Predicting Individual Physiological Responses to Pollution Using Transformer–Based Time-Series Models

Davide Baino

05.09.2025

To do

- Main Notebook to check with all functions included (do you understand what is happening step by step, can you relate one thing to another?)
    - Which part of the model makes it autoregressive of the 12 hours?

- Preso
- What is query, key and value?
- Test Cases

Agenda

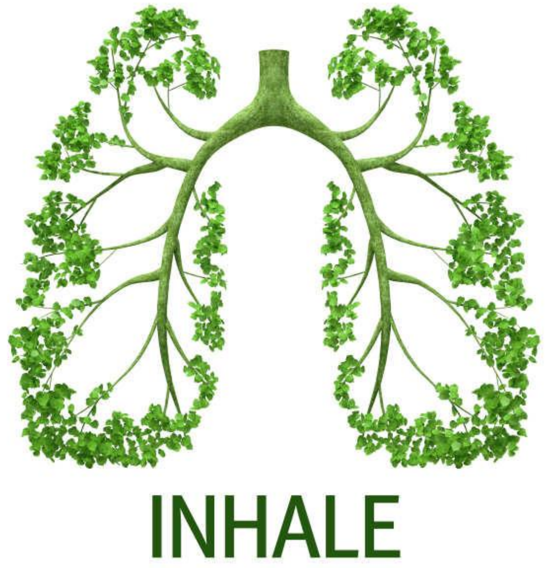
- Aim (1min)
- Dataset - 1min
- Model – 2mins
- Results – 3min
- Conclusions - 1min
- Q&A

Aim

- **Air pollution** contributes to an estimated seven million deaths annually
- Most of research does not show how predicted pollution affect individuals
- The aim is then to **predict individual health responses** to pollution exposure, facilitating early warnings and **personalised health recommendations.**

Why?

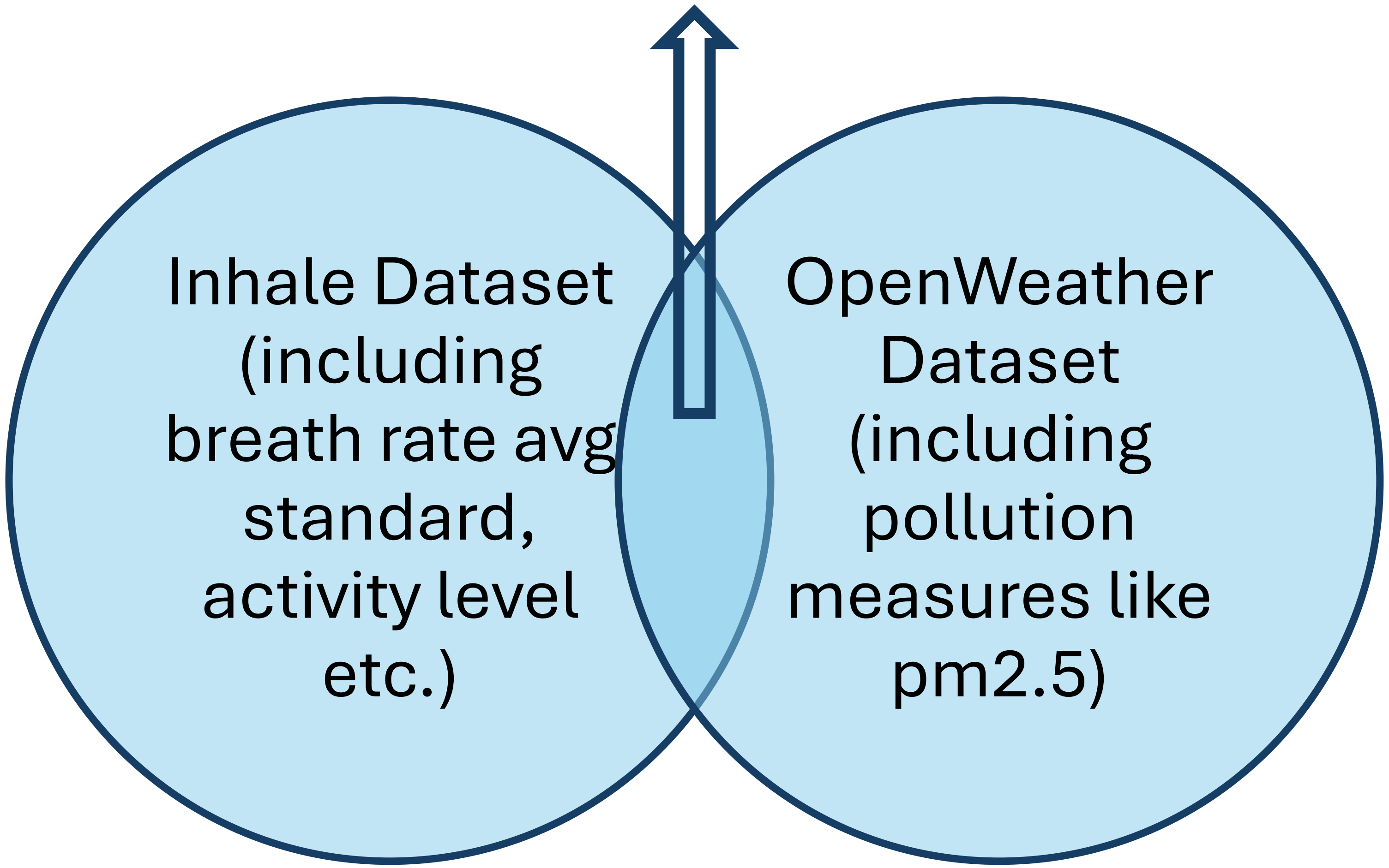
The Inhale Dataset



- Merged respiratory and personally experienced level of pollution on **patient id and timestamp.**
- Added personal levels of inhalation rate by multiplying tidal volume of air inhaled per patient to pollution levels in the air inhaled

