Electronic Supplementary Material (ESI) for Journal of Materials Chemistry A.

This journal is © The Royal Society of Chemistry 2022

Supporting Information for

A MOF-based electronic nose for carbon dioxide sensing with enhanced affinity and selectivity by ionic-liquid embedment

Peng Qin,[a] Salih Okur,[a] Yunzhe Jiang[a] and Lars Heinke [a,*]

[a] Karlsruhe Institute of Technology (KIT), Institute of Functional Interfaces (IFG), Hermann-von-Helmholtz-Platz 1, 76344 Eggenstein-Leopoldshafen, Germany

Corresponding Author

E-mail: Lars.Heinke@kit.edu.

Table S1: Sensor experiments for all different gases and vapors with different concentrations

	analyte	concentration / $\%_{vol}$
1.	CO_2	1.97
2.	CO_2	1.48
3.	CO_2	0.98
4.	CO_2	0.49
5.	CO_2	0.19
6.	CO_2	0.09
7.	N_2	1.97
8.	N_2	1.48
9.	N_2	0.98
10.	N_2	0.49
11.	N_2	0.19
12.	N_2	0.09
13.	ethanol	1.97
14.	ethanol	1.48
15.	ethanol	0.98
16.	ethanol	0.49
17.	ethanol	0.19
18.	ethanol	0.09
19.	methanol	1.97
20.	methanol	1.48
21.	methanol	0.98
22.	methanol	0.49
23.	methanol	0.19
24.	methanol	0.09
25.	CO_2:N_2=1:1	1.97
26.	CO_2:N_2=1:1	1.48
27.	CO_2:N_2=1:1	0.98
28.	CO_2:N_2=1:1	0.49
29.	CO_2:N_2=1:1	0.19
30.	CO_2:N_2=1:1	0.09
31.	CO_2+10%vol humidity	0.19
32.	CO_2+20%vol humidity	0.19
33.	CO_2+30%vol humidity	0.19
34.	H_2O	0.09
35.	H_2O	0.19

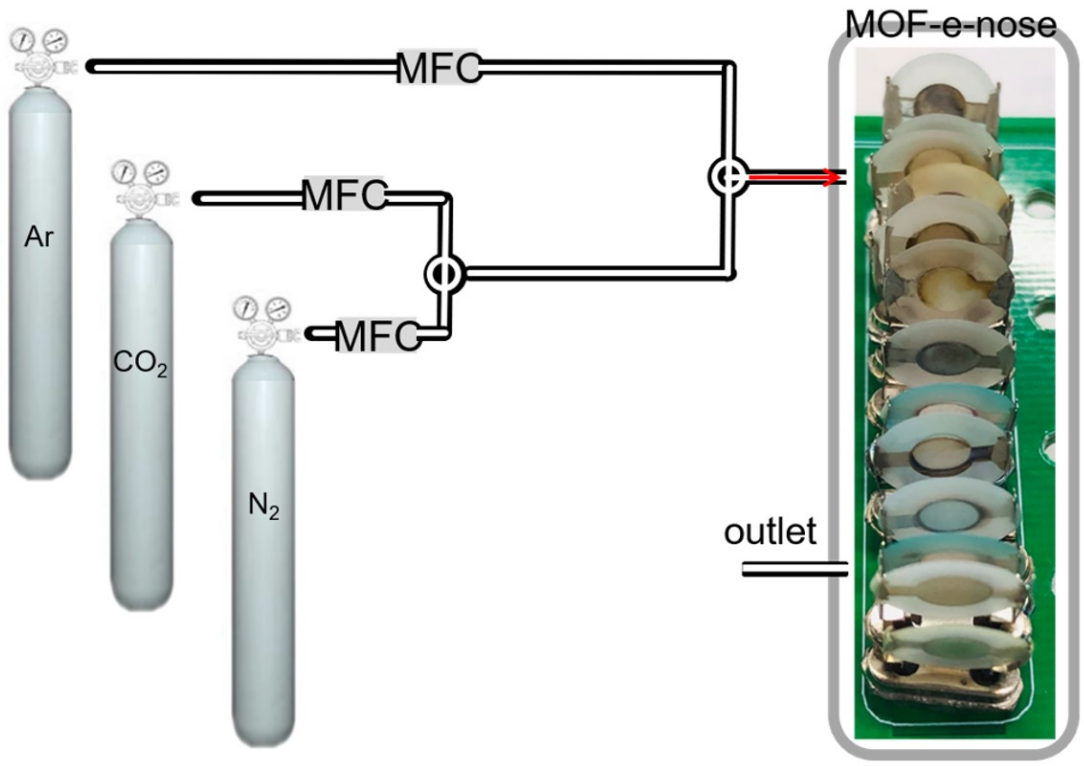


Figure S1. Sketch of the setup. The electronic nose consists of an array of 14 QCM sensors coated with MOF films of HKUST-1 and UiO-66 structures. CO_2 and N_2 concentrations are controlled by their flow rates through mass flow controllers (MFCs). The sensor array is located in a gas flow chamber with an inner diameter of 15 mm and a length of 35 mm.

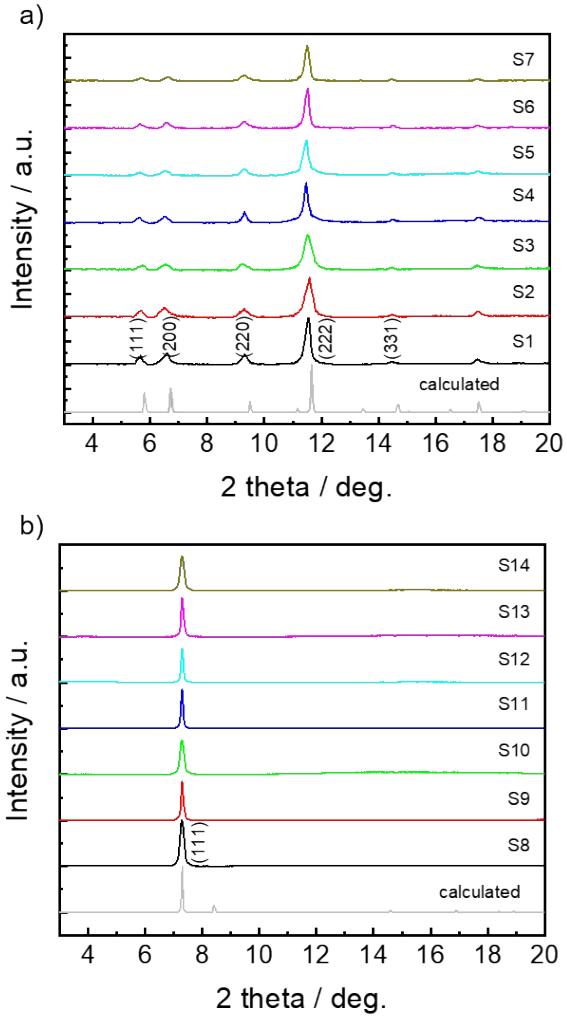


Figure S2. X-ray diffractograms (XRDs) of the MOF films. The MOF names as well as the observed diffraction peaks are labelled. The calculated XRDs are shown below the experimental data. The data show that all samples have the targeted crystalline MOF structures.

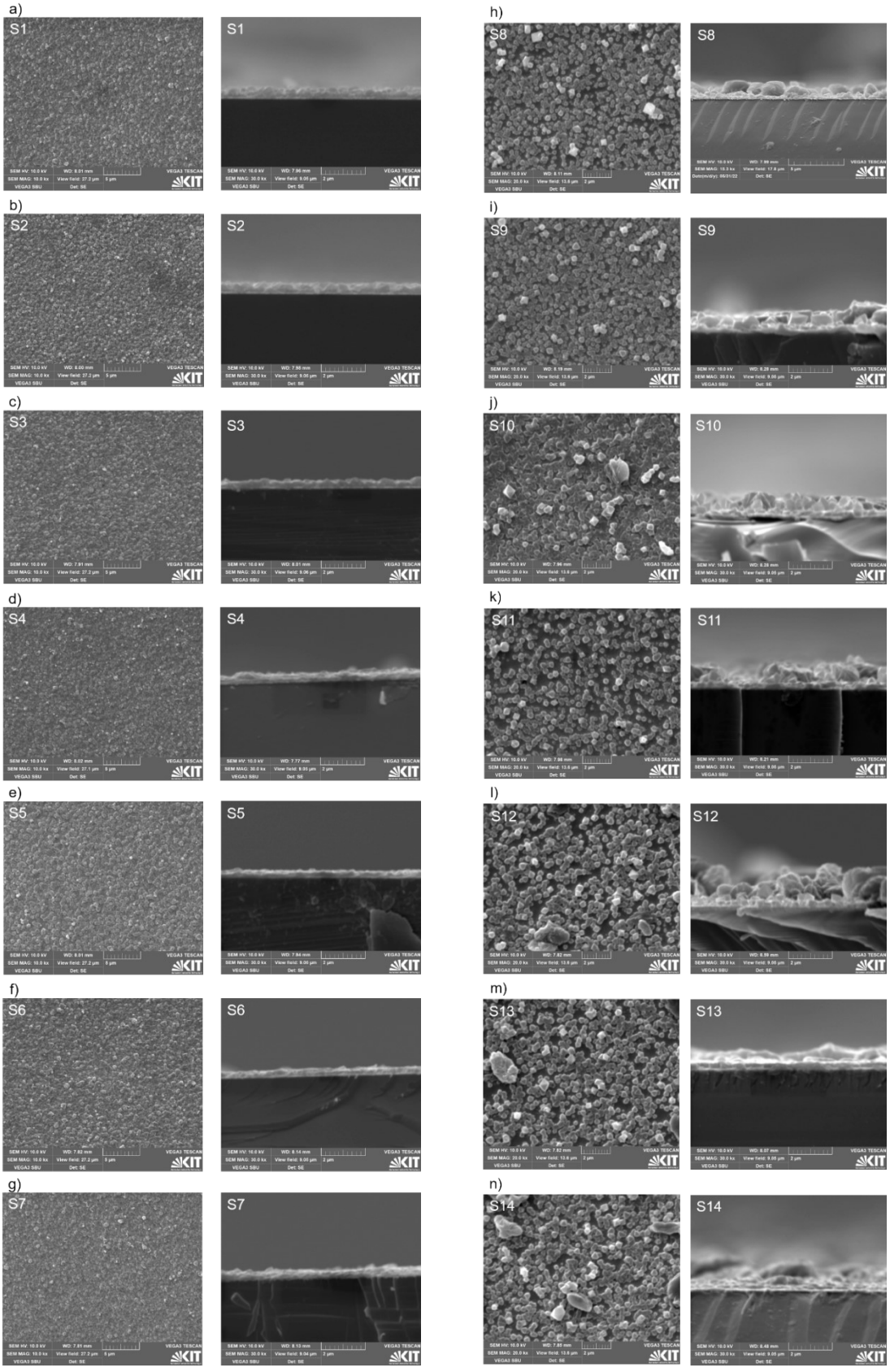


Figure S3. SEM images of all sensors. The left image shows a top view of the sample and the right image shows a side view of the broken sample. The images show the scale bar and measurement parameters. Sensor names are labeled in the images.

