ARTICLE

Rapid Screening of Commercial CBD Oils by Heat-Assisted Dielectric Barrier Discharge Ionization (HA-DBDI) Mass Spectrometry and Correlation-Based Fingerprinting

Received 00th January 20xx,
Accepted 00th January 20xx

DOI: 10.1039/x0xx00000x

Odhisea Gazeli[a,b], Marios C. Christodoulou[c,d], Nikolaos Argirusis[e], George E. Georghiou[a,b], Efstathios A. Elia[*c,f], Agapios Agapiou[c]

The rapid expansion of the cannabidiol (CBD) market has created a need for efficient analytical methods to assess product quality and authenticity. Traditional chromatographic techniques, while accurate, require extensive sample preparation and are unsuitable for high-throughput screening. This study presents a heat-assisted dielectric barrier discharge ionization mass spectrometry (HA-DBDI-MS) approach combined with correlation-based fingerprinting for the rapid grouping of commercial CBD oils. The integrated heating element facilitates thermal desorption of semi-volatile compounds from viscous oil matrices, addressing a limitation in plasma-based ambient ionization. A systematic data processing workflow was developed to mitigate inherent signal variability through spectral averaging and total ion current normalization. The methodology was evaluated using cannabinoid reference standards and four commercial CBD oil samples with varying concentrations, spectrum types, and formulations. Pearson correlation analysis of the normalized spectral fingerprints revealed quantitative relationships consistent with product characteristics, including CBD concentration and spectrum designation. Samples with identical formulation parameters exhibited near-perfect correlation (r = 0.98), while products with distinct compositions showed lower similarity values. The results demonstrate that this approach provides rapid preliminary grouping based on overall phytochemical composition, offering a complementary screening tool to conventional quantitative methods for quality control applications in cannabis-derived products.

[a.] *PHAETHON Centre of Excellence for Intelligent, Efficient and Sustainable Energy Solutions, Nicosia 2109, Cyprus.*
[b.] *ENAL Electromagnetics and Novel Applications Lab, Department of Electrical and Computer Engineering, University of Cyprus, Nicosia 2109, Cyprus.*
[c.] *Department of Chemistry, University of Cyprus, Nicosia, 2109, Cyprus.*
[d.] *Laboratory of Chemical Engineering and Engineering Sustainability, Faculty of Pure and Applied Sciences, Open University of Cyprus, Latsia, 2231, Nicosia, Cyprus*
[e.] *4mat4nrg GmbH, 38678 Clausthal-Zellerfeld, Germany*
[f.] *Medical School, University of Cyprus, Nicosia, 2109, Cyprus*

Supplementary Information

S1. HA-DBDI Source Construction Details

The Heat-Assisted Dielectric Barrier Discharge Ionization (HA-DBDI) source was constructed using easily available and low-cost components. The following section provides detailed specifications for the key components referenced in the main manuscript.