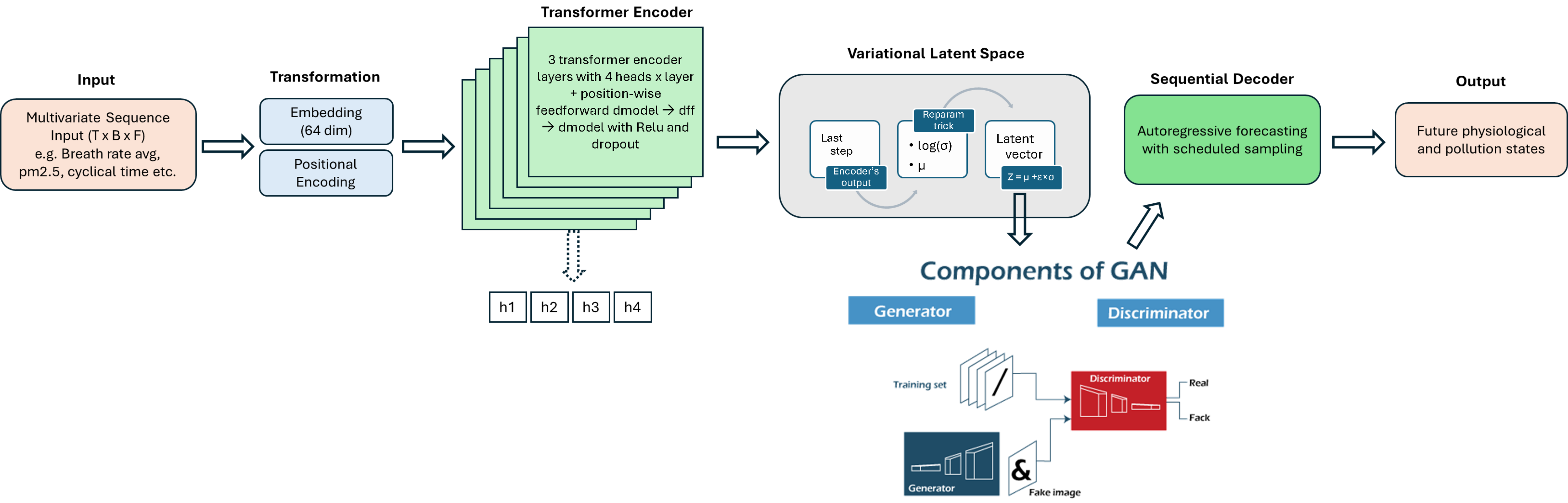
What?

Datasets


Merged on longitude, latitude, timestamp
Inhale Dataset (including breath rate avg standard, activity level etc.)
OpenWeather Dataset (including pollution measures like pm2.5)

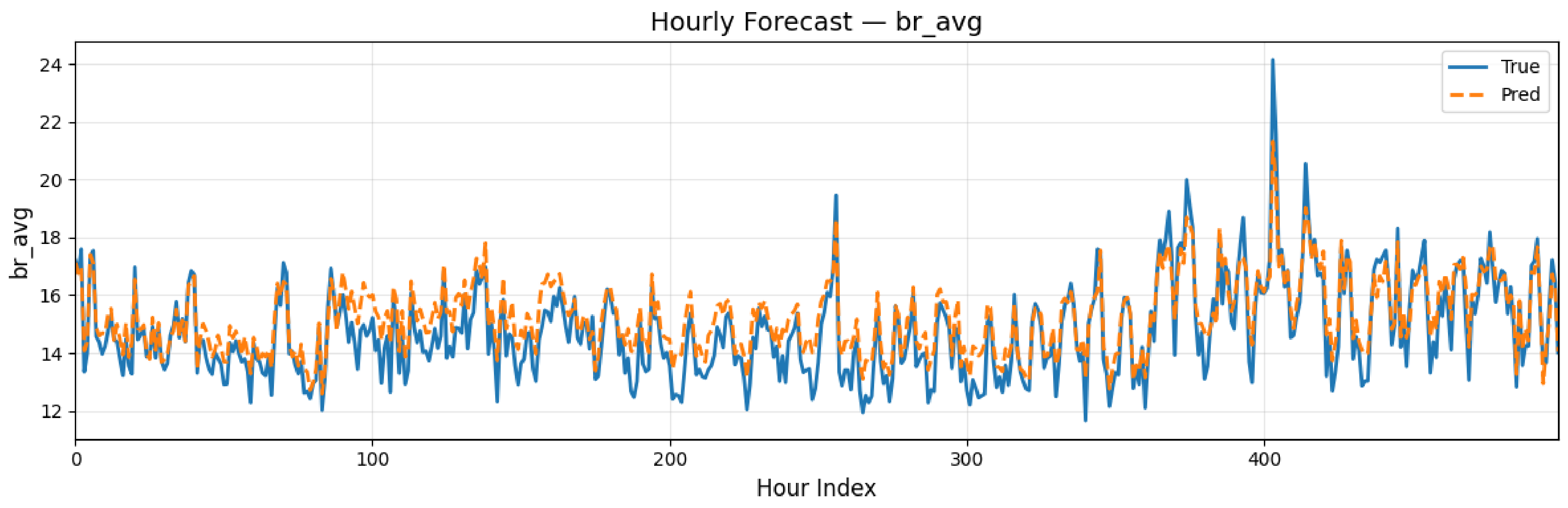
What?

Model

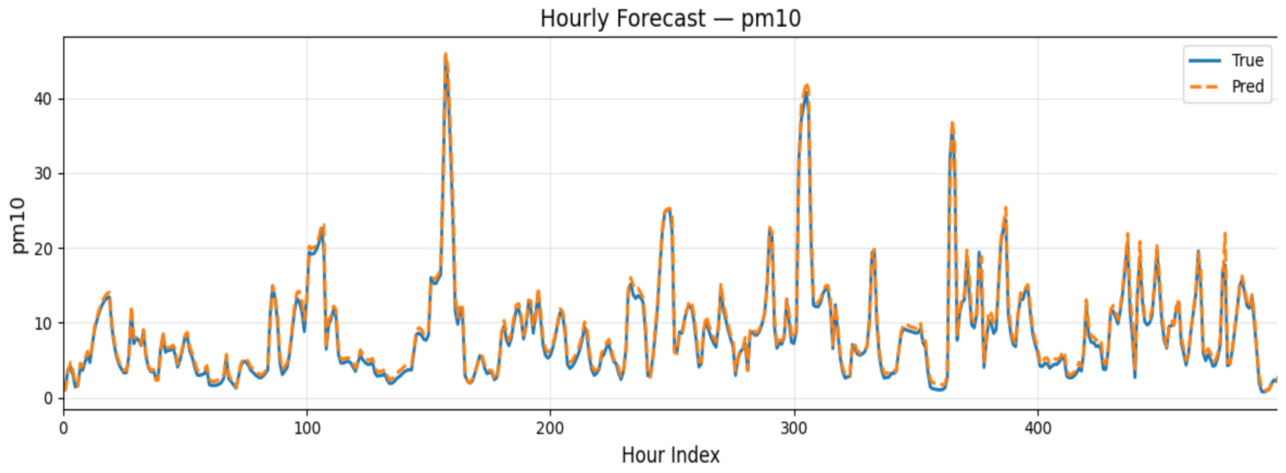

Input
Multivariate Sequence Input (T x B x F) e.g. Breath rate avg, pm2.5, cyclical time etc.
Transformation
Embedding (64 dim)
Positional Encoding
Transformer Encoder
3 transformer encoder layers with 4 heads x layer + position-wise feedforward dmodel → dff → dmodel with Relu and dropout
h1
h2
h3
h4
Variational Latent Space
Last step
Encoder's output
Reparam trick
• log(σ)
• μ
Latent vector
Z = μ +ε×σ
Sequential Decoder
Autoregressive forecasting with scheduled sampling
Output
Future physiological and pollution states
Components of GAN
Generator
Discriminator
Training set
Discriminator
Real
Fack
Generator
Fake image