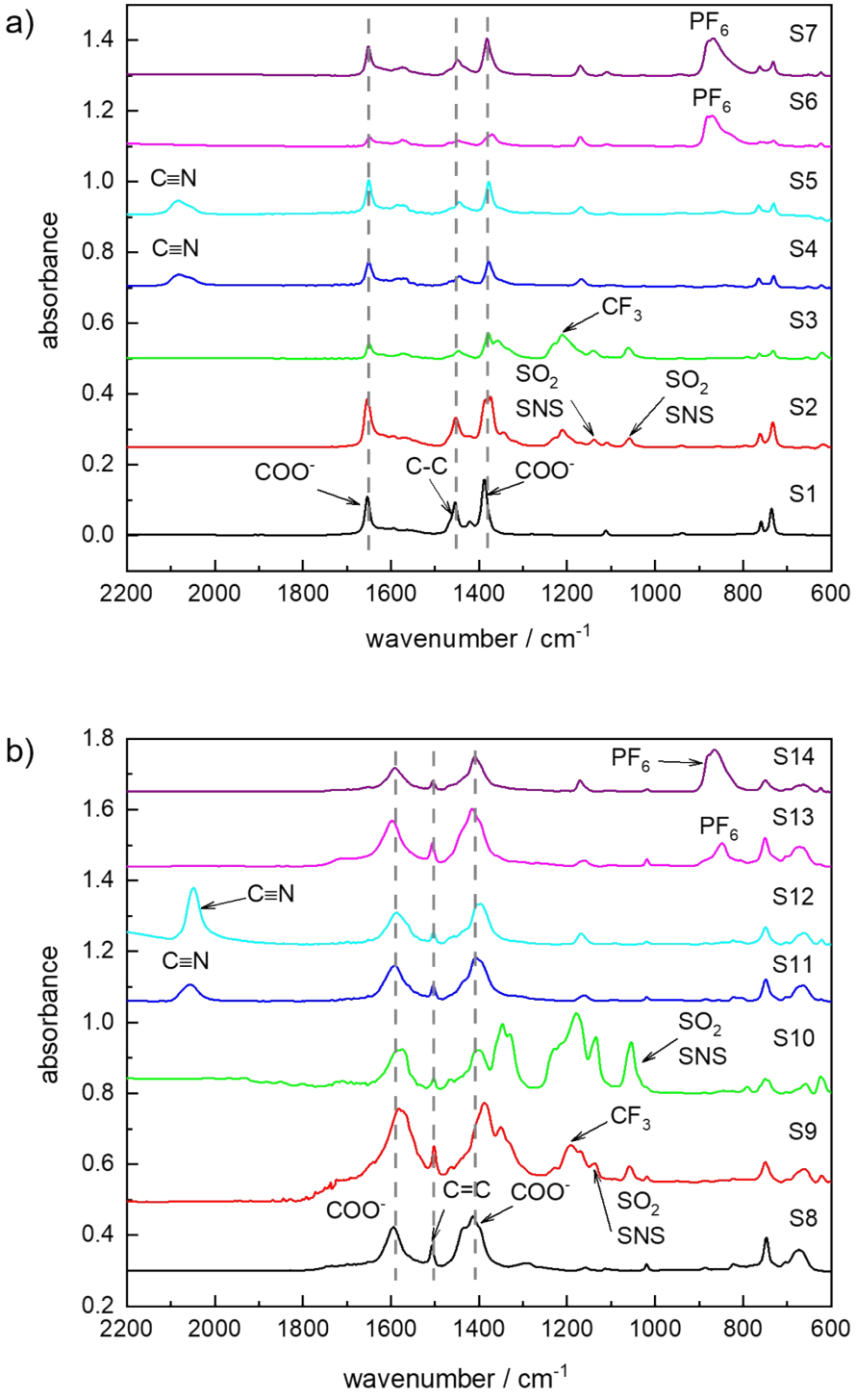
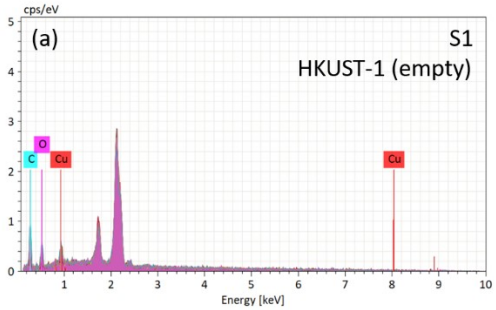
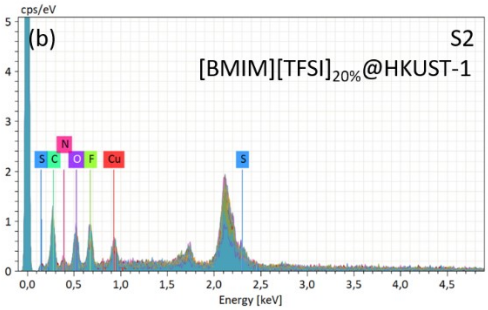
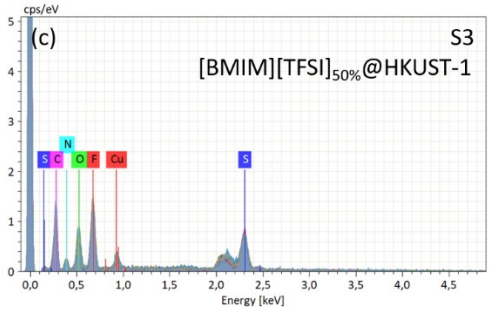
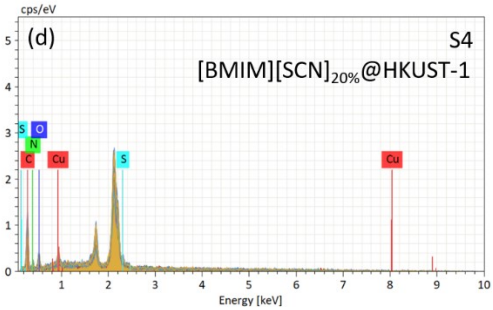


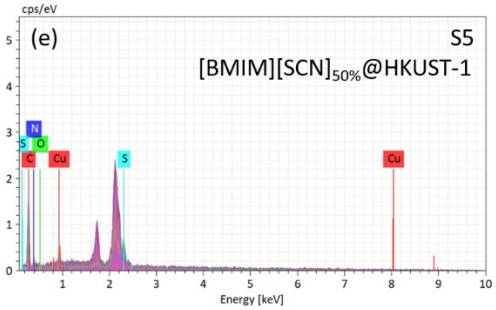
Figure S4: Vibrational spectra of the samples measured *via* infrared absorption reflection spectroscopy (IRRAS). The sample names (see table 1) and the main vibration bands of the MOFs and of the ILs are labelled. IRRAS was performed with a Bruker Vertex 80 spectrometer, using a resolution of 2 cm^{-1}. It was recorded in grazing incidence reflection mode at an angle of incidence of 80° relative to the surface normal.

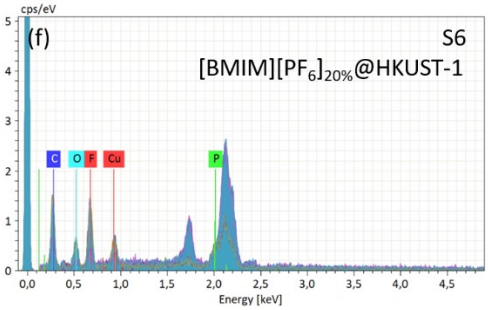

cps/eV
(a)
S1
HKUST-1 (empty)
O
C
Cu
Cu
Energy [keV]

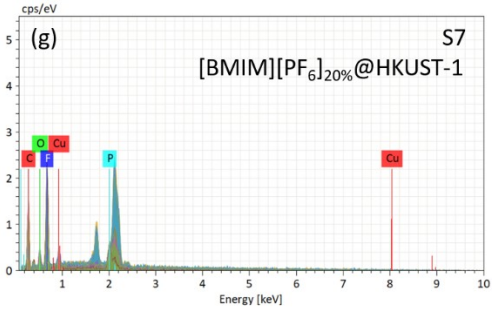

cps/eV
(b)
S2
[BMIM][TFSI]20%@HKUST-1
N
S C O F Cu
S
Energy [keV]


cps/eV
(c)
S3
[BMIM][TFSI]50%@HKUST-1
N
S C O F Cu
S
Energy [keV]


cps/eV
(d)
S4
[BMIM][SCN]20%@HKUST-1
S O
N
C Cu
S
Cu
Energy [keV]


cps/eV
(e)
S5
[BMIM][SCN]50%@HKUST-1
N
S O
C Cu
S
Cu
Energy [keV]


cps/eV
(f)
S6
[BMIM][PF6]20%@HKUST-1
C O F Cu
P
Energy [keV]


cps/eV
(g)
S7
[BMIM][PF6]20%@HKUST-1
O Cu
C F
P
Cu
Energy [keV]

