial decay also results in smaller IQR of reliability bias. These effects become more significant as final sparsity increases.

parameters, we found that the R^2 values of the change in accuracy ANOVA model ($R^2 = 0.9276$, $R^2_{adjusted} = 0.9259$) indicate that this model captures the effects better than the change in reliability bias ANOVA model ($R^2 = 0.5769$, $R^2_{adjusted} = 0.5717$). In the latter model a portion of the variance in the dependent variable remains unaccounted for. We now examine the impact of the pruning interaction effects in greater detail. Throughout the analysis, we use the terms *change in* and *delta* interchangeably.

6.2.1 Impact of Pruning Hyper-Parameters. The interaction effect between final sparsity and pruning schedule is visualized in Figure 9. This interaction significantly affects change in reliability bias due to pruning (as per Table 8). The interaction between final sparsity, pruning schedule and dataset also have a significant effect on change in accuracy (as per Table 7). The figure highlights several interesting observations. When final sparsities are low (i.e., 0.2 and 0.5), the median delta MCC and delta reliability bias are close to zero. Furthermore, the delta MCC and delta reliability bias IQR of models pruned to these sparsities are small. This indicates that these pruned models have low variability in accuracy and reliability bias and that scores lie close to those of the baseline models. However, as the final sparsity increases, the median delta MCC becomes more negative (implying lower accuracy due to pruning) and the IQR increases (indicating greater variability in accuracy due to pruning). Likewise, the median delta reliability bias and the IQR increase, indicating that models become more biased and that reliability bias scores become more variable. For all sparsities there are some models that have a positive change in MCC, thus becoming more accurate, and a negative change in reliability bias, thus becoming less biased, due to pruning. The polynomial decay pruning schedule results in higher median delta MCC scores and lower median delta reliability bias. Polynomial decay also results in smaller IQR of delta reliability bias. These effects become more apparent as final sparsity increases. For developers polynomial decay is thus a more robust pruning schedule to choose.

Figure 10 visualizes the interaction effect of final sparsity and the pruning learning rate. As shown in Table 8 this interaction significantly affects the change in reliability bias due to pruning. Final sparsity and pruning learning rate also have a significant effect on change in accuracy through interactions with sample rate and with dataset (see Table 7). At a low final sparsity of 0.2 the learning rate has no impact on the accuracy and bias of pruned models. As the sparsity increases, this changes dramatically. The smaller the learning rate, the lower the median delta MCC and the larger its IQR. A lower delta MCC results in a greater accuracy drop due to pruning. Similarly, the smaller the learning rate, the higher the median delta reliability bias and the larger

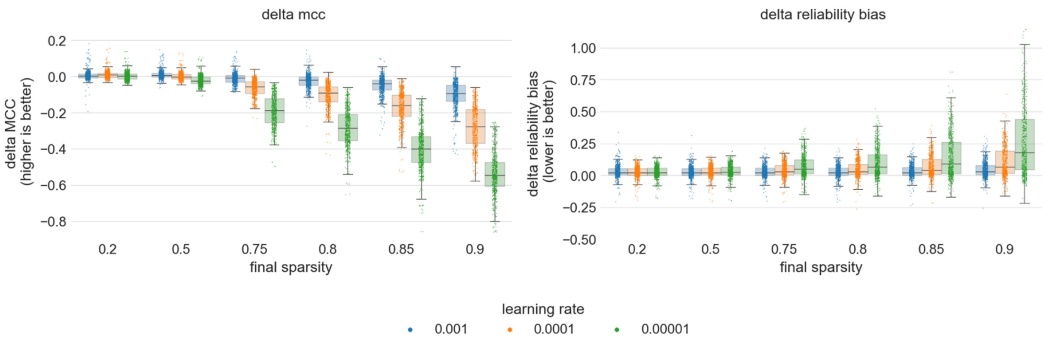

Fig. 10. Interaction effect of final sparsity and pruning learning rate on **change in accuracy** (left) and **change in reliability bias** (right). At final sparsities above 0.5 smaller learning rates significantly reduce MCC scores and increase reliability bias.

its IQR. A higher delta reliability bias results in increase in reliability bias due to pruning. At 90% sparsity the median MCC score of models pruned with a learning rate of 0.00001 reduces by more than 0.5 (maximum value of the MCC metric is 1). This means that the accuracy of pruned models with 90% sparsity is less than half that of baseline models. At the same final sparsity and learning rate the median delta reliability bias increases by 0.18, indicating that substantial performance discrepancies exist between the male and female subgroups.

A possible explanation for our results is that the learning rate optimizes the discovery of structure in the training data to favour one subgroup over the other. This intuition aligns with recent empirical work that shows that top performing deep neural networks can have very similar accuracy, but large variance in other performance aspects such as inference latency due to hyper-parameter tuning [33]. Similarly, a recent study on fixed-seed training of deep learning systems shows high variance in fairness measures if experiments consist of a single run with a fixed seed [48]. Based on our results, developers can choose a larger pruning learning rate, like 0.001, during model optimization to reduce the likelihood of unintended bias and unexpected accuracy degradation, especially when pruning models to high sparsities. While this rule-of-thumb is useful given our current knowledge, further research is needed to fully characterize the impact of the pruning learning rate on model performance and bias. We thus suggest that developers empirically validate and optimize the learning rate during pruning.

To conclude our detailed analysis of interaction effects arising during model pruning, we examine how the interactions between dataset, architecture, sample rate, and pruning learning rate affect change in reliability bias in Figure 11. Across datasets, architectures, and sample rates, we observe the general trend, which we have already identified in the previous figure: the smaller the learning rate, the more biased models become. Careful examination of the results across architectures and sample rates also reveals trends similar to those we observed with pre-processing parameters: the increase in reliability bias due to pruning is greater for the light-weight low-latency CNN architecture and the lower sample rate of 8 kHz, as indicated by higher medians and larger IQRs. This trend is stronger at smaller learning rates. As with the pre-processing parameters, the mswc_rw dataset presents an exception to this observation. For this dataset median delta reliability bias across learning rates shows no clear trend, while the IQRs are always large when compared to the IQRs of the other datasets.

Figure 11 reveals a further insight when comparing results across datasets. The reliability bias of models trained on the google_sc and mswc_en datasets is less affected by the pruning learning rate than what is the case for models trained on the remaining datasets. These two English

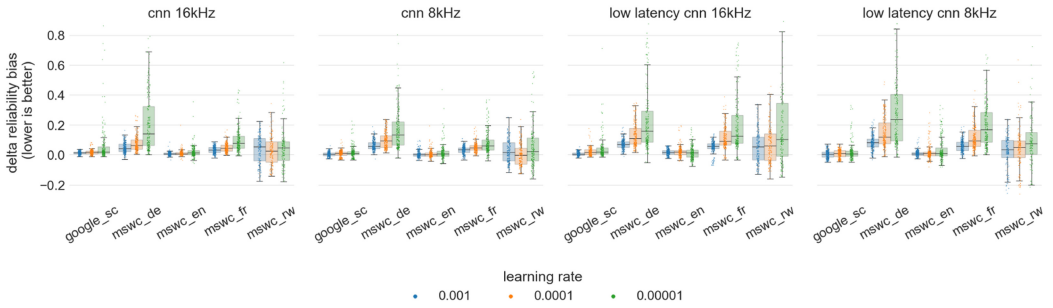

Fig. 11. Interaction effect between dataset, architecture, sample rate and pruning learning rate on **change in reliability bias**. Across datasets the general trend indicates that the smaller the learning rate, the more biased models become. The median change in reliability bias of the google_sc and mswc_en datasets is less affected by the pruning learning rate than that of the remaining datasets.

language datasets have low median delta reliability bias values and a small IQR. A likely contributor to these results is that English is the best resourced language, with the largest available quantity of data. The English datasets in our study thus include more utterances per keyword, more unique speakers per keyword and better representation of speakers and utterances across keywords in the validation and test sets. Data quantity and representation, however, may not explain the entire effect. The mswc_de and mswc_fr datasets have very similar statistics across the keywords, genders and dataset splits, with the mswc_de dataset being 13% larger than the mswc_fr dataset. Yet, the change in reliability bias for mswc_de models is larger and more variable than that of mswc_fr models. Further research is needed to understand the source of this variability. For developers, our results highlight that training, validation and test datasets need to be large enough and representative across groups of users to ensure robust results and avoid bias. Considering that German and French are, after English, two of the best resourced languages in the Mozilla Common Voice corpus,[5] this may mean in practice that developers need to collect context and application specific datasets to evaluate bias.