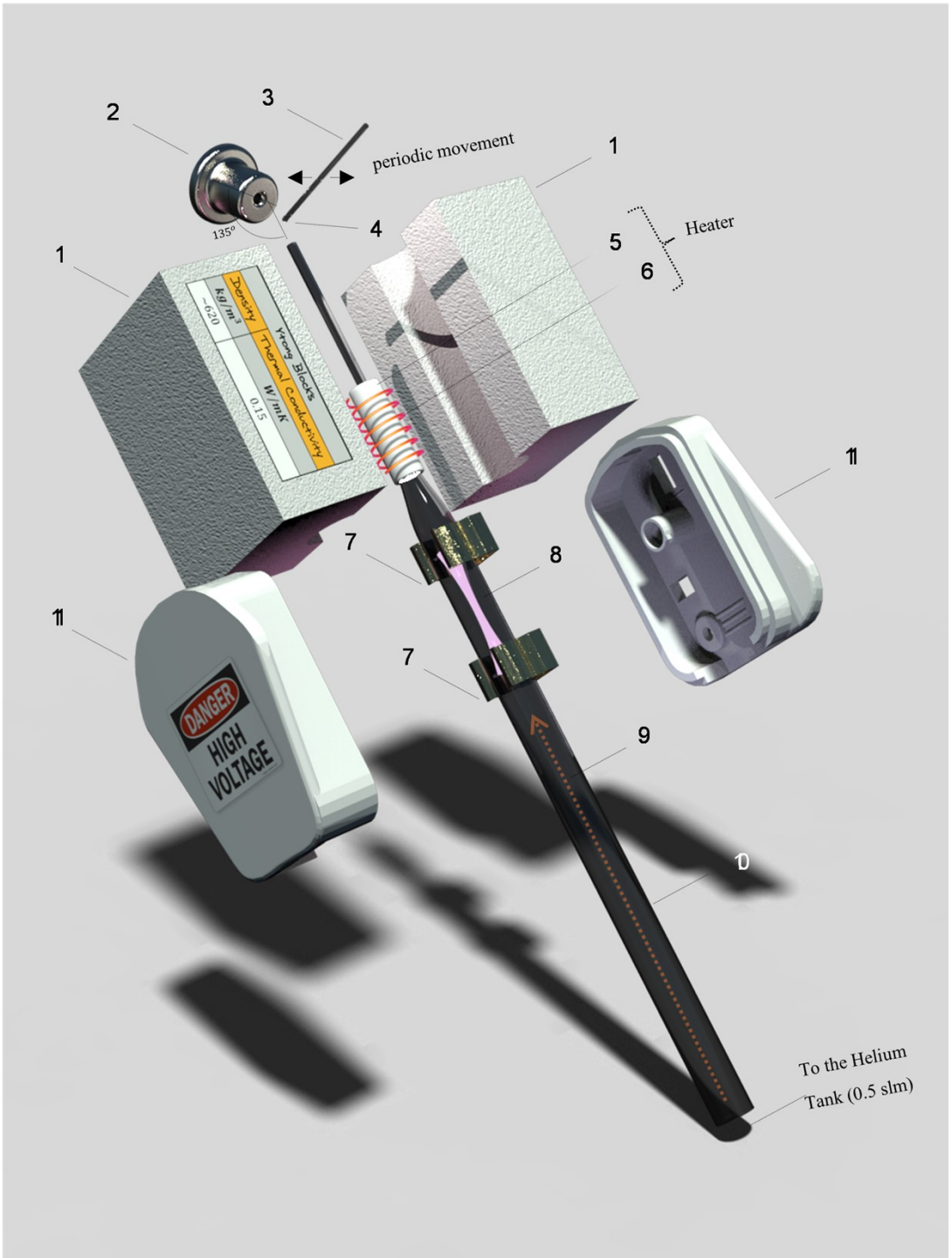


Figure S1: Experimental setup for the analysis of CBD oils using the HA-DBDI ionization source. The components are labelled in the image as follows: 1) Thermal insulation block, 2) Mass Spectrometer (MS) Inlet, 3) disposable pipette tip, 4) Sample, 5) Ceramic Fuse, 6) Nichrome Wire, 7) Fuse Clips, 8) Plasma, 9) Helium Gas Flow, 10) Disposable Pasteur Pipette, 11) UK Plug.

S1.1. Plasma Reactor and Electrodes

The plasma reactor body was a standard 150 mm borosilicate glass Pasteur pipette (Volac/**Figure S1-10**). The high-voltage (HV) and ground electrodes were constructed from the fuse clips (**Figure S1-7**) of a standard UK Type G electrical plug (BS 1363/ **Figure S1-11**). This specific component was selected for its rigid metallic structure and pre-defined geometry, which provided a stable and reproducible electrode gap of 12 mm.

S1.2. Resistive Heating Element

To facilitate the thermal desorption of analytes (**Figure S1-3**) from complex matrices, a resistive heating element was integrated at the exit of the plasma reactor. The heater was constructed as follows.

Core: A standard porcelain fuse (5x20 mm) was used as the insulating core due to its thermal stability and high electrical insulating properties (**Figure S1-5**).

Wire: Six coils of 22-gauge nichrome-chromium wire (80% Ni, 20% Cr) were wrapped around the porcelain fuse without touching each other in order to avoid shorting the circuit. The wire has a specified resistance of 1.0118 Ω ft^{-1} at 20 °C. The total length of the wire used was approximately 264 mm. (**Figure S1-6**)

Power Supply: The heater was powered by a standard laboratory DC power supply set to a constant current of 5 A at 1.6 V, resulting in a total power consumption of 8 W.

Performance: Under these conditions, the heating element raised the temperature of the helium gas stream to approximately 200 °C at the pipette tip.

S1.3. Thermal Insulation

To maintain a stable operating temperature and to protect surrounding components, a thermal insulation block was positioned around the heating element. The block was fabricated from autoclaved aerated concrete, which has a density of approximately 620 kg/m³ and a low thermal conductivity of 0.15 W/m·K (**Figure S1-1**).