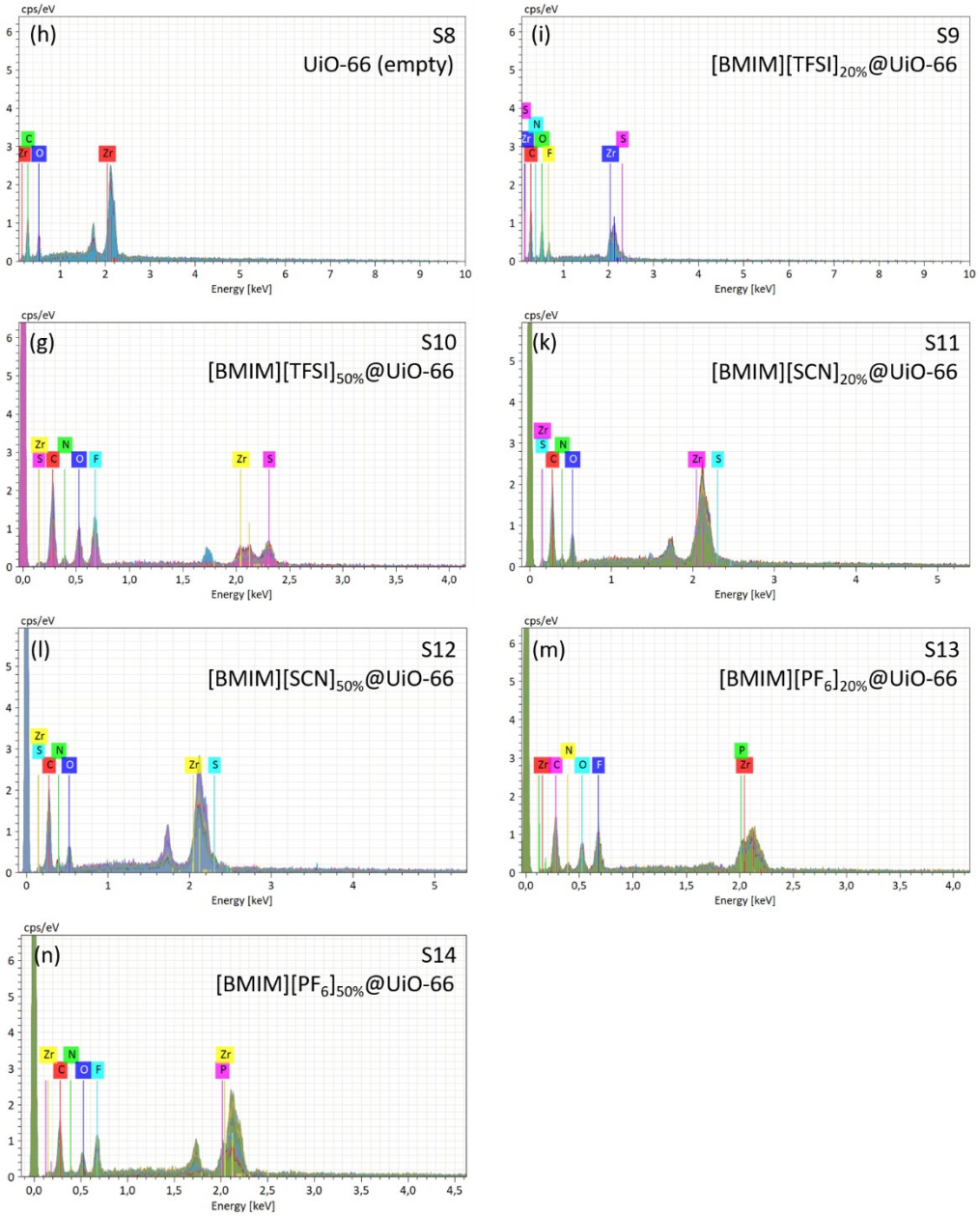


Figure S5: Energy-dispersive X-ray (EDX) spectra of the samples. The observed elements (F, C, N, O, S, P, Cu and Zr) are labelled. The quantification of the elements is given in Table S2. EDX spectroscopy was performed with a TESCAN VEGA3 scanning electron microscope equipped with a Bruker EDX unit.

The EDX spectra were recorded with a TESCAN VEGA3 scanning electron microscope equipped with a Bruker EDX unit.

The EDX spectra are shown in Figures S5. All the elements of the IL and MOF can be found in the spectra, except of hydrogen. For quantifying the IL content, we use elements which only occur in the IL (i.e. N and F) and compare it with the amount of Cu (for HKUST-1) and Zr (for UiO-66). The IL content is determined at many (~5) different positions and the average with standard deviation is shown in Table S2.

The free pore volume is determined by the free pore volume of the empty MOF structure (i.e. 66% for HKUST-1[3] and 60.2% for UiO-66[4] and unit cell sizes of (2.634 nm)[3] and (2.074 nm)[3], respectively), the van-der-Waals-volume of each IL pair (0.308 nm^3 for one pair of [BMIM][TFSI], 0.215 nm^3 for one pair of [BMIM][PF6] and 0.195 nm^3 for one pair of [BMIM][SCN], calculated by the method of ref.[5] and verified by MaterialStudio software) and the IL-loading (determined by EDX).

Table S2: Determined IL loading and free pore volume.

Sample name	type	IL loading [IL pairs per unit cell]	Free pore volume per unit cell [nm^3]
S1	HKUST-1 (empty)	0	12.1
S2	$[BMIM][TFSI]_{20\%}$@HKUST-1	5.3±1.4	10.4
S3	$[BMIM][TFSI]_{50\%}$@HKUST-1	9.8±0.3	9.0
S4	$[BMIM][SCN]_{20\%}$@HKUST-1	6.1±1.1	10.9
S5	$[BMIM][SCN]_{50\%}$@HKUST-1	11.9±0.4	9.7
S6	$[BMIM][PF_6]_{20\%}$@HKUST-1	5.1±1.6	11.0
S7	$[BMIM][PF_6]_{20\%}$@HKUST-1	9.7±1.1	10.0
S8	UiO-66 (empty)	0	5.4
S9	$[BMIM][TFSI]_{20\%}$@UiO-66	1.7±0.6	4.8
S10	$[BMIM][TFSI]_{50\%}$@UiO-66	5.5±0.6	3.7
S11	$[BMIM][SCN]_{20\%}$@UiO-66	2.3±0.6	4.9
S12	$[BMIM][SCN]_{50\%}$@UiO-66	6.2±0.9	4.2
S13	$[BMIM][PF_6]_{20\%}$@UiO-66	2.2±0.9	4.9
S14	$[BMIM][PF_6]_{50\%}$@UiO-66	4.9±0.6	4.3

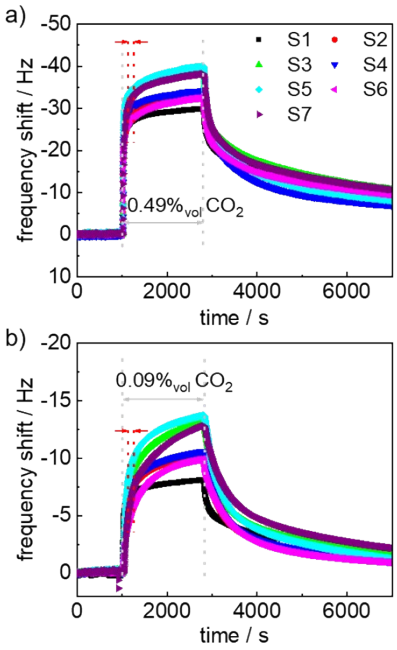


Figure S6. QCM data of the CO_2 exposure with a concentration of 0.49% (a) and 0.09% (b) as a function of time. The color code for the MOF sensors is given in the legend. Here, only the HKUST-1 sensors are shown, the UiO-sensors are shown in Figure S7.

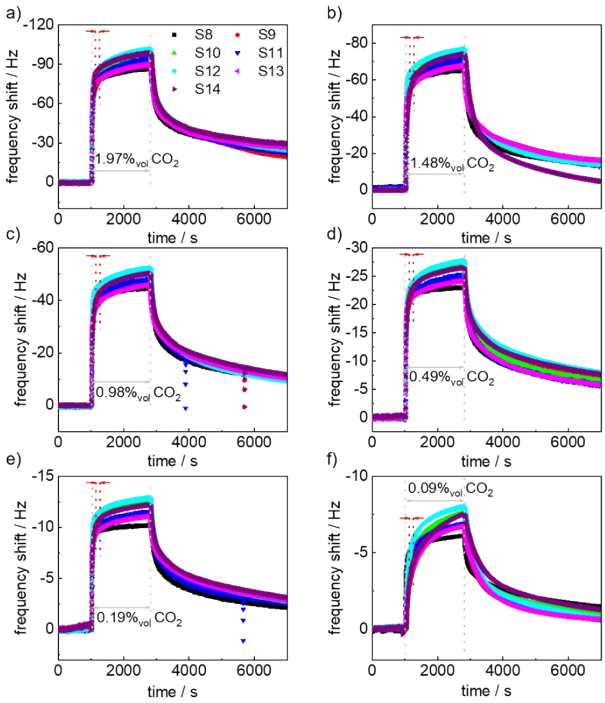


Figure S7. QCM data of the CO_2 exposure with a concentration of 1.97% (a), 1.48% (b), 0.98% (c), 0.49% (d), 0.19% (e) and 0.09% (f) as a function of time. The color code for the MOF sensors is given in the legend. Here, only the UiO-66 sensors are shown, the HKUST-1-sensors are shown in Figures 2 and S6.

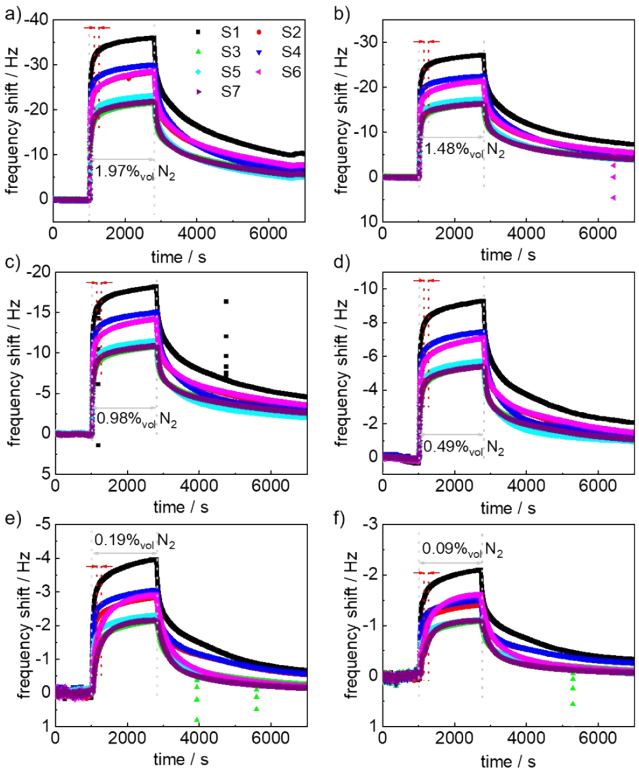


Figure S8. QCM data of the N_2 exposure with a concentration of 1.97% (a), 1.48% (b), 0.98% (c), 0.49% (d), 0.19% (e) and 0.09% (f) as a function of time. The color code for the MOF sensors is given in the legend. Here, only the HKUST-1 sensors are shown, the UiO-sensors are shown in Figure S9.

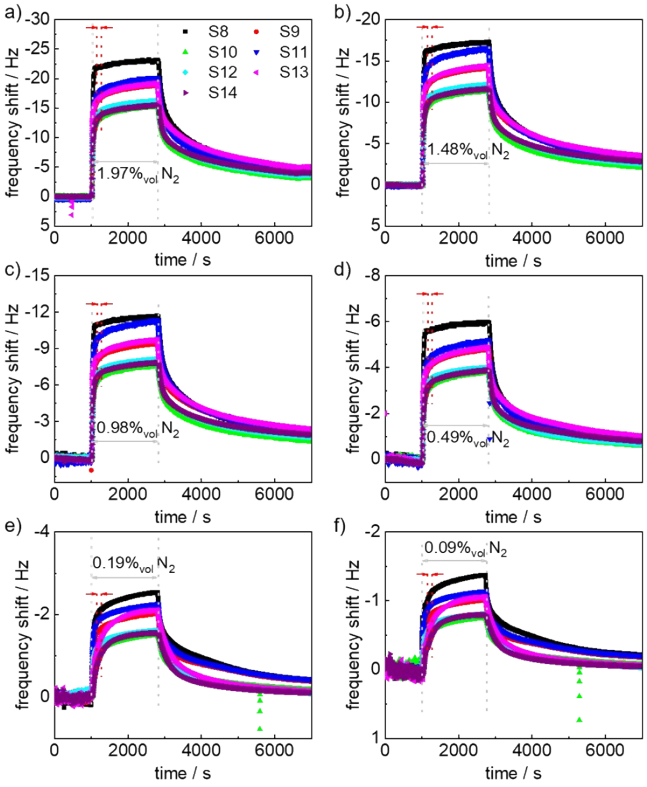


Figure S9. QCM data of the N_2 exposure with a concentration of 1.97% (a), 1.48% (b), 0.98% (c), 0.49% (d), 0.19% (e) and 0.09% (f) as a function of time. The color code for the MOF sensors is given in the legend. Here, only the UiO-66 sensors are shown, the HKUST-1 sensors are shown in Figure S8.