Key insights: Polynomial decay is a more robust pruning schedule than constant sparsity, and a larger pruning learning rate, like 0.001, reduces the likelihood of unintended bias and unexpected accuracy degradation. These design choices are particularly important when pruning models to sparsities greater than 50%, beyond which accuracy and reliability bias can deteriorate dramatically. The increase in reliability bias due to pruning is greater for smaller architectures and at lower sample rates. This trend is stronger at smaller learning rates. Training, validation and test datasets need to be large enough and representative across groups of users to ensure robust results and avoid bias.

6.3 Summary of Results

We conducted empirical experiments for an audio KWS task to investigate the impact of a comprehensive set of design choices on accuracy and reliability bias during model training and optimization. During model training, we investigated design choices related to the data sample rate and pre-processing parameters. We also considered how models trained with a light-weight architecture are affected by the sample rate and pre-processing parameters. Analyzing the results of 17,280 experiments on five datasets showed that median and IQR of reliability bias tend to be

[5] https://commonvoice.mozilla.org/en/languages.

greater at lower sample rates and for light-weight architectures. Whether reliability bias favours or is prejudiced against a group of people was strongly influenced by the training dataset. Overall, male speakers were favoured by models. Model accuracy was lower at lower sample rates and for light-weight architectures. An exception to the overall trends were models trained on the mswc_rw dataset, which had considerably lower accuracy and favoured female speakers.

With regards to pre-processing parameters, we found feature type and dimensions to impact KWS accuracy and reliability bias. These effects were further influenced by the training dataset. In general, MFCC type features performed better than log Mel spectrograms. However, in some instances they also increased reliability bias, prejudicing models against females and favouring males. For MFCC features, reducing the feature dimensions by using fewer Mel filter banks and fewer cepstral coefficients had a negligible impact on accuracy and reliability bias. This presents an opportunity to reduce computational demands for on-device ML applications.

During model optimization, we investigated design choices related model compression, and specifically pruning. Analyzing the results of 12,168 experiments on five datasets, we found polynomial decay to be a more robust pruning schedule than constant sparsity. The smaller the pruning learning rate, the more pruning increased reliability bias and decreased accuracy of baseline models. We found that a larger pruning learning rate, like 0.001, reduced the change in reliability bias and accuracy. These design choices were particularly important when models were pruned to sparsities greater than 50%. Beyond this, accuracy and reliability bias deteriorated dramatically. As with pre-processing parameters, the increase in reliability bias due to pruning was greater for light-weight architectures and at lower sample rates. This trend was stronger at smaller learning rates. Finally, we found that pruning results varied across datasets, with English language datasets showing a smaller increase in reliability bias due to pruning than other languages. One take-away from this is that training, validation and test datasets need to be large enough and representative across groups of users to ensure robust results and avoid bias.

7 STRATEGIES TO MITIGATE RELIABILITY BIAS

In the previous section, we presented empirical results and an analysis of the impact of design choices on accuracy and reliability bias for an audio KWS task. Taking the insights gained through the study into consideration, we now offer low effort strategies for mitigating reliability bias. We first consider strategies for model selection and then discuss approaches for supporting design choices with targeted experimentation.

7.1 Model Selection

In the decision map presented in Figure 2 model selection can occur after training a new model, downloading pre-trained models or optimizing a model. We consider model selection strategies that account for accuracy and reliability bias after model training and after model optimization. Rather than considering reliability bias and accuracy as a trade-off, we seek approaches that enable engineers to navigate multi-objective search scenarios where high accuracy and low reliability bias are desired. In contrast to multi-objective criteria that have been proposed during model training [36, 45], here we focus on multi-objective model selection as a post-processing intervention.

7.1.1 Model Selection After Training. In Section 6.1, we explored in depth how model training design choices impact reliability bias and accuracy. While our analysis presented important insights of trends that exist across datasets and architectures, we also found that models exist that are accurate and that perform equally well for male and female subgroups. A visual appreciation for this can be gained from Figure 5. Across datasets, architectures, and sample rates there are models that lie on or close to the diagonal on which male and female accuracy is equal. This suggests

Table 9. Table of MCC (Accuracy) Scores and Reliability Bias for Models Trained on the Google_sc Dataset and Selected for Top Performance Based on Three Criteria: (1) High Accuracy, (2) Low Bias, and (3) Low Bias + High Accuracy

model selection criteria	metric	16 kHz CNN	8 kHz CNN	16 kHz low-latency CNN	8 kHz low-latency CNN
high accuracy	MCC score	0.877	0.868	0.804	0.778
	reliability bias	1.2e-2	9.8e-3	6.6e-4	4.1e-2
low bias	MCC score	0.849	0.815	0.762	0.740
	reliability bias	1.8e-4	1.9e-4	1.2e-4	1.6e-4
low bias + high accuracy	MCC score	0.872	0.861	0.804	0.775
	reliability bias	7.7e-4	5.9e-3	6.6e-4	1.8e-3

Comparison across metrics shows that the low bias + high accuracy criteria, which accepts a marginal drop in accuracy of up to 1.5%, selects models with considerably lower bias than the high accuracy strategy.

that pre-processing parameters may exist that produce models with high accuracy and low bias. However, these models do not necessarily have the highest accuracy score. We thus considered search criteria for selecting models based on accuracy and reliability bias. Listed below are three criteria we used to select model d from n trained models D by optimizing for:

(1) high accuracy: select d if $MCC_d = max(MCC_1, \ldots, MCC_n)$ for n in D,
(2) low bias: select d if $reliability\ bias_d = min(reliability\ bias_1, \ldots, reliability\ bias_n)$ for n in D,
(3) low bias + high accuracy: select d if $MCC_d >= 0.985 * max(MCC_1, \ldots, MCC_n)$ for n in D **and** $reliability\ bias_d = min(reliability\ bias_1, \ldots, reliability\ bias_k)$ for k where $MCC_k >= 0.985 * max(MCC_1, \ldots, MCC_n)$ for n in D.

We consider the high accuracy criteria as a baseline, as this is the typical strategy followed by engineers that do not consider bias. The low bias criteria presents the opposite scenario, where only reliability bias informs model selection. Finally, the low bias + high accuracy criteria considers accuracy as a satisficing metric while minimizing reliability bias. This criteria selects the model with the lowest reliability bias, provided that it has an accuracy score within a 1.5% threshold of the maximum accuracy. A reasonable threshold value should be selected in accordance with the application requirements. The multi-objective approach allows us to explore alternative models for deployment.

Table 9 shows the MCC (accuracy) score and reliability bias for the best models trained on the google_sc dataset, selected according to the three criteria. We find that the low bias + high accuracy criteria selects models with a low reliability bias across architectures, while retaining an MCC score close to the high accuracy criteria. For the CNN architectures, this criteria reduces reliability bias by 15.7- and 1.7-fold for models trained with 16 and 8 kHz sample rates, respectively. For the 8 kHz low-latency CNN model, reliability bias is reduced 22.3-fold. For the 16 kHz low-latency CNN architecture the model with the highest accuracy also has the lowest reliability bias and thus experiences no reduction. By comparison, models selected using only low bias as selection criteria have a very low reliability bias. However, this comes at the cost of an accuracy drop between 3.2% and 6.1%, which is considerably greater than the desired 1.5% threshold and will degrade performance for both subgroups.

Instead of selecting maximum or minimum values, the selection criteria can be modified to select the m best models under that criteria. We followed this approach choosing $m = 3$ best models for a dataset, model architecture and sample rate triplet to select baseline models for the pruning experiments. The low bias + high accuracy criteria did not return valid models for all triplets,

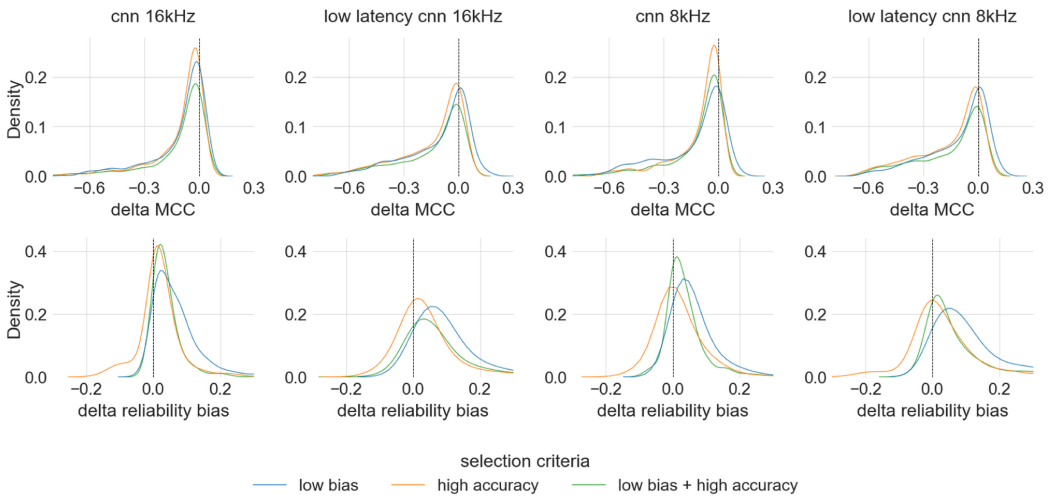

Fig. 12. Change in MCC (accuracy) score and **change in reliability bias** of models after pruning. Models were selected based on three selection criteria: high accuracy (orange), low bias (blue), and high accuracy + low bias (green). Delta MCC is better when greater, delta reliability bias is better when smaller. The selection criteria has no evident effect on delta MCC, but does affect delta reliability bias.

which is the reason for the unequal number of baseline models in Table 3. Next, we consider how these selection criteria hold up after pruning.