S1.4. High-Voltage Power Supply

The plasma (**Figure S1-8**) was sustained by a home-built high-voltage power supply. For the experiments in this study, the power supply was modified with a new transformer (1:1000 primary-to-secondary turns ratio, max 120 W) to allow for fine control over plasma power. The applied voltage waveform was a square wave with a frequency of 20 kHz and a 50% duty cycle.

S2. Schematic of the data processing workflow.

		Mass-To-Charge (m/z) axis				
	Scan / bin	$\left(\frac{m}{z}\right)_1$	$\left(\frac{m}{z}\right)_2$	…	$\left(\frac{m}{z}\right)_{n-1}$	$\left(\frac{m}{z}\right)_n$
Time axis	t_1	$I_1^{t_1}$	$I_2^{t_1}$	…	$I_{n-1}^{t_1}$	$I_n^{t_1}$
	t_2	$I_1^{t_2}$	$I_2^{t_2}$	…	$I_{n-1}^{t_2}$	$I_n^{t_2}$
	…	…	…	…	…	…
	t_{n-1}	$I_1^{t_{n-1}}$	$I_2^{t_{n-1}}$	…	$I_{n-1}^{t_{n-1}}$	$I_n^{t_{n-1}}$
	t_n	$I_1^{t_n}$	$I_2^{t_n}$	…	$I_{n-1}^{t_n}$	$I_n^{t_n}$
	Average	$I_1 = \frac{(I_1^{t_1} + I_1^{t_2} + \cdots + I_1^{t_{n-1}} + I_1^{t_n})}{n}$	$I_2 = \frac{(I_2^{t_1} + I_2^{t_2} + \cdots + I_2^{t_{n-1}} + I_2^{t_n})}{n}$	…	$I_{n-1} = \frac{(I_{n-1}^{t_1} + I_{n-1}^{t_2} + \cdots + I_{n-1}^{t_{n-1}} + I_{n-1}^{t_n})}{n}$	$I_n = \frac{(I_n^{t_1} + I_n^{t_2} + \cdots + I_n^{t_{n-1}} + I_n^{t_n})}{n}$
	Average Normalized (Fingerprint)	$\frac{I_1}{I_1 + I_2 + \cdots + I_{n-1} + I_n}$	$\frac{I_2}{I_1 + I_2 + \cdots + I_{n-1} + I_n}$	…	$\frac{I_{n-1}}{I_1 + I_2 + \cdots + I_{n-1} + I_n}$	$\frac{I_n}{I_1 + I_2 + \cdots + I_{n-1} + I_n}$