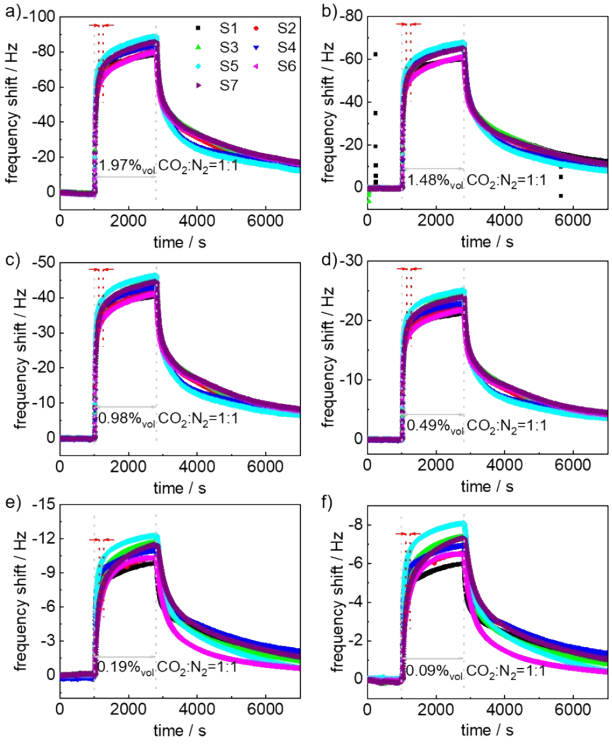


Figure S10. QCM data of the binary CO_2:N_2 = 1:1 exposure with a concentration of 1.97% (a), 1.48% (b), 0.98% (c), 0.49% (d), 0.19% (e) and 0.09% (f) as a function of time. The color code for the MOF sensors is given in the legend. Here, only the HKUST-1 sensors are shown, the UiO-sensors are shown in Figure S11.

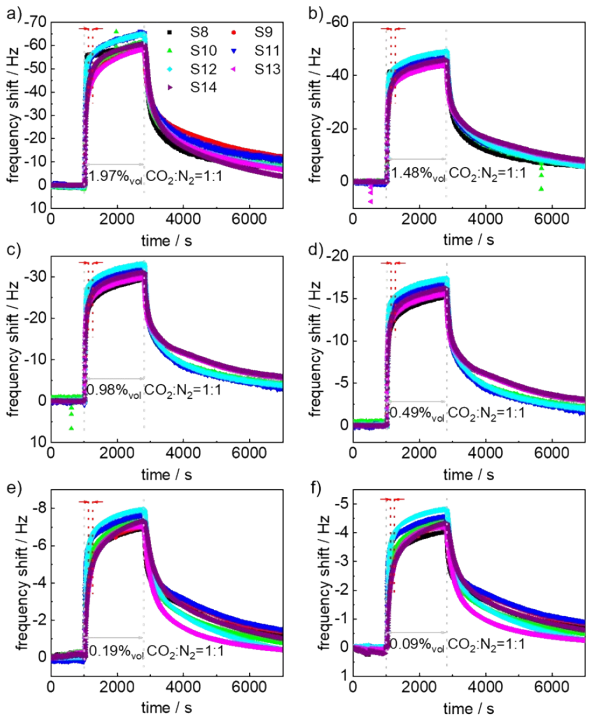


Figure S11. QCM data of the binary CO_2:N_2 = 1:1 exposure with a concentration of 1.97% (a), 1.48% (b), 0.98% (c), 0.49% (d), 0.19% (e) and 0.09% (f) as a function of time. The color code for the MOF sensors is given in the legend. Here, only the UiO-66 sensors are shown, the HKUST-1 sensors are shown in Figure S10.

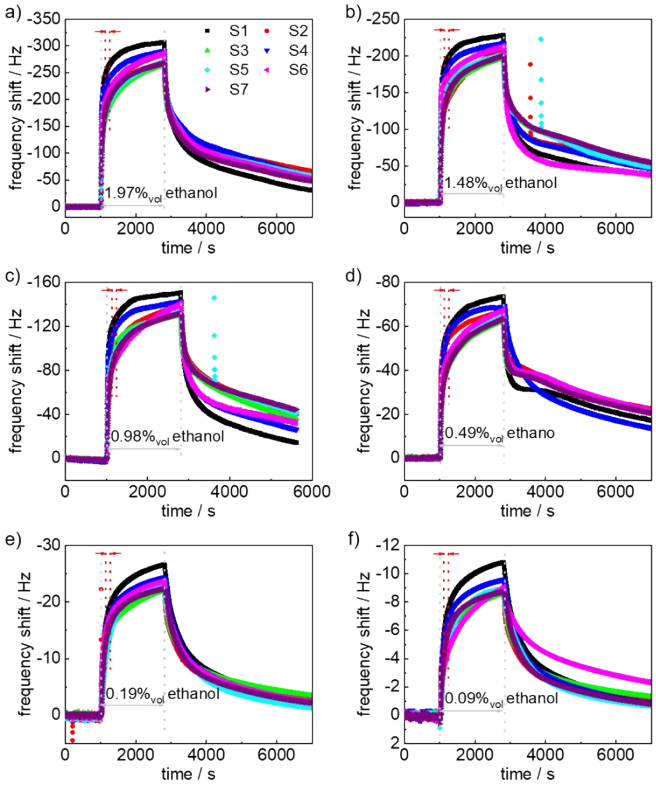


Figure S12. QCM data of the ethanol exposure with a concentration of 1.97% (a), 1.48% (b), 0.98% (c), 0.49% (d), 0.19% (e) and 0.09% (f) as a function of time. The color code for the MOF sensors is given in the legend. Here, only the HKUST-1 sensors are shown, the UiO-sensors are shown in Figure S13.

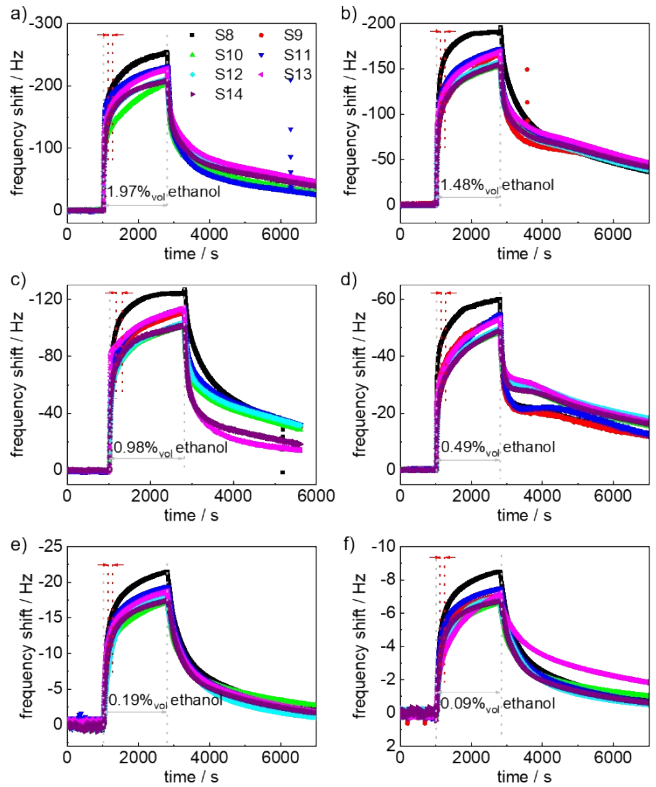


Figure S13. QCM data of the ethanol exposure with a concentration of 1.97% (a), 1.48% (b), 0.98% (c), 0.49% (d), 0.19% (e) and 0.09% (f) as a function of time. The color code for the MOF sensors is given in the legend. Here, only the UiO-66 sensors are shown, the HKUST-1 sensors are shown in Figure S12.

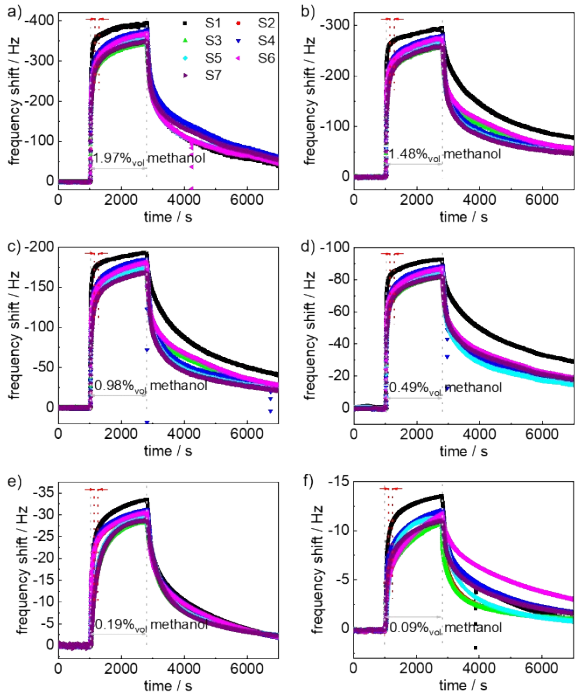


Figure S14. QCM data of the methanol exposure with a concentration of 1.97% (a), 1.48% (b), 0.98% (c), 0.49% (d), 0.19% (e) and 0.09% (f) as a function of time. The color code for the MOF sensors is given in the legend. Here, only the HKUST-1 sensors are shown, the UiO-sensors are shown in Figure S15.

