Supplementary material

Identification of H_2 and NH_3 gases using calorimetric signals and transient response through machine learning

Wenxin Luo[1], Yingcong Zheng[1], Yijun Liu[1, †], and Mingjie Li[1, †]

[1]School of Integrated Circuits, Guangdong University of Technology, Guangzhou 510006, China

[†]Email: yjliu@gdut.edu.cn or limj@gdut.edu.cn

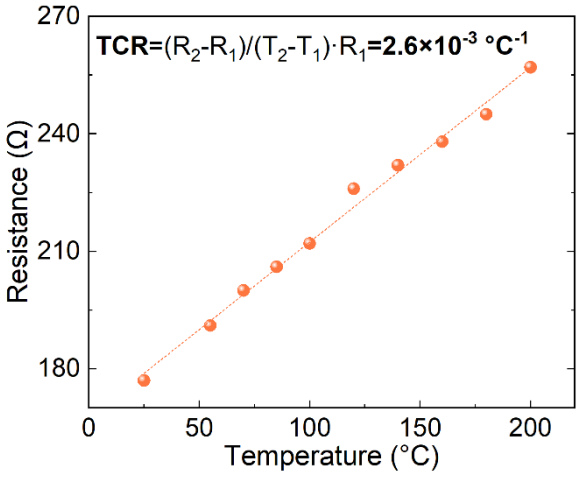

Figure S1. Electrical resistance versus temperature of the microheater.

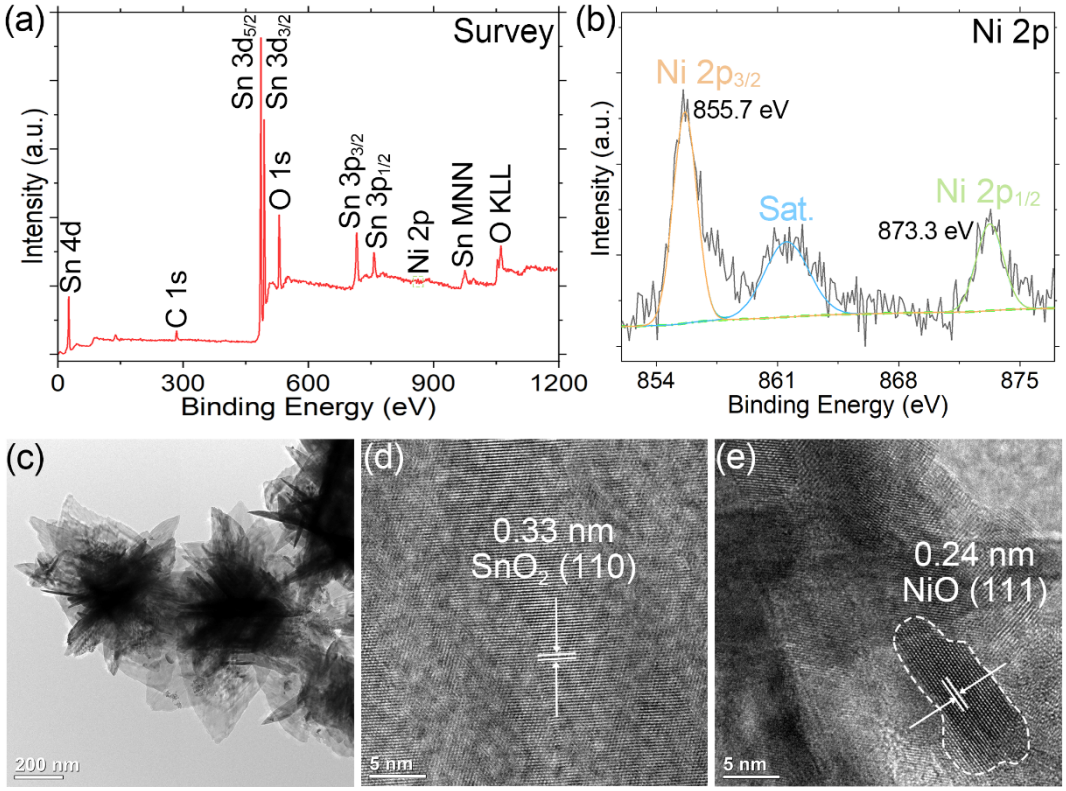

Figure S2. (a) XPS survey spectra of the NiO-SnO_2 nanosheets, and (b) the corresponding core level Ni 2p spectra. (c-e) HRTEM images of NiO-SnO_2 nanosheets.