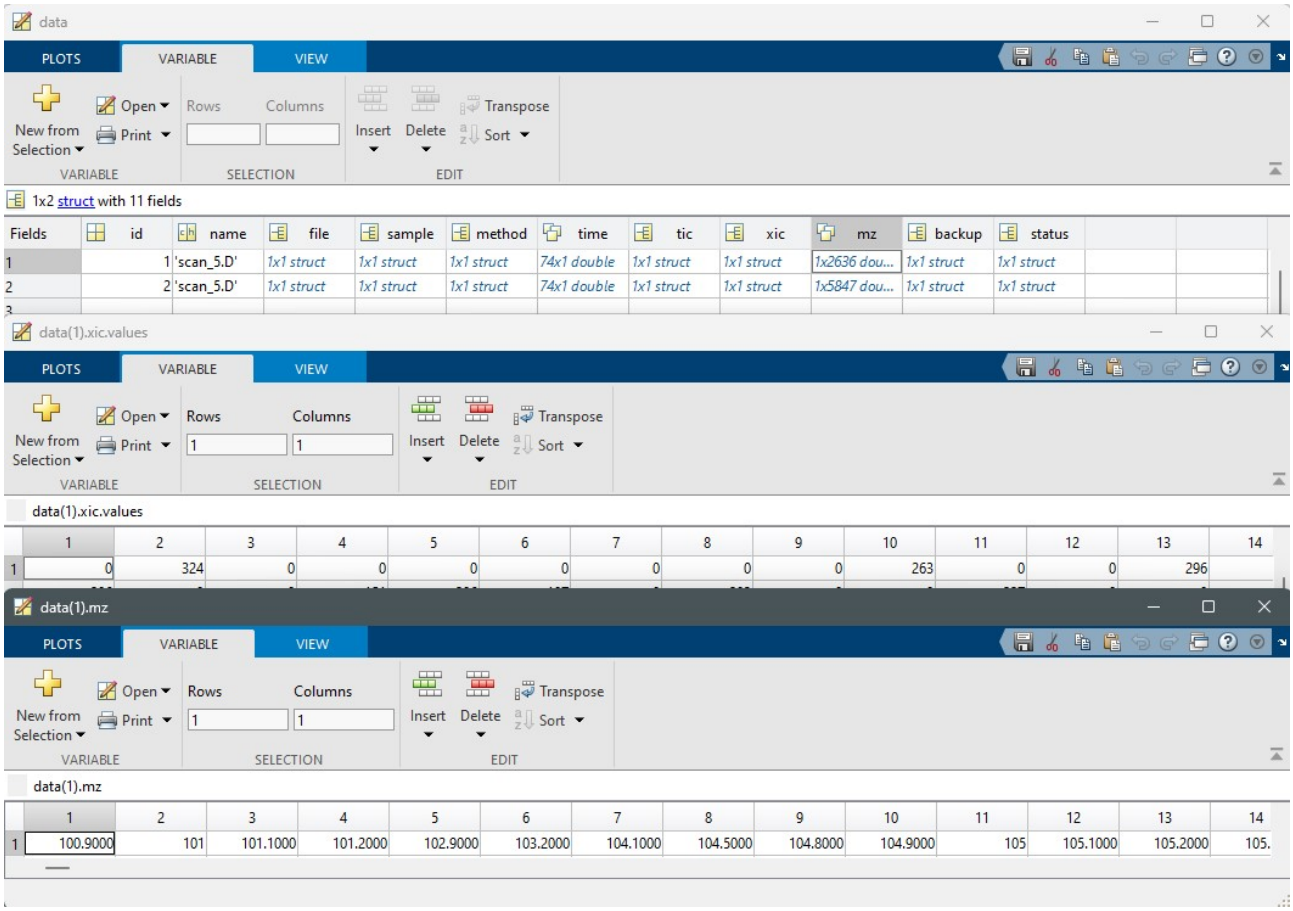
Figure S2.: Schematic representation of the data processing workflow. The raw data is structured as a matrix where each row (t_1 to t_n) represents a single mass spectrum acquired at a specific time point, and each column represents a corrected mass-to-charge bin ($(m/z)_1$ to $(m/z)_n$). The workflow consists of two main steps: (1) Averaging, where the mean intensity (I) for each m/z bin is calculated across all time points, and (2) Normalization, where each average intensity value (I_i) is divided by the sum of all average intensities (the Total Ion Count of the average spectrum, TIC) to produce the final, comparable chemical fingerprint.

S3. MATLAB Data Processing Workflow

This section provides a detailed description of the custom data processing workflow implemented in MATLAB R2024b to convert raw mass spectral data into corrected and normalized chemical fingerprints suitable for comparative analysis.

S3.1. Data Import and Structure

Raw data files in the Agilent .D format were imported using the import function from the Chromatography Toolbox for MATLAB (Version 1.4.0.0). As shown in **Figure S3.1**, this function loads the data into a MATLAB structure variable, where time-resolved spectral information is stored. For each time point, ion intensities are contained within the *data.xic.values* field, while the corresponding mass-to-charge ratios are stored in the *data.mz* field.

