Supplementary Information

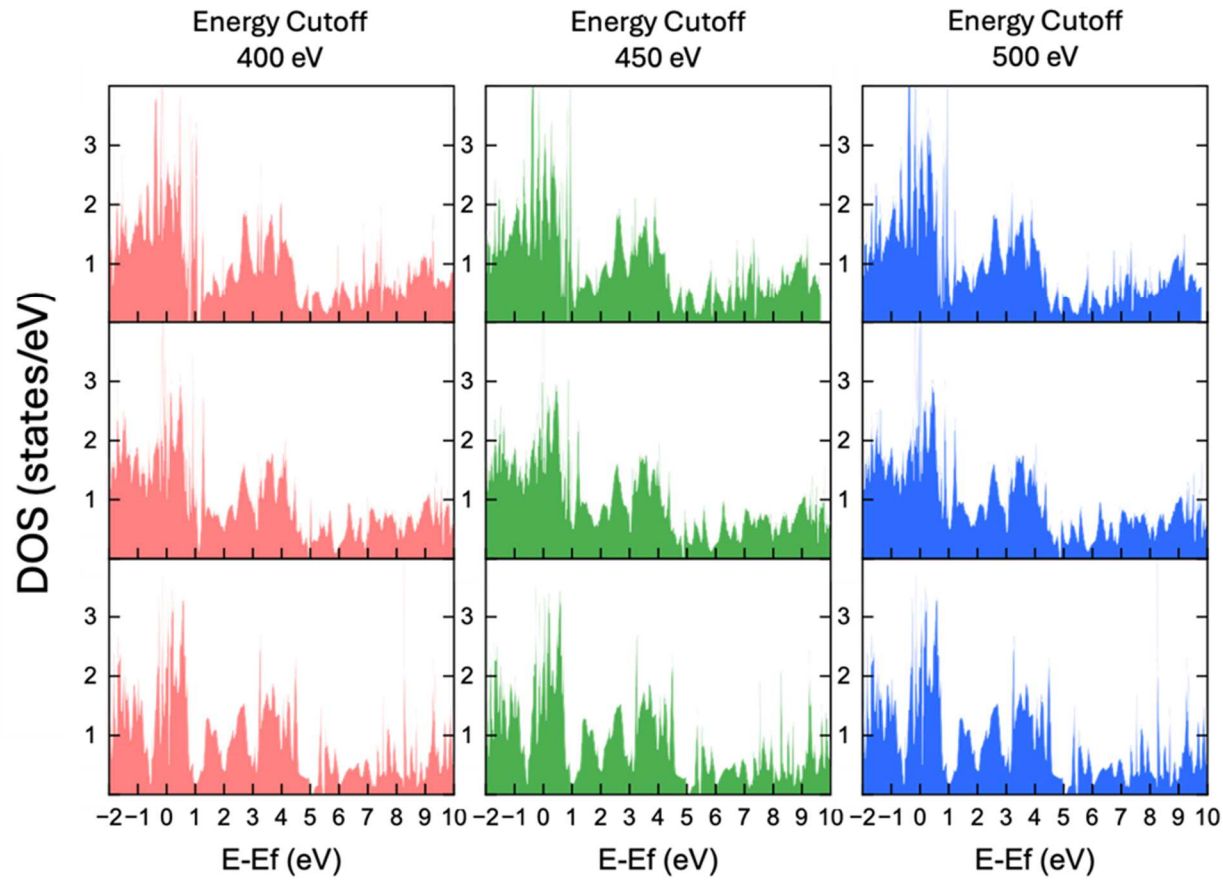

Figure S1. Comparison of the density of states calculated with plane-wave cutoff energies of 400, 450, and 500 eV for slab models with oxygen vacancy concentrations of 0%, 5%, and 10% (from bottom to top)