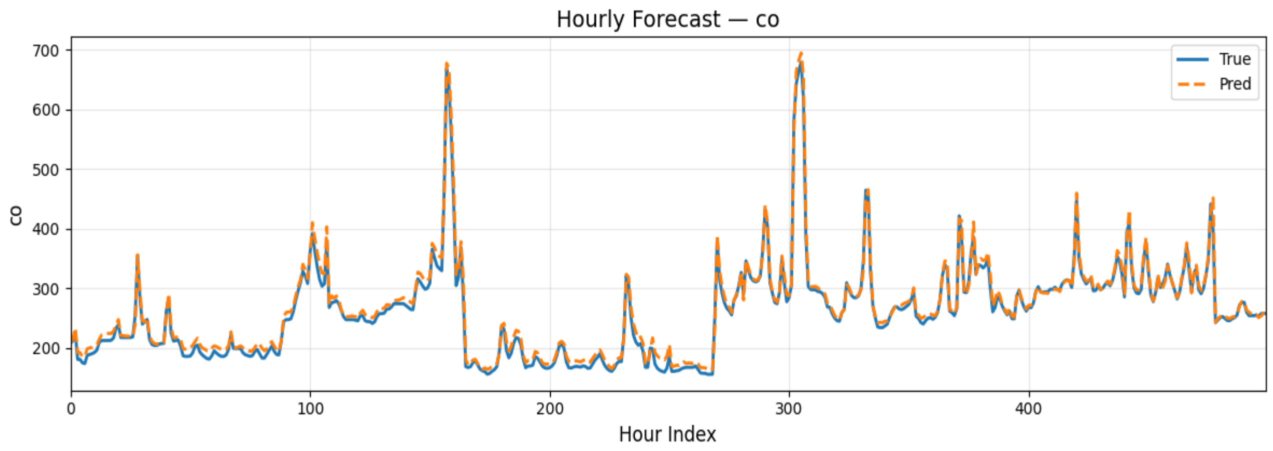
How?

Results – population wide model

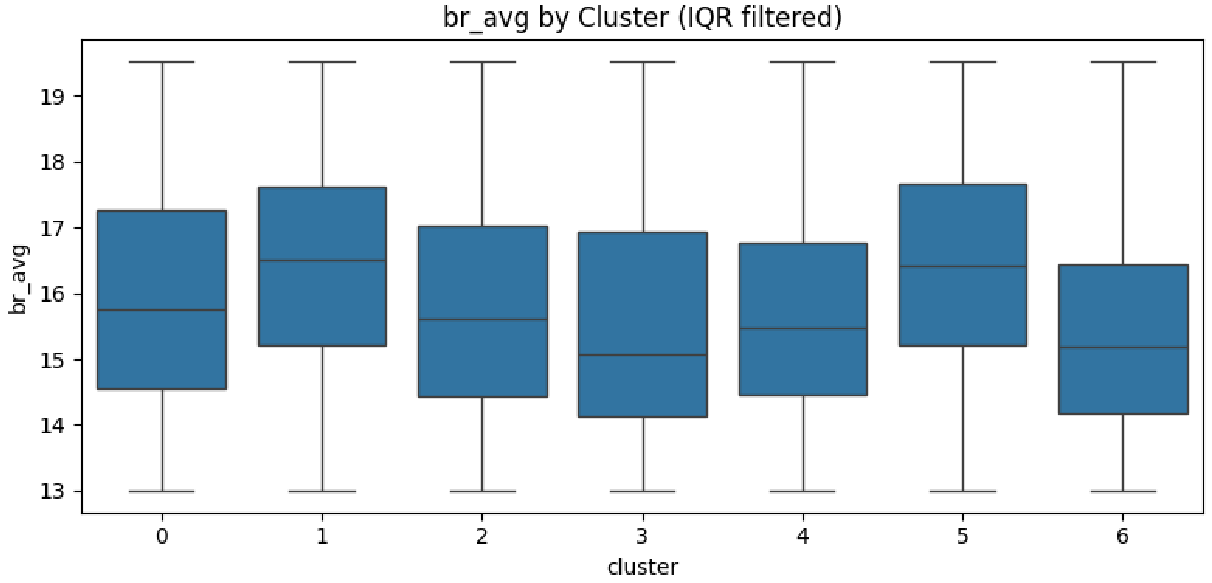


Predicting hourly physiological measures more challenging than pollution measures