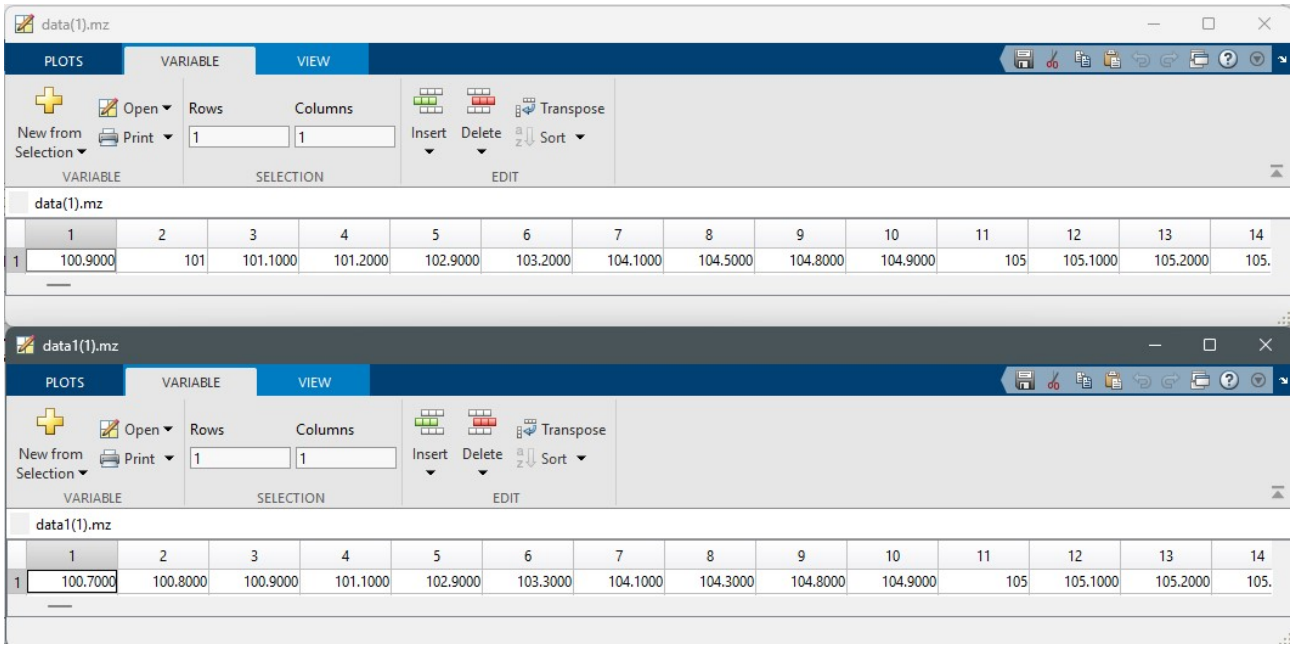


S3.1: Data structure as saved in MATLAB following execution of Chromatography Toolbox commands.

S3.2. The Challenge of Mass Axis Inconsistency and the Need for Data Correction

A critical challenge in processing the raw data was the instrument-generated variability in the mass-to-charge (*m/z*) axis. As illustrated in **Figure S2.2**, the specific *m/z* values recorded for each scan were not identical across analytical runs of individual samples (e.g., the first data point being *m/z* 100.9 in one run and *m/z* 100.7 in another). This inherent instrumental misalignment makes a direct, vector-based comparison of the raw spectra mathematically invalid and would lead to erroneous conclusions.

To overcome this fundamental issue and enable meaningful spectral comparison, a data correction (rebinning) step was implemented as the core of our workflow.



S3.2: Recorded ion masses from two separate spectra, as detected by the mass spectrometer.

