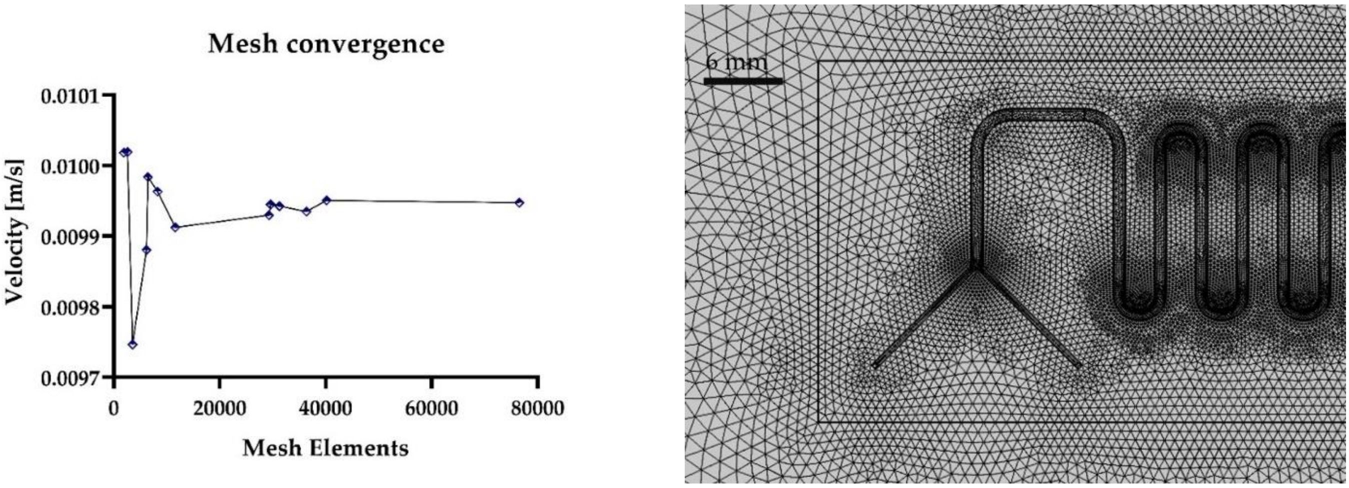
Supplementary Materials



(**a**)

(**b**)

Figure S1. (a) Mesh convergence study for the mixture model velocity. Convergence was evaluated with point evaluation for the velocity field within the micromixing serpentine. The results were assessed considering the active mixing process with the acoustic field, having a more significant variation between mesh sizes than for passive mixing; **(b)** Meshed geometry.

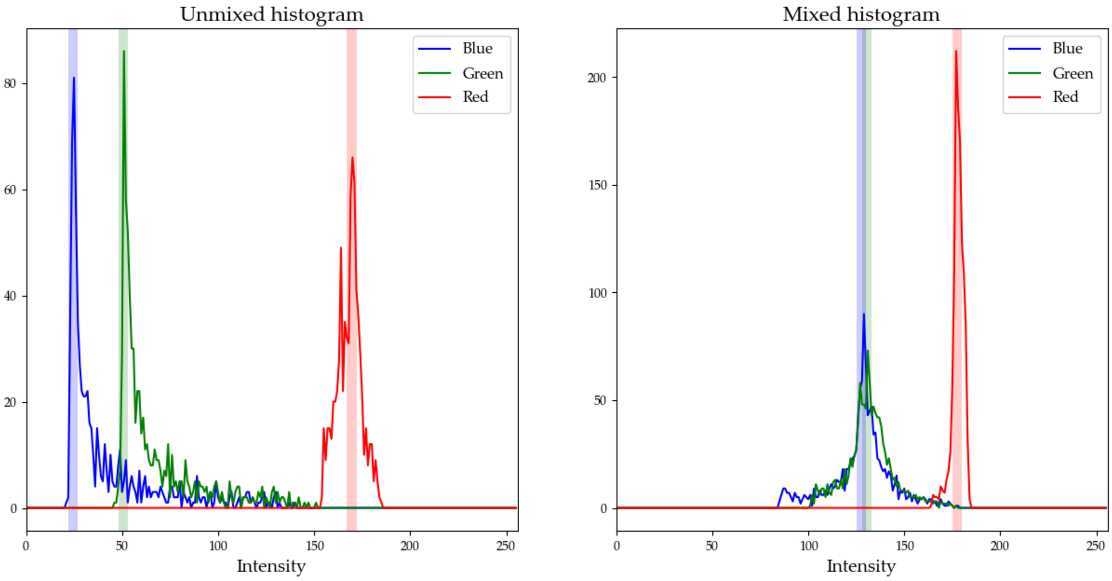
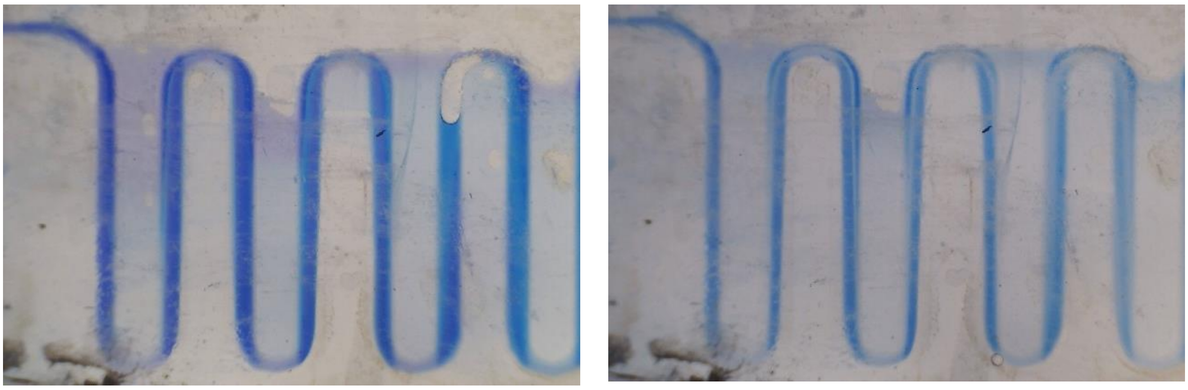