7.1.2 Model Selection After Pruning. For the pruning experiments in Section 6.2, we selected the top three models per architecture and sample rate for each model selection criteria. We now consider the post-pruning performance of models selected with different criteria. In Figure 12, we show the delta (i.e., change in) MCC score (top) and delta reliability bias (bottom) after pruning, for models selected under the three criteria after training. In the density distributions in the figure accuracy increases in the direction of positive change, meaning that delta MCC distributions that peak to the right of zero are desirable. Conversely, reliability bias decreases in the direction of negative change, meaning that delta reliability bias distributions with peaks to the left of zero are desirable. The delta MCC distributions peak just left of zero (CNN architectures) or on zero (low-latency CNN architectures), indicating that the majority of models with these architectures experience a decline in accuracy. The shape of the distributions is similar for different selection criteria under the same architecture and sample rate, with the accuracy of low bias models (which have lower baseline accuracy) increasing slightly more after pruning.

The shapes and peaks of the delta reliability bias distributions vary across model selection criteria. This indicates that the model selection criteria impact reliability bias after pruning. A further confirmation of this is presented in the statistical analysis in Table 8, where the reliability bias of the baseline model has a statistically significant effect on delta reliability bias due to pruning. Analyzing the distributions, we can see that the distributions of models selected for low bias (blue) lie furthest to the right. This means that the reliability bias of these models increases the most after pruning. This is not surprising, as the lower bound of the reliability bias measure is zero and models with low bias are very sensitive to small changes in reliability bias. However, this can also indicate that models selected for low bias may loose some of their good bias properties during pruning. The distributions of models selected for high accuracy (orange) lie furthest to the left. These models typically started out with higher reliability bias after training, which makes them less sensitive to changes in reliability bias and thus better able to retain their reliability bias scores.

Table 10. Mean and Variance of MCC Scores and Reliability Bias Across Pruning Sparsities (0.2, 0.5, 0.75, 0.8, 0.85, 0.9) for the Three Model Selection Criteria

criteria	**high accuracy**				**low bias + high accuracy**				**low bias**			
metric	**MCC score**		***reliability bias***		**MCC score**		***reliability bias***		**MCC score**		***reliability bias***	
	mean	**var**	**mean**	**var**	**mean**	**var**	**mean**	**var**	**mean**	**var**	**mean**	**var**
16kHz CNN	0.885	1.2e-02	1.4e-02	9.6e-03	0.879	1.1e-02	4.0e-03	5.5e-03	0.823	5.3e-02	6.5e-04	4.0e-04
8kHz CNN	0.876	9.2e-03	6.5e-03	1.8e-03	0.870	8.9e-03	1.1e-03	6.7e-04	0.851	1.9e-02	2.9e-04	2.6e-04
16kHz llCNN	0.808	1.8e-02	8.9e-03	7.0e-03	0.804	1.7e-02	1.4e-03	1.1e-03	0.772	2.3e-02	4.9e-04	4.1e-04
8kHz llCNN	0.785	2.5e-02	1.0e-02	7.6e-03	0.781	2.4e-02	1.8e-03	2.1e-03	0.761	3.5e-02	4.9e-04	4.9e-04

Models have been trained on the google_sc dataset. Models with lower bias can be selected for all sparsities at an accuracy cost of less than 1.5%.

In Figure 15 in the Appendix, we visualize the distribution of MCC scores and reliability bias for the selection criteria. Right of the peak (i.e., in the higher accuracy range), the MCC score distributions for the high accuracy criteria and the low bias + high accuracy criteria lie very close to each other. After pruning models selected for high accuracy and for low bias + high accuracy thus have similar MCC scores. For reliability bias the distribution of the low bias + high accuracy criteria lies between the low bias and high accuracy distributions. The low bias + high accuracy criteria thus results in models with lower bias after pruning than the high accuracy criteria. Overall, this makes the low bias + high accuracy criteria a good choice to select a range of models for pruning.

Finally, we reapply the same model selection criteria that we previously applied after training, *after pruning*. Table 10 shows the mean and standard deviation of accuracy and reliability bias across sparsities for the three selection criteria for *pruned* models trained on google_sc. We observe that mean reliability bias can be improved by an order of magnitude by choosing the low bias + high accuracy criteria rather than the high accuracy criteria. Models selected with the low bias criteria suffer a large drop in accuracy. While the low bias criteria offers lower reliability bias than the low bias + high accuracy criteria, the latter already has a low mean and variance in reliability bias, making additional reductions less impactful. For all models the variance of metrics across sparsities is relatively low, which is supported by our earlier observation that models trained on the google_sc dataset are less affected by pruning hyper-parameters (see Figure 11). Across all datasets the low bias + high accuracy criteria selects models with similar accuracy and lower reliability bias than the high accuracy criteria. This outcome is not surprising, as the purpose of a multi-objective criterion is precisely to satisfy multiple objectives. The value of our analysis lies in empirically validating the obvious rather than in finding surprise: engineers can reduce bias in audio KWS with little effort by applying a multi-objective criterion during model selection, choosing models that satisfy an accuracy condition while minimizing bias.

7.1.3 Summary of Model Selection Strategy. Engineers should use a multi-objective criterion that considers accuracy and reliability bias to select models that have high accuracy and low bias after training or after pruning. We propose that engineers set a tolerance that controls the drop in accuracy from the maximum value, thus using accuracy as a satisficing metric while minimizing reliability bias. The tolerance value should be determined from application requirements. If model training is followed by pruning, then a small number of top models should be selected for pruning using high accuracy and low bias + high accuracy strategies.

7.2 Supporting Design Decisions with Targeted Experimentation

In Section 3, we presented a map of design choices arising in the on-device ML workflow. We then showed empirically that these design choices can lead to disparate performance of audio KWS

Table 11. Recommended Design Choice Variables and Values for Audio KWS to Mitigate Bias while Reducing Resource Consumption During Experimentation

Design action	Design choice	Choice variable (unit)	Variable values
Train new model	input features | sample rate	sample rate (kHz)	*determined by application*
Train new model	input features | pre-processing	feature type	MFCC
Train new model	input features | pre-processing	# Mel filter banks	20, 32
Train new model	input features | pre-processing	# MFCCs	10, 11
Train new model	input features | pre-processing	frame length (ms)	20, 25, 30, 40
Train new model	input features | pre-processing	frame step (% frame length)	40, 50, 60
Train new model	input features | pre-processing	window function	Hamming
Optimize model	light-weight architecture	model architecture	*determined by application*
Optimize model	model compression | pruning	final sparsity (%)	*determined by application*
Optimize model	model compression | pruning	pruning frequency	10, 100
Optimize model	model compression | pruning	pruning schedule	polynomial decay
Optimize model	model compression | pruning	pruning learning rate	1e-4, 1e-5 for sparsities < 50%; 1e-3 for sparsities > 50%
Model selection	selection strategy	criteria	high accuracy, low bias + high accuracy
Model selection	selection strategy	# best models	3

models for males and females. Our analysis in Section 6 demonstrates that even when engineers make reasonable decisions about training and optimization parameters (see Table 1) their choices can lead to models with widely different accuracy and bias properties. Especially when training light-weight architectures or processing data at low sample rates, systematic experimentation is a necessary strategy to support design decisions and mitigate bias. We have demonstrated that iterating over pre-processing parameters during training, and pruning hyper-parameters during model compression can help engineers train models with high accuracy and low bias. However, experimentation comes at a cost: each iteration requires computational resources, takes time and consumes energy. Where a single audio KWS model takes only a couple of minutes to train, we trained 17,280 models, pruned 12,168 models, and ran our experiments for several days. This is a costly undertaking.

