Toxic Gas Identification - Bertin Technologies

Killian Steunou
killian.steunou@ens-paris-saclay.fr
March 18, 2025

Abstract

This paper addresses the challenge of toxic gas identification and characterization using sensor data significantly influenced by varying humidity levels, which present a notable distribution shift between training and test datasets. To bridge this domain gap, we explore and benchmark several modeling strategies, including strategic data partitioning that emulates test-set humidity distributions, standard machine learning models (Random Forest and XGBoost), a two-stage classification-regression approach, and an end-to-end deep learning architecture called RAMTNet, specifically designed for multi-task regression. Additionally, we implement Unsupervised Domain Adaptation through adversarial training, encouraging domain-invariant feature representations. Our experiments reveal that careful simulation of test conditions and domain adaptation substantially mitigate overfitting caused by humidity-induced distribution shifts. The best-performing approach, a two-stage model combining classification and regression, achieves a weighted RMSE of 0.154256, surpassing the provided baseline (0.1567).

1 Introduction

This report aims to explore different methods to address the task of toxic gas characterization. The goal is to accurately identify and characterize various gases using data from 13 different sensors, including measurements significantly affected by humidity. A critical complication arises due to a notable distribution shift in humidity levels between the training and test datasets. We approach this issue by employing strategic data partitioning to mimic test conditions, alongside applying Unsupervised Domain Adaptation (UDA) in end-to-end deep learning models, inspired by [1], to learn robust domain-invariant feature representations.
We use different modeling choices through hyperparameters tuning for Random Forest Regressor and XGBoost, two-stage approaches and end-to-end deep learning models, that will be detailed in the report.
The effectiveness of these strategies is evaluated using the competition-specific error metric, with the ultimate aim of surpassing the provided baseline performance.

2 Objective and Data Presentation

2.1 Overview of the Data

The labeled dataset is made of 202,933 samples, with 13 features:

- Humidity: arbitrary values between 0 and 1
- Sensor group M4, M5, M6, M7: a first group of sensor located physically close in space;
- Sensor group M12, M13, M14, M15: a second group of sensor located physically close in space;
- Additional sensors R, S1, S2, S3.

The target includes 23 categories (c01, ..., c23) and a corresponding alarm value (ranging between 0 and 1). Thus, the goal is to identify active categories and predict their respective alarm levels. For

example, if the alarm level for a sample is 0.14 and the active categories are c03, c04, and c22, then its label would be a vector $c = \{0, 0, \underbrace{0.14}_{\text{3rd position}}, \underbrace{0.14}_{\text{4th position}}, 0, \ldots, 0, \underbrace{0.14}_{\text{22nd position}}, 0\}$.

2.2 Evaluation Metric

For each sample, the error is computed as:

Error = $\frac{1}{23} \sum_{i=1}^{23} f_i * (c_i - \hat{c}_i)^2$

where:

- $\hat{c}_i$ is the predicted alarm level for gas category i.
- f_i is a weighting factor that increases the penalty when an alarm should have been triggered:
  - $f_i = 1$ if the true value $\hat{c}_i < 0.5$.
  - $f_i = 1.2$ if $\hat{c}_i >= 0.5$

For a whole set of predictions, the metric is the mean of this error, on which a square root is applied.

2.3 Feature Analysis

2.3.1 Feature Correlation

Figure 1 shows the feature correlations in the train set. We can clearly see sensors M4 to M7 being highly correlated between them, as well as sensors M12 to M15, and the additional sensors R, S1, S2 and S3 show some positive correlation between them. There is also some inter-groups correlation, S1 being positively correlated with all sensors in the M12-M15 group, and negatively correlated with the sensors in the M4-M7 group. There is a negative correlation between sensors of the M4-M7 group and sensors of the M12-M15 group.
These inter-group and intra-group relationships should be taken into account when designing a model, or creating new interaction features. Interestingly, humidity shows no correlation with any feature, suggesting its impact on the sensors' measures might be either indirect or non-linear (but present since it was stated in the data presentation of the challenge that "water has an impact on the measures").

The features in the test set show similar linear correlations as the train features.

2.3.2 Feature Distribution

Another important aspect that is key to this challenge is the data distribution, especially the difference in humidity distribution in the train set and in the test set, as shown in Figure 2.

As we can see, most ($\approx$50%) of the humidity values in the train set are very low, almost zero (< 0.0004). **74.12%** of the samples in the train set show a humidity level **below** 0.2, while **79.66%** of the samples in the test set show a humidity level **above** 0.2. Overall the test set humidity is much more uniformly distributed between 0 and 1.
This indicates a clear distribution shift for humidity between train and test sets. Other sensor distributions are very similar between the train and test set.

Most models achieve a very low validation error in the range of 0.03, training on 50% of the labeled data, without rescaling. But when evaluated on the test data, the error is much higher, around 0.16, suggesting a strong overfitting on the labeled data. We will explore strategies to reduce this validation-test gap in Section 4.