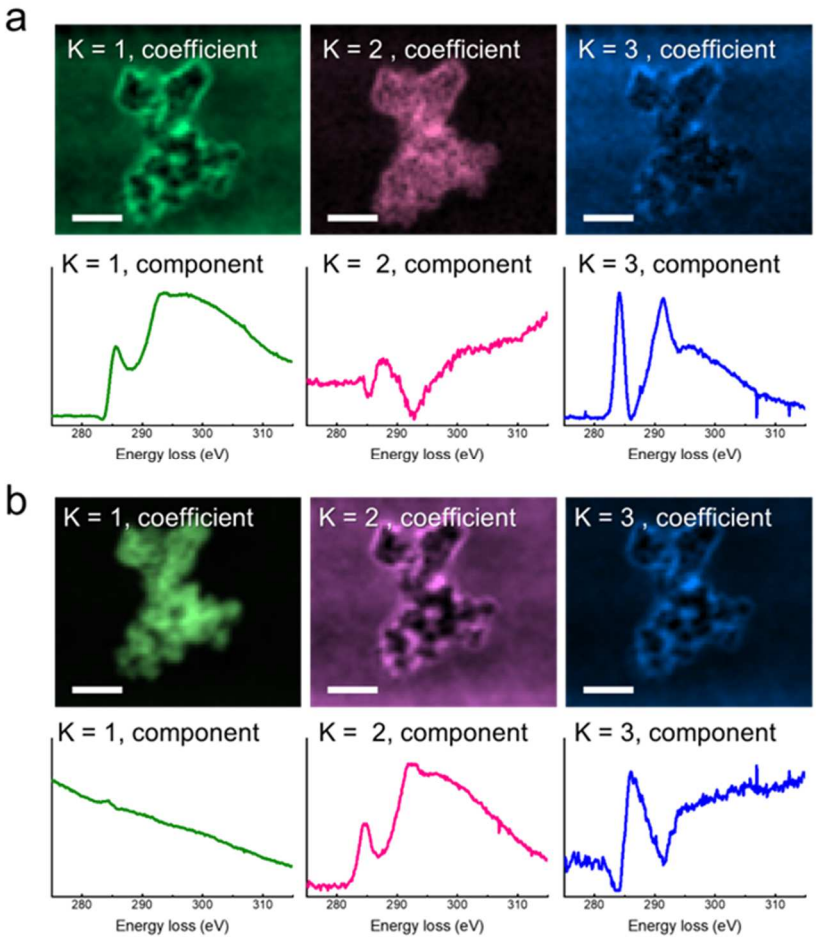

Figure S2. Effect of power-law background subtraction of the C *K* EELS SI in NMF-driven decomposition and clustering. (a, b) Comparison of clustering results for the C *K* EELS SI data before and after background subtraction.

Note S1: The clustering results of NMF decomposition for the background-subtracted C *K* EELS SI showed that the morphological distributions of the spectral feature components were nonphysical, because the first- and second-ranked coefficient maps (K = 1 and 2) indicated the spatial distributions of the deconvoluted spectral feature components for the ionomer morphology, and the greatest variance regarding the overall shape of the particle morphology was lost. These morphologically irrelevant classifications concerning the ionomer component make reliably evaluating the spatial distribution of the ionomer over the IrO_{2-x} particles difficult. However, the NMF decomposition and clustering for the C *K* EELS SI data without background subtraction exhibited contextually meaningful performance for the spatial separation of the three major feature components: overall particle shape (K = 1), graphene overlayer (K = 2), and ionomer (K = 3), in order of their total spatial abundances.

