```python
from dataclasses import dataclass

import matplotlib.pyplot as plt
import numpy as np
from scipy.ndimage import gaussian_filter1d
from scipy.optimize import curve_fit
from scipy.signal import find_peaks

ion_products = {1: 'H', 2: 'H₂', 10: 'Ar⁴', 12: 'C', 13: 'Ar³', 16: 'O', 17: 'OH', 18: 'H₂O',
                20: 'Ar²', 28: 'N₂', 32: 'O₂', 40: 'Ar', 43: 'CO₂'}

colors = ['#bf212f', 'black', '#264b96', '#27b376', '#AA5486', '#FFEB3B']
line_styles = ['-', '-', '-', '-']
@dataclass
class PeakSettings:
    match_peaks: bool = False
    wlen: int = 300
    width: float = 5
    min_height_percent: float = 0.005
    normalize: bool = False
    integrate: bool = False


@dataclass
class Transformation:
    function: callable = None
    original_points: np.array = None
    target_points: np.array = None
    smoothing: float = 0.01

@dataclass
class ZoomInSettings:
    enabled: bool = False
    xlim: tuple = (0, 0)
    position: tuple = (0.15, 0.35, 0.5, 0.4)  # (x, y, width, height) in figure coordinates
    linewidth: float = 1.5


def plot(file_names, transformation=Transformation(), interval=None, max_time=8,
         peak_settings=PeakSettings(), zoom_settings=ZoomInSettings(), weights=None, product_labels=ion_products):
    if interval is None:
        interval = [0, 0]

    plt.figure(figsize=(10.5, 5))
    main_axes = plt.gca()  # Main plot axes
    zoom_axes = None

    if zoom_settings.enabled:
        zoom_axes = plt.axes(zoom_settings.position)  # Create inset

    anchor = 0
    x_values = []
    count_values = []
    peak_values = []

    for i, file_name in enumerate(file_names):
        x, count = np.loadtxt(file_name, unpack=True)
        weight = (weights[i] if weights is not None else 1)

        # Adjust interval
        channels = len(x)
        if interval == [0, 0]:
            interval = [0, channels]

        x = x[interval[0]:interval[1]]
        count = count[interval[0]:interval[1]]

        # Find peaks
        peaks, peak_infos = find_peaks(count, height=max(count) * peak_settings.min_height_percent,
                                       wlen=peak_settings.wlen, width=peak_settings.width)
        peak_heights = peak_infos['peak_heights']

        if peak_settings.match_peaks:
            if anchor == 0:
                anchor = peaks[peak_heights.argmax()] + interval[0]
            else:
                factor = (peaks[peak_heights.argmax()] + interval[0]) / anchor
                x = x / factor

        # Apply transformation if given
        if transformation.function is not None:
            params, _ = curve_fit(transformation.function, transformation.original_points, transformation.target_points)
            x = transformation.function(x, *params)
        else:
            x = x / channels * max_time

        largest = integrate_peaks(main_axes, x, count, peaks, peak_infos, colors[i], peak_settings.integrate
, weight, product_labels)
        if peak_settings.normalize:
            count = count / largest * weight


        x_values.append(x)
        count_values.append(count)
        peak_values.append(peaks)

        count_smoothed = gaussian_filter1d(count, transformation.smoothing)

        label = file_name.split('/')[-1].split('.')[0]

        # Plot on main axes
        main_axes.plot(x, count_smoothed, color=colors[i], linestyle=line_styles[i], lw=1.1, label=label, alpha=0.7)

        # Plot on zoomed-in inset if enabled
        if zoom_settings.enabled:
            zoom_axes.plot(x, count_smoothed, color=colors[i], linestyle=line_styles[i], lw=1.1, alpha=0.6)

    # separate lists for inset and main
    peaks_inset = []
    for i, peaks in enumerate(peak_values):
        peaks_inset.append([peak for peak in peaks if zoom_settings.xlim[0] <= x_values[i][peak] <= zoom_settings.xlim[1]])
    peaks_main = []
    for i, peaks in enumerate(peak_values):
        peaks_main.append([peak for peak in peaks if not zoom_settings.xlim[0] <= x_values[i][peak] <= zoom_settings.xlim[1]])

    if transformation.function is not None:
        # Annotate peaks in main plot (excluding zoomed-in region)
        annotate_peaks(main_axes, x_values, count_values, max(count), peaks_main, product_labels)