3 Related work

The majority of the recent literature on gas classification focuses on deep learning methods, achieving great results using various sensor features.
[2] introduces the use of very deep convolutional neural network for gas classification, using various sensor measurements, where one sensor is not associated with a single measure, which is quite

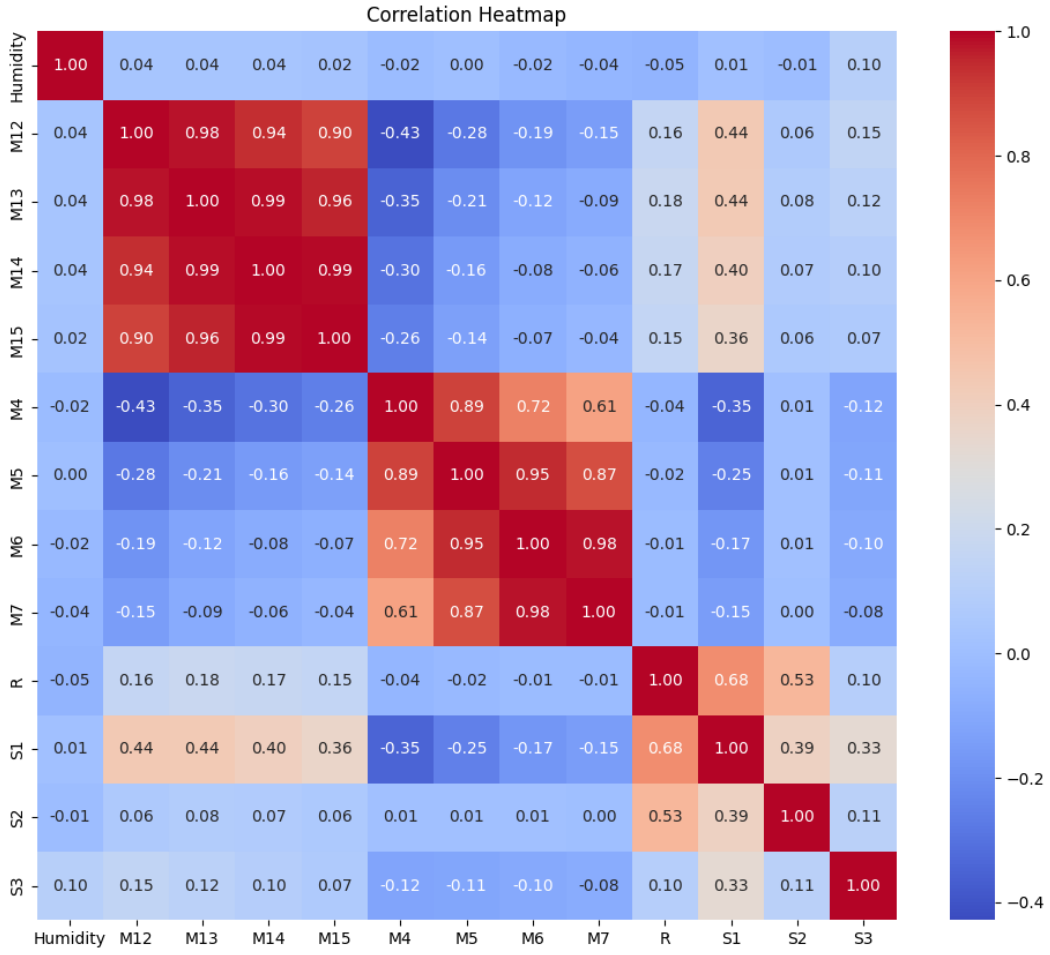


Figure 1: Heatmap showing pairwise Pearson correlations among sensor measurements in the training dataset, highlighting strong intra-group correlations and significant negative inter-group correlations.

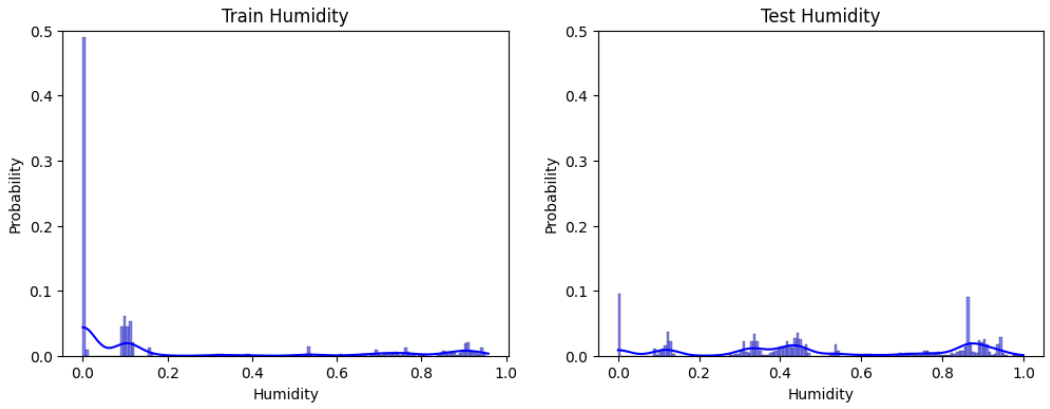


Figure 2: Comparison of humidity distributions between training and test datasets, illustrating a pronounced shift with significantly lower humidity levels in the training set compared to the more uniformly distributed test set.

different from our data.
To tackle the sensor drift issue, that can be seen as a similar issue as our humidity distribution shift,

[1] employs Unsupervised Domain Adaptation through domain adversarial neural networks.
Finally, [3] provides interesting insights on the link between complexity of neural networks and overfitting.