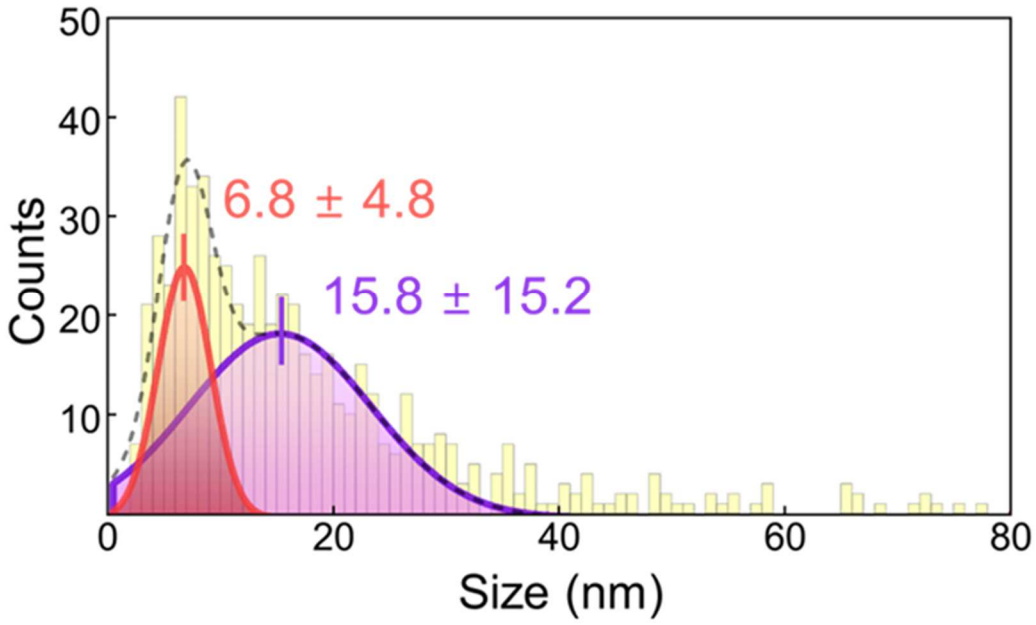

Figure S3. Bimodal distribution of the IrO_{2-x} particle sizes measured from five HAADF STEM images of the sample.

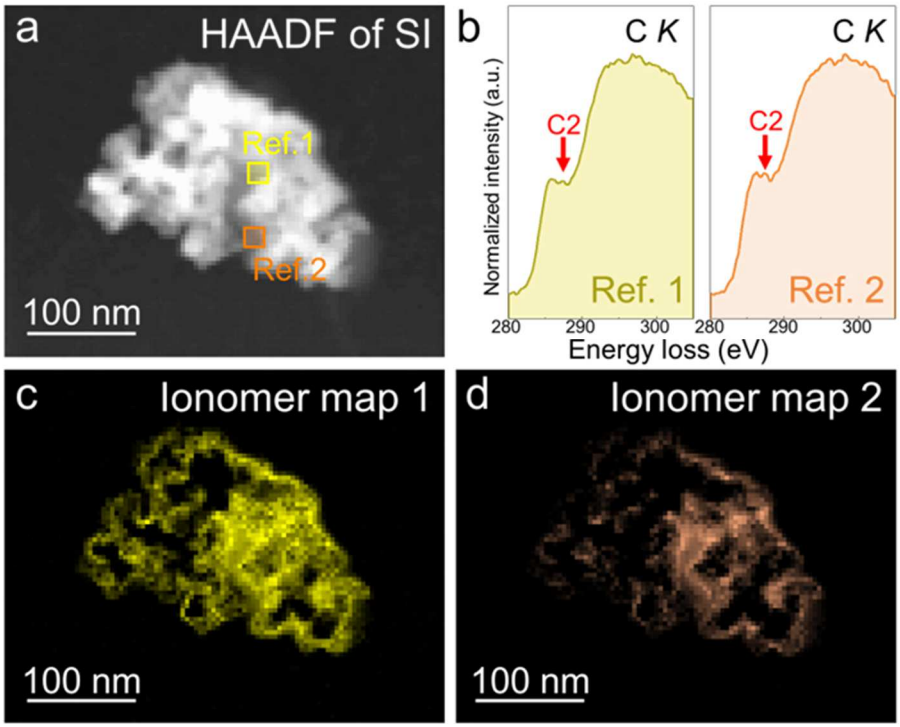

Figure S4. Dependence of the chosen references on MLLS fitting results of C *K* EELS SI data. (a) HAADF image of the sample for the EELS SI data acquisition. (b) Reference C *K* spectra obtained from the two different locations (Ref. 1 and Ref. 2) to acquire a fit coefficient map visualizing the ionomer distribution on the IrO_{2-x} particles. (c, d) Fit coefficient maps for the ionomer distribution obtained after the MLLS fitting using the two different references. This result indicates that the MLLS fits with an arbitrary reference component, which could result in an unreliable evaluation of the ionomer distribution.