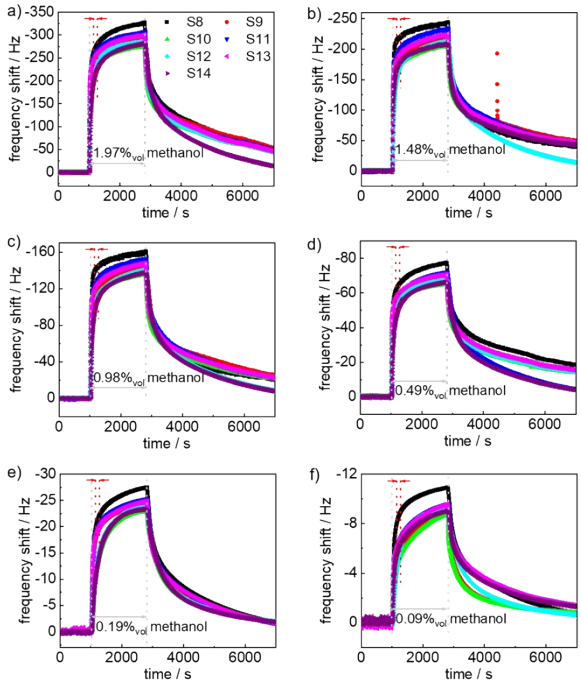


Figure S15. QCM data of the methanol exposure with a concentration of 1.97% (a), 1.48% (b), 0.98% (c), 0.49% (d), 0.19% (e) and 0.09% (f) as a function of time. The color code for the MOF sensors is given in the legend. Here, only the UiO-66 sensors are shown, the HKUST-1 sensors are shown in Figure S14.

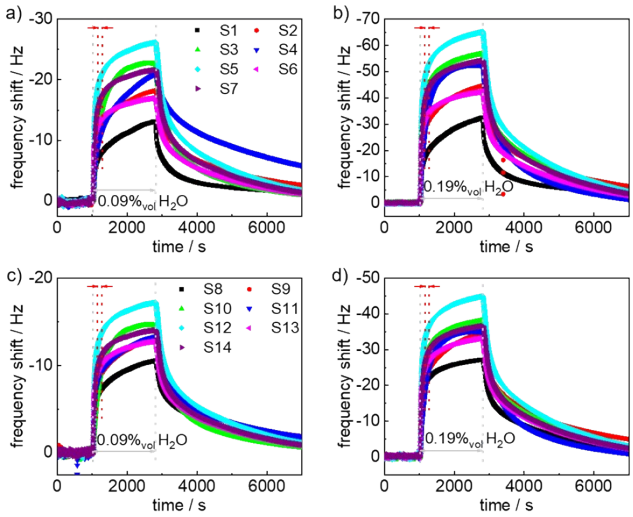


Figure S16. QCM data of the water exposure with a concentration of 0.09% (a), 0.19% (b), 0.09% (c), 0.19% (d) as a function of time. The gases and concentrations are labelled in the panels. The color code for the MOF sensors is given in the legend. a-b) the HKUST-1 sensors are shown, c-d) the UiO-66 sensors are shown.

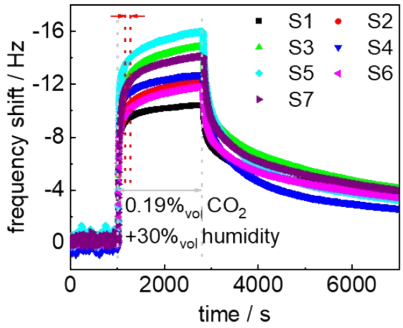


Figure S17. QCM data of the CO_2 at 0.19% at 30% humidity exposure. The color code for the MOF sensors is given in the legend. Here, only the HKUST-1 sensors are shown, the UiO-sensors are shown in Figure S18.

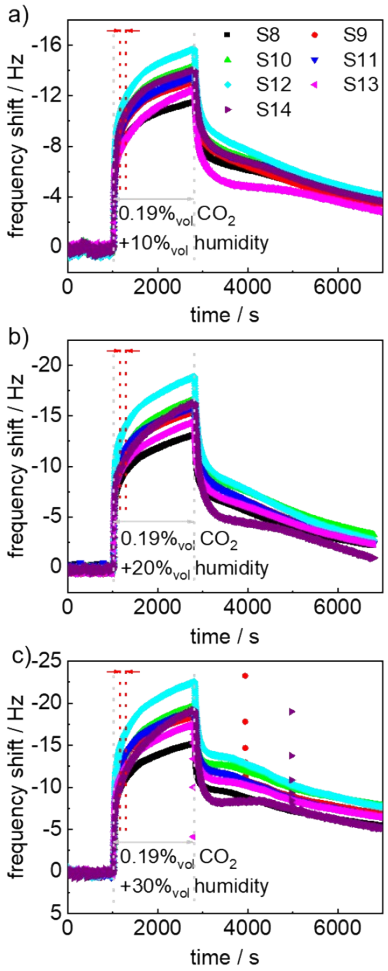


Figure S18. QCM data of the CO_2 exposure at 0.19% with different 30% humidity levels. a): 10% humidity, b): 20% humidity and c): 30% humidity. Gas molecules and humidity are indicated on the panel. Here, only the UiO-66 sensors are shown, the HKUST-1 sensors are shown in Figure S17 and 2.

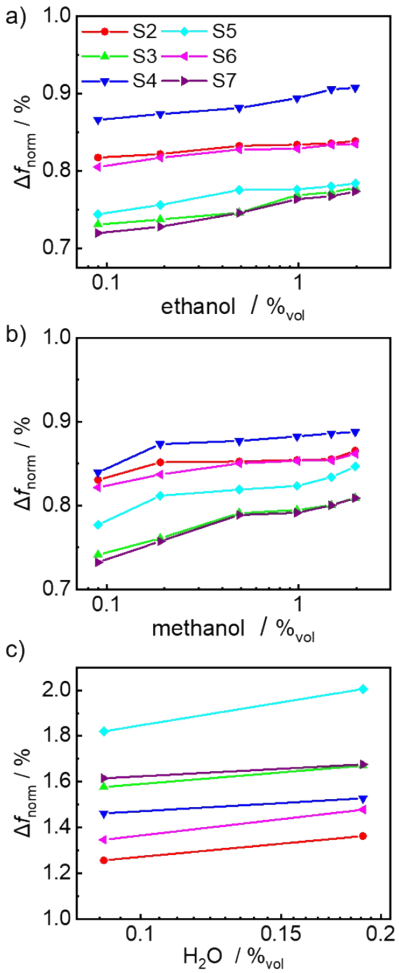


Figure S19. Ratio (Δf_{norm}) of the analyte uptake by the sensors loaded with IL and the analyte uptake by the empty HKUST-1 sensor *versus* the analyte concentration. a) ethanol, b) methanol and c) water. Here, only the HKUST-1 sensors are shown, the UiO-sensors are shown in Figure S21.

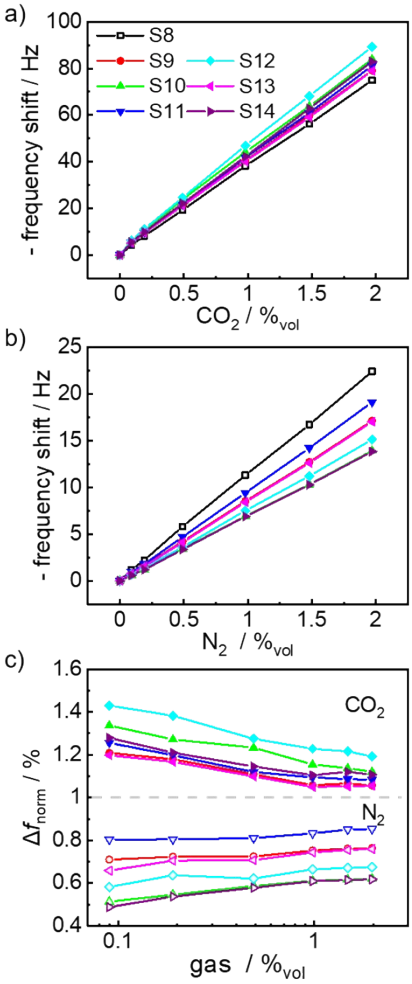


Figure S20. Negative values of the frequency shift *versus* concentration of a) CO_2 and b) N_2. c) Ratio (Δf_{norm}) of the analyte uptake by the sensors loaded with IL and the analyte uptake by the empty sensor *versus* the analyte concentration. Here, only the UiO-66 sensors are shown, the HKUST-1-sensors are shown in Figure 3.

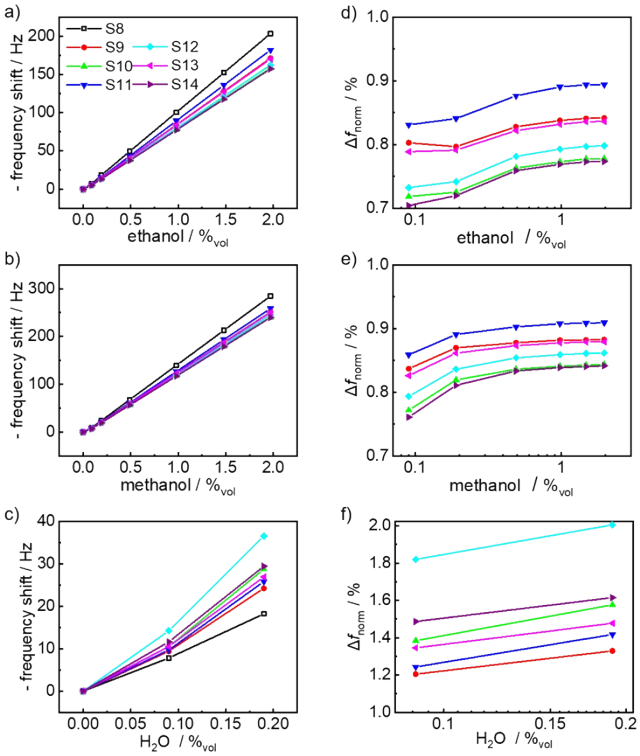


Figure S21. Negative values of the frequency shift *versus* analyte concentration of a) ethanol, b) methanol and c) water. d-f) Ratio (Δf_{norm}) of the analyte uptake by the sensors loaded with IL and the analyte uptake by the empty sensor *versus* the analyte concentration. Here, only the UiO-66 sensors are shown, the HKUST-1-sensors are shown in Figure 3 and S19

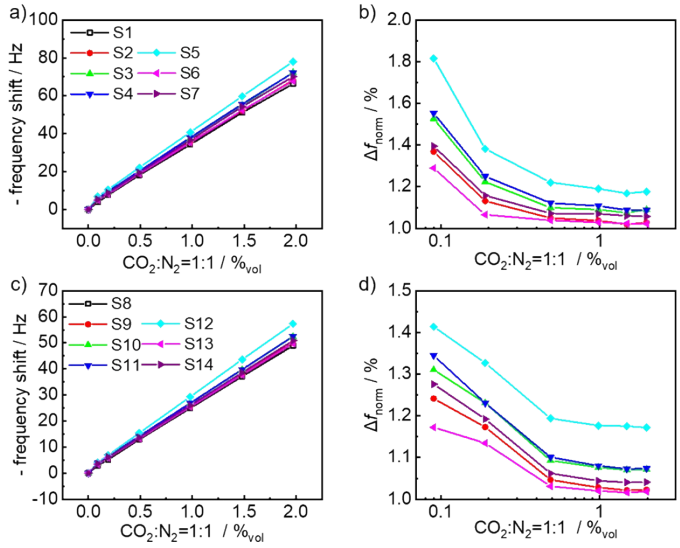


Figure S22. Negative values of the frequency shift *versus* concentration of a CO_2:N_2 = 1:1 binary gas mixture. a) the HKUST-1 sensors are shown , c) the UiO-66 sensors are shown. b and d) Ratio (Δf_{norm}) of the analyte uptake by the sensors loaded with IL and the analyte uptake by the empty sensor *versus* the analyte concentration.

