Accurate lightweight calibration methods for mobile low-cost particulate matter sensors [⋆]

Per-Martin Jørstad[1], Marek Wojcikowski[2], Tuan-Vu Cao[3], Jean-Marie Lepioufle[3], Krystian Wojtkiewicz[4], and Phuong Hoai Ha[1⋆⋆]

[1] UiT The Arctic University of Norway, Norway
{per.m.jorstad, phuong.hoai.ha}@uit.no

[2] Gdansk University of Technology, Gdansk, Poland
marwojci1@pg.edu.pl

[3] Norwegian Institute for Air Research, Oslo, Norway
{tvc,jml}@nilu.no

[4] Wrocław University of Science and Technology, Poland
krystian.wojtkiewicz@pwr.edu.pl

Abstract. Monitoring air pollution is a critical step towards improving public health, particularly when it comes to identifying the primary air pollutants that can have an impact on human health. Among these pollutants, particulate matter (PM) with a diameter of up to 2.5 micrometers (or PM2.5) is of particular concern, making it important to continuously and accurately monitor pollution related to PM. The emergence of mobile low-cost PM sensors has made it possible to monitor PM levels continuously in a greater number of locations. However, the accuracy of mobile low-cost PM sensors is often questionable as it depends on geographical factors such as local atmospheric conditions.

This paper presents new calibration methods for mobile low-cost PM sensors that can correct inaccurate measurements from the sensors in real-time. Our new methods leverage Neural Architecture Search (NAS) to improve the accuracy and efficiency of calibration models for mobile low-cost PM sensors. The experimental evaluation shows that the new methods reduce accuracy error by more than 26% compared with the state-of-the-art methods. Moreover, the new methods are lightweight, taking less than $2.5ms$ to correct each PM measurement on Intel Neural Compute Stick 2, an AI-accelerator for edge devices deployed in air pollution monitoring platforms.

1 Introduction

Air pollution monitoring is crucial to improving public health. According to the World Health Organization (WHO), air pollution places 99% of the global pop-

[⋆] This work was supported in part by the National Centre for Research and Development (grant NOR/POLNOR/HAPADS/0049/2019-00), Research Council of Norway (grant 270053) and European Commission (grant 101086541). The authors would like to thank the Agency of Regional Atmospheric Monitoring Gdansk-Gdynia-Sopot in Poland for providing data from the reference air pollution measurement stations.

[⋆⋆] Corresponding author

ulation at risk of developing various diseases and causes 6.7 million deaths each year [14]. The primary air pollutants responsible for affecting human health include Particulate Matter (PM) and nitrogen dioxide (NO2). Out of the contaminants listed, particles with a diameter of up to 2.5 μm (also known as $PM_{2.5}$) are particularly hazardous. They have the ability to breach the body's natural barriers and enter the bloodstream, leading to cardiovascular and/or respiratory problems [21]. As a result, continuous and accurate monitoring of pollution related to PM is of great significance [4, 13]. Dependable measurements of PM are essential for creating early warning systems that can provide information on sudden spikes or prolonged high levels of pollutants.

Current PM monitoring relies on a limited number of costly air monitoring stations due to the inaccuracy of mobile low-cost PM sensors (MLCS). Although the fixed stations can provide high quality measurements, their complex measurement techniques result in very high purchase and maintenance cost, preventing them from being deployed widely and monitoring many locations. The mobile low cost PM sensors, on the other hand, enable covering more locations quickly and cheaply. The MLCS sensors enable localizing hot spots and mapping the spatial and temporal dynamics of air pollution, particularly in urban areas. However, the accuracy of the MLCS sensors is often questionable since it depends on geographical factors such as local atmospheric conditions and pollutant concentration levels [3]. Because of the small size and low cost requirements of the MLCS sensors, the MLCS sensors cannot be equipped with built-in calibrator, air filtering equipment, temperature controller (cooler and heater) nor relative humidity controllers that are used in the air monitoring stations.

Improving the accuracy of mobile low-cost PM sensors with in-situ software-based calibration models is an emerging research area [11,17,20,22]. The accuracy of the calibration methods can be evaluated using root mean square error RMSE ($\mu g/m^3$), which varies from 2.82 [20] to 8.48 [22]. The seminal work [20] has developed a kriging-based model to correct the measurement from MLCS sensors. The kriging-based correction model is established using optimized lower and upper bounds, which are tailored to the environmental conditions for which both PM measurements from the reference station and sensor are available. Therefore, the model's corrective capability is restricted to these specified ranges. Alternatively, machine learning-based (ML-based) methods have been proposed to improve the MLCS measurement accuracy [9, 11]. The PM measurement accuracy of the ML-based methods is still limited with reported RMSE ($\mu g/m^3$) from 4.21 to 5.45 [11].