Rather than replicating our approach, engineers should take the resource footprint and cost of model training into account, and target their experiments to iterate over values that are likely to yield high accuracy, low bias models. To this end, we propose revised design choice variable values based on the insights we gained through our experiments in Table 11. Given these reduced options, engineers only need to train 48 models per sample rate and architecture, and run at most 24 pruning experiments for low sparsities (12 experiments for higher sparsities of more than 50%). This targeted approach to experimentation is thus a feasible strategy for engineers to use data-driven decision making to mitigate bias in on-device ML workflows.

8 DISCUSSION

We now take a higher-level perspective to reflect on the overarching implications of our work on bias in on-device ML. We first discuss reliability bias as a source of unfairness and discrimination in on-device ML and then reflect on limitations of the study.

8.1 Reliability Bias as a Source of Unfairness and Discrimination in On-device ML

On-device ML applications are becoming increasingly prevalent in our day-to-day lives, as consumers' privacy concerns and large volumes of sensor data are motivating a shift to run deep

neural networks on devices rather than the cloud. Despite the prevalence of on-device ML applications, known bias challenges in ML systems, and the material consequences of system failure, bias in on-device ML is understudied. In this article, we set out to study sources of bias in on-device ML that have not been considered in the domain, and that are overlooked in current research on ML fairness.

When interacting with services that make use of on-device ML, users are justified to expect reliable performance, irrespective of their demographic, social or economic attributes. We defined reliability bias as systematic device failures due to on-device ML performance disparities across user groups. Reliability bias is a particular concern for on-device ML as it counter-acts the promise of technology-enabled service access, an important value proposition of on-device ML. If reliability bias remains unidentified and is not accounted for, then it can be a source of unfairness in on-device ML systems. Unfair on-device ML systems that are deployed at scale can lead to a discriminatory service infrastructure that restricts who has access to services, and how these services can be accessed.

Our empirical study shows that design choices made by engineers in each stage of the on-device ML workflow can introduce reliability bias when deploying ML as a system component. While bias in on-device ML can be cast as an AI ethics concern, we consider it important to also approach it as a matter of responsible design. Based on our findings, we do not consider reliability bias as an immutable property of a particular model or system. Instead, we position that reliability bias arises from design choices that amplify or reduce disparate predictive performance across groups of users based on their personal attributes. Engineers thus have an active role to play to detect and mitigate reliability bias. While some design choices may lie beyond the immediate control of engineers, they have full control over others. Measuring bias in the on-device ML development workflow is the first step that engineers should take to practice responsible design and make a commitment to building fairer technology systems.

We have focused our evaluation of reliability bias on performance discrepancies in predictive accuracy. In on-device ML applications, system efficiency is another important performance metric that interacts with accuracy. For example, a KWS system with poor predictive performance can require several user attempts to activate the system. This can increase computations, which leads to increased power consumption and faster drainage of a device's battery. Reliability bias should thus also be considered for system efficiency. The bias measure that we have proposed can be extended easily to characterize reliability bias due to system (in)efficiency. We will investigate this in future work. Additionally, we note that our empirical study is focused on audio-based ML. Although audio is a prominent data modality in on-device ML, we are cognizant that other data types (e.g., images) are also used. Future work can extend our methodology to different modalities and new learning tasks to investigate reliability bias in them.

8.2 Limitations

In quantifying an unobservable, abstract construct like fairness, bias measures often make assumptions about what is fair. Yet, fairness is a contested construct [28] that is underpinned by the values of those that define it. This makes it important to state assumptions explicitly to avoid mismatches between the construct that is measured, and its quantified operationalization. In this study, we have made the assumption that false-positive and false-negative error rates are equally important across all keyword classes. We have operationalized this assumption by using the MCC to quantify reliability bias. While the MCC is an accepted metric for multiclass classification, it does not capture the disparate impact that false positives and false negatives may carry in particular application scenarios. For example, a KWS system in an emergency care application that has a high false-negative rate for the keyword "help" is likely to have a more detrimental impact on affected

users than a home entertainment system with a high false-positive rate for the keyword "lights on." Characterising harms associated with applications and identifying acceptable error rates is an important area for future research.

We motivated our use of a parity-based bias measure by claiming that ground truth labels in KWS are exactly known and undisputed. While this avoids bias propagation through labelling choices, constructing groups remains a normative design decision that requires careful consideration. In our audio KWS study, we constructed groups based on a speaker's gender. Our approach to labelling voice samples with gender was limited to a binary gender classification system and a crowd-sourced labelling campaign. Even though the MSWC gender labels are self-annotated, binary gender representation removes individuals that do not fit within this classification system from the bias evaluation. Crowd-sourced labelling can introduce further misclassification [52]. Male and female voices can be higher or lower pitched than what a data worker perceives as normal for that gender, and misclassified accordingly. Gender is also just one of many demographic attributes that influences the human voice [56]. Subgroups established along other speaker attributes can reveal further dimensions of bias and should be investigated in future work.

While dataset representation was not the focus of this study, it oftentimes is an important contributor to bias. We took this into consideration and carefully constructed gender-balanced dataset splits when we designed our experiments. Our gender-balancing protocol resulted in balanced datasets across keyword-speaker pairs (see Table 2) but unequal utterances/keyword across dataset splits and genders. When the number of unique speakers and utterances/keyword in a dataset are small, it becomes difficult to construct representative datasets, which can affect the reliability of results. In this study, our dataset construction choices may explain some of the performance deviations we observed for the MSWC Kinyarwanda dataset (see Figure 11), which contained an order of magnitude fewer different speakers than the other datasets. Finding ways for creating balanced datasets and evaluating bias when data availability across groups is variable remains an important open challenge.

We note that our study is limited to investigating bias in CNN architectures. These architectures are very popular for speech and vision related tasks in on-device ML. We chose to focus on one architecture to study how light-weight architectures, a model optimization design choice to reduce the model size, impacts reliability bias. Future work should also examine reliability bias in other architectures. Furthermore, while this study investigates the design choices that we deemed most likely to impact bias, future work should examine the impact of design choices that we did not examine, such as quantization. Despite these limitations, our investigation of performance disparity provides necessary insights that highlight the need of addressing bias in on-device settings. Studying bias in the emerging field of on-device ML is thus an important research direction for the software engineering community.

9 CONCLUSION

Billions of devices deploy on-device ML today. Despite bias and fairness being a major area of concern in ML, they have not been considered in on-device ML settings. Biased performance impacts device reliability, and can result in systematic device failures due to performance disparities across user groups. This can inconvenience and even harm users. Our study is the first investigation of bias in development workflows in the emerging on-device ML domain, and lays an important foundation for building fairer on-device ML systems.

In this study, we investigate the propagation of bias through design choices in the on-device ML workflow, and identify *reliability bias* as a potential source of unfairness. *Reliability bias* arises from disparate on-device ML performance due to demographic attributes of users, and results in systematic device failure across user groups. Drawing on definitions of group fairness, we quantify

reliability bias and use the measure in empirical experiments to evaluate the impact of design choices on bias in an audio KWS task, a dominant application of on-device ML. Our results validate that seemingly innocuous design choices—a light-weight architecture, the data sample rate, pre-processing parameters of input features, and pruning hyper-parameters for model compression—can result in disparate predictive performance across male and female groups.

Given their context dependence and ubiquitous nature, developing inclusive on-device ML systems ought to be an important priority for engineers. Our findings caution that design choices in the development workflow can have major consequences for the propagation of *reliability bias* and consequently fairness of on-device ML. Based on our findings, we suggest strategies for model selection and targeted experimentation to help engineers navigate the gap between technical choices, deployment constraints, accuracy and bias during on-device ML development. Taken together, our work highlights that engineers and the decisions they make have an important role to play to ensure that the social requirement of inclusive on-device ML is realized within the constrained on-device setting.