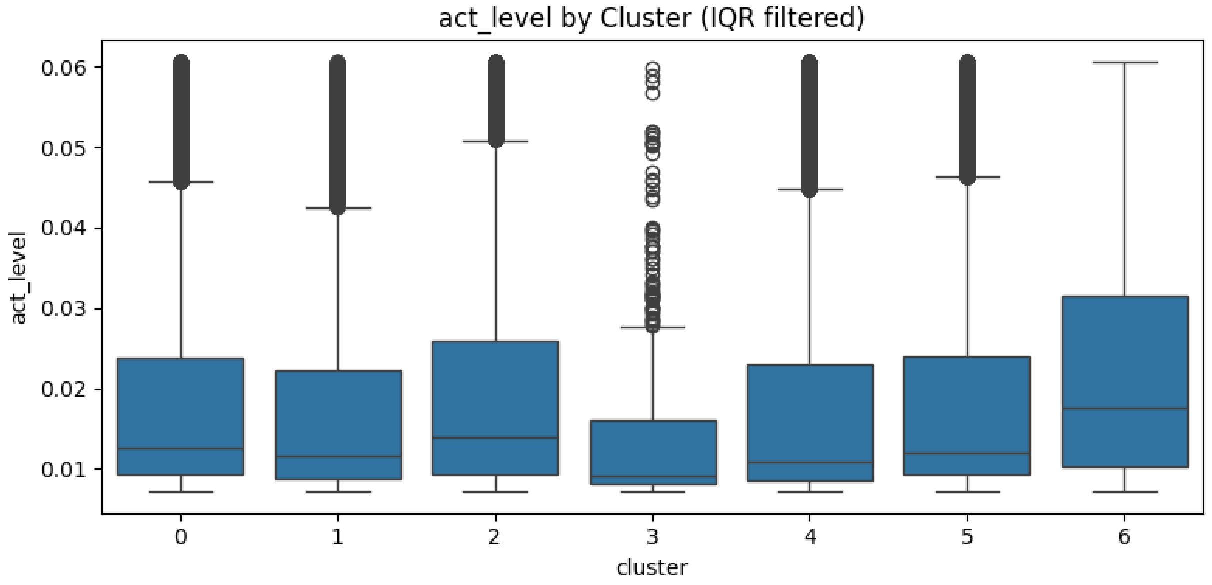


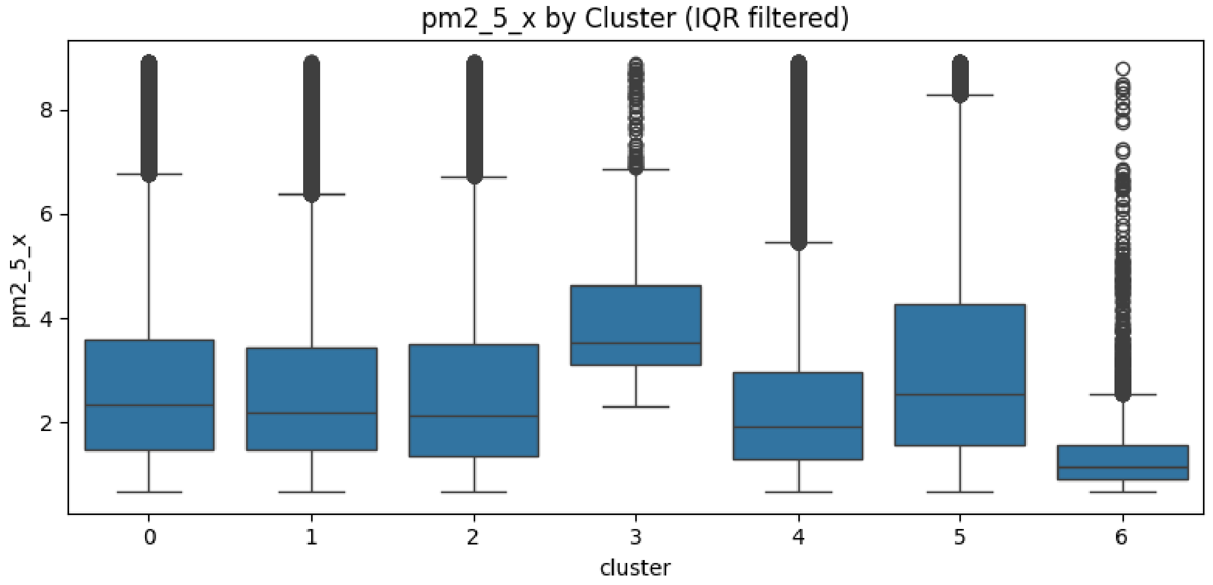


Reaction to pollution by clusters

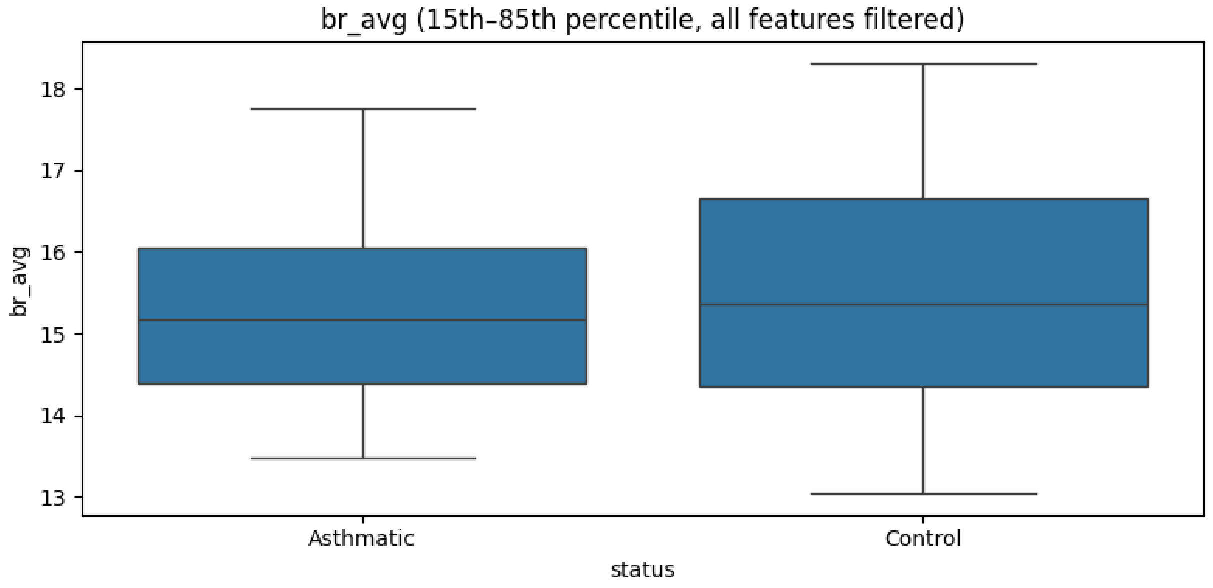


- ❑ Cluster 3 - highest levels of PM2.5 and PM10 levels and shows one of the largest increases in average breathing rate.
- ❑ Cluster 4 - lower PM2.5 and PM10, higher activity levels are associated with the strongest breathing-rate response.
- ❑ Cluster 0 - has modest PM2.5 but elevated O3 and SO2 relative to other clusters, consistent with vehicles and indoor pollution influences.