In this work, we propose new ML-based methods (called NAS-RS and NAS-RE) that significantly improve the PM measurement accuracy of mobile low cost sensors, achieving RMSE ($\mu g/m^3$) of 1.89, reducing accuracy error by more than 26% compared with the state-of-the-art methods. Moreover, our new methods are lightweight and their correction of each PM measurement takes less than 2.5 ms on Intel Neural Compute Stick 2 (NCS2) [6], an AI-accelerator for edge devices deployed in air pollution monitoring platforms [1]. Our new methods leverage Neural Architecture Search (NAS) [24] to improve the accuracy of calibration models for MLCS. NAS is a growing field within the domain of deep learning model design. NAS aims to find the best architecture for a new model by automating architecture engineering, avoiding the time-consuming and error-prone manual development of neural architectures. Even though NAS has several potential benefits, it is still a young field [10,15,16,24]. There are few active NAS frameworks for the mainstream audience, making NAS difficult to leverage by users on their practical problems. Furthermore, the few libraries that do exist, often have a tendency to simplify the NAS-process to hyper-parameter tuning [23]. Although this works, to a degree, it does not fully utilize NAS capabilities. In this work, we leverage NAS to find the most optimal architecture for a regression model to correct the PM measurement from MLCS that can run on edge devices (e.g., Intel Neural Compute Stick 2 (NCS2) [6]). Furthermore, to examine the different features of NAS, we compare two NAS methods: a random search method (NAS-RS) and a highly sophisticated evolution method called regularized evolution (NAS-RE).

The rest of the paper is organized as follows. Section 2 describes the design and implementation of the new calibration methods. Section 3 presents the model training and testing results. Section 4 discusses the results and Section 5 concludes the paper.

2 Design and Implementation

2.1 Data Preprocessing

Since the input data consists of several sensor measurements, with many of vastly different ranges, the model was given standardized data as input. That is, the model was given data where all the input values had a mean of 0 and a standard deviation of 1. Equation 1 show the standardization formula.

$$y = \frac{(x - \mu_x)}{\sqrt{\sigma_x^2}} \tag{1}$$

where x is some input feature, μ_x denotes the mean of the input features, and σ_x^2 denotes the input feature's variance.

2.2 Base Model Design for MLCS Calibration

To correct the inaccurate measurement from MLCS, we first build a small regression model as the base model. Figure 1 shows this base model. It consists of three fully connected layers, each with a width - set empirically - to 64 and initialized using Kaiming's initialization [5]. Following each fully connected layer is ReLU activation function to act as a non-linearity. Both the input and the output of the model is configurable.

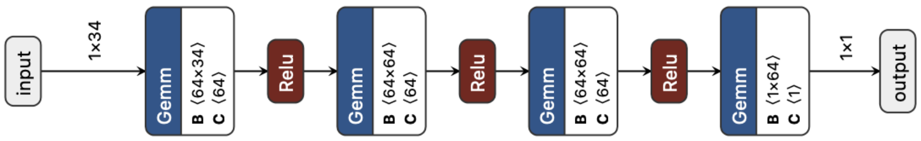
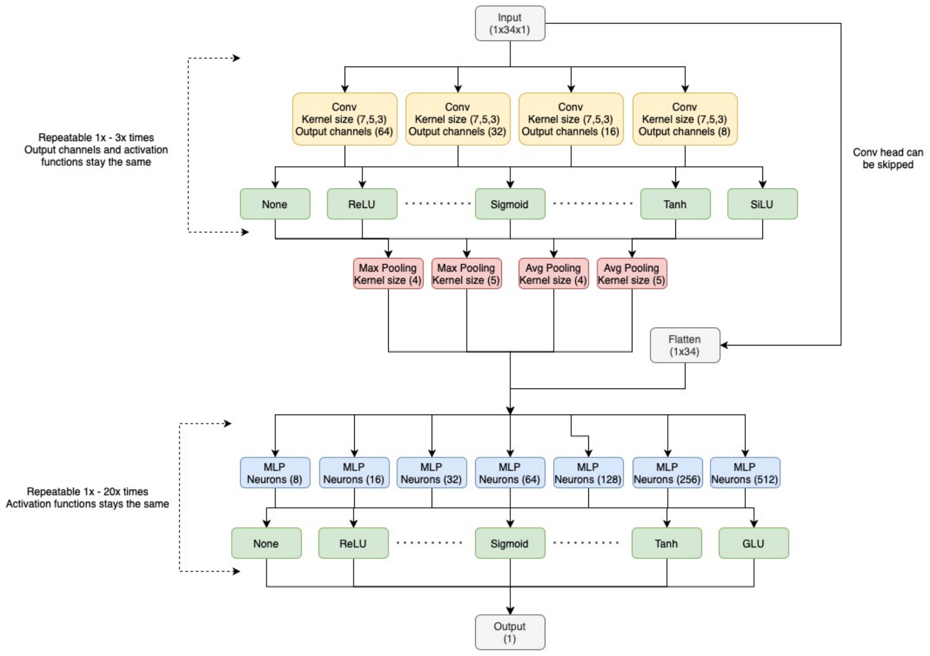