A APPENDIX

A.1 Experiment Setup: Datasets

Google Speech Commands keyword classes: "bed":0, "bird":1, "cat":2, "dog":3, "down":4, "eight":5, "five":6, "four":7, "go":8, "happy":9, "house":10, "left":11, "marvin":12, "nine":13, "no":14, "off":15, "on":16, "one":17, "right":18, "seven":19, "sheila":20, "six":21, "learn":22, "stop":23, "three":24, "tree":25, "two":26, "up":27, "wow":28, "yes":29, "zero":30, "backward":31, "follow":32, "forward":33, "visual":34

MSWC English keyword classes: "about": 0, "after": 1, "also": 2, "been": 3, "could": 4, "first": 5, "from": 6, "have": 7, "however": 8, "just": 9, "know": 10, "like": 11, "many": 12, "more": 13, "most": 14, "only": 15, "other": 16, "over": 17, "people": 18, "said": 19, "school": 20, "some": 21, "that": 22, "they": 23, "this": 24, "three": 25, "time": 26, "used": 27, "were": 28, "what": 29, "when": 30, "will": 31, "with": 32, "would": 33, "your": 34

MSWC German keyword classes: "aber": 0, "alle": 1, "auch": 2, "dann": 3, "dass": 4, "diese": 5, "doch": 6, "durch": 7, "eine": 8, "gibt": 9, "haben": 10, "hauptstadt": 11, "heute": 12, "hier": 13, "immer": 14, "jetzt": 15, "kann": 16, "können": 17, "mehr": 18, "muss": 19, "nach": 20, "nicht": 21, "noch": 22, "oder": 23, "schon": 24, "sein": 25, "sich": 26, "sind": 27, "wenn": 28, "werden": 29, "wieder": 30, "wird": 31, "wurde": 32, "zwei": 33, "über": 34

MSWC French keyword classes: "alors": 0, "aussi": 1, "avec": 2, "bien": 3, "cent": 4, "cette": 5, "comme": 6, "c'est": 7, "dans": 8, "deux": 9, "donc": 10, "elle": 11, "fait": 12, "huit": 13, "mais": 14, "mille": 15, "monsieur": 16, "même": 17, "nous": 18, "numéro": 19, "plus": 20, "pour": 21, "quatre": 22, "saint": 23, "sept": 24, "soixante": 25, "sont": 26, "tout": 27, "trois": 28, "très": 29, "vingt": 30, "vous": 31, "également": 32, "était": 33, "être": 34

MSWC Kinyarwanda keyword classes: "abantu": 0, "ariko": 1, "avuga": 2, "bari": 3, "benshi": 4, "buryo": 5, "cyane": 6, "gihe": 7, "gukora": 8, "gusa": 9, "hari": 10, "ibyo": 11, "icyo": 12, "igihe": 13, "imana": 14, "imbere": 15, "kandi": 16, "kuba": 17, "kugira": 18, "kuko": 19, "kuri": 20, "mbere": 21, "muri": 22, "ndetse": 23, "neza": 24, "ntabwo": 25, "nyuma": 26, "perezida": 27, "rwanda": 28, "ubwo": 29, "umuntu": 30, "umwe": 31, "yagize": 32, "yari": 33, "yavuze": 34

A.2 Statistical Analysis: Design Choices Arising During Model Training

Model 1. First factorial ANOVA interaction model for model training design choices

```
model_inital = 'metric ~ C(dataset_name, Sum)+C(model_arch, Sum)C(resample_rate,
    Sum)+C(mfccs, Sum)+C(mel_bins, Sum)+C(frame_length, Sum)+C(frame_step, Sum)+C(
    window_fn, Sum)+C(dataset_name, Sum)*C(model_arch, Sum)*C(resample_rate, Sum)*
    C(mfccs, Sum)*C(mel_bins, Sum)+C(dataset_name, Sum)*C(model_arch, Sum)*C(
    resample_rate, Sum)*C(frame_length, Sum)*C(frame_step, Sum)*C(window_fn, Sum)'
```

Model 2. Final factorial ANOVA interaction model for effect of model training design choices on MCC

```
model_final_mcc = 'mcc ~ C(dataset_name, Sum)+C(model_arch, Sum)+C(resample_rate,
    Sum)+C(mfccs, Sum)+C(mel_bins, Sum)+C(dataset_name, Sum)*C(resample_rate, Sum)
    *C(mfccs, Sum)+C(model_arch, Sum)*C(mfccs, Sum)*C(mel_bins, Sum)+C(
    dataset_name, Sum)*C(model_arch, Sum)*C(mfccs, Sum)+C(frame_length, Sum)+C(
    frame_step, Sum)+C(model_arch, Sum)*C(frame_length, Sum)*C(frame_step, Sum)'
```

Model 3. Final factorial ANOVA interaction model for effect of model training design choices on reliability bias

```
model_final_bias = 'reliability_bias ~ C(dataset_name, Sum)+C(model_arch, Sum)+C(
    resample_rate, Sum)+C(dataset_name, Sum)*C(resample_rate, Sum)+C(mfccs, Sum)+C
    (mel_bins, Sum)+C(dataset_name, Sum)*C(model_arch, Sum)*C(mfccs, Sum)+C(
    dataset_name, Sum)*C(mel_bins, Sum)+C(frame_length, Sum)'
```

A.3 Impact of Pre-processing Parameters

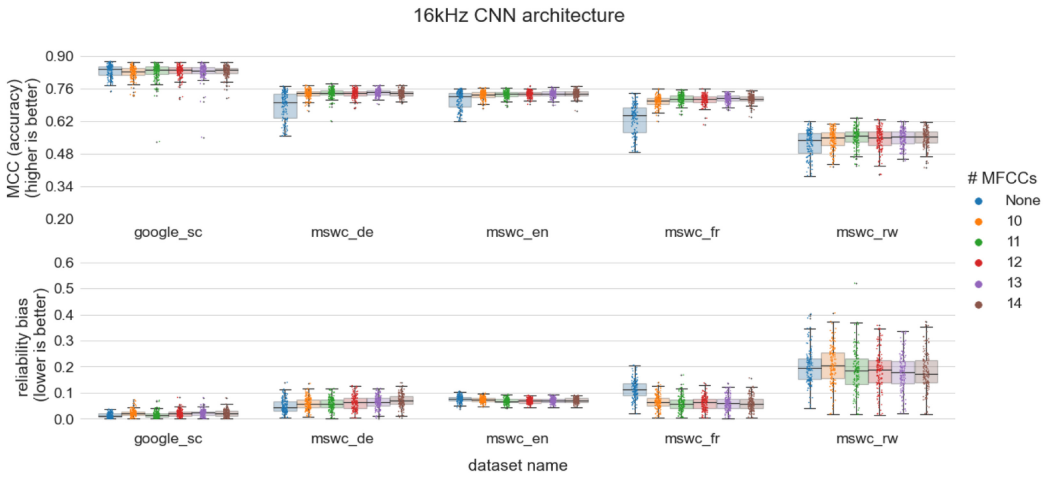

Fig. 13. Effect of MFCC dimensions on accuracy and reliability bias for 16 kHz CNN models. Models without MFCC features, i.e., models that directly use log Mel spectrograms as input features, tend to perform worse than those that use MFCC features.

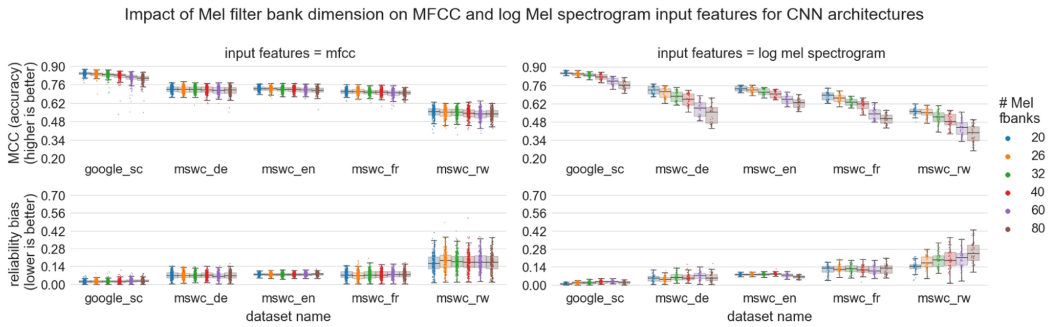

Fig. 14. Effect of log Mel spectrogram dimensions (# Mel fbanks) on accuracy and reliability bias, disaggregated by input feature type for CNN architectures. The number of Mel filter banks clearly impacts models that directly use log Mel spectrograms as input features.