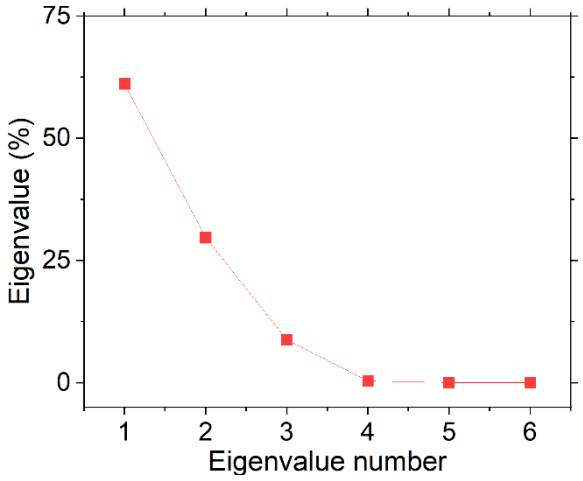

Figure S3. Scree plots showing the cumulative variance explained by the principal components.

- **Extracting sensing feature parameters**

When a gas sensor is exposed to target gas, its response is influenced by various factors, including the type of adsorbate, the mechanisms of physisorption and/or chemisorption involved in the adsorption process, the gas concentration, and the exposure duration. The transient response of a gas sensor can, therefore, be characterized by multiple parameters. To analyze the response curves, several key characteristics were extracted. The normalized dynamic response is defined by the following ratio [1]:

$$S_n(t) = \frac{i(t) - i(0)}{i(T) - i(0)} \quad (1)$$

where *i(0)* and *i(t)* represent the current (i.e., either MOS or ΔT current) at the initial time of gas injection and at time *t*, respectively. Also, eight specific response points were selected at regular time intervals: *T/8, T/4, 3T/8, T/2, 5T/8, 3T/4, 7T/8* and *T*, where *T*

denotes the total measurement time. To extract a characteristic parameter from this dynamic curve, we were inspired from the fill factor concept of photovoltaic cells [2]. A curve was constructed to describe the evolution of the product $(T\text{-}t)\cdot T\cdot S_n$ as a function of time. The maximum point on this curve, located at t_{max} enables the mathematical definition of the curve's characteristic R_c:

$$R_c = \frac{(1+(dy/dx)^2)^{3/2}}{d^2y/dx^2} \tag{2}$$

Further, the dynamic response can be parameterized by its length, denoted as C_{abs}. This parameter corresponds to the cumulative sum of the curves derived from the eight defined response points (from *T/8 to T*), representing the electrical or calorimetric signals over a specific range from the starting point.