Fig. 1. Base Model



Input (1x34x1)
Repeatable 1x - 3x times Output channels and activation functions stay the same
Conv Kernel size (7,5,3) Output channels (64)
Conv Kernel size (7,5,3) Output channels (32)
Conv Kernel size (7,5,3) Output channels (16)
Conv Kernel size (7,5,3) Output channels (8)
Conv head can be skipped
None
ReLU
Sigmoid
Tanh
SiLU
Max Pooling Kernel size (4)
Max Pooling Kernel size (5)
Avg Pooling Kernel size (4)
Avg Pooling Kernel size (5)
Flatten (1x34)
Repeatable 1x - 20x times Activation functions stays the same
MLP Neurons (8)
MLP Neurons (16)
MLP Neurons (32)
MLP Neurons (64)
MLP Neurons (128)
MLP Neurons (256)
MLP Neurons (512)
None
ReLU
Sigmoid
Tanh
GLU
Output (1)

Fig. 2. Model Search Space

2.3 NAS Model Search Space for MLCS Calibration

Figure 2 depicts our NAS model search space created for the base regression model. It is essentially a jumbo net expanding the depth of the network with an optional convolution head and a bigger fully connected tail.

The convolution head is the first section of the expanded network and consists of up to three convolution layers followed by a single max-pooling or average-pooling layer. The head functions as an optional and lightweight feature extractor motivated by the design of popular computer vision models like the VGG16 [19]. Its purpose is essentially to try and find features that the fully connected tail can use later for the actual regression. Since the head is optional, it can be skipped, allowing the data to flow directly to the fully connected tail.

In the head, the search space gives the convolution layers an independent choice of kernel size and a shared choice of activation function and output channel. This means that each layer will have its own kernel size (e.g., 3, 5, or 7), but will share an output channel size (e.g., 8, 16, 32 or 64) and activation function (listed below). For the pooling layer, the search space gives it the option to have a kernel size of either 4 or 5. The values given by the search space were chosen empirically.

Following the convolution head is the fully connected tail. As in the base model, the fully connected layers are to do the regression and yield the final output. As shown in Figure 2, the search space allows the tail to have a depth ranging from 1-20, where each layer has a shared width (e.g., 8, 16, 32, 64, 128, 256 or 512) and activation function. The values given by the search space were chosen empirically.

For both the convolution head and the fully connected tail, the following activation functions can be selected: ELU, Hardshrink, HardSigmoid, Hardtanh, HardSwish, LeackyReLU, LogSigmoid, PReLU, ReLU, ReLU6, RReLU, CELU, GELU, Sigmoid, SiLU, Mish, SoftPlus, Softshrink, Softsign, Tanh, Tahnshrink, GLU and No-activation, for a total of 22 choices. However, the convolution layers were not able to utilize GLU, and thus the convolution head has 21 choices of activation functions.

With the convolution head consisting of 3 optional convolution layers with an independent kernel of 3 different choices, 21 shared activation functions for the convolutions, 22 shared activation functions for the fully connected layers, 4 different output channel sizes for the convolutions, 2 optional pooling layers, 2 options for the pooling kernel-size, up to 20 fully connected layers, and 7 width choices for the fully connected layers, the model space features a total of $3^3 * 21 * 22 * 4 * 2 * 2 * 20 * 7 = 27.9$ million candidate architectures!

2.4 NAS Methods for MLCS Calibration

We leverage two NAS methods for MLCS calibration, namely Brute Force NAS and Evolution-based NAS, to get insights into the affects of NAS methods on MLCS calibration. NAS can be described as a gradient-based method that exploits reinforcement learning (RL) to iteratively find better networks. The method was first introduced in [24], and is based on the idea that most DNNs have the structure and connectivity that can be expressed by a variable-length string. This allows most DNN architectures to be defined by generative mechanisms that can define such a string. If the mechanism is trainable, it can be taught to generate better and better architectures over time.