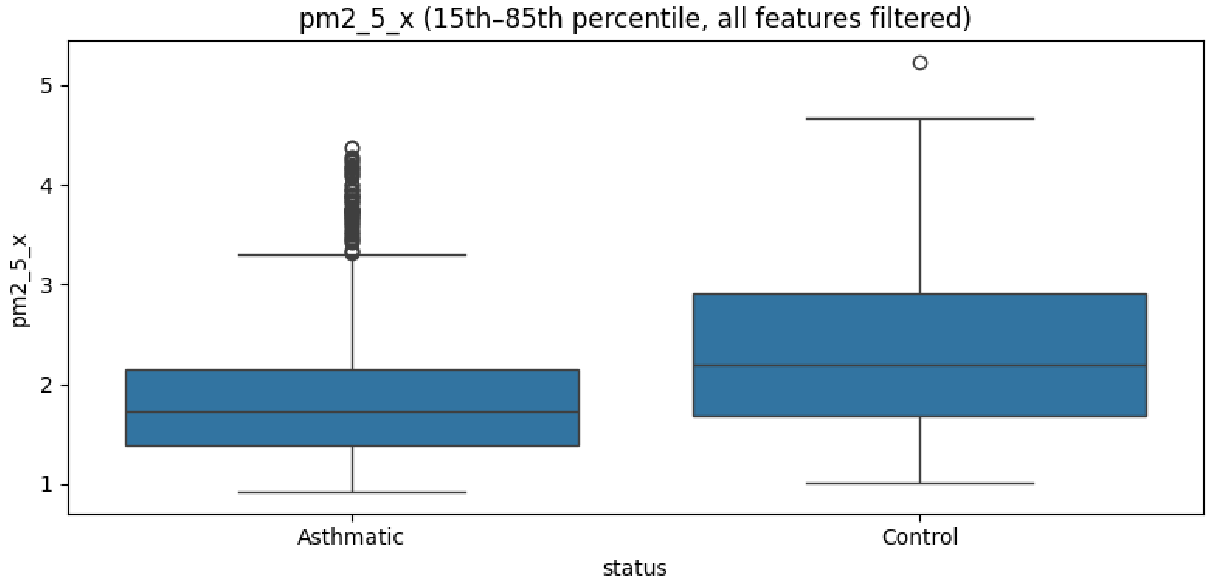


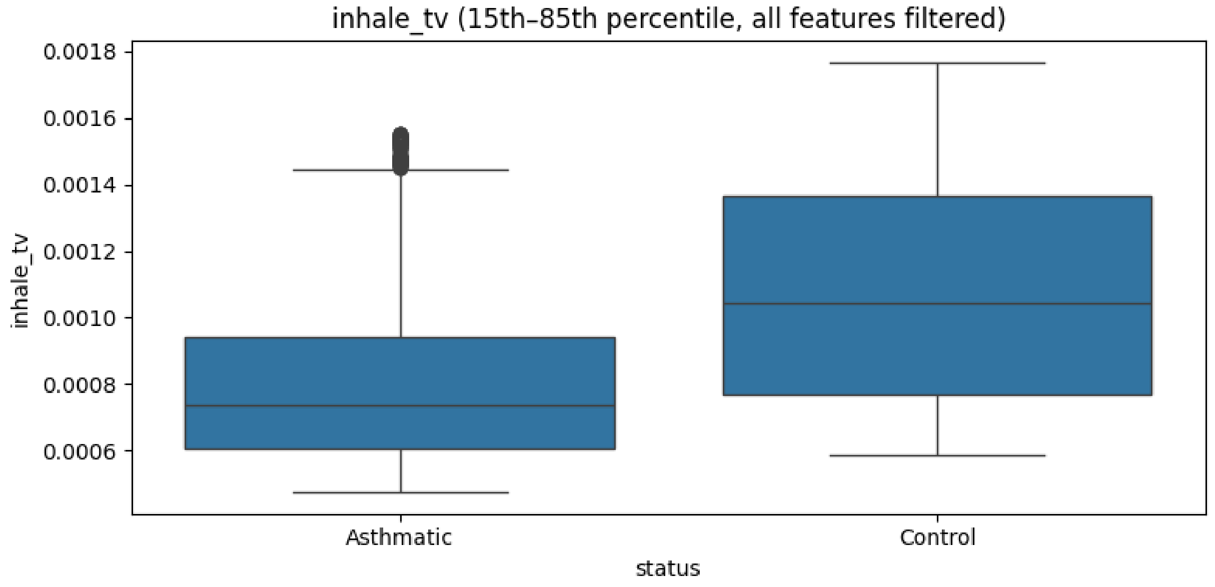


Reaction to pollution by health group

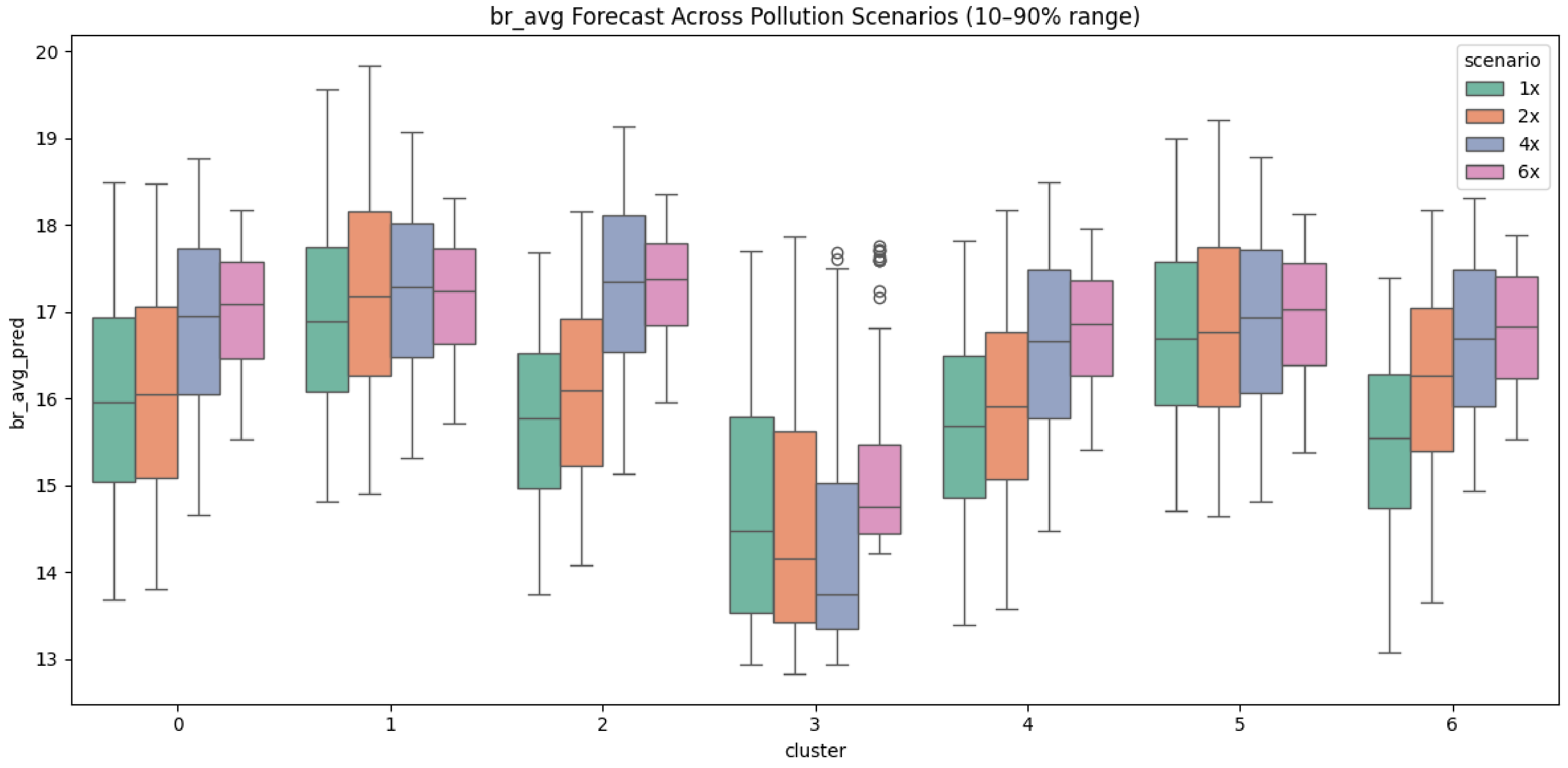


- ❑ Control (healthy group) is reacting more to pollution than asthmatic
- ❑ Control has higher level of pm2.5, pm10 . Greater inhalation and they have greater level of activity
- ❑ Control inhale more polluted air than healthy