A.4 Statistical Analysis: Design Choices Arising During Model Optimization

Model 4. First factorial ANOVA interaction model for pruning hyper-parameters

```
model_inital = 'delta_metric ~ mcc_baseline + reliability_bias_baseline + C(
    dataset_name, Sum)+C(model_arch, Sum)+C(resample_rate, Sum)+C(
    pruning_learning_rate, Sum)+C(pruning_schedule, Sum)+C(pruning_frequency, Sum)
    +C(pruning_final_sparsity, Sum)+C(dataset_name, Sum)*C(model_arch, Sum)*C(
    resample_rate, Sum)*C(pruning_learning_rate, Sum)*C(pruning_schedule, Sum)*C(
    pruning_frequency, Sum)*C(pruning_final_sparsity, Sum)'
```

Model 5. Final factorial ANOVA interaction model for effect of pruning design choices on change in MCC

```
model_final_delta_mcc = 'delta_mcc ~ mcc_baseline + reliability_bias_baseline + C(
    dataset_name, Sum)+C(model_arch, Sum)+C(resample_rate, Sum)+C(
    pruning_learning_rate, Sum)+C(pruning_schedule, Sum)+C(pruning_frequency, Sum)
    +C(pruning_final_sparsity, Sum)+C(model_arch, Sum)*C(pruning_learning_rate,
    Sum)*C(pruning_schedule, Sum)*C(pruning_final_sparsity, Sum)+C(dataset_name,
    Sum)*C(model_arch, Sum)*C(resample_rate, Sum)*C(pruning_final_sparsity, Sum)+C
    (dataset_name, Sum)*C(pruning_learning_rate, Sum)*C(pruning_schedule, Sum)+C(
    dataset_name, Sum)*C(pruning_schedule, Sum)*C(pruning_final_sparsity, Sum)+C(
    dataset_name, Sum)*C(pruning_learning_rate, Sum)*C(pruning_final_sparsity, Sum
    )+C(resample_rate, Sum)*C(pruning_learning_rate, Sum)*C(pruning_final_sparsity
    , Sum)'
```

Model 6. Final factorial ANOVA interaction model for effect of pruning design choices on change in reliability bias

```
model_final_delta_bias = 'delta_reliability_bias ~ mcc_baseline+
    reliability_bias_baseline+C(dataset_name, Sum)+C(model_arch, Sum)+C(
    resample_rate, Sum)+C(pruning_learning_rate, Sum)+C(pruning_schedule, Sum)+C(
    pruning_frequency, Sum)+C(pruning_final_sparsity, Sum)+C(dataset_name, Sum)*C(
    model_arch, Sum)*C(resample_rate, Sum)*C(pruning_learning_rate, Sum)+C(
    dataset_name, Sum)*C(pruning_learning_rate, Sum)*C(pruning_final_sparsity, Sum
    )+C(pruning_schedule, Sum)*C(pruning_final_sparsity, Sum)+C(model_arch, Sum)*C
    (pruning_final_sparsity, Sum)'
```

A.5 Model Selection After Pruning

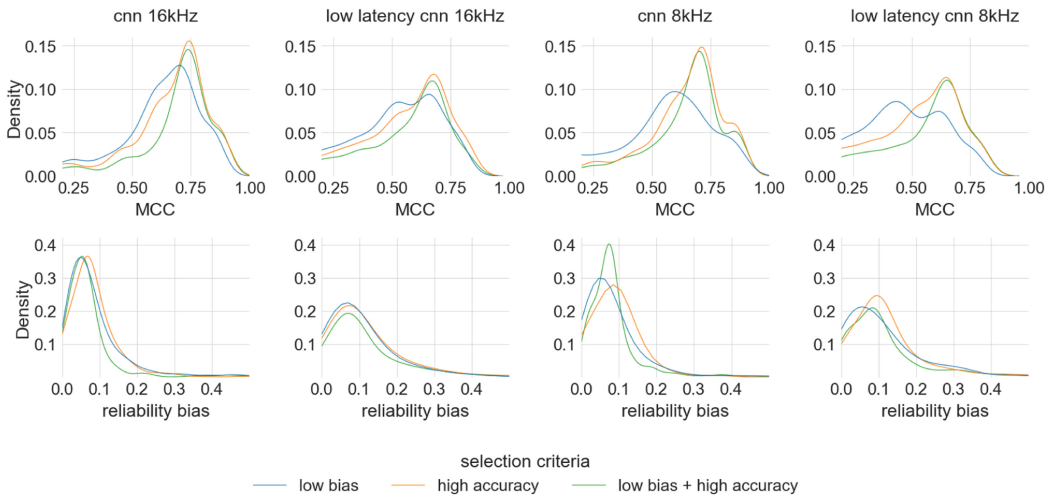

Fig. 15. MCC (accuracy) score and reliability bias of models after pruning. Models were selected based on three selection criteria: high accuracy, low bias, and high accuracy + low bias. After pruning MCC is greatest for models selected with a criteria that considers high accuracy. Similarly, reliability bias is lower for criteria that consider low bias.

ACKNOWLEDGMENTS

We thank Roel Dobbe, Sem Nouws, and Dewant Katare for their feedback and useful suggestions on the work.

REFERENCES

[1] Raziel Alvarez and Hyun Jin Park. 2019. End-to-end streaming keyword spotting. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP'19)*. 6336–6340.

[2] Colby Banbury, Vijay Janapa Reddi, Peter Torelli, Jeremy Holleman, Nat Jeffries, Csaba Kiraly, Pietro Montino, David Kanter, Sebastian Ahmed, Danilo Pau, Urmish Thakker, Antonio Torrini, Peter Warden, Jay Cordaro, Giuseppe Di Guglielmo, Javier Duarte, Stephen Gibellini, Videet Parekh, Honson Tran, Nhan Tran, Niu Wenxu, and Xu Xuesong. 2021. MLPerf Tiny Benchmark. Retrieved from http://arxiv.org/abs/2106.07597.

[3] Colby R. Banbury, Vijay Janapa Reddi, Max Lam, William Fu, Amin Fazel, Jeremy Holleman, Xinyuan Huang, Robert Hurtado, David Kanter, Anton Lokhmotov, David Patterson, Danilo Pau, Jae-sun Seo, Jeff Sieracki, Urmish Thakker, Marian Verhelst, and Poonam Yadav. 2020. Benchmarking TinyML Systems: Challenges and Direction. Retrieved from http://arxiv.org/abs/2003.04821.

[4] Abeba Birhane. 2022. The unseen Black faces of AI algorithms. *Nature* 610, 7932 (2022), 451–452. https://doi.org/10.1038/d41586-022-03050-7

[5] Tolga Bolukbasi, Kai-wei Chang, James Zou, Venkatesh Saligrama, and Adam Kalai. 2016. Man is to computer programmer as woman is to homemaker? Debiasing word embeddings. In *Proceedings of the 30th International Conference on Neural Information Processing Systems (NIPS'16)*. 4356–4364.

[6] Joy Buolamwini and Timnit Gebru. 2018. Gender shades: Intersectional accuracy disparities in commercial gender classification. In *Proceedings of Machine Learning Research: Conference on Fairness, Accountability, and Transparency*, Vol. 81. 1889–1896.

[7] Han Cai, Chuang Gan, Ligeng Zhu, and Song Han. 2020. TinyTL: Reduce memory, not parameters for efficient on-device learning. *Advances in Neural Information Processing Systems* (Dec. 2020).

[8] Joymallya Chakraborty, Suvodeep Majumder, and Tim Menzies. 2021. *Bias in Machine Learning Software: Why? How? What to do?* Vol. 1. ACM, 429–440. https://doi.org/10.1145/3468264.3468537

[9] Guoguo Chen, Carolina Parada, and Georg Heigold. 2014. Small-footprint keyword spotting using deep neural networks. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP'14)*. IEEE.

[10] Jiasi Chen and Xukan Ran. 2019. Deep learning with edge computing: A review. *Proc. IEEE* 107, 8 (8 2019), 1655–1674. https://doi.org/10.1109/JPROC.2019.2921977

[11] CPS Public Working Group. 2017. *Framework for Cyber-Physical Systems: Volume 1, Overview.* Technical Report. National Institute of Standards and Technology. https://doi.org/doi.org/10.6028/NIST.SP.1500-201

[12] Brian D'Alessandro, Cathy O'Neil, and Tom Lagatta. 2017. Conscientious classification: A data scientist's guide to discrimination-aware classification. *Big Data* 5, 2 (2017), 120–134. https://doi.org/10.1089/big.2016.0048

[13] Swarnava Dey, Arijit Mukherjee, Arpan Pal, and P. Balamuralidhar. 2019. Embedded deep inference in practice: Case for model partitioning. In *Proceedings of the 1st Workshop on Machine Learning on Edge in Sensor Systems (SenSys'19).* 25–30. https://doi.org/10.1145/3362743.3362964

[14] Sauptik Dhar, Junyao Guo, Jiayi Liu, Samarth Tripathi, Unmesh Kurup, and Mohak Shah. 2021. On-device machine learning: An algorithms and learning theory perspective. *ACM Trans. Internet Things* 2, 3 (2021). Retrieved from http://arxiv.org/abs/1911.00623.