The NAS process consists of two main parts: a static search space, containing all possible model configurations; and an iterative training loop that uses the search space to train the generative mechanism (usually called the controller). More specifically, a round of NAS can be expressed as follows:

1. The controller samples a network (referred as child network) from the search space.
2. The child network is trained and tested using a predefined training and validation set.

3. The child network produces a validation score that the controller can use to update itself.
4. The controller receives reward signal and updates itself, hopefully getting slightly better. After the update, the controller once again samples a network from the search space to create a new child network. Then, the entire process repeats again.

Brute Force NAS. Brute force NAS considers algorithms that search their search space unintelligently without using a controller for guidance, e.g., Grid-Search [12], a method that systematically tries all options one-by-one. A variant of Brute Force NAS is Random-Search NAS (NAS-RS), which chooses a candidate from the search space at random. Although unintelligent, NAS-RS has been reported to yield worthwhile results [2].

Evolution-based NAS. Evolution-based NAS is a method inspired by real life biological evolution. Hence, instead of using a controller, it aims to directly improve the model space by letting good models mutate and bad models die. By doing this repeatedly, only good models will be the ones standing at the end [16].

An example of this approach can be found in Regularized Evolution [16] (RE, but also known as Aging Evolution - AE). It follows the same principle as other evolution based methods. However, instead of killing the worst performing models in the population (or search space), it rather kills the oldest one. The change acts as some sort of regularization, preventing overfitting.

3 Evaluation

To train and test both the base model and the child networks of the NAS process, we utilized a dataset of measurements from the HAPADS air pollution monitoring platform (see Figure 3) [1]. The HAPADS platform deploys the mobile low cost SPS30 PM sensor [18] and provides the dataset of PM measurements for evaluation in this paper. The dataset contains 765 air quality measurements by the HAPADS platform over one month period for an area in Gdansk, Poland. The dataset includes inaccurate measurements from the SPS30 PM sensor in the HAPADS platform and highly accurate measurements from a nearby modern weather station ARMAG. To form a regression problem, we used all measurement data from the sensor (e.g., PM, air pressure, humidity, temperature) in the HAPADS platform to predict the accurate PM2.5 measurements

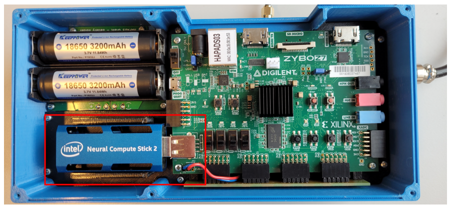


Fig. 3. HAPADS air pollution monitoring platform [1]

Table 1. Base model Mean Absolute Error (MAE) and Mean Square Error (MSE), where columns Avg and Std are the average and standard deviation.

	TensorFlow					PyTorch					Avg	Std
MAE	1.88	1.83	1.84	1.84	1.90	1.80	1.72	1.75	1.74	1.70	1.80	0.06
MSE	6.90	6.42	6.63	7.13	7.01	6.38	5.88	6.62	6.16	6.52	6.57	0.36

Table 2. Mean preprocessing time and mean model runtime on the test set, using an ONNX and OpenVINO IR version of the base model, where columns Avg and Std are the average and standard deviation.

	ONNX - PyTorch						OpenVINO IR - PyTorch						Avg	Std
	Warm Start			Cold Start			Warm Start			Cold Start				
Preprocessing (µs)	10.73	6.35	9.93	5.87	7.92	23.3	6.39	7.01	10.50	11.19	22.74	22.51	12.03	6.48
Model (ms)	1.78	1.75	1.75	1.70	1.85	2.00	1.76	1.78	1.80	1.74	1.86	1.83	1.80	0.07

from the modern weather station ARMAG (i.e., the ground truth). During the development of the base model and NAS models, roughly 60% of the dataset was used as a training set while the remaining 40% was used for validation/testing. We used mean absolute error (MAE) and mean square error (MSE) for the loss and accuracy, respectively.

3.1 Base Model

We trained the base model for 100 epochs on the train set using the Adam optimizer. We utilized a machine with Intel i7-7700 4-core CPU for training. For testing, we used the Intel Neural Compute Stick 2 (NCS2), an AI-accelerator for edge devices deployed in the HAPADS air pollution monitoring platform (see Figure 3) [1].

Table 1 shows the test loss (MAE) and test accuracy (MSE) after ten runs of the base model. Among these runs, five runs were done using a model instance created using the TensorFlow (TF) framework while the remaining five runs were achieved using a model instance from the PyTorch (PT) framework. Model instances were trained from the bottom before testing and tested in both the ONNX and OpenVINO IR formats supported by NCS2. The results show that base model achieves relatively good results with a mean test MAE of 1.80 and mean test MSE of 6.57. The results also show that the PT implementation is slightly better than the TF one. However, the difference is quite small. Interestingly, there were no differences between running the models in the ONNX and OpenVINO IR formats.