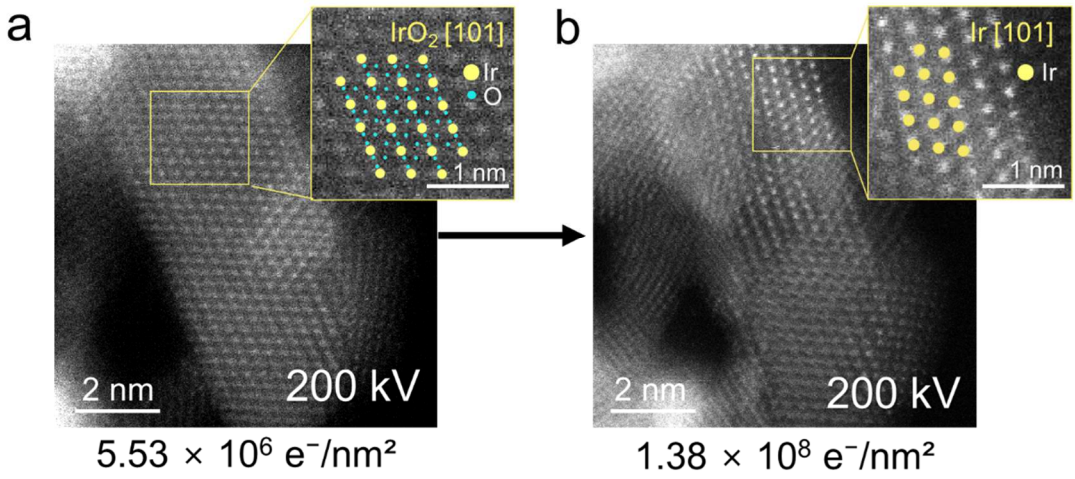

Figure S5. HAADF STEM images consecutively acquired as a function of electron dose, which show the electron-beam-induced phase transition of (a) IrO_{2-x} to (b) Ir metal.

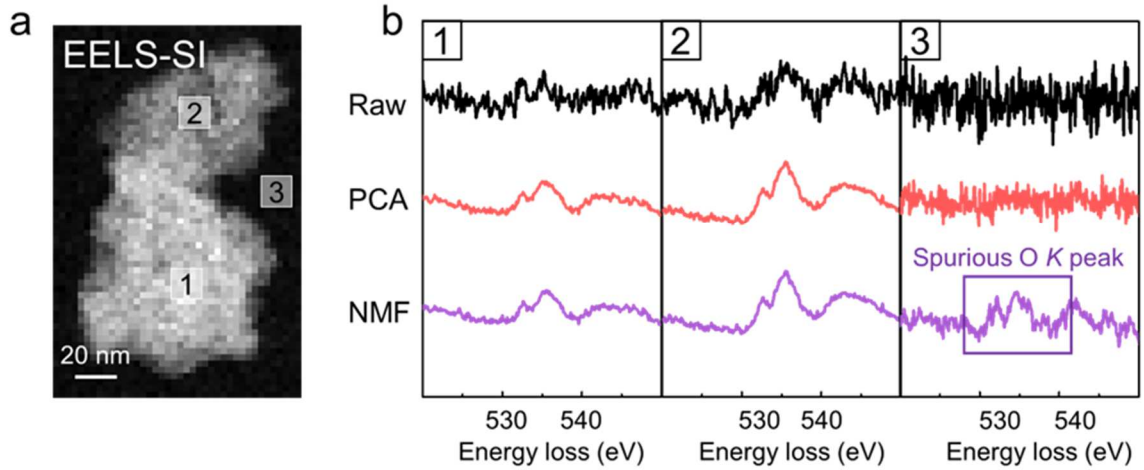

Figure S6. Comparison of noise-removal performance of PCA and NMF algorithms. (a) ADF image of the EELS SI data of the IrO_{2-x} particles. (b) Comparison of O *K*-edges for the three selected areas (no. 1–3) before and after the removal of noise components ($k \geq 4$) using PCA and NMF. Note that a spurious O *K* peak is generated in region 3 after the NMF, which does not include IrO_{2-x} particles.