S3.3. Implementation of the Spectral Processing Workflow in MATLAB

```
% Chromatography Toolbox Script
% This script demonstrates how to use the Chromatography Toolbox to import
% chromatography data.
% Supported file extensions include:
% - Agilent (.D)
% - Agilent (.MS)
% - netCDF (.CDF)

clc;          % Clear the command window of any previous text.
clear all;    % Clear all variables from the MATLAB workspace to start fresh.

% Create an instance of the Chromatography object.
obj = Chromatography(); % Initializes the toolbox object, giving access
%to its functions.

% Display the toolbox version.
obj.version; % Accesses the 'version' property of the object and %displays it.

% Import chromatography data from an Agilent '.D' file.
data = obj.import('.D'); % Calls the 'import' method to load the data
%into the 'data' structure.
```

```
% ========================================================================
% MATLAB Script for Processing HA-DBDI-MS Data
%
% Version: 2.4 (Final & Fast)
% ========================================================================
%% --- 0. Initialization ---

tic;
% Starts a stopwatch timer to measure the script's execution time.
if ~exist('data', 'var')
% Checks if the variable 'data' does NOT exist in the workspace.
    error('The "data" variable was not found. Please run Script 1 first to import the .D
file.'); % Stops execution and displays an error if %'data' is missing.
end

%% --- 1. TIC Visualization and Analysis Window Selection ---
%Prints a status message to the command window.
fprintf('Step 1: Visualizing Total Ion Chromatogram (TIC)...\n');
%Extracts the time vector (retention time in minutes) from the data.
time_vector = data(1).time;
%Calculates the Total Ion Current by summing all intensities for each %time point (sum
%along dimension 2).
tic_vector = sum(data(1).xic.values, 2) ;
figure;  % Creates a new figure window for plotting.
plot(time_vector, tic_vector', '-b', 'LineWidth', 1.5); % Plots the
%time vector vs. the TIC vector with a blue, solid line of width 1.5.
title('Total Ion Chromatogram (TIC)');   % Sets the %title of the plot.
xlabel('Time (minutes)');          % Sets the %label for the x-axis.
ylabel('Intensity');               % Sets the %label for the y-axis.
grid on;                % Adds a grid %to the plot for better readability.
ax = gca; ax.FontSize = 12;
```

```
% Gets the handle to the current axes (gca) and sets the font size to 12.
fprintf('Examine the plot, then set "start_time_min" and "end_time_max" in Section
2.\n'); % Instructs the user on the next step.

%% --- 2. Define Analysis Parameters ---
start_time_min = min(data(1).time); % Sets the start time for analysis
%to the earliest time point in the data.
end_time_max   = max(data(1).time); % Sets the end time for analysis to
%the latest time point in the data.

%% --- 3. Data Correction and Rebinning (Vectorized) ---
%Prints a status message.
fprintf('Step 3: Rebinning raw data onto a corrected m/z axis...\n');
mz_initial = 100     % Defines %the lower m/z limit for the analysis.
mz_final   = 1000;   % Defines %the upper m/z limit for the analysis.
resolution = 0.1     % Defines %the resolution of MS
corrected_mz_axis = (mz_initial: resolution :mz_final)';     % Creates
%the new, uniform m/z axis from 100 to 1000 with a step of 0.1. The '
%transposes it to a column vector.

num_scans = length(data(1).time);%Determines the total number of scans (time points) in
% the data.

rebinned_data = zeros(num_scans, length(corrected_mz_axis)); % Pre-allocates a matrix of
% zeros to store the rebinned data for efficiency.

% Check if .mz is a cell array or a standard array.
is_mz_cell = iscell(data(1).mz); % Checks the data type of
%the m/z field. Returns true if it's a cell array.

for i = 1:num_scans % NOTE: This loop iterates through ALL scans.
% The original `1:2` was likely %for testing.
    % Get the raw m/z values for the current scan based on its type.
    if is_mz_cell     % If the %m/z data is stored in a cell array...
        raw_mz = data(1).mz{i}; % %...extract the m/z vector for scan 'i' using cell
% array indexing {}.
    else %Otherwise (if it's a standard matrix)...
 % If not a cell, it implies a single, shared m/z axis or a %matrix of m/z values.
        raw_mz = data(1).mz;    % Assign the whole matrix first.
% If it's a matrix with a row for each time point, select the %correct row.
        if size(raw_mz,1) == num_scans
% Check if the number of rows %matches the number of scans.
            raw_mz = data(1).mz(i,:); % Select the i-th row using
% standard matrix indexing ().
        end
    end
    raw_mz; % This line displays the
%raw_mz vector for the current scan (useful for debugging).
    raw_intensity = data(1).xic.values(i, :); % Extracts the raw
%intensity values for the current scan 'i'.

% Trim vectors to actual data length to avoid errors with padding %zeros.
    valid_length = find(raw_mz > 0, 1, 'last'); % Finds the index of
%the last m/z value that is greater than zero.
    raw_mz = raw_mz(1:valid_length);  % Trims the m/z
```

```
%vector to remove any trailing zeros.
    raw_intensity = raw_intensity(1:valid_length); % Trims the
%intensity vector to match the new length of the m/z vector.
    % Rebinning process: Assign each raw m/z value to a bin on the corrected axis.
    [~, ~, bin_indices] = histcounts(raw_mz, [corrected_mz_axis;
corrected_mz_axis(end)+0.1]); % Determines which bin each raw_mz value
%falls into. The '~' ignores unwanted outputs.
    valid_indices = bin_indices > 0;
% Creates a logical mask to find which m/z values fell within a valid
%bin (index > 0).
    binned_intensities = accumarray(bin_indices(valid_indices)',
raw_intensity(valid_indices)', [length(corrected_mz_axis) 1], @sum); % Sums up all
% intensities that fall into the same bin.

    rebinned_data(i, :) = binned_intensities';
% Stores the resulting vector of binned intensities as a new row in the
%rebinned_data matrix.
end
fprintf('Data correction and rebinning complete.\n'); % Prints a status message.
```

```
%% --- 4. Spectral Averaging ---
analysis_indices = find(time_vector >= start_time_min & time_vector <= end_time_max);
% Finds the indices of the scans that are within the specified time window.
if isempty(analysis_indices)
% Checks if the time window is empty.
    error('The specified time window does not contain any data points.');
% Stops execution if no data points are %found.
end
fprintf('Step 4: Averaging spectra from %.2f min to %.2f min...\n', start_time_min,
end_time_max); % Prints a status message with the selected time range.
average_spectrum = mean(rebinned_data(analysis_indices, :), 1);
% Calculates the average spectrum by taking the mean of each column
%over the selected time window.
std_tic = std(rebinned_data(analysis_indices, :), 1);
% Calculates the standard deviation for each m/z bin across the
%selected time window.
fprintf('Averaging complete.\n');
% Prints a status message
```

```
%% --- 5. Normalization and Final Output ---
fprintf('Step 5: Normalizing the average spectrum...\n'); % Prints a status message.
% Calculates the total intensity (TIC) of the single average spectrum.
total_intensity_of_avg = sum(average_spectrum);
% Checks if the total intensity is positive to avoid division by zero.
if total_intensity_of_avg > 0
    normalized_avg_spectrum = average_spectrum ./ total_intensity_of_avg; % Normalizes
%  the spectrum by dividing each %intensity value by the total intensity.
else            % If total intensity is zero...
    normalized_avg_spectrum = average_spectrum;                          % ...the
%  normalized spectrum is also all zeros.
end

fingerprint = [corrected_mz_axis, normalized_avg_spectrum'];             % Creates the
% final 'fingerprint' matrix with m/z in the first column %and normalized intensities in
% the second.
```



```
fprintf('Workflow complete. Final variable "fingerprint" has been created.\n'); % Prints
% a final confirmation message.
```

```
%% --- 6. Visualization of Final Fingerprint (Optional) ---
figure;                    %Creates a new figure window.
stem(fingerprint(:,1), fingerprint(:,2)); %Creates a stem plot of the final fingerprint
% (m/z vs. normalized %intensity).
title('Final Normalized Average Mass Spectrum (Fingerprint)'); % Sets the plot title.
xlabel('Mass-to-Charge (m/z)');                                % Sets the x-axis label.
xlim([314 316])                                                % Zooms in on a specific
%  m/z range. Comment out this line to see the full %spectrum.
ylabel('Normalized Intensity');                                % Sets %the y-axis label.
grid on;                                                  % Adds a grid to the plot.
ax = gca; ax.FontSize = 12;   % Gets the current axes handle and sets the font size.
```

```
%% --- 7. Cleanup ---
toc;  % Stops the %stopwatch timer and displays the total elapsed time.
TIC = [time_vector, tic_vector];% Creates a new 2-%column matrix 'TIC' for easy export.
clearvars -except TIC average_spectrum fingerprint; % Clears all %variables from the
% workspace EXCEPT the final important results.
```