4 Feature Engineering and Modeling

4.1 Data preprocessing

Most measures of sensor groups M4-M7 and M12-M15 are highly concentrated around zero, but with some light, yet long, skewness (M4-M7 are left-skewed, M12-M15 are right-skewed). This long-tailed pattern appears as well in the train set as in the test set, so we decided to keep these outliers in the training data.
Traditional machine learning models (Random Forest, linear regression, k-nearest neighbors, ...) are not very sensitive to the scale of the input data. Deep learning models, on the opposite, can be sensitive to high input values, especially if they did not see such inputs very often during training. In practice, we found little difference in performance of our deep learning model when rescaling the data or not, thanks to the use individual feature embeddings. The architecture used is detailed in Section 4.5.

Given the limited number of daily submissions permitted for test evaluation, extensive experimentation was impractical, necessitating the creation of a challenging validation set that accurately mimics the test distribution. We test three strategies:

- **Strategy 0.** Simple random split, keeping 90% of labeled data for training, 10% for validation.
- **Strategy 1.** Since the test set has much higher humidity values than the train set, we split the data in the following way:
  1. First, take all labeled data with humidity < 0.2, and assign it to a train set.
  2. Then, add to this train set 25% of the data with humidity ≥ 0.2
  3. Finally, use the remaining 75% of the data with humidity ≥ 0.2 as the validation set.

  That way, we reproduce the shift that we have seen between train and test sets, making it even a bit larger. The validation errors with this validation set now correspond more to the test errors, and we might be able to assess if a model overfits or not.
- **Strategy 2.** As we can see in Figure 2, there is in fact 10% of the test samples with near-zero humidity. This second strategy consists of randomly removing training samples with near-zero humidity, until they represent 10% of the dataset. The resulting distribution is shown in Figure 3. The major drawback with this strategy is that we remove 44.77% of the labeled data, reducing from 202,933 to 112,077 samples. The benefit is that now the labeled samples are more uniformly distributed, which can especially be important for deep learning models.

4.2 Feature Engineering

A very common way to include interactions between two variables (for examples a and b) is to create a new variable (c) computed from the two others (such as $c = a \times b$, or $c = \frac{a}{b}$). We create such interaction variables between humidity and each sensor (product **and** ratio), along with two other variables: humidity^2 and $\sqrt{\text{humidity}}$, to include non linear relationships. Using this feature engineering strategy yields 39 variables (13 original ones + 2 out of humidity + 2*12 humidity interaction variables). While this may be a good idea to explain the relationships between variables, it seems to worsen generalization, showing a high error when evaluating a model trained on these 39 features on the test set. We find that modeling choices are more impactful than engineered features; henceforth, raw sensor measures are used directly as model inputs (models trained on rescaled measures show a higher test error).

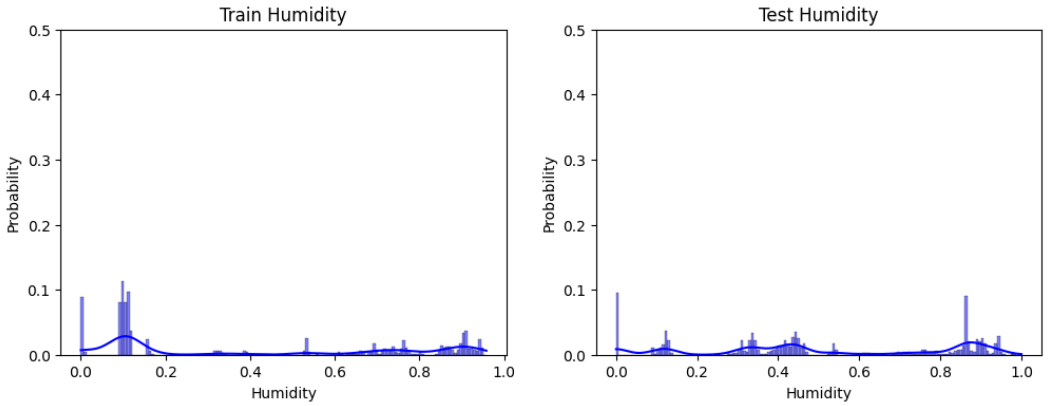


Figure 3: Adjusted humidity distributions for training and test datasets after strategically reducing near-zero humidity samples from the training set to better align it with the test dataset distribution.