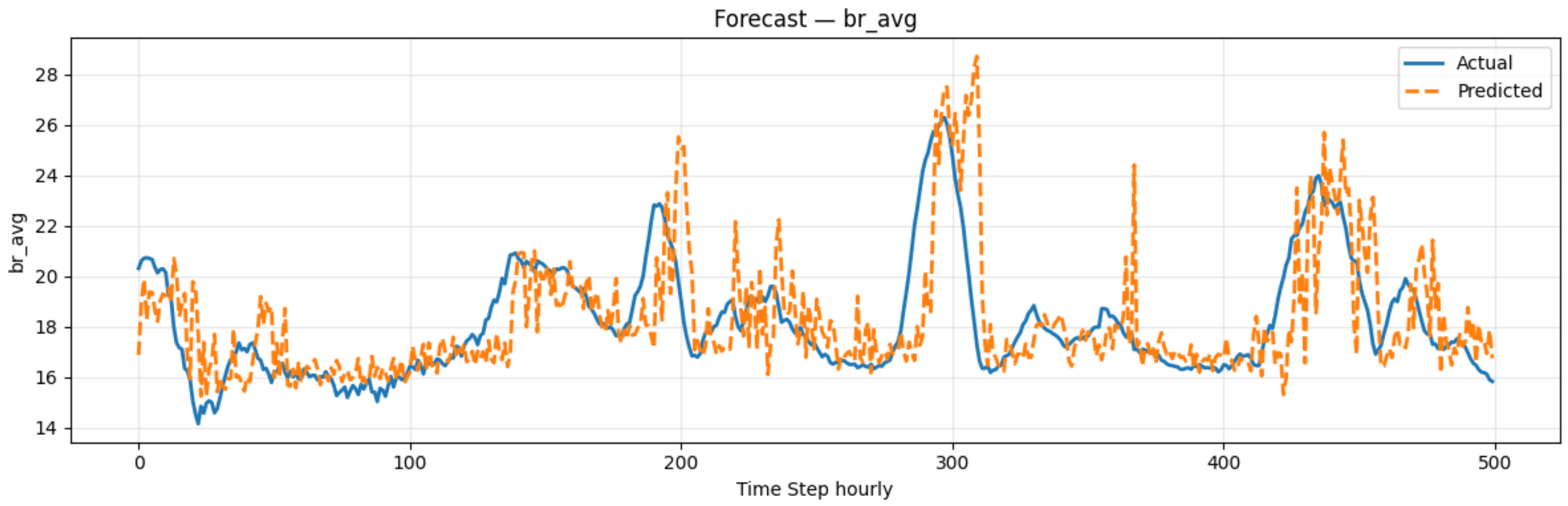


Perturbation multiplied by 1, 2, 4, 6 times of pollution measures



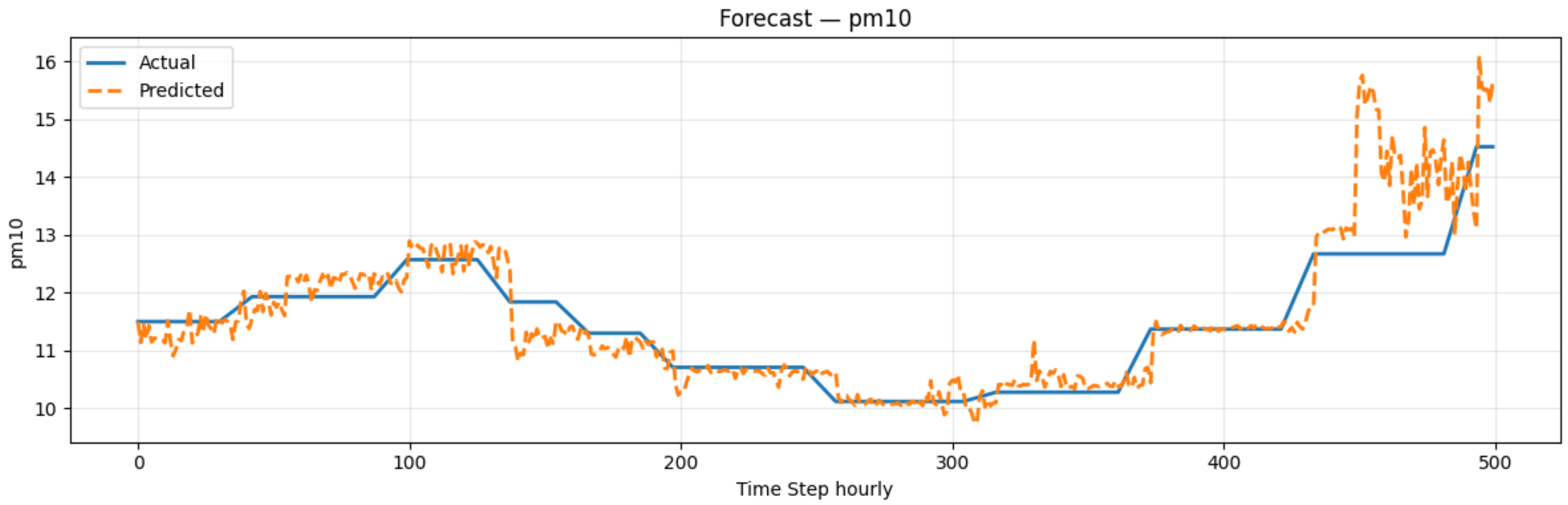
- ❑ Greater levels of pollution suggest higher volatility within breathing rate, meaning more unstable breathing.
- ❑ Clusters 0, 2, 4 and 6 seem to show the strongest reactivity, with breath rate rising by up to 10%.
- ❑ Clusters 1 and 5 display only mild changes in average breath rate but reflect more erratic standard deviations, indicating spikier breathing patterns.

Fine tuned model can predict next hours for unseen data

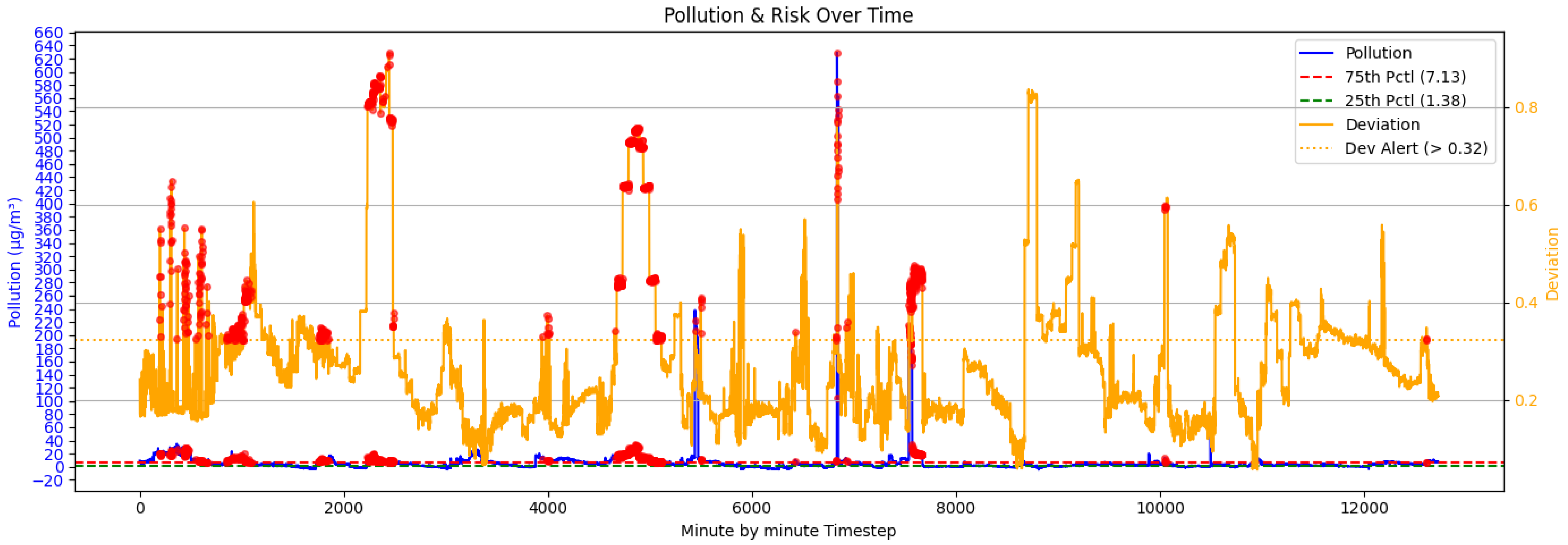


- ❑ Model successfully predicted next hour breath rate average so it learns from population and apply to unseen data