Figure S2. Color intensity histogram for unmixed and mixed fluids within the microfluidic device. The color composition was obtained from captured images shown in Figure S3.

$$I_i = \sqrt{0.241(red_{channel})^2 + 0.691(green_{channel})^2 + 0.068(blue_{channel})^2} \quad (1A)$$

Where the red, green, and blue channels are given by the pixel composition of an RGB image.



(**a**) (**b**)

Figure S3. Top view of microfluidic device channels during mixture test evaluation in the presence of methylene blue for different FRRs. (**a**) FRR 1:1; (**b**) FRR 1:3.

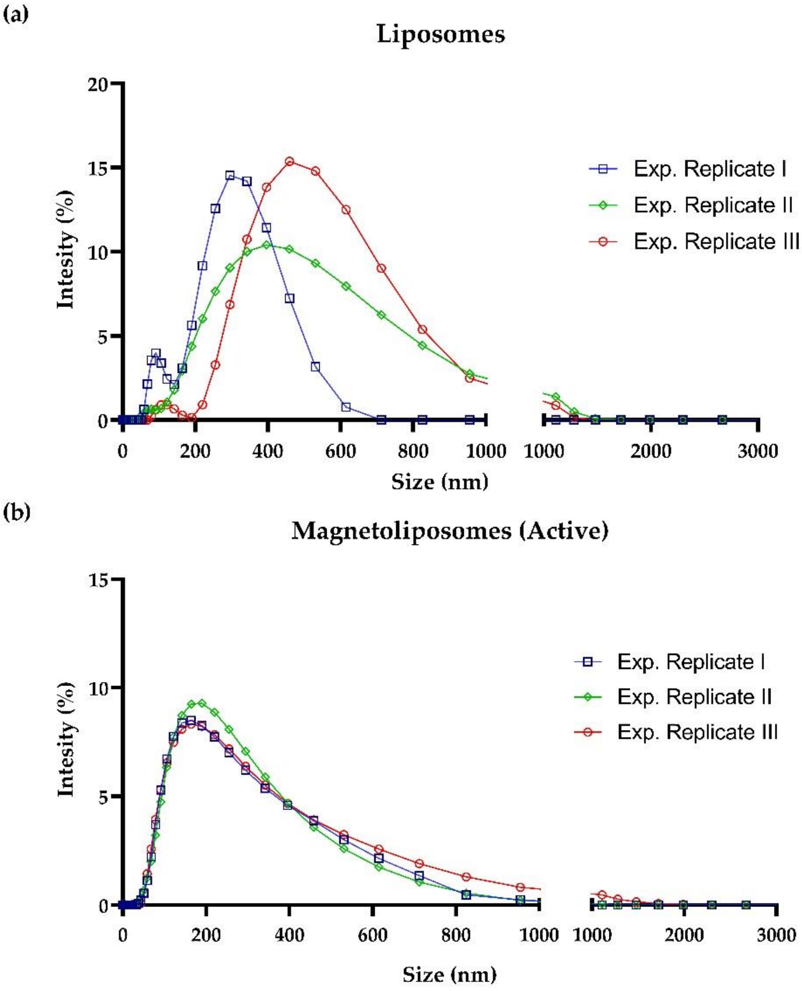


Figure S4. DLS measurements of liposomes and magnetoliposomes. (a) Liposomes used for encapsulation experiments; (b) Magnetoliposomes obtained following acoustic encapsulation (active) at a FRR = 1:3.