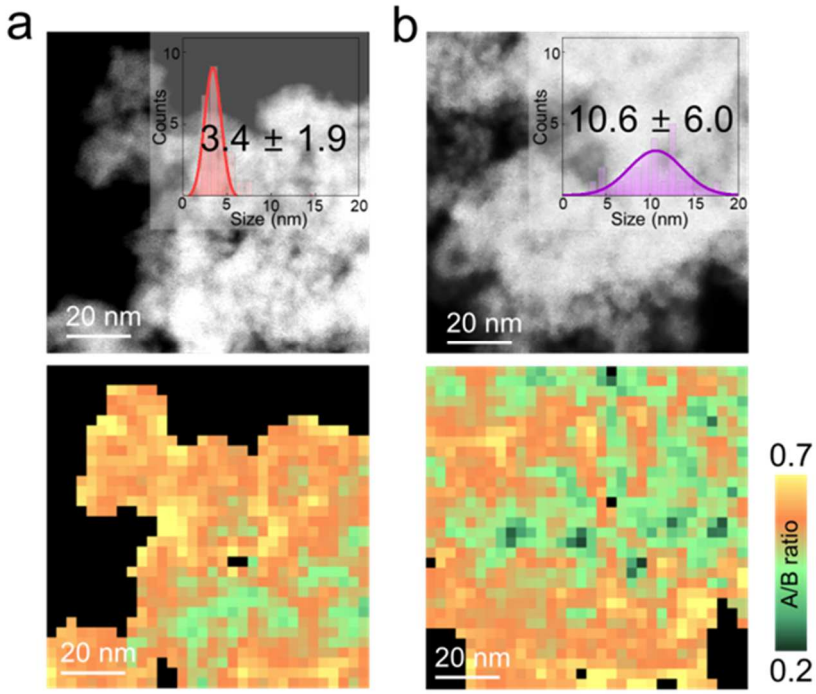

Figure S7. Particle-size dependence of the A/B ratio values of O *K*-edges. (top) HAADF images of (a) small and (b) large IrO_{2-x} particles with size distributions of 3.4 ± 1.9 nm and 10.6 ± 6.0 nm, respectively. (bottom) A/B ratio maps corresponding to the two imaging areas. The measured value of the A/B ratio (5.8 ± 0.1) in small particles was estimated to be larger than that (5.3 ± 0.1) in large particles, suggesting that oxygen vacancies are more concentrated in smaller IrO_{2-x} particles with sizes of < 5 nm.

Appendix. Source code scripts

This appendix describes the core analysis functions used for STEM-EELS signal preprocessing (Script S1), C *K*-edge phase analysis (Script S2), and O *K*-edge oxygen vacancy quantification (Script S3). Installation, package dependencies, and command-line usage are documented in the repository at https://github.com/SKKU-STEM/ESICharWE/

Script S1. preprocessing.py — Signal Preprocessing

Four preprocessing functions are applied sequentially to the raw spectrum-image array of shape (H, W, E), where H and W are the spatial dimensions, and E is the number of energy channels.

S1-1. preprocess_signal

Absolute-value correction removes artefactual negative intensities. The energy axis is cropped to the target range via HyperSpy's isig slicer. Stride-based spatial binning sums overlapping k×k patches, reducing spatial dimensions to (H − k + 1, W − k + 1) while improving SNR by a factor of k^2.

```
import numpy as np


def preprocess_signal(sig, energy_min=None, energy_max=None,
                      normalize=False, stride_binning=None):
    sig.data = np.abs(sig.data)


    if energy_min is not None and energy_max is not None:
        sig = sig.isig[float(energy_min):float(energy_max)]


    if stride_binning is not None:
        data = sig.data
        H, W, E = data.shape
        k = stride_binning
        new_H, new_W = H - k + 1, W - k + 1
        binned = np.zeros((new_H, new_W, E), dtype=data.dtype)
        for i in range(new_H):
```

```
            for j in range(new_W):
                binned[i, j] = data[i:i+k, j:j+k, :].sum(axis=(0, 1))
        sig = sig._deepcopy_with_new_data(binned)
        sig.axes_manager[0].size = new_W
        sig.axes_manager[1].size = new_H


    if normalize:
        sig.data /= np.max(sig.data, axis=-1, keepdims=True)
    return sig
```