4.3 Standard ML Models

Now that we have created reasonably trustworthy training and validation sets, we can experiment with different models, starting with the "classical" machine learning models: Random Forest and XGBoost.
We train both models as multi-output regressors, running a hyperparameters search for each (comparing performance using strategies 0, 1, and 2). This straightforward approach gives good results, as presented in Table 4, the random forest model beats the baseline while XGBoost ends up just above. The best parameters found are described in Table 1.

Hyperparameter	Value
RandomForestRegressor	
`n_estimator`	100
`max_depth`	None
`min_samples_split`	0.005
`min_samples_leaf`	20
XGBoost	
`n_estimator`	100
`max_depth`	7
`learning_rate`	0.1
`subsample`	0.8
`colsample_bytree`	0.8

Table 1: Optimal hyperparameters identified through hyperparameter tuning for Random Forest Regressor and XGBoost models. Parameters not specified are default from Scikit-Learn [4].

4.3.1 Multi Linear Regression with Feature Selection

Another simple approach but less effective is to fit 23 linear models to the train set, one for each category. This "brute-force" model is best used with our 39 engineered features. The training procedure is the following for each category:

1. Fit an OLS model on all features.
2. Remove features that are non-significant (p-value < 0.05).
3. Fit an OLS model on significant features.

This model did not generalize well at all, but motivated the following two-stage approach.

4.4 Two-stage Approach

This approach is more aligned with the objective of this challenge, which has been presented in two parts: classify a gas and predict its alarm level. To this end, we implement a two-stage model that is made of two models trained for different tasks:

- A multi-label classifier whose objective is to predict which categories are active for a given input
- A regressor whose objective is to predict the alarm level for a given input.

The final output is then crafted by placing the predicted alarm level at the predicted categories positions, setting the others to 0. A Random Forest classifier achieves more than 99% accuracy on the validation set, which is important as misclassifications cost a lot in this setting. A significant advantage of this two-stage approach is its ability to produce exact zeros in non-active categories, reducing the prediction error compared to other models that output small non-zero predictions for inactive categories. For the regressor model, we chose a Random Forest. Both classifier and regressor have the following parameters `n_estimators`: 100, `max_depth`: 15, `min_samples_split`: 0.005, `min_samples_leaf`: 30, and this two-stage model achieves a test error of 0.155773, surpassing the baseline.

4.5 End-to-end Deep Learning for Multi-task Regression

As suggested by the literature, deep learning models seem to work well for gas classification, but little work has bee done on regression, even less on multi-regression.
In this part we present an end-to-end model that predicts directly the target vector from the input sensor measures. This model is made of an *encoder*, whose role is to create a meaningful embedding (a vector) from the different sensors, and then apply a multi-regressor *head* that predicts the values of each category. The model is called Residual Attention Multi-Task Network (RAMTNet), its architecture is summarized in Figure 4 and detailed below.

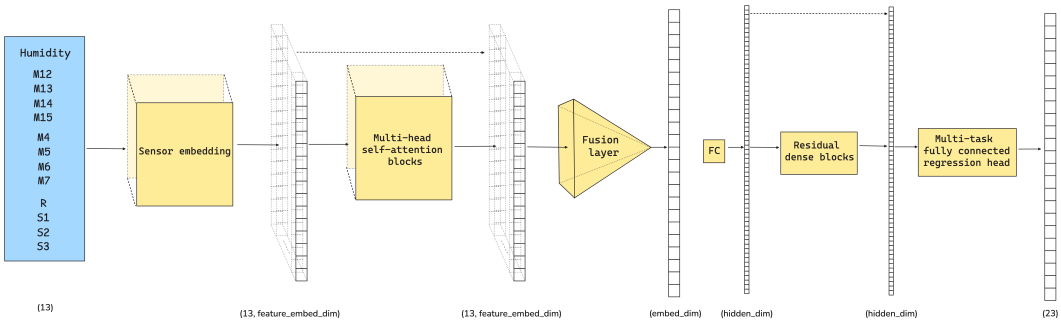


Figure 4: Overview of the proposed Residual Attention Multi-Task Network (RAMTNet) architecture, illustrating sensor-specific embeddings, residual multi-head self-attention layers, final embedding fusion, and the head used for multi-task regression.

4.5.1 RAMTNet

RAMTNet is made of two components: an encoder which takes sensor measures as input and produces an embedding, followed by a head made of residual dense blocks and 23 fully connected networks to predict the outputs.

Encoder. The encoder of RAMTNet is made of a first component that produces one vector (referred to as a *feature embedding*) per sensor measure, followed by several multi-head self-attention (with residual connections) blocks, and finally a linear layer that fuses the 13 embeddings into a single higher-dimensional one.

- **Feature Embedding**: each input (sensor measure, 1-dimensional) is processed by a linear layer that maps it to a k-dimensional vector. This results in a matrix $A \in \mathbb{R}^{13\times k}$ containing all feature embeddings.