S3.4. Correlation Analysis

For the pairwise comparison of the final normalized spectra, the standard built-in MATLAB function *corrcoef(M)* was used, which calculates the Pearson correlation coefficients for a matrix where each column represents a sample's fingerprint.

S4. Spectral Reproducibility.

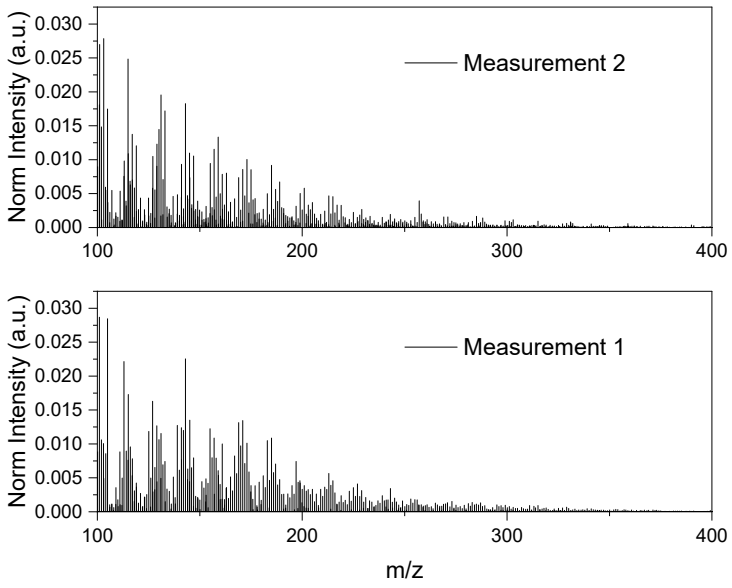


Figure S4: Spectral reproducibility of the HA-DBDI method. The plot shows two independent mass spectral acquisitions of the same CBD standard solution (1 µg/mL) under identical conditions, exhibiting high consistency in the spectral pattern.