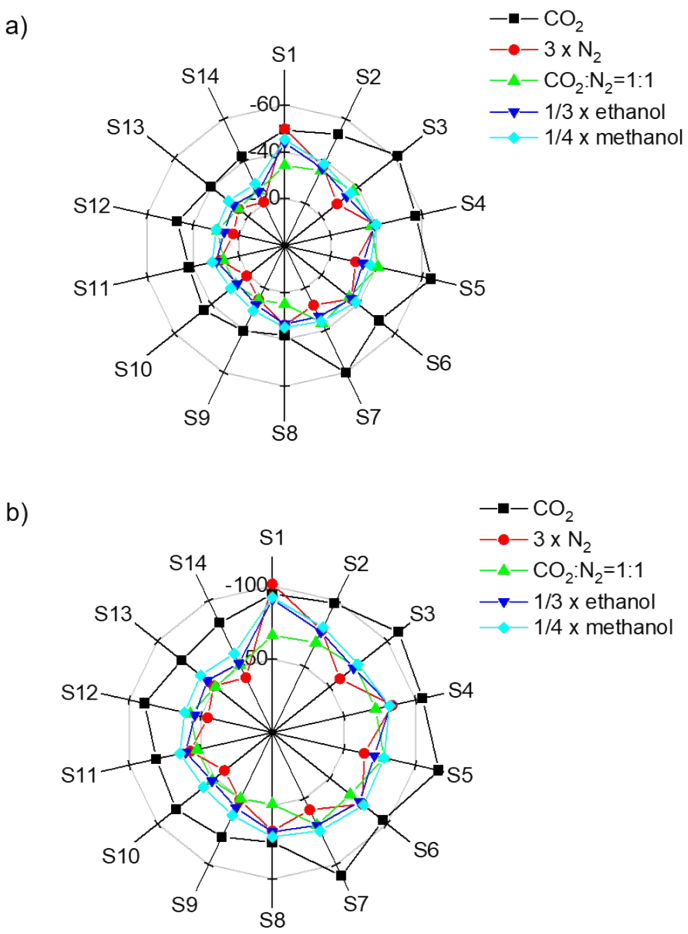


Figure S23. Radar plot of the frequency shift Δf for exposure to CO_2, N_2, CO_2:N_2=1:1, ethanol and methanol at a concentration of a) 0.98%$_{vol}$ and b) 1.97%$_{vol}$. Each axis represents a different sensor, see table 1. For a better visibility, the data of nitrogen has been multiplied by 3, and the ethanol and methanol data have been divided by 3 and 4, respectively.

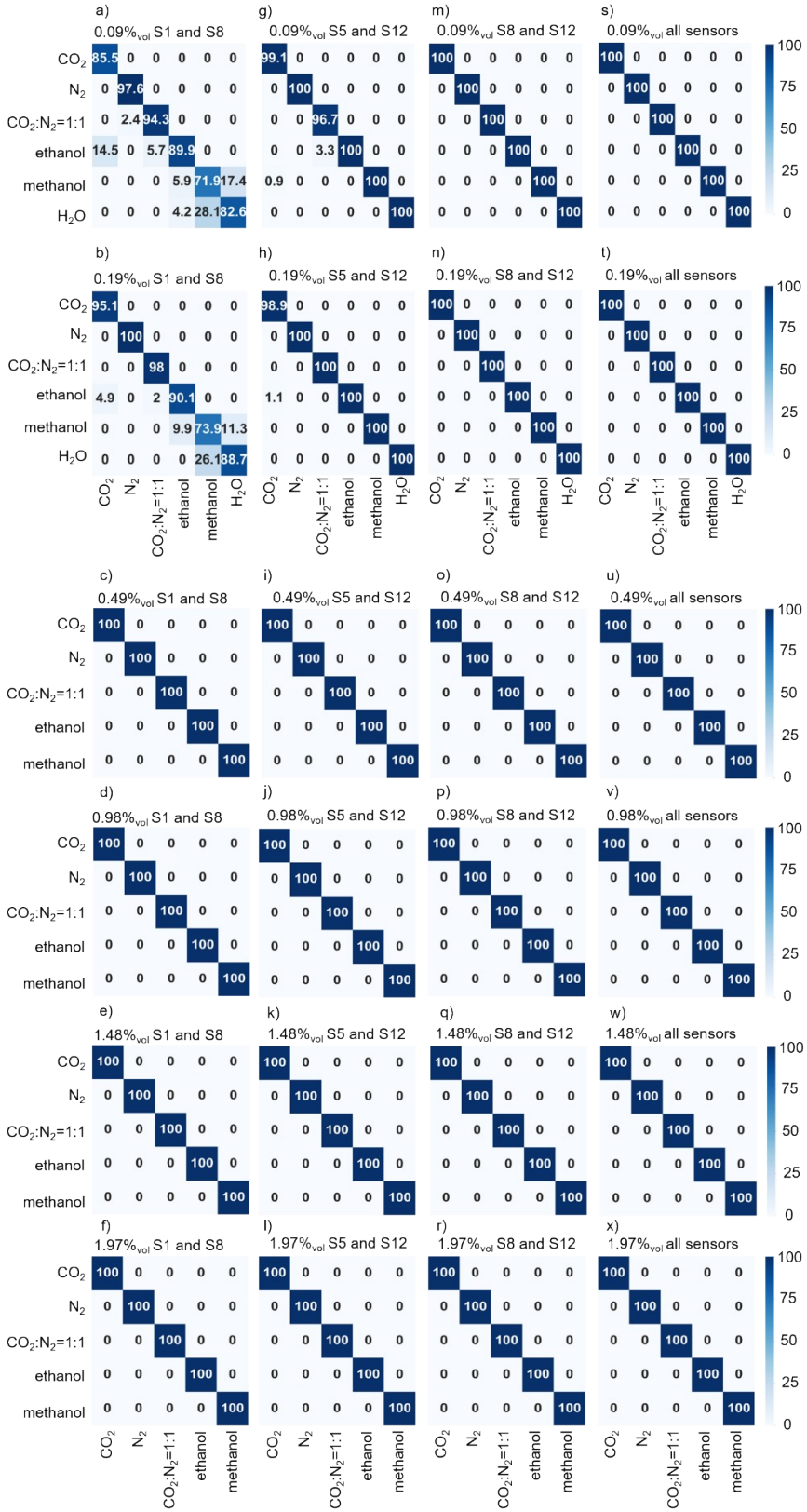


Figure S24. a)-f) Confusion matrixes based on the data from sensors S1 and S8 (empty HKUST-1 and UiO-66 sensors). a) at 0.09%$_{vol}$, b) at 0.19%$_{vol}$, c) at 0.49%$_{vol}$, d) at 0.98%$_{vol}$, e) at 1.48%$_{vol}$ and f) at 1.97%$_{vol}$. g)-l) Confusion matrixes based on the data from sensors S5 and S12. g) at 0.09%$_{vol}$, h) at 0.19%$_{vol}$, i) at 0.49%$_{vol}$, j) at 0.98%$_{vol}$, k) at 1.48%$_{vol}$ and l) at 1.97%$_{vol}$. m)-r) Confusion matrixes based on the data from sensors S8 and S12. m) at 0.09%vol, n) at 0.19%vol, o) at 0.49%vol, p) at 0.98%vol, q) at 1.48%vol and r) at 1.97%vol. s)-x) Confusion matrixes based on the data from entire sensor array. s) at 0.09%vol, t) at 0.19%vol, u) at 0.49%vol, v) at 0.98%vol, w) at 1.48%vol and x) at 1.97%vol.

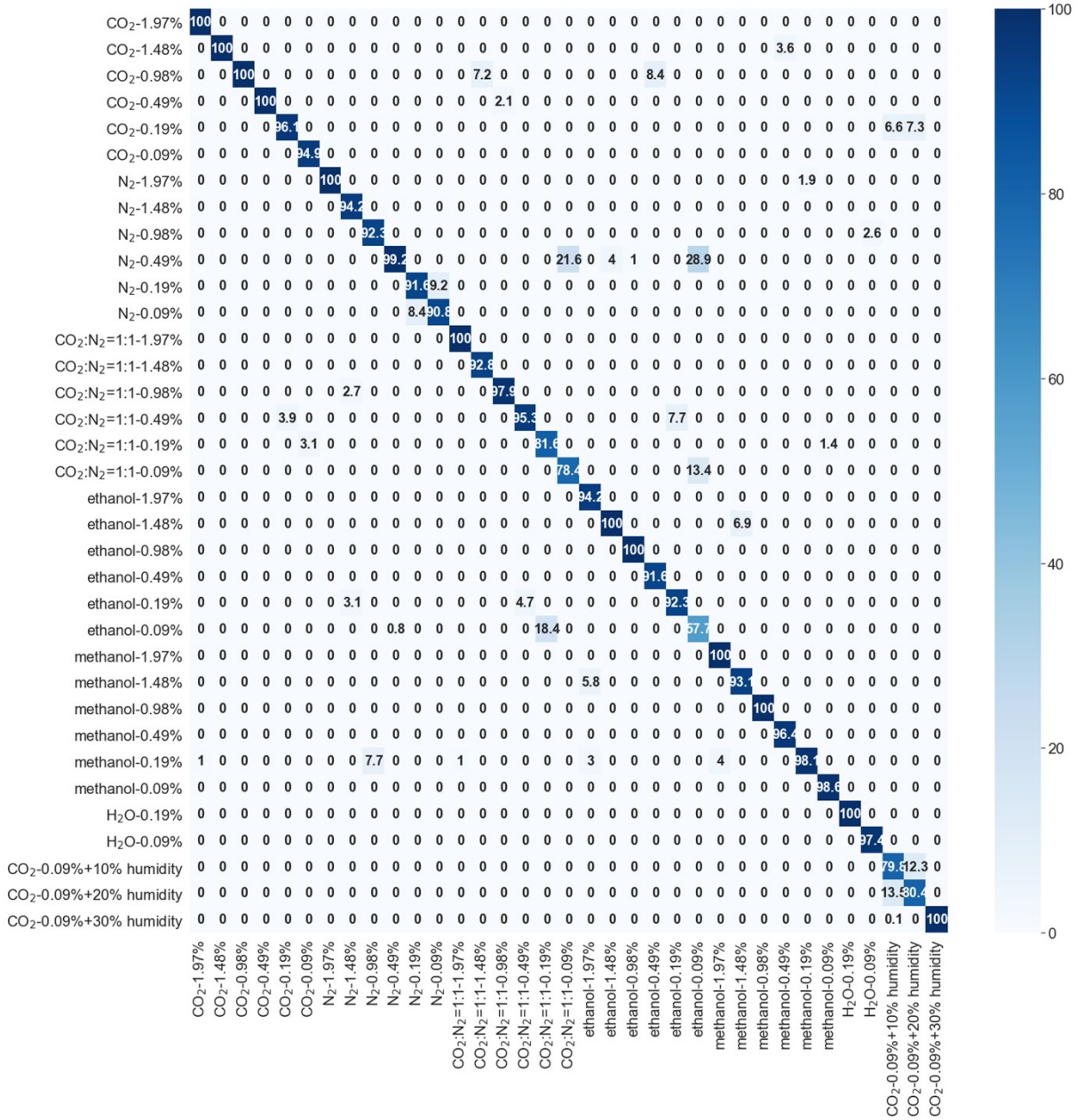


Figure S25. Confusion matrix based on the data from the entire sensor array. The sensor data contains 35 classes for CO_2, N_2, CO_2:N_2=1:1, ethanol, methanol and H_2O at different concentrations. The average accuracy is 93.8%.