- **Residual multi-head self-attention blocks**: the matrix A is then processed sequentially through several multi-head self-attention blocks, denoted RMHA : $\mathbb{R}^{13\times k} \to \mathbb{R}^{13\times k}$. This makes the encoder learn complex relationships between sensors. A residual connection is then used, resulting in a matrix $B = A + \text{RMHA}(A) \in \mathbb{R}^{13\times k}$.
- **Fusion Layer**: Finally, the matrix B is flattened into a vector in $\mathbb{R}^{13^* k}$, and mapped with a linear layer and ReLU activation function to a m-dimensional embedding vector $E \in \mathbb{R}^m$.

Head. The head of RAMTNet is made of the following components:

- **Residual dense blocks**: a first linear layer with ReLU activation is used to map the embedding $E \in \mathbf{R}^m$ to a vector $C \in \mathbb{R}^l$. This vector is then processed sequentially through several residual blocks, which are each a Multi-Layer Perceptron (MLP) with two hidden layers, ReLU activations and batch normalization, such as RB : $\mathbb{R}^l \to \mathbb{R}^l$, $\text{RB}(C) = C + \text{MLP}(C)$. This part outputs a vector $D \in \mathbb{R}^l$.
- **Multi-task regression head**: finally, D is processed by 23 different MLPs with two hidden layers, one ReLU activation and a Sigmoid output activation, each MLP predicting one category.

Dropout layers are also used to prevent overfitting. Exact parameters used for training can be found in Appendix A.

This architecture leverages a powerful yet efficient encoder capable of identifying complex patterns in the input data, thereby reducing the need for a computationally heavy regression head. This model achieves the lowest error on all validation sets, but unfortunately is does not generalize well and perform poorly on the test set. This is why we introduce the following method.

4.6 Adversarial Unsupervised Domain Adaptation

We adapted the methodology from [1] to train RAMTNet with adversarial unsupervised domain adaptation (UDA). The method introduces an adversarial setting with a domain classifier whose objective is to classify source embeddings and target embeddings apart, while the encoder tries to fool it by bringing embeddings from both domains close. It is decomposed into two parts:

1. Train the encoder on the source domain (labeled data)
2. Introduce the domain classifier and train it along with the pretrained encoder with a modified loss.

To apply adversarial UDA to RAMTNet, we introduce a domain classifier D alongside the encoder G and the multi-task head H. Let us denote the source domain by $D^s = \{(x_i^s, y_i^s)\}$ and the unlabeled target domain by $D^t = \{x_j^t\}$. We first pretrain G and H on D^s only, minimizing the weighted RMSE $\mathcal{L}_{\text{task}}$ on each output. Then, in a second phase, we jointly train G, H, and D using a combined loss:

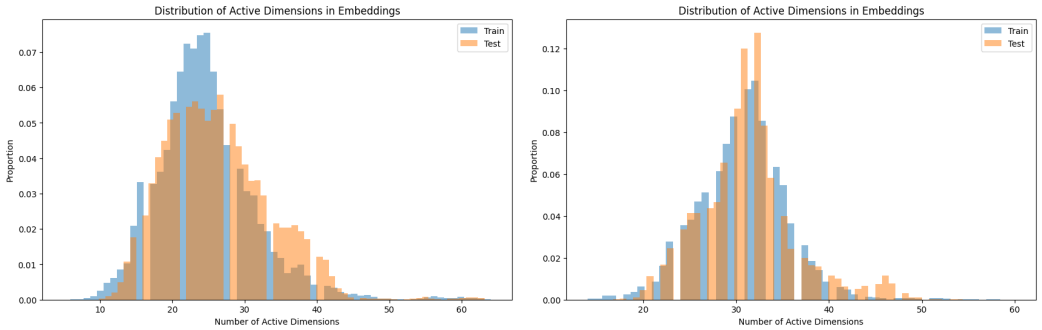
$$\min_{G,H} \max_{D} \big[\mathcal{L}_{\text{task}}(H(G(x^s)), y^s) \; - \; \lambda\, \mathcal{L}_{\text{domain}}(D(G(x^s)), D(G(x^t)))\big].$$

Here, $\mathcal{L}_{\text{domain}}$ is a cross-entropy term that measures how accurately the domain classifier D distinguishes source embeddings (with domain label 0) from target embeddings (domain label 1). During forward passes, the encoder G produces embeddings E, which are fed both to H (for predicting gas values) and to D (for predicting domain). By training D to classify source vs. target correctly, while G is penalized if D succeeds, G learns to align the distributions of the two domains in the embedding space. The scalar $\lambda \in [0, 1]$ balances the main regression objective with the adversarial objective.

In practice, we implement this adversarial interplay via a gradient reversal layer (GRL), which multiplies the gradient from D by $-\lambda$ when updating G. If λ is annealed from 0 to 1 across training epochs ($\lambda_p = \frac{2}{1+e^{-\gamma\, p}} - 1$ with $\gamma = 5$ and $0 \leq p \leq 1$ depending on the epoch), then early training focuses on perfecting $\mathcal{L}_{\text{task}}$, and later training focuses on minimizing the discrepancy between source and target embeddings. As a result, the final encoder G produces domain-invariant features that improve generalization on unlabeled target data. While [1] uses $\gamma = 10$, we reduce it to 5 for λ to

increase more slowly across epochs, which appeared to be more beneficial for our case. The different evolutions of λ_p depending on γ are shown in Appendix B, Figure 6.