S1-2. denoise_signal — PCA reconstruction

The spectrum image is reshaped to a 2-D matrix (H·W × E), decomposed by PCA, and reconstructed from the leading n_components principal components. This suppresses high-frequency noise while preserving spectral fine structure. For O K edge data, n_components = 4 was fixed based on scree-plot inspection.

```
from sklearn.decomposition import PCA


def denoise_signal(sig, method='PCA', n_components=3):
    h, w, e = sig.data.shape
    flat    = sig.data.reshape(-1, e)
    if method != 'PCA':
        raise ValueError(f"Unsupported method: '{method}'. Use 'PCA'.")
    model = PCA(n_components=n_components)
    comp  = model.fit_transform(flat)
    recon = model.inverse_transform(comp)
    return sig._deepcopy_with_new_data(recon.reshape(h, w, e))
```

S1-3. smooth_signal — Savitzky–Golay filter

A Savitzky–Golay filter (window length = 21 channels, polynomial order = 2) is applied along the energy axis to reduce residual noise after PCA reconstruction.

```
from scipy.signal import savgol_filter
```

```
def smooth_signal(sig, window_length=21, polyorder=2):
    if sig.data.ndim != 3:
        raise ValueError('Signal data must be 3-dimensional (H, W, E).')
    smoothed = savgol_filter(sig.data, window_length, polyorder, axis=-1)
    return sig._deepcopy_with_new_data(smoothed)
```

S1-4. subtract_background_signal — power-law background removal

A power-law model $I(E) = A \cdot E^{-r}$ is fitted independently at each pixel to the pre-edge energy window [500, 528 eV] using scipy.optimize.curve_fit. The fitted background is subtracted from the full spectrum.

```
from scipy.optimize import curve_fit
from tqdm import tqdm


def subtract_background_signal(sig, fit_range=(500, 528)):
    def _power_law(x, A, r):
        return A * np.power(x, -r)


    energy_axis = sig.axes_manager[-1].axis
    h, w, _     = sig.data.shape
    corrected   = np.empty_like(sig.data)
    mask        = (energy_axis >= fit_range[0]) & (energy_axis <= fit_range[1])
    x_fit       = energy_axis[mask]


    for i in tqdm(range(h), desc='Background subtraction'):
        for j in range(w):
            y     = sig.data[i, j, :]
            y_fit = np.maximum(y[mask], 1e-6)
            try:
                popt, _ = curve_fit(_power_law, x_fit, y_fit,
                                    p0=(1., 2.), maxfev=5000)
```

```
                corrected[i, j] = y - _power_law(energy_axis, *popt)
            except RuntimeError:
                corrected[i, j] = y
    return sig._deepcopy_with_new_data(corrected)
```