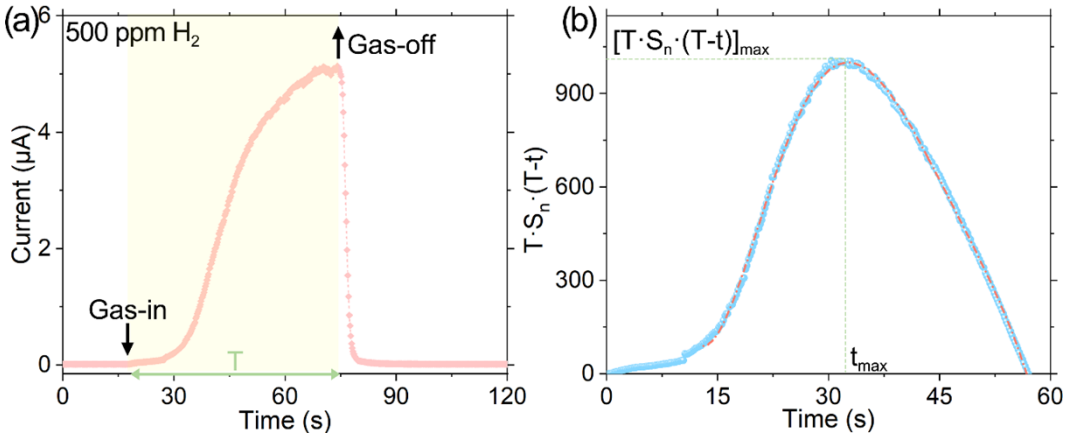

Figure S4. (a) Transient current response to 500 ppm H_2, and (b) the corresponding Fill Factor resolve of product $(T\text{-}t)\cdot T\cdot S_n$ as a function of time.

- **Data set for PCA and ML algorithm**

Table S1. Extracted gas sensing features for ML algorithm.

Gas (ppm)		Features					
H_2	NH_3	S	ΔT	$C_{abs(S)}$	$C_{abs(\Delta T)}$	$R_{c(S)}$	$R_{c(\Delta T)}$
50	0	17.79	4.82	30.22	67.40	10.41	115.06
100	0	40.89	6.84	38.00	71.51	22.34	120.42
150	0	71.55	8.96	42.20	75.86	29.24	122.31
200	0	101.03	11.14	45.81	80.52	124.00	126.26
300	0	175.63	15.81	48.80	89.72	139.76	145.37
500	0	310.61	26.34	51.62	107.82	292.62	191.45
0	50	4.08	2.81	66.62	80.83	20.92	72.95
0	100	6.07	2.82	67.02	81.43	23.37	106.37
0	150	23.80	2.83	68.62	81.83	27.53	141.41
0	200	46.20	2.84	68.82	82.03	32.34	391.06
0	300	68.88	2.84	72.23	82.63	37.04	530.74
0	500	108.10	2.85	75.03	83.03	41.20	803.36

Table S2. Specific ML algorithm parameters for various model construction.

Model	ML algorithm parameters						
PCA	**n_components**	**copy**	**whiten**	**svd_solver**	**tol**	**iterated_power**	**n_oversamples**
	2	True	False	auto	0	auto	10
KNN	**n_neighbors**	**weights**	**algorithm**	**leaf_size**	**p**	**metric**	**metric_params**
	3	distance	auto	30	2	manhattan	None
SVM	**C**	**kernel**	**degree**	**gamma**	**coef0**	**shrinking**	**probability**
	0.3	rbf	3	scale	0	True	True
RF	**n_estimators**	**criterior**	**max_depth**	**min_samples_leaf**	**max_features**	**max_leaf_nodes**	**n_jobs**
	100	gini	3	2	sqrt	None	-1

- **Code for ML algorithm with PCA preprocessing**

- **Table S3.** The code for KNN classifier.

Algorithm code

Input: Raw data of initial array data

Process:

```
1: X = data.iloc[:, :-1]
2: y = data.iloc[:, -1]
3: X_train, X_test, y_train, y_test = train_test_split(X,y,
4: test_size=0.3,random_state=42)
5: y_train_origin = y_train.reset_index(drop=True)
6: scaler = StandardScaler()
4: X_train_scaled  = scaler.fit_transform(X_train)
5: X_test_scaled = scaler.fit_transform(X)
6: pca = PCA(n_components=2)
7: X_train_pca = pca.fit_transform(X_train_scaled)
8: X_test_pca = pca.fit_transform(X_test_scaled)
9: k = 4
10: kf = KFold(n_splits=k, shuffle=True, random_state=32)
11: y_pred_all = []
12: y_val_all = []
13: knn = KNeighborsClassifier(
14: n_neighbors=3,
15: weights='distance',
12: metric='  manhattan',
13: p=2)
14: for train_index, test_index in kf.split(X_train_pca,y_train_origin)
15: X_train, X_val = X_train_pca[train_index], X_train_pca[test_index]
16: y_train, y_val = y_train_origin[train_index], y_train_origin[test_index]
17: knn.fit(X_train, y_train)
18: y_pred = knn.predict(X_val)
19: y_pred_all.extend(y_pred)
20: y_val_all.extend(y_val)
21: y_pred = knn.predict(X_test_pca)
22: accuracy = accuracy_score(y, y_pred)
23: print(f'The final model accuracy: {accuracy:.2f}')
24: plt.rcParams['axes.unicode_minus'] = False
25: plt.figure(figsize=(8, 6))
26: cm = confusion_matrix(y, y_pred)
27: sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
28: plt.title('Confusion matrix')
29: plt.xlabel('Predicted')
30: plt.ylabel('Observed')
```