[15] William R. Dieter, Srabosti Datta, and Wong Key Kai. 2005. Power reduction by varying sampling rate. In *Proceedings of the International Symposium on Low Power Electronics and Design.* 227–232.

[16] Roel Dobbe, Thomas Krendl Gilbert, and Yonatan Mintz. 2021. Hard choices in artificial intelligence. *Artific. Intell.* 300 (2021), 103555. https://doi.org/10.1016/j.artint.2021.103555

[17] Song Han, Huizi Mao, and William J. Dally. 2016. Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. In *Proceedings of the 4th International Conference on Learning Representations (ICLR'16).* 1–14.

[18] John H. L. Hansen and Taufiq Hasan. 2015. Speaker recognition by machines and humans: A tutorial review. *IEEE Signal Process. Mag.* 32, 6 (2015), 74–99. https://doi.org/10.1109/MSP.2015.2462851

[19] Moritz Hardt, Eric Price, and Nathan Srebro. 2016. Equality of opportunity in supervised learning. *Advances in Neural Information Processing Systems*, 3323–3331.

[20] Andrew J. Hawkins. 2019. Serious Safety Lapses Led to Uber's Fatal Self-driving Crash, New Documents Suggest. Retrieved from https://www.theverge.com/2019/11/6/20951385/uber-self-driving-crash-death-reason-ntsb-dcouments.

[21] Yanzhang He, Rohit Prabhavalkar, Kanishka Rao, Wei Li, Anton Bakhtin, and Ian McGraw. 2017. Streaming small-footprint keyword spotting using sequence-to-sequence models. In *Proceedings of the IEEE Automatic Speech Recognition and Understanding Workshop (ASRU'17).*

[22] Takuya Higuchi, Mohammad Ghasemzadeh, Kisun You, and Chandra Dhir. 2020. Stacked 1D convolutional networks for end-to-end small footprint voice trigger detection. In *Proceedings of the Annual Conference of the International Speech Communication Association (INTERSPEECH'20).* https://doi.org/10.21437/Interspeech.2020-2763

[23] Kenneth Holstein, Jennifer Wortman Vaughan, Hal Daumé, Miroslav Dudík, and Hanna Wallach. 2019. Improving fairness in machine learning systems: What do industry practitioners need? In *Proceedings of the Conference on Human Factors in Computing Systems.* 1–16. https://doi.org/10.1145/3290605.3300830

[24] Sara Hooker, Nyalleng Moorosi, Gregory Clark, Samy Bengio, and Emily Denton. 2020. Characterising Bias in Compressed Models. Retrieved from https://arxiv.org/abs/2010.03058.

[25] Jennifer Horkoff. 2019. Non-functional requirements for machine learning: Challenges and new directions. In *Proceedings of the IEEE International Conference on Requirements Engineering.* 386–391. https://doi.org/10.1109/RE.2019.00050

[26] Max Hort and Federica Sarro. 2022. *Privileged and Unprivileged Groups: An Empirical Study on the Impact of the Age Attribute on Fairness.* Vol. 1. ACM. https://doi.org/10.1145/3524491.3527308

[27] Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens Van Der Maaten, and Kilian Weinberger. 2018. Multi-scale dense networks for resource efficient image classification. In *Proceedings of the 6th International Conference on Learning Representations (ICLR'18).* 1–14.

[28] Abigail Z. Jacobs and Hanna Wallach. 2021. Measurement and fairness. In *Proceedings of the ACM Conference on Fairness, Accountability, and Transparency (FAccT'21).* 375–385. https://doi.org/10.1145/3442188.3445901

[29] Vijay Janapa Reddi, Brian Plancher, Susan Kennedy, Laurence Moroney, Pete Warden, Lara Suzuki, Anant Agarwal, Colby Banbury, Massimo Banzi, Matthew Bennett, Benjamin Brown, Sharad Chitlangia, Radhika Ghosal, Sarah Grafman, Rupert Jaeger, Srivatsan Krishnan, Maximilian Lam, Daniel Leiker, Cara Mann, Mark Mazumder, Dominic Pajak, Dhilan Ramaprasad, J. Evan Smith, Matthew Stewart, Dustin Tingley, and Harvard University. 2022. Widening access to applied machine learning with TinyML. Retrieved from https://hdsr.mitpress.mit.edu/pub/0gbwdele/release/3. https://doi.org/10.1162/99608f92.762d171a

[30] Chanwoo Kim and Richard M. Stern. 2016. Power-normalized cepstral coefficients (PNCC) for robust speech recognition. *IEEE/ACM Trans. Audio Speech Lang. Process.* 24, 7 (2016), 1315–1329. https://doi.org/10.1109/TASLP.2016.2545928

[31] Allison Koenecke, Andrew Nam, Emily Lake, Joe Nudell, Minnie Quartey, Zion Mengesha, Connor Toups, John R. Rickford, Dan Jurafsky, and Sharad Goel. 2020. Racial disparities in automated speech recognition. *Proc. Natl. Acad. Sci. U.S.A.* 117, 14 (2020), 7684–7689. https://doi.org/10.1073/pnas.1915768117/-/DCSupplemental.y.

[32] Timothy B. Lee. 2019. How Terrible Software Design Decisions Led to Uber's Deadly 2018 Crash. Retrieved from https://arstechnica.com/cars/2019/11/how-terrible-software-design-decisions-led-to-ubers-deadly-2018-crash/.

[33] Lizhi Liao, Heng Li, Weiyi Shang, and Lei Ma. 2022. An empirical study of the impact of hyperparameter tuning and model optimization on the performance properties of deep neural networks. *ACM Trans. Softw. Eng. Methodol.* 31, 3 (2022), 1–40. https://doi.org/10.1145/3506695

[34] Lucas Liebenwein, Cenk Baykal, Brandon Carter, David Gifford, and Daniela Rus. 2021. Lost in pruning: The effects of pruning neural networks beyond test accuracy. *Proc. Mach. Learn. Syst.* 3 (2021), 93–138.

[35] Jiayi Liu, Samarth Tripathi, Unmesh Kurup, and Mohak Shah. 2020. Pruning algorithms to accelerate convolutional neural networks for edge applications: A survey. Retrieved from http://arxiv.org/abs/2005.04275.

[36] Suyun Liu and Luis Nunes Vicente. 2022. Accuracy and fairness trade-offs in machine learning: A stochastic multi-objective approach. *Comput. Manage. Sci.* 19, 3 (2022), 513–537. https://doi.org/10.1007/s10287-022-00425-z

[37] Xuechen Liu, Md Sahidullah, and Tomi Kinnunen. 2020. A comparative Re-assessment of feature extractors for deep speaker embeddings. In *Proceedings of the Annual Conference of the International Speech Communication Association, (INTERSPEECH'20)*. 3221–3225. https://doi.org/10.21437/Interspeech.2020-1765

[38] Akhil Mathur, Tianlin Zhang, Sourav Bhattacharya, Petar Velickovic, Leonid Joffe, Nicholas D. Lane, Fahim Kawsar, and Pietro Lió. 2018. Using deep data augmentation training to address software and hardware heterogeneities in wearable and smartphone sensing devices. In *Proceedings of the 17th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN'18)*. IEEE, 200–211.

[39] Mark Mazumder, Sharad Chitlangia, Colby Banbury, Yiping Kang, Juan Ciro, Keith Achorn, Daniel Galvez, Mark Sabini, Peter Mattson, David Kanter, Greg Diamos, Pete Warden, Josh Meyer, and Vijay Janapa Reddi. 2021. Multilingual spoken words corpus. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks (NeurIPS'21)*.

[40] Ninareh Mehrabi, Fred Morstatter, Nripsuta Saxena, Kristina Lerman, and Aram Galstyan. 2019. A survey on bias and fairness in machine learning. Retrieved from https://arxiv.org/abs/1908.09635.

[41] Shira Mitchell, Eric Potash, Solon Barocas, Alexander D'Amour, and Kristian Lum. 2021. Algorithmic fairness: Choices, assumptions, and definitions. *Annu. Rev. Stat. Appl.* 8 (2021), 141–163. https://doi.org/10.1146/annurev-statistics-042720-125902

[42] Alessandro Montanari, Manuja Sharma, Dainius Jenkus, Mohammed Alloulah, Lorena Qendro, and Fahim Kawsar. 2020. ePerceptive: Energy reactive embedded intelligence for batteryless sensors. In *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*. 382–394.