- ❑ Fine-tuned model is also able to predict next few hours of pollution level, so it understands the relationship between the twp



Alert system



There is a personalised baseline of expected physiology for individuals and deviations from it at every new timestep

Pollution exposure and physiological deviation are assessed against individualised thresholds. High risk is greater than 75%

An alert is triggered only when both thresholds are exceeded, so that we can infer that pollution and physiological are driven by one another

Conclusions

- ❑ A Transformer VAE-GAN was trained on cleaned data to forecast future physiological and pollution measures, showing strong accuracy for both population trends and unseen individuals.
- ❑ Clustering of latent spaces revealed groups with different sensitivities to pollution, with some clusters showing strong physiological reactions under high exposure.
- ❑ A threshold-based alert system was developed to warn individuals of likely adverse reactions, supporting behaviour changes to reduce exposure.

Main Notebook altogether

- Sampling_interval = 60, history_hours = 1, forecast_hours = 12, block_minutes = 60
- Window_size = 1* 60 // 60 = 1, block_size = 1, num_blocks = 12. This mean divide the 12 forecast_blocks In 12 parts, where each time I predict one block
- Batch_size is a training sample where each sampe as input and targets defined by SlidingWindowDataset

Part 1

Main Notebook altogether

- Sampling_interval = 60, history_hours = 1, forecast_hours = 12, block_minutes = 60
- Window_size = 1, block_size = 1, num_blocks = 12. This means I divide the 12 forecast_blocks In 12 parts, where each time I predict one block
- Batch_size is a training sample where each sample as input and targets defined by SlidingWindowDataset
- So for the data loader I am using window_size of 1 to predict 12 and in the dataloader I concatenate all of this altogether in 64 samples of this 1 past (input) to predict 12 steps but block size of 1.

Part 1

Main Notebook altogether

- Sampling_interval = 60, history_hours = 1, forecast_hours = 12, block_minutes = 60
- Window_size = 1* 60 // 60 = 1, block_size = 1, num_blocks = 12. This mean divide the 12 forecast_blocks In 12 parts, where each time I predict one block
- Batch_size is a training sample where each sampe as input and targets defined by SlidingWindowDataset

Dataset Summary

- Respiratory data I dropped ["INH016", "INH019","INH026", "INH126", "INH136"] because missing values of br_avg and br_std
- Merged respiratory and pollution on patient id and timestamp
- When filtering I:
  - Define the time difference between current point and previous one
  - Instantiate new segment if difference is greater than 1min (non continuous)
  - Keeping only segments which have at least 30rows of 1 minute data

Sliding Window Dataset

- n_samples = rows while n_features = features or columns
- How many valid windows can I extract from the sequence (goal is input and target) → num_windows = (n_samples - window_size - forecast_steps) // step + 1
- How many sliding windows can you extract given the window length, forecast length, and stride?
- The for loop:
    - start = i * step → step is my stride (needed it cause took me too much to train at the start)
    - window = data[start:start + window_size] (past values [8]) first goes from 0 to 8
    - target = data[start + window_size:start + window_size + forecast_steps] (goes from 8 to * + forecast_steps

Model.py – Positional Encoding

- div_term = torch.exp(torch.arange(0, d_model, 2, dtype=torch.float) * (-math.log(10000.0) / d_model)):

This is the denominator term as per paper formula

Div_term div decides the “speed” of sine/cosine oscillations in each embedding dimension

https://medium.com/@manindersingh120996/hands-on-with-transformers-recreating-attention-is-all-you-need-in-pytorch-step-by-step-ecfbf3e1985b

- Let’s not foroget we created an empty matrix. We are then adding in each row but colum 0,2,4 sine and 1,3, 5 cosine
    - pe[:, 0::2] = torch.sin(position * div_term)
    - pe[:, 1::2] = torch.cos(position * div_term)

- Adding a batch dimension with unsqueeze and register_buffer, save pe which is the positional encoding matrix on the same device as the model so can be trained on GPUs

Model.py – Custom Transformer Encoder

- MultiheadAttention:
  - multihead_attn = nn.MultiheadAttention(embed_dim, num_heads)
  - attn_output, attn_output_weights = multihead_attn(query, key, value)
  - attn_output → encoded representation of the input in term of attention (seq_len, batch, d_model)
  - The it goes through the dropout.
  - Then I basically add together the original embeddings with attention outputs that have been going through multiread self-attention
  - Position wise feedforward network:
    - src2 = self.linear2(self.dropout(self.activation(self.linear1(src))))
    - The result is then passed through a linear transformation – where I go from d_model 64 to dim_feedforward which is 512 (hidden size of feedforward layers)
    - Then a relu activation (I need to introduce nonlinear transformation) ReLU keeps positive inputs unchanged and zeroes negatives → gradient is either 1 or 0 → avoids vanishing gradients in practice.
    - LayerNorm is basically normalization ensuring each feature belong to the same scale

Model.py – Variational TimeSeriesTransformer

- We start with transforming input features = feature cols to d_model with a linear transformation → we want to improve expressiveness
- To the d_model we add positional encoding!
- We are then implementing Custom Transformer Encoder Later! It was custom because we started by storing attention weights but then we didn't implement it. Dim_feedforward = 512 (hidden size of the FFN inside each encoder layer)
- We instantiate self.mean_layers and self.logvar_layer. They take the encoder output and spit out two different vectors. They are just two simple vectors but through reparametisation and training, they will learn to be those
- Then let's take last step of encoder layer that really summarises the dataset and instantiate two linear layers (mu and logvar) Reparameterization ensures we can still train the model with backprop despite randomness.

Model.py – Latent Discriminator

- LatentDiscriminator is a binary classifier that judges whether a latent vector comes from the true Gaussian prior or from the encoder. It’s used to align the encoder’s latent distribution with $N(0,I)$

Train – how does training the model work?