After training, D is discarded and only G and H are used at inference. Figure 5 illustrates how the distribution of active embedding dimensions changes after including UDA: by enforcing alignment, the encoder activation patterns become more consistent between train (source) and test (target). By leveraging unlabeled target data through adversarial training, this method effectively reduces the distribution gap between source and target domains, thus enhancing generalization to previously unseen humidity conditions. Attention maps of the encoder (Apendix C, Figure 7) also witness a significant change with UDA, bringing strong diversification in feature attention compared to the encoder's attention maps before UDA.



(a) RAMTNet (b) RAMTNet + UDA

Figure 5: Comparison of embedding activation distributions between training and test sets before (left) and after (right) applying adversarial Unsupervised Domain Adaptation, demonstrating improved alignment of feature representations across domains.

We train RAMTNet on three different train sets from the three strategies, evaluating only on the validation set from strategy 1, assuming the model would struggle more with high-humidity samples. We compare validation errors on different sub-split of validation data, corresponding to different ranges of humidity. The results are presented in Tables 2 and 3.

Strat	UDA	Train loss	Val loss
0	No	0.07853	0.08856
0	Yes	0.14311	0.14825
1	No	0.07929	0.10536
1	Yes	0.15243	0.16183
2	No	0.09314	0.08674
2	Yes	0.12190	0.12254
All labeled data	No	0.07659	0.08771
All labeled data	Yes	0.13262	0.14575

Table 2: Training and validation errors for different train sets for RAMTNet with and without UDA.

Strat	UDA	[0.0, 0.68]	[0.68, 0.76]	[0.76, 0.87]	[0.87, 0.91]	[0.91, 1.0]
0	No	0.10382	0.09219	0.08124	0.09470	0.06614
0	Yes	0.14720	0.16015	0.14023	0.14340	0.14948
1	No	0.11381	0.13234	0.09600	0.09866	0.07806
1	Yes	0.16316	0.17435	0.16092	0.14730	0.16228
2	No	0.09805	0.08714	0.08226	0.09615	0.06633
2	Yes	0.11696	0.13122	0.12932	0.11241	0.12177
All labeled data	No	0.09821	0.09608	0.08045	0.09384	0.06566
All labeled data	Yes	0.14621	0.15285	0.16055	0.12698	0.13988

Table 3: Comparison of RAMTNet error on humidity-split validation sets with and without UDA.

We notice than the results are significantly better without UDA across all validation splits and training strategies, which is expected since we add a second objective to the loss of the network with UDA. Interstingly, the training/validation error gap is reduced with UDA, which is exactly what we aim to achieve with this method. Training on the whole labeled dataset yields more differences across validation splits with UDA than without.
The most important result of this benchmark is that strategy 2 is the one reducing most the training validation error gap, and this is even more visible for UDA. Strategy 2 thus seems to be a good training set to prevent overfitting.

5 Test Results

Evaluated on the test set, some models manage to beat the baseline by a few points. Interestingly, the simpler models (random forest) appear to be the best, and while RAMTNet alone could not beat the baseline, its UDA variant did. The issue with UDA is the adversarial nature of the procedure, which makes training unstable, requiring careful hyperparameters tuning: size of the encoder and head, dropout rates, base learning rate, number of warm-up epochs for the encoder, and size of the domain classifier. Full results per model are shown in Figure 4.

Method	Test Weighted RMSE
RandomForestRegressor	**0.154256**
Two-StageModel	0.155773
RAMTNet + UDA	0.156510
Baseline	**0.1567**
XGBoost	0.156914
RAMTNet	0.160069
Multi-Linear Regression[1]	0,694516

Table 4: Comparison of weighted RMSE performance on the test dataset across evaluated models. Hyperparameter details are provided in corresponding subsections.

6 Future work

Despite encouraging results achieved by our current methodologies, several avenues remain open for future exploration to further enhance model performance and robustness.

Hyperparameter Tuning. RAMTNet is by design inherently modular, and involves several hyperparameters: dimension of individual feature embeddings, number of self-attention blocks, number of heads in each self-attention block, dimension of the fused embedding, number of residual blocks, hidden dimension of the residual blocks, and hidden dimension of the multi-task head. All these have an impact on the number of learnable parameters of the model, especially the head hyperparameters. For example, with the architecture detailed in A, the encoder has 29,408 learnable parameters and the head has 274,967. While this does not represent an unusual amount of parameters for a deep learning model, the size of the head may cause overfitting or be a bottleneck for practical use where low latency is needed, when deployed on edge device.

Improved Domain Adaptation Techniques. Implementing the full UDA method introduced by [1] could benefit our model. The full method involves using one encoder per sensor and make the individual sensor embeddings from source and target domains closer. As RAMTNet uses a (very simple) individual feature encoding step, this could easily be done.