Table 2 shows the mean latency of data pre-processing (in microseconds) and model (in millisecond) for each test sample. The test was done 12 times, 6 times using the OpenVINO IR model format and 6 times using the ONNX model format. In both cases, the model came from a PT instance. For both the OpenVINO IR and ONNX runs, 3 of the 6 runs were done from a cold state, namely from a nearly idle state (i.e., 3-minute pause between each run). The other 3 runs came from a warm state, namely the underlying hardware and network had already processed 1000 samples unrelated to the test set before starting the test. There were also no pause between test runs while the model was in a warm state. The table shows that the average latency of data pre-processing is negligible to that of model (12.03μs vs. 1.80ms). Both the ONNX and OpenVINO IR implementations have similar average latency.

3.2 NAS Models

Setup We evaluated NAS capabilities to find good architectures by exploring the model space (defined in Section 2.3) using Regularized Evolution (NAS-RE) and Random Search (NAS-RS) described in Section 2.4. During the evaluation, both search strategies found 200 architectures and they repeated 3 times. NAS-RE was run with a population size (model space size) of 100, a tournament size of 25 and mutation probability of 0.05 in a specific direction. NAS-RE could use the same configuration twice (i.e., deduplication set to false) while NAS-RS did not use a model configuration more than once (i.e., deduplication set to true) and ignored failed models, following the NNI framework recommendation [12]. The NAS processes were conducted using the NNI framework on a MacBook Pro with 2,2 GHz 4-core Intel Core i7 processor. The metrics used for the evaluation were the same as for the base model evaluation: MAE for loss and MSE for accuracy.

Results Table 3 and table 4 show the 10 best models after the 3 runs of NAS-RE and NAS-RS, respectively. The columns are divided into three main sections (Conv1D, Pool and MLP) and depict the choices of the 11 options discussed in Section 2.3 for each model, together with the model's final accuracy MSE on the test set.

The Conv1D section describes the convolution part of the convolution head. Columns "Num Layers", "Kernel Size", "Out Channels" and "Activation Function" refer to the numbers of convolution layers, the kernel size of each layer, the shared output channel size, and the shared activation function that follows each layer, respectively. For example, the model at the last row in Table 3 has 2 convolution layers. The first layer has a kernel size of 7 while the second has a kernel size of 3. Both layers have 8 output channels and use HardSigmoid as their activation function.

The Pool section describes the pooling part of the convolution head. In this section, columns "Kernel Size" and "Type" denote the filter size of the pooling window and the kind of pooling operation, respectively. For example, the model at the last row in Table 3 has an average pooling layer with the window size of 4.

Lastly, the MLP (Multilayer Perceptron) section describes the fully connected part of the models. Columns "Num Layers", "Num Neurons" and "Activation Function" refer to the number of MLP layers, the shared width of all layers and the shared activation function that follows each layer. For example, the model at the last row in Table 3 has a three MLP layers with a width of 32 and a GELU activation function.

Table 3. Top 10 best models using NAS-RE

Conv1D						Pool		MLP			Final MSE
Num Layers	Kernel Size			Out Channels	Activation Function	Kernel Size	Type	Num Neurons	Num Layers	Activation Function	
N/a	N/a	N/a	N/a	N/a	N/a	N/a	N/a	32	18	N/a	3.57
3	5	3	3	16	N/a	5	Max	8	3	Mish	3.74
N/a	N/a	N/a	N/a	N/a	N/a	N/a	N/a	512	2	Sigmoid	3.84
3	5	3	3	16	N/a	4	Max	8	3	Mish	3.92
2	7	3	N/a	8	Sigmoid	4	Avg	32	3	GELU	3.95
N/a	N/a	N/a	N/a	N/a	N/a	N/a	N/a	32	2	Sigmoid	3.97
2	7	7	N/a	8	HardTanh	5	Avg	512	10	LeakyReLU	3.99
2	7	3	N/a	8	Sigmoid	4	Avg	32	3	GELU	4.00
N/a	N/a	N/a	N/a	N/a	N/a	N/a	N/a	512	2	Sigmoid	4.01
2	7	3	N/a	8	HardSigmoid	4	Avg	32	3	GELU	4.02