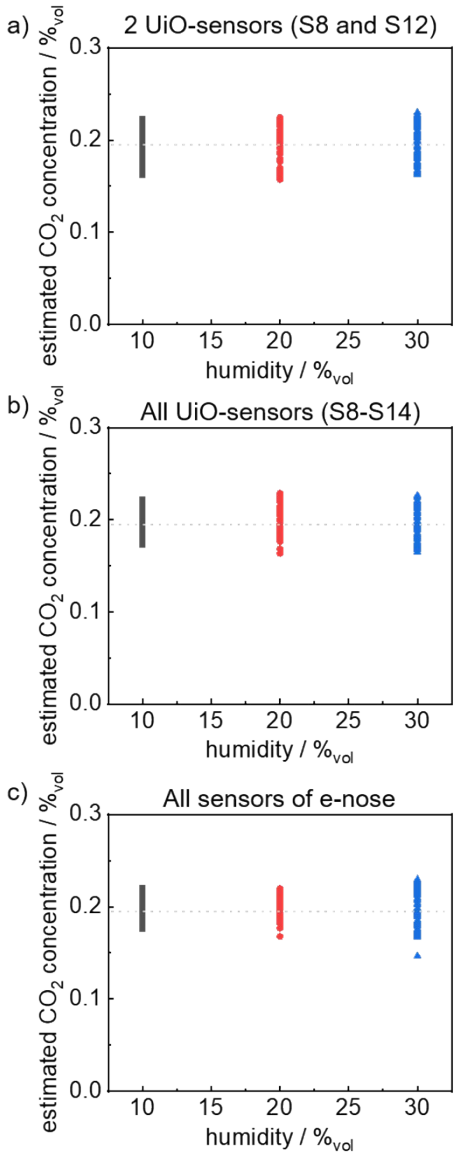


Figure S26. Comparison of the estimated gas concentration at different humidity levels and the actual CO_2 concentration of 0.19%$_{vol}$ (dotted line). a) based on the data from S8 and S12, b) based on the data from all UiO-sensors (S8-S14) and c) based on the data from the entire e-nose.

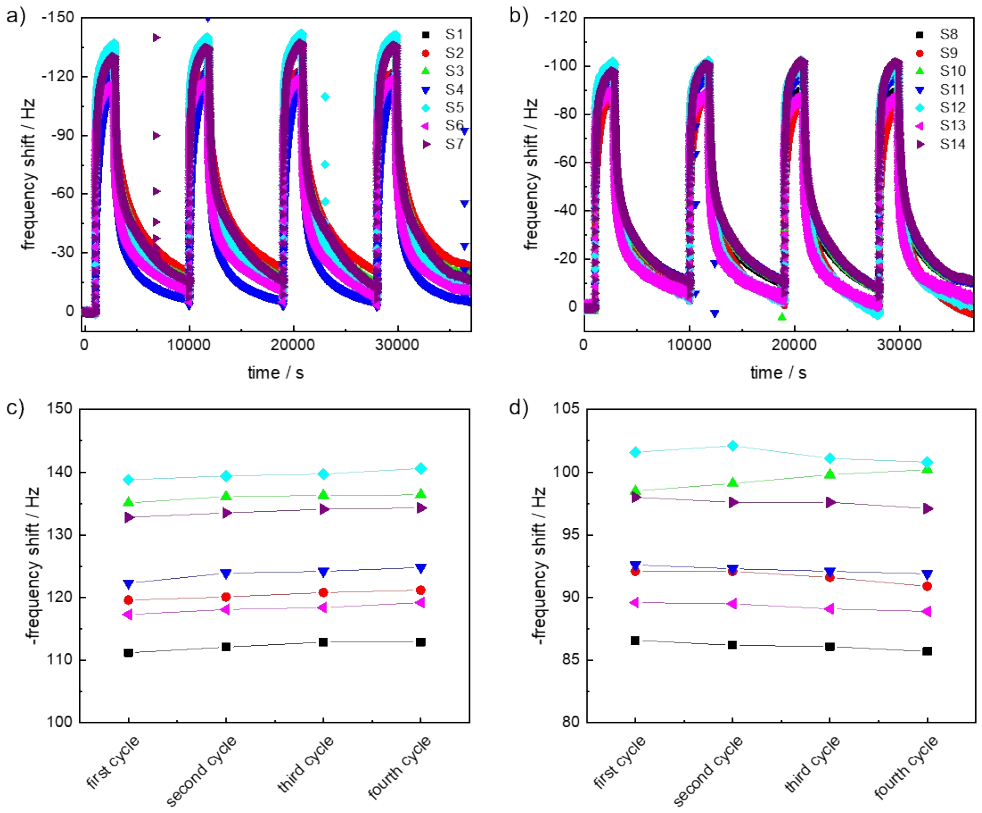


Figure S27: QCM data of a) HKUST-1 sensor array and b) UiO-66 sensor array for 4 cycles of adsorption at 1.97% CO_2 concentration. Comparison of the frequency shifts observed at the end of exposure for the different cycles for the HKUST-1 sensors (c) and the UiO-66 sensors (d). The average standard deviation for the four cycles and all samples is 0.61 Hz.

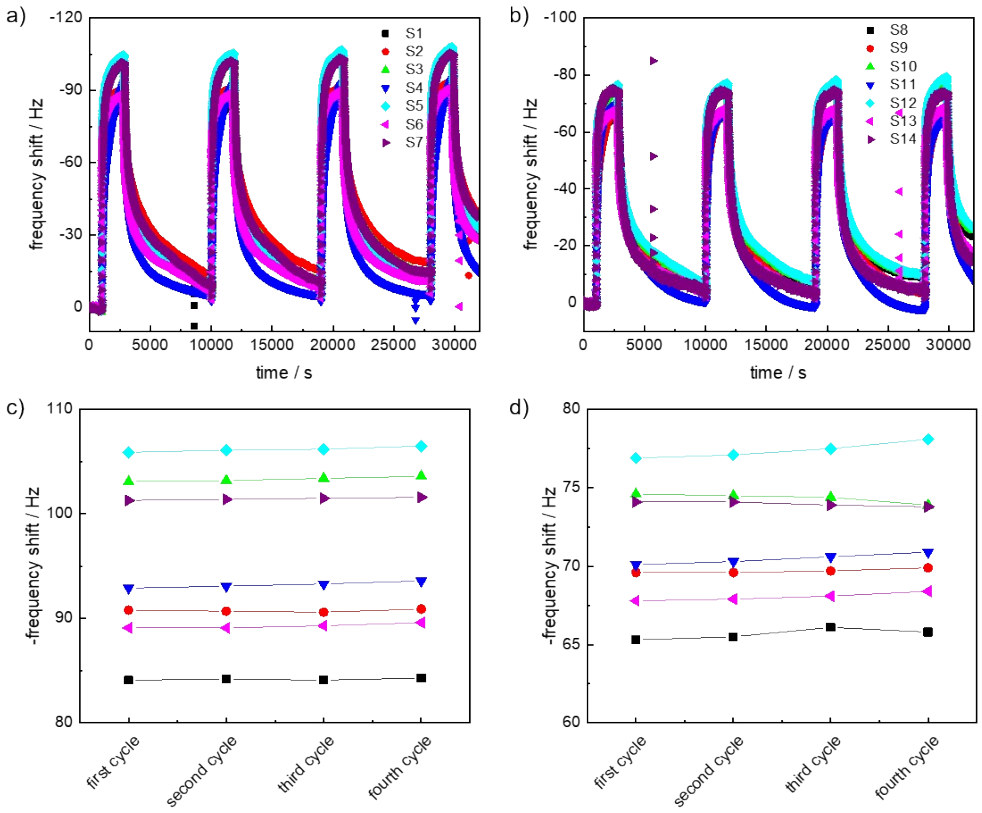


Figure S28: QCM data of a) HKUST-1 sensor array and b) UiO-66 sensor array for 4 cycles of adsorption at 1.48% CO_2 concentration. Comparison of frequency shifts observed at the end of exposure for the different cycles for the HKUST-1 sensors (c) and the UiO-66 sensors (d). The average standard deviation for the four cycles and all samples is 0.52 Hz.

Table S3: Time constants of a mono-exponential fit for the uptake and for the release of CO_2 with a concentration of 1.97%. The average time constants are 1.3±0.4 min for the uptake and 6.2±0.3 min for the release. The response time $t_{response}$ and recovery time $t_{recovery}$ (which means, the time until 99% of the final signal are reached) can be calculated from the time constants $\tau_{uptake/release}$ of the mono-exponential fits by $t_{response} = 5\times \tau_{uptake}$ and $t_{recovery} = 5\times \tau_{release}$. As result, the response time of the sensor array is approximately 6min and the recovery time is 30min. It should be stated that this time response is significantly slower than the response of other CO_2 sensors.[6-8] For practical applications, the sensor response time needs to be improved, for instance by minimizing the defect density in the MOF material[9] and by decreasing the MOF film thickness (without significantly decreasing the sensitivity). After improving the time response, we believe such IL@MOF materials can also be applied in dynamic CO_2 monitoring.

Remarkably, the sensor array provides already a precise classification of the analytes before the sensor response time, see Figure 6 and the discussion.