Script S1: Supplementary Python Script

Mixing_Code.ipynb

This code quantifies the mixing quality, measured using the standard deviation of the pixel intensity within a window. Given by the following equation:

```
\begin{equation}
\sigma = 1-\sqrt{\frac{1}{N}\sum{(\frac{I_{i}-I_{mix}}{I_{umix}-
I_{mix}})^{2}}}
\end{equation}

where $I_{i}$ is the pixel intensity of a given pixel in the position i,
$I_{umix}$ is the intensity of the same pixel before mixing, and
$I_{mix}$ is the intensity of the same pixel when it is reached a full
mix."""

# First, we import the libraries
import matplotlib.pyplot as plt
import numpy as np
import cv2
import imageio as io
import matplotlib
                                        # Image visualization

# read control images mix and unmix liquids
mix_image = io.imread('Mezclado.PNG')
umix_image = io.imread('Sin_Mezlcar.PNG')

# read image to evaluate mixture
image = io.imread('3a1/3a1finalfP7.PNG')

plt.figure(figsize=(10,10)) # Increase the image size by 10x10
plt.imshow(image)
plt.xticks([])
plt.yticks([])
plt.close()

plt.figure(figsize=(10,10)) # Increase the image size by 10x10
plt.imshow(umix_image)
plt.xticks([])
plt.yticks([])
plt.close()

plt.figure(figsize=(10,10)) # Increase the image size by 10x10
plt.imshow(mix_image)
plt.xticks([])
plt.yticks([])
plt.close()
                                        # Image Analysis

color = ('b','g','r')
labels = ('Blue','Green','Red')

font = {'family' : 'Palatino Linotype','size': 10}
matplotlib.rc('font', **font)

plt.figure(figsize=(15,7))
plt.subplot(1,2,1)

# Coulor image information of all channels (rgb)
for i, c in enumerate(color):
    hist = cv2.calcHist([umix_image], [i], None, [256], [0, 256])
    plt.plot(hist, color = c, label=labels[i])
    plt.title('Unmix histogram',fontsize = 16,family='Palatino Linotype')
    plt.xlim([0,256])

plt.axvspan(167, 172, facecolor='red', alpha=0.2)
plt.axvspan(48, 53, facecolor='green', alpha=0.2)
plt.axvspan(22, 27, facecolor='blue', alpha=0.2)

plt.xlabel('Intensity',fontsize=14)
plt.legend(fontsize=12)

plt.subplot(1,2,2)
for i, c in enumerate(color):
    hist = cv2.calcHist([mix_image], [i], None, [256], [0, 256])
    plt.plot(hist, color = c, label=labels[i])
    plt.title('Mixed histogram',fontsize = 16,family='Palatino Linotype')
    plt.xlim([0,256])

plt.axvspan(175, 180, facecolor='red', alpha=0.2)
plt.axvspan(128, 133, facecolor='green', alpha=0.2)
plt.axvspan(125, 130, facecolor='blue', alpha=0.2)
plt.xlabel('Intensity',fontsize=14)
plt.legend(fontsize=12)
plt.show()

umix_mean = np.sqrt( 0.241*(160**2) + 0.691*(   50** 2) + 0.068*( 20** 2 ) )
mix_mean   = np.sqrt( 0.241*(175**2) + 0.691*( 130** 2) + 0.068*( 125** 2) )

# Calculation of the standard deviation “quality of mixing”

umix = umix_mean # Intensity of the material
mix =   mix_mean # Intensity of full Mixing
N = np.size(image)
vec_sum = []
for i in range(np.size(image,axis=0)):
  for j in range(np.size(image,axis=1)):

    I_i = np.sqrt( 0.241* (image[i,j,0] ** 2) + 0.691* (image[i,j,1] ** 2) + 0.068* (image[i,j,2]**2) )
    vec_sum.append( (I_i-mix)/(umix-mix) )

sigma = 1 - np.sqrt( (1/N)*np.sum(np.power(vec_sum,2)) )

print('Our sigma is:')
print(sigma)
```