S5. Mass Spectra at High Concentration and Blank Analysis.

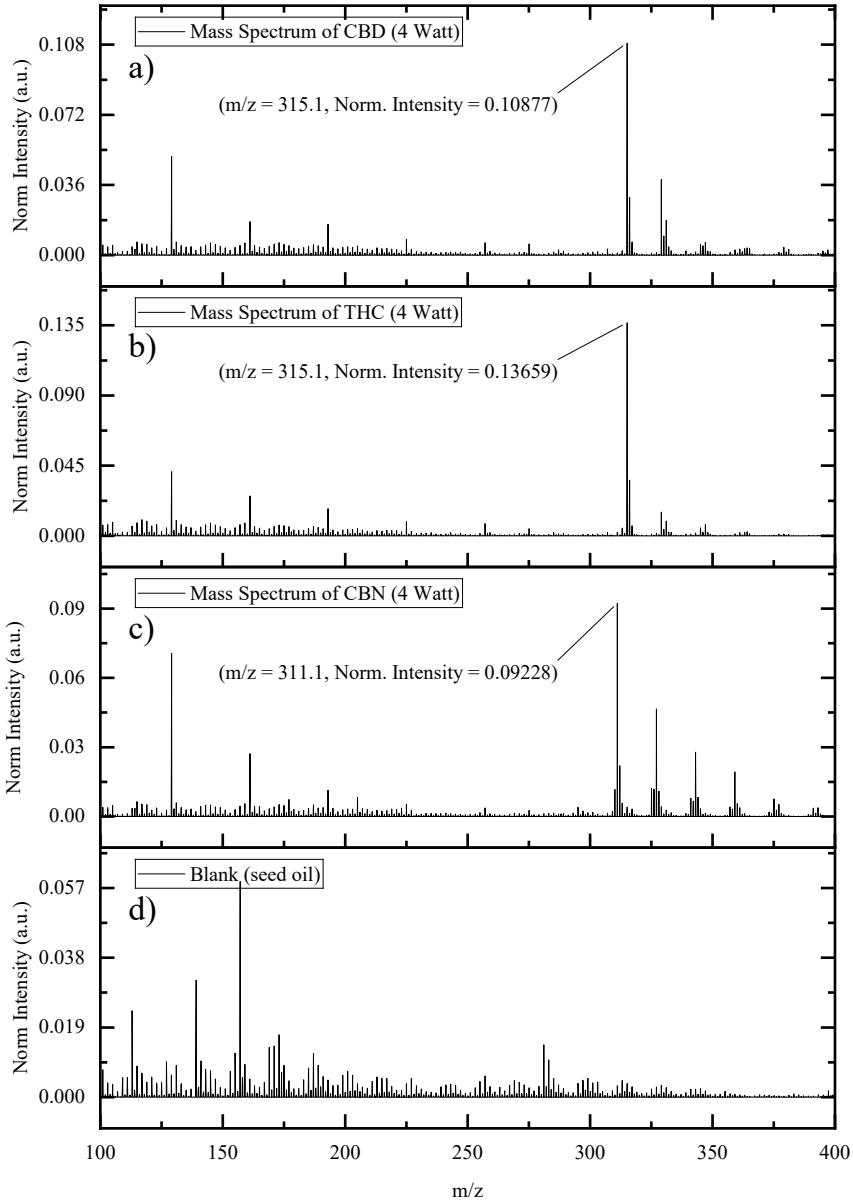


Figure S5: Validation of signal identity through high-concentration standards and blank matrix analysis. Panels (a), (b), and (c) shows mass spectra of CBD, CBN, and THC at 100 µg/mL, respectively. At this higher concentration, the protonated molecule [M+H]+ at m/z 315.1 (for CBD/THC) becomes the base peak, confirming that the low signal-to-noise ratio observed at 1 µg/mL is due to solvent suppression. Panel (d) shows the spectrum of a cannabinoid-free seed oil blank, confirming the absence of any interfering background signal at m/z 315.1.

Acknowledgements

This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement 810686. E. A. E. received funding from the European Union under Marie Skłodowska-Curie Actions (grant no. 101109014).