Sample name	type	uptake time constant / min	release time constant / min
S1	HKUST-1 (empty)	0.64	5.86
S2	$[BMIM][TFSI]_{20\%}$@HKUST-1	1.45	6.03
S3	$[BMIM][TFSI]_{50\%}$@HKUST-1	1.52	6.06
S4	$[BMIM][SCN]_{20\%}$@HKUST-1	0.87	5.96
S5	$[BMIM][SCN]_{50\%}$@HKUST-1	1.01	6.07
S6	$[BMIM][PF_6]_{20\%}$@HKUST-1	1.82	6.11
S7	$[BMIM][PF_6]_{20\%}$@HKUST-1	1.84	6.66
S8	UiO-66 (empty)	0.76	6.11
S9	$[BMIM][TFSI]_{20\%}$@UiO-66	1.83	6.28
S10	$[BMIM][TFSI]_{50\%}$@UiO-66	2.00	6.38
S11	$[BMIM][SCN]_{20\%}$@UiO-66	1.07	6.34
S12	$[BMIM][SCN]_{50\%}$@UiO-66	1.12	6.61
S13	$[BMIM][PF_6]_{20\%}$@UiO-66	1.29	6.51
S14	$[BMIM][PF_6]_{50\%}$@UiO-66	1.45	6.56

Neural network code

```
import torch
import pandas as pd
import os
import numpy as np
filepath = "../dataset/…/"
Data = []
for file in os.listdir(filepath):
    if (not file.startswith('~')):
        data = pd.read_excel(filepath+file)
        Data.append(data)
gas=[]
for i in range(len(Data)):
    gas.append(os.listdir(filepath)[i][0:4])

gas

def data_preprocessing(Data, Col, start, end):
    New_Data=[]
    for i in range(len(Data)):
        data = Data[i]
        new_data = data.iloc[start: end]
        new_data = new_data[Col[i]]
        New_Data.append(new_data)
    return New_Data


def choose_points(data,mode,zone,num_of_points):
    time = data['Time']
    matrix_mx3 = data[data.columns[1:]].values
    if mode==1:
        index = np.sum(time<zone)
        return matrix_mx3[max(0,index-num_of_points):index]
    elif mode==2:
        index1 = np.sum(time<zone[0])
        index2 = np.sum(time<zone[1])
        return matrix_mx3[index1:index2]

def plot_curve(Data,zone,num_of_points,Columns,sort=0,rolling=False,w = 5,mode=1):
    D = []
    length = len(Data[0].columns)-1
    if type(zone)!=list:
        zone = [zone]*len(Data)
    plt.figure(figsize=(15,6*len(Data)))
    for i in range(len(Data)):
        data = Data[i]
        time = data['Time'].values
        if mode==1:
            time = time[time<zone[i]]
            time = time[-num_of_points:]
        elif mode==2:
            time = time[time<zone[-1]]
            time = time[time>zone[0]]
        d = pd.DataFrame(columns=data.columns)
        d['Time']=time[(w-1)*int(rolling):]
        if rolling:
            points = choose_points(data,mode,zone[i],num_of_points-w+1)
        else:
            points = choose_points(data,mode,zone[i],num_of_points)
        for column in range(length):
            d.iloc[:,column+1]=points[:,column]
        D.append(d)
        if sort!=0:
            for column in range(points.shape[1]):
                points[:,column] = points[:,column][np.argsort(points[:,column])]
        plt.subplot(len(Data),1,i+1)
        for j in range(length):
            plt.plot(time[(w-1)*int(rolling):],points[:,j],'x',label = Columns[i][j+1])
        plt.legend()
        plt.title(gas[i])
    return D

    col1 = ['Time','c5','c12']
    col2 = ['Time','c5','c12']
    col3 = ['Time','c5','c12']
    col4 = ['Time','c5','c12']
```

```
col5 = ['Time','c5','c12']
col6 = ['Time','c5','c12']

Col = [col1,col2,col3,col4,col5,col6]
Processed_Data = data_preprocessing(Data, Col, 3, 100)

learning_rate = 1e-4
batch_size = 64
epochs = 1000

def data_to_tensor(data):
    X = []
    Y = []
    for i in range(len(data)):
        val = data[i].values[:, 1:].astype(float)
        X.append(val)
        Y.append([i] * val.shape[0])
    Y = np.array(Y).reshape(-1)
    X = np.array(X).reshape(Y.shape[0], -1)
    return torch.FloatTensor(X), torch.LongTensor(Y), len(data)

def confusion_matrix(pred, Y, n):
    ret = np.zeros((n, n))
    for i in range(pred.shape[0]):
        ret[int(Y[i]), int(pred[i])] += 1
    return pd.DataFrame(ret.astype(int))


X, Y, n = data_to_tensor(Processed_Data)
# TODO: Normalize
Z = torch.zeros(Y.shape[0],6)
Z[Y==0]=torch.FloatTensor([1.97,1.48,0.98,0.49,0.19,0.09])
Z[Y==1]=torch.FloatTensor([1.97,1.48,0.98,0.49,0.19,0.09])
Z[Y==2]=torch.FloatTensor([1.97,1.48,0.98,0.49,0.19,0.09])
Z[Y==3]=torch.FloatTensor([1.97,1.48,0.98,0.49,0.19,0.09])
Z[Y==4]=torch.FloatTensor([1.97,1.48,0.98,0.49,0.19,0.09])
Z[Y==5]=torch.FloatTensor([1.97,1.48,0.98,0.49,0.19,0.09])
Z[Y==6]=torch.FloatTensor([1.97,1.48,0.98,0.49,0.19,0.09])
Z[Y==7]=torch.FloatTensor([1.97,1.48,0.98,0.49,0.19,0.09])
Z[Y==8]=torch.FloatTensor([1.97,1.48,0.98,0.49,0.19,0.09])
Z[Y==9]=torch.FloatTensor([1.97,1.48,0.98,0.49,0.19,0.09])
Z[Y==10]=torch.FloatTensor([1.97,1.48,0.98,0.49,0.19,0.09])
Z[Y==11]=torch.FloatTensor([1.97,1.48,0.98,0.49,0.19,0.09])
Z[Y==12]=torch.FloatTensor([1.97,1.48,0.98,0.49,0.19,0.09])
Z[Y==13]=torch.FloatTensor([1.97,1.48,0.98,0.49,0.19,0.09])
Z[Y==14]=torch.FloatTensor([1.97,1.48,0.98,0.49,0.19,0.09])
Z[Y==15]=torch.FloatTensor([1.97,1.48,0.98,0.49,0.19,0.09])
Z[Y==16]=torch.FloatTensor([1.97,1.48,0.98,0.49,0.19,0.09])
Z[Y==17]=torch.FloatTensor([1.97,1.48,0.98,0.49,0.19,0.09])
# shuffle
indices = torch.randperm(X.shape[0])
train_indices, test_indices = indices[ : int(0.9 * X.shape[0])], indices[int(0.9 * X.shape[0]):]
X_train = X[train_indices]
Y_train = Y[train_indices]
X_test = X[test_indices]
Y_test = Y[test_indices]

Z_train = torch.zeros(Y_train.shape[0],6)
Z_train[Y_train==0]=torch.FloatTensor([1.97,1.48,0.98,0.49,0.19,0.09])
Z_train[Y_train==1]=torch.FloatTensor([1.97,1.48,0.98,0.49,0.19,0.09])
Z_train[Y_train==2]=torch.FloatTensor([1.97,1.48,0.98,0.49,0.19,0.09])
Z_train[Y_train==3]=torch.FloatTensor([1.97,1.48,0.98,0.49,0.19,0.09])
Z_train[Y_train==4]=torch.FloatTensor([1.97,1.48,0.98,0.49,0.19,0.09])
Z_train[Y_train==5]=torch.FloatTensor([1.97,1.48,0.98,0.49,0.19,0.09])
Z_train[Y_train==6]=torch.FloatTensor([1.97,1.48,0.98,0.49,0.19,0.09])
Z_train[Y_train==7]=torch.FloatTensor([1.97,1.48,0.98,0.49,0.19,0.09])
Z_train[Y_train==8]=torch.FloatTensor([1.97,1.48,0.98,0.49,0.19,0.09])
Z_train[Y_train==9]=torch.FloatTensor([1.97,1.48,0.98,0.49,0.19,0.09])
Z_train[Y_train==10]=torch.FloatTensor([1.97,1.48,0.98,0.49,0.19,0.09])
Z_train[Y_train==11]=torch.FloatTensor([1.97,1.48,0.98,0.49,0.19,0.09])
Z_train[Y_train==12]=torch.FloatTensor([1.97,1.48,0.98,0.49,0.19,0.09])
Z_train[Y_train==13]=torch.FloatTensor([1.97,1.48,0.98,0.49,0.19,0.09])
Z_train[Y_train==14]=torch.FloatTensor([1.97,1.48,0.98,0.49,0.19,0.09])
Z_train[Y_train==15]=torch.FloatTensor([1.97,1.48,0.98,0.49,0.19,0.09])
Z_train[Y_train==16]=torch.FloatTensor([1.97,1.48,0.98,0.49,0.19,0.09])
Z_train[Y_train==17]=torch.FloatTensor([1.97,1.48,0.98,0.49,0.19,0.09])
```

```
import torch.nn as nn

net = nn.Sequential(
    nn.Linear(X.shape[-1], 128),
    nn.ReLU(),
    nn.Linear(128, 128),
    nn.ReLU(),
    nn.Linear(128, 128),
    nn.ReLU(),
    nn.Linear(128, 128),
    nn.ReLU(),
    nn.Linear(128, n),
)

loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(net.parameters(), lr=learning_rate)

for e in range(epochs):
    print(f"---Epoch {e}---")
    for batch in range(X_train.shape[0] // batch_size):
        x = X_train[batch * batch_size: (batch + 1) * batch_size]
        y = Y_train[batch * batch_size: (batch + 1) * batch_size]
        pred = net(x)
        loss = loss_fn(pred, y)

        # Backpropagation
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        if batch % 5 == 0:
            loss = loss.item()
            print(f"batch {batch}, loss: {loss:>7f}")
with torch.no_grad():
    pred = torch.argmax(net(X_train), dim = -1)
tmp = confusion_matrix(pred, Y_train, n)
tmp.to_excel("1.xlsx")

with torch.no_grad():
    pred = torch.argmax(net(X_test), dim = -1)
confusion_matrix(pred, Y_test, n)

net = nn.Sequential(
    nn.Linear(X.shape[-1], 128),
    nn.ReLU(),
    nn.Linear(128, 128),
    nn.ReLU(),
    nn.Linear(128, 6),
)
loss_fn = nn.MSELoss()
optimizer = torch.optim.SGD(net.parameters(), lr=learning_rate)

for e in range(epochs):
    print(f"---Epoch {e}---")
    for batch in range(X_train.shape[0] // batch_size):
        x = X_train[batch * batch_size: (batch + 1) * batch_size]
        z = Z_train[batch * batch_size: (batch + 1) * batch_size].view(-1, 6).float()
        pred = net(x)
        #print(x.shape,pred.shape,z.shape)
        loss = loss_fn(pred, z)

        # Backpropagation
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        if batch % 5 == 0:
            loss = loss.item()
            print(f"batch {batch}, loss: {loss:>7f}")

import matplotlib.pyplot as plt

plt.scatter(net(X_test)[:100,0].detach().numpy(), Z_test[:100,0].numpy())
df = pd.DataFrame(np.concatenate([net(X_test)[:100,0].detach().numpy().reshape(-1,1), Z_test[:100,0].numpy().reshape(-
1,1)], axis = -1))
df.to_excel("2.xlsx")
```