Hybrid Modeling Approaches Combining traditional machine learning models such as Random Forests with deep learning architectures could leverage the strengths of both approaches. Specifically, deep models could capture complex sensor interactions and nonlinearities, while ensemble models could manage robust predictions and uncertainty estimation.

[1] Multi-linear regression shows notably poorer performance, likely due to the humidity-induced distribution shifts and non-linearity of the interactions.

Feature Representation Learning Exploring advanced representation learning techniques, such as variational autoencoders or contrastive learning, could further improve sensor embeddings by explicitly modeling latent interactions. These techniques might help to generate more meaningful embeddings, leading to increased robustness against distribution shifts.

Addressing these areas would significantly strengthen the reliability of toxic gas characterization models in dynamic environmental conditions.

7 Conclusion

In this report, we investigated various modeling strategies to tackle the challenging problem of toxic gas identification under significant humidity-induced distribution shifts. Through extensive experimentation with traditional machine learning models, two-stage classification-regression approaches, and a new deep learning architecture (RAMTNet) enhanced by unsupervised domain adaptation, we reduced the performance gap between training and test conditions. Our findings highlight the importance of realistic data partitioning and advanced domain adaptation techniques in building robust predictive models. These promising results set a solid foundation for further enhancements in model generalization to real-world scenarios.

References

[1] Lakmal Meegahapola, Hamza Hassoune, and Daniel Gatica-Perez. M3bat: Unsupervised domain adaptation for multimodal mobile sensing with multi-branch adversarial training. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 8(2):1–30, May 2024.

[2] Pai Peng, Xiaojin Zhao, Xiaofang Pan, and Wenbin Ye. Gas classification using deep convolutional neural networks. *Sensors*, 18(1), 2018.

[3] Jolanta Wawrzyniak. Methodology for quantifying volatile compounds in a liquid mixture using an algorithm combining b-splines and artificial neural networks to process responses of a thermally modulated metal-oxide semiconductor gas sensor. *Sensors*, 22(22), 2022.

[4] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[5] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019.

[6] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[7] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019.

Supplementary Materials

A Implementation Details

RAMTNet Architecture The RAMTNet model comprises two primary components: an encoder and a multi-task regression head.

Encoder: Each of the 13 sensor inputs is independently mapped into an 16-dimensional embedding vector using individual linear layers. These embeddings are processed through 2 residual multi-head self-attention blocks, each consisting of 4 attention heads and residual connections. The self-attention blocks are followed by a fusion linear layer that combines the flattened embeddings into a final 128-dimensional vector.

Head: The embedding vector from the encoder is fed into a regression head composed of a single residual dense block, consisting of two hidden layers with 128 units each, ReLU activations, batch normalization, and dropout layers. The multi-task regression head itself consists of 23 separate MLPs, each containing two hidden layers of 64 units and ReLU activations, followed by a sigmoid activation function.

Training Hyperparameters RAMTNet was trained using the optimizer AdamW [5] with a learning rate of 2.5×10^{-4}, a weight decay of 1×10^{-5} and trained for a total of 100 epochs. Dropout rates were set to 0.5 for the encoder and 0.5 for the head, respectively. Training was conducted with a batch size of 2048 samples, optimizing the model using the provided weighted RMSE loss function. A plateau learning rate schedule was used to divide the learning rate by 2 when the validation loss did not improve for 10 epochs.

Adversarial Unsupervised Domain Adaptation A gradient reversal layer is introduced between the encoder and a domain discriminator to enforce domain invariance in the encoder's output embeddings. The domain discriminator architecture has a fully connected neural network with one hidden layer of 32 units, followed by a ReLU activation. The output layer is followed by a sigmoid activation function. The adversarial training used the gradient reversal technique with λ increasing from 0 to 1 during training, progressively increasing the adversarial loss contribution over 60 UDA epochs, after pretraining the encoder alone for 100 epochs. The domain classifier is optimized with Adam [6], and a learning rate of 2.5×10^{-4}.

RAMTNet was developped using the PyTorch framework [7].

Reproducibility The code used to get the results is released at https://github.com/killian31/IdGas.

B Evolution of λ_p for different values of γ

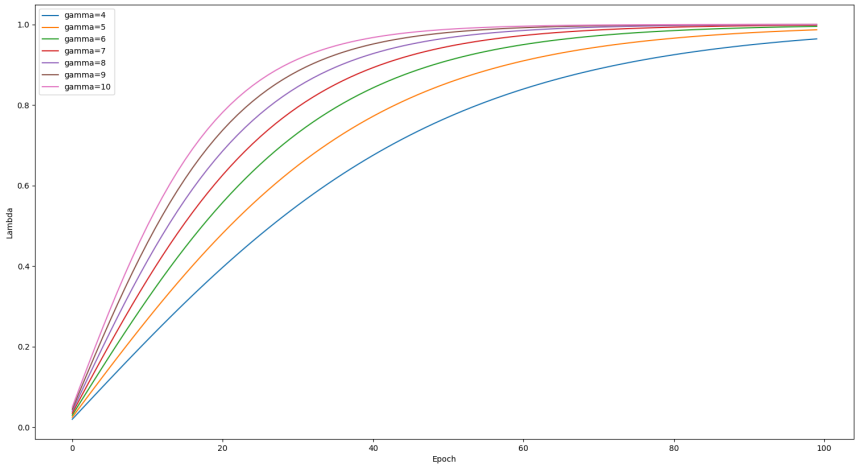


Figure 6: Evolution of λ across epochs for different γ.