Table 4. Top 10 best models using NAS-RS

Conv1D						Pool		MLP			Final MSE
Num Layers	Kernel Size			Out Channels	Activation Function	Kernel Size	Type	Num Neurons	Num Layers	Activation Function	
N/a	N/a	N/a	N/a	N/a	N/a	N/a	N/a	8	1	HardSigmoid	3.94
3	7	7	7	8	HardTanh	5	Max	128	6	N/a	3.95
3	3	5	7	16	CELU	4	Max	512	4	SiLU	4.01
2	3	3	N/a	64	RReLU	4	Max	256	1	HardSigmoid	4.05
3	3	3	7	16	SiLU	5	Avg	256	8	N/a	4.10
2	7	7	N/a	16	HardSigmoid	4	Max	256	1	SiLU	3.97
2	7	7	N/a	8	HardTanh	5	Avg	512	10	LeakyReLU	4.2
N/a	N/a	N/a	N/a	N/a	N/a	N/a	N/a	8	8	N/a	4.23
N/a	N/a	N/a	N/a	N/a	N/a	N/a	N/a	8	14	N/a	4.26
3	7	5	5	16	N/a	4	Avg	8	3	ReLU	4.27

The results show that the overall best model - in terms of accuracy - was a pure MLP network, found by NAS-RE (the top model in Table 3). The network has 18 layers of which each has 32 neurons and no activation function. Interestingly, the MSE of the network is only 0.17 better than that of the second best model, a combination of convolution and MLP networks also found by NAS-RE. Comparing the two search strategies, it is clear that NAS-RE achieves better accuracy than NAS-RS. While NAS-RE achieved a top-5 MSE of 3.95, NAS-RS only achieved a top-1 MSE of 3.94 and a top-5 MSE of 4.10.

Figure 4 and Figure 5 shows the PM measurements of the mobile low cost SPS30 PM sensor [18] (HAPADS) against the ground truth from the reference station (ARMAG) before and after corrected by our NAS-RE calibration method using the best model (i.e., top model in Table 3), respectively. The results show that our calibration methods can significantly improve the accuracy of the mobile low cost PM sensor. Table 5 shows the accuracy comparison of our new calibration methods (NAS-RE and NAS-RS) with state-of-the-art methods for mobile low-cost PM sensors.

3.3 Evaluation of NAS-RE and NAS-RS for Particulate Matter Monitoring Platform

To evaluate the viability of the new calibration models NAS-RE and NAS-RS in practice, we run the top NAS-RE and NAS-RS models on Intel Neural Compute Stick 2 (NCS2), an AI-accelerator for edge devices deployed in the HAPADS air pollution monitoring platform (see Figure 3). Since HardSigmoid, the activation function used in the best model from NAS-RS, is not supported by the NCS2, we emulated HardSigmoid using HardTanh. Namely, we replaced HardSigmoid

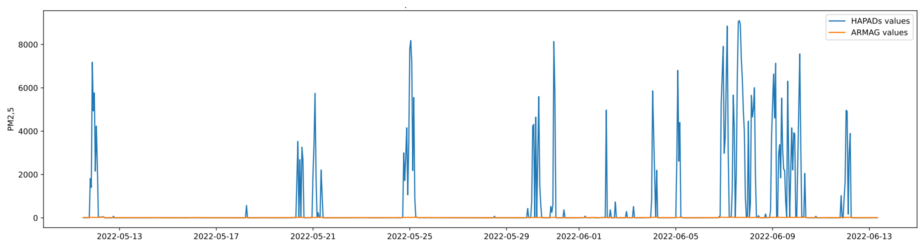


Fig. 4. The PM measurement of a low cost PM sensor (HAPADS) without correction and the ground truth from a reference station (ARMAG). Figure 5 zooms in on the the ground truth.

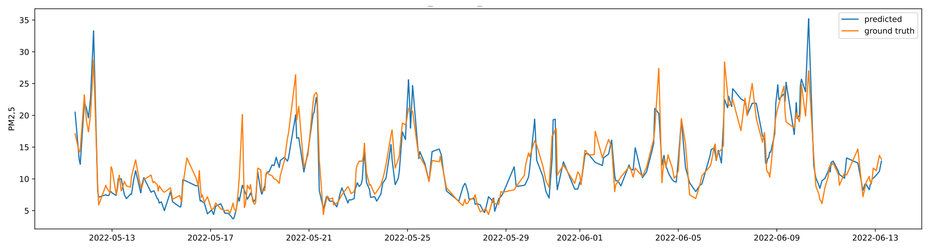


Fig. 5. The PM measurement of a low cost PM sensor (HAPADS) corrected by our NAS-RE method and the ground truth (ARMAG).

with HardTanh using a lower limit of -3 and and upper limit of 3, then scaling the result by a factor of 1/6 and adding 0.5 to it.