- Doing 3 epochs. After putting model in training mode.
- Using the train_loader with the same logic as SlidingWindowDataset so I create an input and a target
    - targets   = full_targets.view(B, num_blocks, block_size, F_in) --. This is key because I am splitting the full_targets into blocks. So that the model can implement rollout training

    - 1. model discriminator and generator
        - Creating z_fake → model.reparameterize (mu, logvar) It's fake because it's model-generated latent codes, not guaranteed to follow Gaussian.
        - Creating z_real → Then we sample a real latent vector from a standard Gaussian prior (same shape as z_fake).
        - Then make a tensor full of 0 and a tensor full of ones
        - The discriminator is then trained

Visualisation

- # Inverse transforming back to original
- I don't actually need it anymore but at the start I was not predicting them all.
- If that's the case, when I inverse transform the scaler is expecting all the columns
- if scaler is not None:

Create zero-filled arrays with the full original number of columns the scaler expects.

- dummy_cols = scaler.n_features_in_
- preds_full = np.zeros((len(preds), dummy_cols))
- trues_full = np.zeros((len(trues), dummy_cols))

- # Fill first columns with predicted/true features
- preds_full[:, :preds.shape[1]] = preds
- trues_full[:, :trues.shape[1]] = trues

- # Inverse transforming then slice back to feature dimension
- preds = scaler.inverse_transform(preds_full)[:, :preds.shape[1]]
- trues = scaler.inverse_transform(trues_full)[:, :trues.shape[1]]

Utils

- def extract_latents_by_condition(model, dataloader, device):
- def normalize_risk_vector(vec, risk_indices, scaler):
- def get_risk_from_prediction(pred_vector, baseline_vector, risk_indices, scaler, method="euclidean", green_thresh=0.3, yellow_thresh=1.0, return_distance=False):
- def predict_with_model(model, dataloader, device):
- def scan_individual_risk(model,dataset,scaler,baseline_vector,risk_indices, device,):
- def scan_risk(model,dataset,scaler,baseline_vector,risk_indices,pollution_index,device):
- def compute_reactivity_score(deviations, pollution_vals):
- def predict_pollution_level(combined: pd.DataFrame, multiplier: float, model, feature_cols: list, scaler, window_size: int, forecast_steps: int, pollution_cols: list, physiological_cols: list, device, PatientLatentDataset
- def get_baseline_prediction_vector(model, dataloader, device, scaler, risk_indices)

Utils – extract_latents_by_condition

- Extracting both the latent vector z and the metadata coming with it.
- This is because of the patientlatentdataset
- Z = sampled latent vector, it encodes the past context upto that window
- Metadata does not come from the model, but it comes from the dataset and it is aligned with z. Metadata comes from the dataset stoered future row (GT at forecast horizon).
    - It is basically telling me how does the current latent vector going to impact the next future statest (not a prediction for now but in the dataset).

- I then reformat the metadata batch to turn it into a Pandas DataFrame.

```
• for x, meta_batch in dataloader:
•     # Reformat for transformer (seq_len, batch, features) and move to device
•     x = x.permute(1, 0, 2).to(device)
•     # Forward pass: only need latent mean and variance
•     _, mu, logvar, _ = model(x)
•     # Sample latent vector z using reparameterization trick
•     z = model.reparameterize(mu, logvar).cpu().numpy()
•     z_list.append(z)

•     # Converting to store the data in dataloader
•     # k is f.i. patient_id
•     # v is the context of the data where every row is raw physiological and pollution form of the data
•     clean_meta = {}
•     for k, v in meta_batch.items():
•         clean_meta[k] = np.array(v).tolist()
```

Utils – Normalize risk vector

- Create a dummy dataset with same number of features as original dataset. We inverse transform with the same number of features that the scaler was trained on so that the scaler does not throw us an error

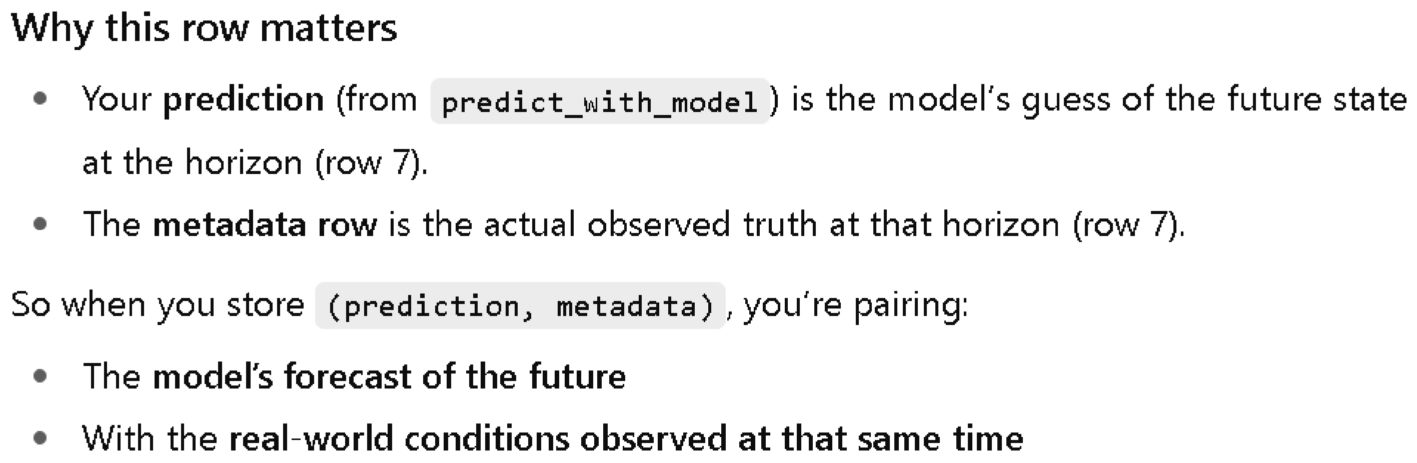
Utils – get_risk_from_prediction

- Comparing predicted with baseline vector
- If distance is less than threshold then green etc

Utils – Predict with model

- Create a dummy dataset with same number of features as original dataset. We inverse transform with the same number of features that the scaler was trained on so that the scaler does not throw us an error

predict_with_model



Why this row matters

- Your **prediction** (from `predict_with_model`) is the model's guess of the future state at the horizon (row 7).
- The **metadata row** is the actual observed truth at that horizon (row 7).

So when you store `(prediction, metadata)`, you're pairing:

- The **model's forecast of the future**
- With the **real-world conditions observed at that same time**

Utils – get_baseline_prediction_vector

- I predict the first horizon step for every window, collect those vectors, inverse-scale, keep only risk-related features, and then average them
- The resulting mean vector is my baseline profile for the individual. But I am not using the full horizon forecasts (steps 2…N are ignored).

Utils – Scan_individual_risk

- It’s taking the normal state of an individual with get_baseline_prediction_vector
- Then I compute the Eucldiean distance between the prediction and the normal. If the threshold is greater than 0.3 then green, red or yellow

Utils – Scan_risk

- It’s taking the normal state of an individual with get_baseline_prediction_vector
- Then I compute the Eucldiean distance between the prediction and the normal. If the threshold is greater than 0.3 then green, red or yellow

Compute_reactivity_score

- Find IQR pollution values
- Then I am collecting deviations when pollution_vals are lower than p (low_exposure or high_exposure)
- The last row is saying take the mean if list is empty, if not 0.0

Predict_pollution_level

- It’s the multiplier