        # Annotate peaks in inset (only zoomed-in region)
        if zoom_settings.enabled:
            max_height_inset = max(count[peak] for peaks in peaks_inset for peak in peaks)
            annotate_peaks(zoom_axes, x_values, count_values, max_height_inset * 3, peaks_inset, product_labels)

    # Zoom settings
    if zoom_settings.enabled:
        zoom_axes.set_xlim(zoom_settings.xlim)

        # Auto-scale inset y-axis
        zoom_axes.set_ylim((0, max_height_inset))
        yticks = np.arange(0, max_height_inset, 0.1)
        zoom_axes.set_yticks(yticks)

        # Draw a rectangle around the zoomed-in region in the main plot
        x_rect = zoom_axes.get_xlim()
        y_rect = zoom_axes.get_ylim()
        main_axes.plot([x_rect[0], x_rect[0], x_rect[1], x_rect[1], x_rect[0]],
                       [y_rect[0], y_rect[1], y_rect[1], y_rect[0], y_rect[0]],
                       linestyle="--", color="gray", linewidth=zoom_settings.linewidth)
        zoom_axes.spines['top'].set_visible(False)

    # Formatting main plot
    main_axes.spines['right'].set_visible(False)
    main_axes.spines['top'].set_visible(False)
    main_axes.set_xlabel('m/q [u/e]' if transformation.function is not None else r't in $\mu$s')
    main_axes.set_ylabel('Normalisierte Counts' if peak_settings.normalize else 'Counts')
    legend = main_axes.legend(loc="upper left")
    for line in legend.get_lines():
        line.set_linewidth(4)
    plt.tight_layout()
    plt.show()

def integrate_peaks(axes, x, count, peaks, peak_infos, color, show_integration, weight, product_labels):
    """ Returns the total counts of the largest peak """
    largest_count = 0
    total_counts = []
    for j, peak in enumerate(peaks):
        m_over_q = round(x[peak], 0)
        lower_bound = peak_infos['left_bases'][j]
        upper_bound = peak_infos['right_bases'][j]

        integrated_count = np.trapz(count[lower_bound:upper_bound], x=x[lower_bound:upper_bound])
        largest_count = max(largest_count, integrated_count)
        if show_integration and m_over_q in product_labels:
            total_counts.append((x[lower_bound], x[upper_bound], integrated_count))
            axes.axvspan(x[lower_bound], x[upper_bound], hatch='////', facecolor=color, alpha=0.15, edgecolor='grey')

        print(f'{product_labels.get(m_over_q, m_over_q)}⁺: {integrated_count:.0f}')

    if show_integration:
        total = 0
```

```python
        for data in total_counts:
            lower, higher, count = data
            weighted = count/largest_count * weight
            total += weighted
            axes.text((lower + higher)/2, .5, f'{weighted:.2e}', rotation=90, ha='center', fontsize=9)

    return largest_count


def annotate_peaks(axes, x_values, count_values, max_height, peak_values, product_labels):
    """ Annotates peaks on the given axes (either main plot or zoomed-in inset). """
    # create a dictionary with m/q as key where all height values for that peak are stored with their indices in the value lists
    m_over_q_peaks = {}
    for i, peaks in enumerate(peak_values):
        for peak in peaks:
            m_over_q = round(x_values[i][peak],0)
            info = (i,count_values[i][peak], peak)
            if m_over_q in m_over_q_peaks:
                m_over_q_peaks[m_over_q].append(info)
            else:
                m_over_q_peaks[m_over_q] = [info]

    plotted = []
    for m_over_q, infos in m_over_q_peaks.items():
        indices = [info[0] for info in infos]
        heights = np.array([info[1] for info in infos])
        peaks = [info[2] for info in infos]
        peak_idx = heights.argmax()
        list_idx = indices[peak_idx]

        if m_over_q in product_labels and m_over_q not in plotted:
            height = max(heights)
            axes.text(x_values[list_idx][peaks[peak_idx]], height + max_height * .035, f'{m_over_q:.0f}', ha='center', fontsize=11)
            txt = product_labels[m_over_q]
            axes.text(x_values[list_idx][peaks[peak_idx]], height + max_height * .08, txt + '⁺', ha='center', fontsize=12 if 'Ar' in txt else 9)
            plotted.append(m_over_q)

```