Script S2. analysis_ck.py — C *K*-edge Analysis

C *K*-edge spectra (270–330 eV, stride-2 spatial binning) are decomposed by non-negative matrix factorization (NMF) with NNDSVD initialization. The NMF loading matrix W (H·W × n) is clustered by K-means to assign each spatial pixel to a phase. Pixels with integrated intensity below the spatial mean are masked as background.

```
import numpy as np
import hyperspy.api as hs
from sklearn.decomposition import NMF
from sklearn.cluster import KMeans
from preprocessing import preprocess_signal


def perform_ck_analysis(data_sig, n_components):
    EELS_sig = preprocess_signal(data_sig.copy(),
                    energy_min=270., energy_max=330.,
                    normalize=False, stride_binning=2)


    # --- NMF decomposition ---
    h, w, c   = EELS_sig.data.shape
    data_flat = EELS_sig.data.reshape(-1, c)
    data_flat[data_flat < 0] = 0
    nmf = NMF(n_components=n_components, init='nndsvd', max_iter=20000)
    W   = nmf.fit_transform(data_flat)   # loading matrix  (H*W, n)
    H   = nmf.components_                # spectral basis   (n, E)


    # --- K-means clustering on loading vectors ---
    km     = KMeans(n_clusters=n_components, n_init='auto', max_iter=1000)
```

```
    labels = km.fit_predict(W).reshape(h, w)


    # --- Background mask ---
    intensity_map = EELS_sig.data.sum(axis=2)
    particle_mask = (intensity_map >= intensity_map.mean()).astype(int)
    label_map     = labels * particle_mask


    return W, H, label_map
```

Script S3. analysis_ok.py — O *K*-edge Analysis

O *K*-edge spectra (500–560 eV) are preprocessed with stride-2 spatial binning, PCA denoising (4 components), Savitzky–Golay smoothing (window = 21, order = 2), and pixel-wise power-law background subtraction. The B peak is identified as the local maximum nearest to channel 340 in the spatially summed spectrum; the A peak is located 30 channels prior. Per-pixel peak positions are refined within search windows of ±5 (A) and ±10 (B) channels. Oxygen vacancy concentration is calibrated as V_o (%) = [I(A)/I(B) − 0.492] / 0.017, clipped at zero. IrO_{2-x} particle pixels are isolated by 3-class K-means on integrated intensity.

```
import numpy as np
from sklearn.cluster import KMeans
from scipy.ndimage import maximum_filter
from preprocessing import (preprocess_signal, denoise_signal,
                           smooth_signal, subtract_background_signal)


def perform_ok_analysis(data_sig):
    # --- Preprocessing ---
    sig = preprocess_signal(data_sig,
              energy_min=500., energy_max=560., stride_binning=2)
    sig = denoise_signal(sig, method='PCA', n_components=4)
    sig = smooth_signal(sig, window_length=21, polyorder=2)
    sig = subtract_background_signal(sig, fit_range=(500, 528))
```

```
    # --- Peak detection on spatially summed spectrum ---
    sum_data     = sig.sum().data
    peak_indices = np.where(maximum_filter(sum_data, 101) == sum_data)[0]
    B_peak_pos   = int(peak_indices[np.argmin(np.abs(peak_indices - 340))])
    A_peak_pos   = B_peak_pos - 30

    # --- Particle mask (K-means on integrated intensity) ---
    sig_sum = sig.data.sum(axis=2)
    km      = KMeans(n_clusters=3, n_init='auto', max_iter=1000)
    labels  = km.fit_predict(sig_sum.reshape(-1, 1))
    means   = [sig_sum.flatten()[labels == i].mean() for i in range(3)]
    bg_lbl  = int(np.argmin(means))
    mask    = (labels != bg_lbl).reshape(sig_sum.shape).astype(float)

    # --- Per-pixel A/B ratio ---
    diff    = np.gradient(sig.data, axis=2)
    a_range = slice(A_peak_pos - 5,  A_peak_pos + 6)
    b_range = slice(B_peak_pos - 10, B_peak_pos + 11)
    h, w    = sig_sum.shape
    A_pos   = np.argmin(np.abs(diff[:, :, a_range]),  axis=-1) + a_range.start
    B_pos   = np.argmax(sig.data[:, :, b_range],      axis=-1) + b_range.start
    rows, cols = np.arange(h)[:, None], np.arange(w)
    I_A = sig.data[rows, cols, A_pos]
    I_B = sig.data[rows, cols, B_pos]

    # --- Vo calibration ---
    AB_map = np.divide(I_A, I_B, out=np.zeros_like(I_A), where=(I_B != 0))
    vo_map = np.clip((AB_map - 0.492) / 0.017, 0, None)

    return AB_map * mask, vo_map * mask
```