Table 6 and table 7 show the test loss (MAE) and test accuracy (MSE) after 10 runs using the best NAS-RE model and best NAS-RS model, respectively. All model instances were trained from the bottom before testing and tested in both the ONNX and OpenVINO IR formats. The tables show that the top-1 NAS-RE model actually performed worse than anticipated, having a mean MSE of 5.69. Meanwhile the top-1 NAS-RS model performed better than anticipated and reported a mean MSE of 3.90.

Tables 8 and 9 show the mean latency of data pre-processing (in microseconds) and model (in millisecond) for each test sample. The test was done 12 times, 6 times using the OpenVINO IR model format and 6 times using the ONNX model format. In both cases, the model came from a PyTorch instance. For both the OpenVINO IR and ONNX runs, 3 out of the 6 runs were done from a cold state, namely from a nearly idle state (3 minute pause between each run). The other 3 runs came from a warm state, namely the underlying hardware and network had already processed 1000 samples unrelated to the test set before starting the test. There is also no pause between test runs while the model was in a warm state. The tables show that the average latency of data pre-processing is negligible to that of model (e.g., $10.62\mu s$ vs. $1.74ms$ in Table 9). Both the ONNX and OpenVINO IR implementations have similar model latency. The re-

Table 5. Accuracy comparison of our new calibration methods (NAS-RE and NAS-RS) with previous methods for mobile low-cost sensors (MLCS). Except for the Cluster analysis method [22] using the PMS7003 PM sensor, all methods use the SPS30 PM sensor [18] with the same measurement.

Methods	RMSE ($\mu g/m^3$)	NAS-RE error reduction (%)
None (raw measurement)	1695.63	100%
Kriging-based [20]	2.82	33%
Cluster analysis [22]	8.48	78%
Linear regression	3.97	52%
Random forest [9]	2.55	26%
Gradient Boosting Regression Tree [8]	2.58	27%
AdaBoost [7]	2.77	32%
NAS-RS (this paper)	1.98	5%
NAS-RE (this paper)	1.89	0%

Table 6. Mean Absolute Error and Mean Square Error - Best NAS-RE Model, where columns Avg and Std are the average and standard deviation.

	OpenVINO/ONNX - PyTorch										Avg	Std
MAE	1.90	1.72	1.79	1.65	1.75	1.66	1.63	1.57	1.58	1.74	1.70	0.09
MSE	7.94	6.28	5.79	5.00	5.59	5.42	4.95	4.86	4.95	6.07	5.69	0.88

sults show that the best NAS-RS model (with average model runtime of $1.74ms$) is faster than the best NAS-RE model (with average model runtime of $2.44ms$). Both models are fast enough to correcting PM measurements from the HAPADS platform where measurements occur every 10 minutes.

4 Discussion

Even though the top-1 NAS-RE model failed to bring satisfactory results on the Intel NSC2 (i.e., its high MSE of 5.69), it is clear that the best NAS-RS model succeeded and outperformed all tested models (i.e., its lowest MSE of 3.90). Why it is so effective is likely because of its simplicity. Having only 8 neurons, for instance, reduces the risk of overfitting, which is a problem with many deeper neural networks. Because of its shallowness, its HardSigmoid non-linearity could work optimally with less risk of running into the vanishing gradient problem.

In terms of latency, we see the same results as discussed above: the top-1 NAS-RE model is worse than the top-1 NAS-RS model that outperforms all models. Considering the total number of neurons in each model, this is not a surprising result. The baseline model has 3*64 = 192 neurons in its hidden layers and the top-1 NAS-RE model has 32*18 = 576 neurons in its hidden layers while the top-1 NAS-RS model only has 8 neurons in its hidden layer. Since fewer neurons imply fewer parameters and faster data throughput, it is clear that the model with the fewest neurons (i.e., NAS-RS) will be faster than all other models, at least when comparing MLPs.

Table 7. Mean Absolute Error and Mean Square Error - Best NAS-RS Model, where columns Avg and Std are the average and standard deviation.

	OpenVINO/ONNX - PyTorch										Avg	Std
MAE	1.35	1.49	1.42	1.48	1.39	1.35	1.49	1.43	1.36	1.41	1.42	0.05
MSE	3.69	4.09	3.85	4.07	3.96	3.66	4.19	3.91	3.82	3.85	3.90	0.16

Table 8. Mean preprocessing time and mean model runtime of the best NAS-RE model, where columns Avg and Std are the average and standard deviation.