C Visualizing Attention Maps of RAMTNet

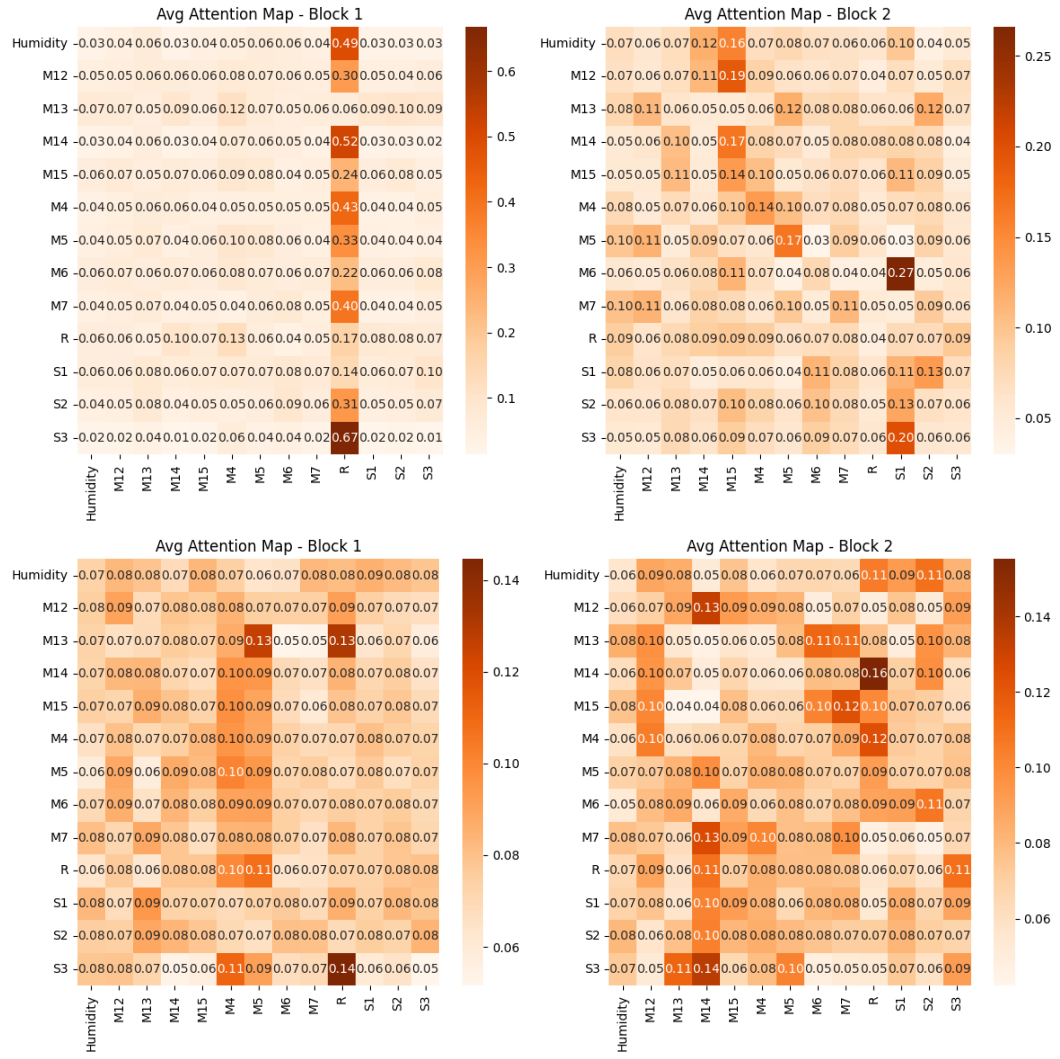


Figure 7: Comparison of average self-attention maps without (top) and with (bottom) adversarial unsupervised domain adaptation (UDA). Attention values represent the average interactions between input features (rows: queries, columns: keys) across the training set. The introduction of UDA results in a more balanced distribution of attention weights, reducing over-reliance on dominant features (e.g., *R* and *Humidity*), suggesting improved robustness and generalization capability.

After incorporating unsupervised domain adaptation, as shown in Figure 7, attention weights exhibit a more balanced distribution across input features, reducing the previous over-reliance on particular features such as *R*. The observed diversification in feature attention suggests that adversarial domain adaptation effectively promotes richer representations, potentially enhancing generalization and robustness in the presence of sensor drift or domain shifts.