[43] Pardis Emami Naeini, Sruti Bhagavatula, Hana Habib, Martin Degeling, Lujo Bauer, Lorrie Cranor, Norman Sadeh, Santa Clara, Pardis Emami-naeini, Sruti Bhagavatula, Hana Habib, Martin Degeling, Lujo Bauer, Lorrie Faith Cranor, and Norman Sadeh. 2017. Privacy expectations and preferences in an IoT world this paper is included in the proceedings of the privacy expectations and preferences in an IoT world. In *Proceedings of the 13th Symposium on Usable Privacy and Security (SOUPS'17)*. Retrieved from https://www.usenix.org/conference/soups2017/technical-sessions/presentation/naeini.

[44] Danielle J. Navarro, David R. Foxcroft, and Thomas J. Faulkenberry. 2019. Factorial Anova. In *Learning Statistics with JASP: A Tutorial for Psychology Students and Other Beginners*. 327–380. https://doi.org/10.1002/9781119121077.ch10

[45] Kirtan Padh, Diego Antognini, Emma Lejal-Glaude, Boi Faltings, and Claudiu Musat. 2021. Addressing fairness in classification with a model-agnostic multi-objective algorithm. In *Proceedings of the 37th Conference on Uncertainty in Artificial Intelligence (UAI'21)*. 600–609.

[46] Dana Pessach and Erez Shmueli. 2022. A review on fairness in machine learning. *Comput. Surveys* 55, 3 (2022), 1–44. https://doi.org/10.1145/3494672

[47] ProKNX. 2022. Smart Home Technology Keeps People Happy at Home for Longer. Retrieved from https://www.proknx.com/en/news/2020/smart-home-technology-keeps-people-happy-at-home-for-longer/.

[48] Shangshu Qian, Hung Viet Pham, Thibaud Lutellier, Zeou Hu, Jungwon Kim, Lin Tan, Yaoliang Yu, Jiahao Chen, J. P. Morgan, and Sameena Shah. 2021. Are my deep learning systems fair? An empirical study of fixed-seed training. In *Proceedings of the 35th Conference on Neural Information Processing Systems (NeurIPS'21)*. Retrieved from https://github.com/lin-tan/fairness-variance/.

[49] Inioluwa Deborah Raji, Timnit Gebru, Margaret Mitchell, Joy Buolamwini, Joonseok Lee, and Emily Denton. 2020. Saving face: Investigating the ethical concerns of facial recognition auditing. In *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society (AIES'20)*. 145–151. https://doi.org/10.1145/3375627.3375820

[50] Apple Inc. (Press release). 2021. Apple Advances Its Privacy Leadership with iOS 15, iPadOS 15, macOS Monterey, and watchOS 8. Retrieved from https://www.apple.com/newsroom/2021/06/apple-advances-its-privacy-leadership-with-ios-15-ipados-15-macos-monterey-and-watchos-8/.

[51] Tara N. Sainath and Carolina Parada. 2015. Convolutional neural networks for small-footprint keyword spotting. In *Proceedings of the Annual Conference of the International Speech Communication Association (INTERSPEECH'15)*. 1478–1482. https://doi.org/10.21437/interspeech.2015-352

[52] Susumu Saito, Yuta Ide, Teppei Nakano, and Tetsuji Ogawa. 2021. VocalTurk: Exploring feasibility of crowdsourced speaker identification. In *Proceedings of the Annual Conference of the International Speech Communication Association (INTERSPEECH'21)*. 2932–2936. https://doi.org/10.21437/Interspeech.2021-464

[53] D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-François Crespo, and Dan Dennison. 2015. Hidden technical debt in machine learning systems. *Advances in Neural Information Processing Systems*.

[54] Julien Siebert, Lisa Joeckel, Jens Heidrich, Adam Trendowicz, Koji Nakamichi, Kyoko Ohashi, Isao Namba, Rieko Yamamoto, and Mikio Aoyama. 2021. Construction of a quality model for machine learning systems. *Softw. Qual. J.* 0123456789 (2021). https://doi.org/10.1007/s11219-021-09557-y

[55] Harvineet Singh, Rina Singh, Vishwali Mhasawade, and Rumi Chunara. 2021. Fairness violations and mitigation under covariate shift. In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency (FAccT'21)*. 3–13. https://doi.org/10.1145/3442188.3445865

[56] Rita Singh. 2019. *Profiling Humans from Their Voice*. https://doi.org/10.1007/978-981-13-8403-5

[57] Scott Small, Sara Khalid, Paula Dhiman, Shing Chan, Dan Jackson, Aiden Doherty, and Andrew Price. 2021. Impact of reduced sampling rate on accelerometer-based physical activity monitoring and machine learning activity classification. *J. Measure. Phys. Behav.* 4, 4 (2021), 298–310.

[58] Rachael Tatman. 2017. Gender and dialect bias in YouTube's automatic captions. Retrieved from https://aclanthology.org/W17-1606/. https://doi.org/10.18653/v1/w17-1606

[59] Wiebke Toussaint, Akhil Mathur, Aaron Yi Ding, and Fahim Kawsar. 2021. Characterising the role of pre-processing parameters in audio-based embedded machine learning. In *Proceedings of the 3rd International Workshop on Challenges in Artificial Intelligence and Machine Learning for Internet of Things (AIChal-lengeIoT'21)*. ACM, 439–445. https://doi.org/10.1145/3485730.3493448

[60] Wiebke Toussaint, Dave Van Veen, Courtney Irwin, Yoni Nachmany, Manuel Barreiro-Perez, Elena Díaz-Peláez, Sara Guerreiro de Sousa, Liliana Millán, Pedro L. Sánchez, Antonio Sánchez-Puente, Jesús Sampedro-Gómez, P. Ignacio Dorado-Díaz, and Víctor Vicente-Palacios. 2020. Design considerations for high impact, automated echocardiogram analysis. In *Proceedings of the International Conference on Machine Learning (ICML'20)*. Retrieved from http://arxiv.org/abs/2006.06292.

[61] George Tucker, Minhua Wu, Ming Sun, Sankaran Panchapagesan, Gengshen Fu, and Shiv Vitaladevuni. 2016. Model compression applied to small-footprint keyword spotting. In *Proceedings of the Annual Conference of the International Speech Communication Association (INTERSPEECH'16)*. 1878–1882. https://doi.org/10.21437/Interspeech.2016-1393

[62] Jennifer Pattison Tuohy. 2021. Amazon Alexa's New Elder Care Service Launches Today. Retrieved from https://www.theverge.com/2021/12/7/22822026/amazon-alexa-together-elder-care-price-features-release-date.

[63] Sahil Verma and Julia Rubin. 2018. Fairness definitions explained. In *Proceedings of the International Conference on Software Engineering*. 1–7. https://doi.org/10.1145/3194770.3194776

[64] Hugo Villamizar, Marcos Kalinowski, and Helio Lopes. 2022. A catalogue of concerns for specifying machine learning-enabled systems. Retrieved from http://arxiv.org/abs/2204.07662.

[65] Sandra Wachter, Brent Mittelstadt, and Chris Russell. [n.d.]. Bias preservation in machine learning: The legality of fairness metrics under EU non- discrimination law. *West Virginia Law Review*, 1–51. Retrieved from https://ssrn.com/abstract=3792772.

[66] Yuxuan Wang, Pascal Getreuer, Thad Hughes, Richard F. Lyon, and Rif A. Saurous. 2017. Trainable frontend for robust and far-field keyword spotting. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP'17)*. 5670–5674. https://doi.org/10.1109/ICASSP.2017.7953242

[67] Pete Warden. 2018. Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition. Retrieved from https://arxiv.org/abs/1804.03209.

[68] Tien Ju Yang, Andrew Howard, Bo Chen, Xiao Zhang, Alec Go, Mark Sandler, Vivienne Sze, and Hartwig Adam. 2018. NetAdapt: Platform-aware neural network adaptation for mobile applications. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 11214 LNCS, 289–304. https://doi.org/10.1007/978-3-030-01249-6{_}18

[69] Yundong Zhang, Naveen Suda, Liangzhen Lai, and Vikas Chandra. 2017. Hello edge: Keyword spotting on microcontrollers. Retrieved from https://arxiv.org/abs/1711.07128.

Received 20 May 2022; revised 1 March 2023; accepted 6 March 2023