	ONNX - PyTorch						OpenVINO IR - PyTorch						Avg	Std
	Warm Start			Cold Start			Warm Start			Cold Start				
Preprocessing (µs)	10.73	9.94	9.16	25.21	20.33	10.80	9.69	10.41	9.66	9.87	9.34	20.89	13.00	5.41
Model (ms)	2.27	2.30	2.33	2.52	2.76	2.41	2.28	2.29	2.69	2.54	2.60	2.36	2.44	0.16

Table 9. Mean preprocessing time and mean model runtime of the best NAS-RS model, where columns Avg and Std are the average and standard deviation.

	ONNX - PyTorch						OpenVINO IR - PyTorch						Avg	Std
	Warm Start			Cold Start			Warm Start			Cold Start				
Preprocessing (µs)	9.43	16.26	7.39	15.73	13.13	8.74	6.73	8.45	9.98	9.59	12.46	9.60	10.62	2.96
Model (ms)	1.72	1.76	1.71	1.73	1.74	1.91	1.74	1.81	1.63	1.78	1.75	1.68	1.74	0.06

5 Conclusion

We have introduced new accurate lightweight calibration methods for improving accuracy of measurements from mobile low-cost particulate matter sensors. The new methods utilize Neural Architecture Search (NAS) to improve the accuracy and efficiency of calibration models compared with the state-of-the-art methods. The experimental results show that the new calibration methods achieves the best root mean square error (RMSE) of 1.89, reducing accuracy error by more than 26% compared with the state-of-the-art methods. The new methods take less than $2.5ms$ to correct an inaccurate measurement from a mobile low-cost PM sensors on the HAPADS air pollution monitoring platform, making the methods suitable for real-time calibration. The new accurate lightweight methods would contribute to resolving the accuracy challenge of mobile low-cost PM sensors, enabling them to be deployed in monitoring PM continuously and accurately in many locations.

References

1. Hapads: Highly accurate and autonomous programmable platform for providing air pollution data services to drivers and the public. Accessed 21.03.2023.
2. J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems*, volume 24, 2011.
3. European Commission. Review of sensors for air quality monitoring, jrc technical report, 2019.

4. A. Gressent, L. Malherbe, A. Colette, H. Rollin, and R. Scimia. Data fusion for air quality mapping using low-cost sensor observations: Feasibility and added-value. *Environment International*, 143:105965, 2020.
5. Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, 2015.
6. Intel. Intel neural compute stick 2 (intel ncs2), Accessed Feb. 26, 2023.
7. Scikit learn developers. An adaboost regressor, Accessed Feb. 26, 2023.
8. Scikit learn developers. Histogram-based gradient boosting regression tree, Accessed Feb. 26, 2023.
9. J.-M. Lepioufle, L. Marsteen, and M. Johnsrud. Error prediction of air quality at monitoring stations using random forest in a total error framework. *Sensors*, 21(6), 2021.
10. H. Liu, K. Simonyan, and Y. Yang. Darts: Differentiable architecture search, 2018.
11. H.-Y. Liu, P. Schneider, R. Haugen, and M. Vogt. Performance assessment of a low-cost pm2.5 sensor for a near four-month period in oslo, norway. *Atmosphere*, 10(2), 2019.
12. Microsoft. Neural Network Intelligence, 2021.
13. M. Méndez, M.G. Merayo, and M Núñez. Machine learning algorithms to forecast air quality: a survey. *Artif Intell Rev*, 2023.
14. World Health Organization. Air pollution data portal, Accessed Feb. 25, 2023.
15. Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. Efficient neural architecture search via parameter sharing, 2018.
16. Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search, 2018.
17. Xie S and et al. Feasibility and acceptability of monitoring personal air pollution exposure with sensors for asthma self-management. *Asthma Res Pract.*, 7(1):13, 2021.
18. Sensirion. Sps30 particulate matter (pm) sensor, Accessed Feb. 26, 2023.
19. Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2014.
20. M. Wojcikowski, B. Pankiewicz, A. Bekasiewicz, T.-V. Cao, J.-M. Lepioufle, I. Vallejo, R. Odegard, and H. P. Ha. A surrogate-assisted measurement correction method for accurate and low-cost monitoring of particulate matter pollutants. *Measurement*, 200:111601, 2022.
21. Liyao Yang, Cheng Li, and Xiaoxiao Tang. The impact of pm2.5 on the host defense of respiratory system. *Frontiers in Cell and Developmental Biology*, 8, 2020.
22. J. Yun and J. Woo. Iot-enabled particulate matter monitoring and forecasting method based on cluster analysis. *IEEE Internet of Things*, 8(9):7380–7393, 2021.
23. Q. Zhang, Z. Han, F. Yang, Y. Zhang, Z. Liu, M. Yang, and L. Zhou. Retiarii: A deep learning exploratory-training framework. In *OSDI*, 2020.
24. Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning, 2016.