Output:

```
print(f'The final model accuracy: {accuracy:.2f}')
plt.show()
```

Table S4. The code for SVM classifier.

Algorithm code

Input: Raw data of initial array data

Process:

```
X = data.iloc[:, :-1]
y = data.iloc[:, -1]
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)
k = 4
kf = KFold(n_splits=k, shuffle=True, random_state=26)
y_pred_all = []
y_test_all = []
for train_index, test_index in kf.split(X_pca):
X_train, X_test = X_pca[train_index], X_pca[test_index]
y_train, y_test = y[train_index], y[test_index]
svm = SVC(
C=0.3 ,
kernel='rbf',
probability=True)
svm.fit(X_train, y_train)
y_pred = svm.predict(X_test)
y_pred_all.extend(y_pred)
y_test_all.extend(y_test)
y_pred = svm.predict(X_pca)
accuracy = accuracy_score(y, y_pred)
plt.rcParams['axes.unicode_minus'] = False
plt.figure(figsize=(8, 6))
cm = confusion_matrix(y, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion matrix')
plt.xlabel('Predicted')
plt.ylabel('Observed')
```

Output:

```
print(f'Model accuracy: {accuracy:.2f}')
plt.show()
```

Table S5. The code for RF classifier.

Algorithm code

Input: Raw data of initial array data

Process:

```
X = data.iloc[:, :-1]
y = data.iloc[:, -1]
X_train, X_test, y_train, y_test = train_test_split(X,y,
test_size=0.3,random_state=42)
y_train_origin = y_train.reset_index(drop=True)
scaler = StandardScaler()
X_train_scaled  = scaler.fit_transform(X_train)
X_test_scaled = scaler.fit_transform(X)
pca = PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.fit_transform(X_test_scaled)
k = 4
kf = KFold(n_splits=k, shuffle=True, random_state=6)
y_pred_all = []
y_val_all = []
forest = RandomForestClassifier(n_estimators=100,
random_state=11,
n_jobs=-1,
max_depth=3,
min_samples_leaf = 2)
for train_index, test_index in kf.split(X_train_pca,y_train_origin)
X_train, X_val = X_train_pca[train_index], X_train_pca[test_index]
y_train, y_val = y_train_origin[train_index], y_train_origin[test_index]
knn.fit(X_train, y_train)
y_pred = knn.predict(X_val)
y_pred_all.extend(y_pred)
y_val_all.extend(y_val)
y_pred = knn.predict(X_test_pca)
plt.figure(figsize=(8, 6))
cm = confusion_matrix(y, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion matrix')
plt.xlabel('Predicted')
plt.ylabel('Observed')
```

Output:

```
print(f'The final model accuracy: {accuracy:.2f}')
plt.show()
```

References

[1] Bouricha B.; Souissi R.; Bouguila N., et al., A real-time sharp selectivity with in_2s_3 gas sensor using a nonlinear dynamic response for vocs. Measurement, 2021, 185, 110070
[2] Kim M.-S.; Kim B.-G.; Kim J., Effective variables to control the fill factor of organic photovoltaic cells. ACS Appl. Mater. Interfaces, 2009, 1(6), 1264-1269