*Supplementary Information for*

# FLIM-FRAPP: Near-Simultaneous Characterization of Multi-Scale Polymer Dynamics via Fluorescence Microscopy

M. K. Jutze,[a] Walter W. Young,[a] Jasney Combs,[b] Owen C. Drescher,[c] Siddh Merchant,[d] and Reika Katsumata*[a]

[a] University of Massachusetts Amherst Department of Polymer Science and Engineering, 120 Governors Drive, Amherst MA 01003 USA
[b] Ball State University Department of Biology, 622 N Martin Street, Muncie IN 47303 USA
[c] University of Massachusetts Amherst Department of Physics, 710 North Pleasant Street, Amherst MA 01003 USA
[d] University of Massachusetts Amherst Department of Chemical Engineering, 686 North Pleasant Street, Amherst MA 01003 USA
*Corresponding author: rkatsumata@umass.edu

## *Materials*

Copper(I) bromide (Cu(I)Br), triethylamine (TEA), and basic alumina were purchased from Acros Organics (Fair Lawn, NJ, USA). Methyl acrylate (MA), methyl methacrylate (MMA), and ethyl α-bromoisobutyrate (EBiB) were purchased from Sigma-Aldrich (St. Louis, MO, USA). N, N, N', N", N"-Pentamethyl diethylenetriamine (PMDETA) and 2-bromoisobutyryl bromide (BiBBr) were purchased from TCI (Portland, OR, USA). Inhibitor removal resin and 4-chloro-7-nitrobenzofurazan (NBD-Cl) were purchased from Alfa Aesar (Haverhill, MA, USA). Glacial acetic acid, tetrahydrofuran (THF), toluene, ethanol, sodium bicarbonate, hydrochloric acid, methyl amino ethanol (MEA), and hexanes were purchased from Fisher Scientific (Waltham, MA, USA). Dichloromethane (DCM) was purchased from Supelco (Millipore) (Burlington, MA).

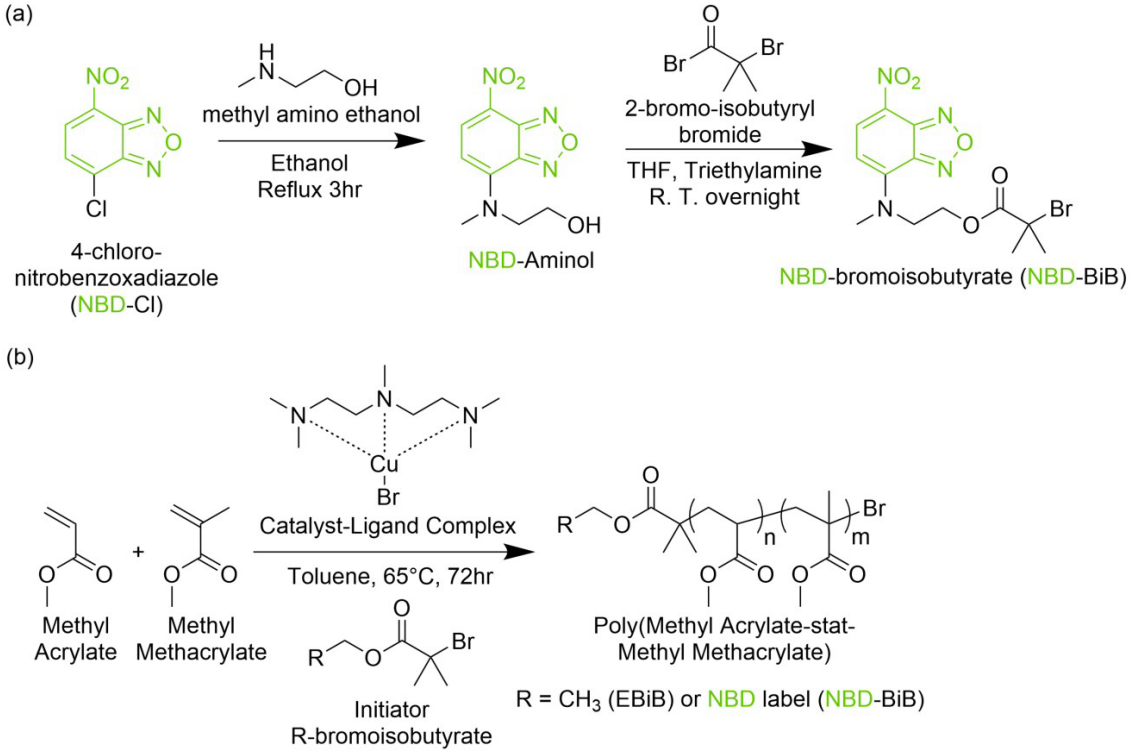
MA and MMA were purified by stirring in inhibitor removal resin and subsequent basic alumina column filtration. Cu(I)Br was isolated by stirring in glacial acetic acid and subsequent vacuum filtration and drying under reduced pressure. All other materials were used as received.

## *Synthesis of Unlabelled P(MA-stat-MMA) and Labelled NBD-P(MA-stat-MMA)*

Labelled NBD-P(MA-stat-MMA) was synthesized using a fluorescently-labelled atom-transfer radical polymerization (ATRP) initiator, NBD- BiB, according to procedures modified from Katzenstein et al (**Scheme S1a**).[1] NBD-Cl and MEA (molar ratio 1:3, respectively) were added to excess ethanol and refluxed for 3 hours at 60 °C. NBD-aminol precipitated during the reaction and subsequent cool to room temperature was collected by vacuum filtration and purified by two recrystallizations in ethanol, before final filtration and drying under reduced pressure. The dried NBD-aminol was dissolved in excess THF, followed by additions of TEA and BiBBr (molar ratio of NBD-aminol, TEA, and BiBBr of 1:5:1.5, respectively). The reaction proceeded overnight at room temperature, after which the THF was removed under reduced pressure and the reaction mixture was redissolved in dichloromethane. NBD-BiB was separated via liquid-liquid extraction with three cycles consisting of successive washes with 1M hydrochloric acid, concentrated aqueous sodium bicarbonate, and DI water. The organic fraction was isolated, and NBD-BiB was collected under reduced pressure.

Unlabelled poly(methyl acrylate–stat–methyl methacrylate) (P(MA-stat-MMA)) polymer and P(MA-stat-MMA) labelled with Nitrobenzofurazan (NBD-P(MA-stat-MMA)) were synthesized via ATRP reactions (**Scheme S1b**), following the available literature.[2,3] A freeze-pump-thaw procedure was repeated for a minimum of five times to a solution of purified MA, MMA, PMDETA, toluene, and, in the case of unlabelled synthesis, EBiB. The solution was then cannulated into a vial containing isolated Cu(I)Br and, in the case of labelled synthesis, NBD-BiB having undergone a minimum of five pump – backfill cycles

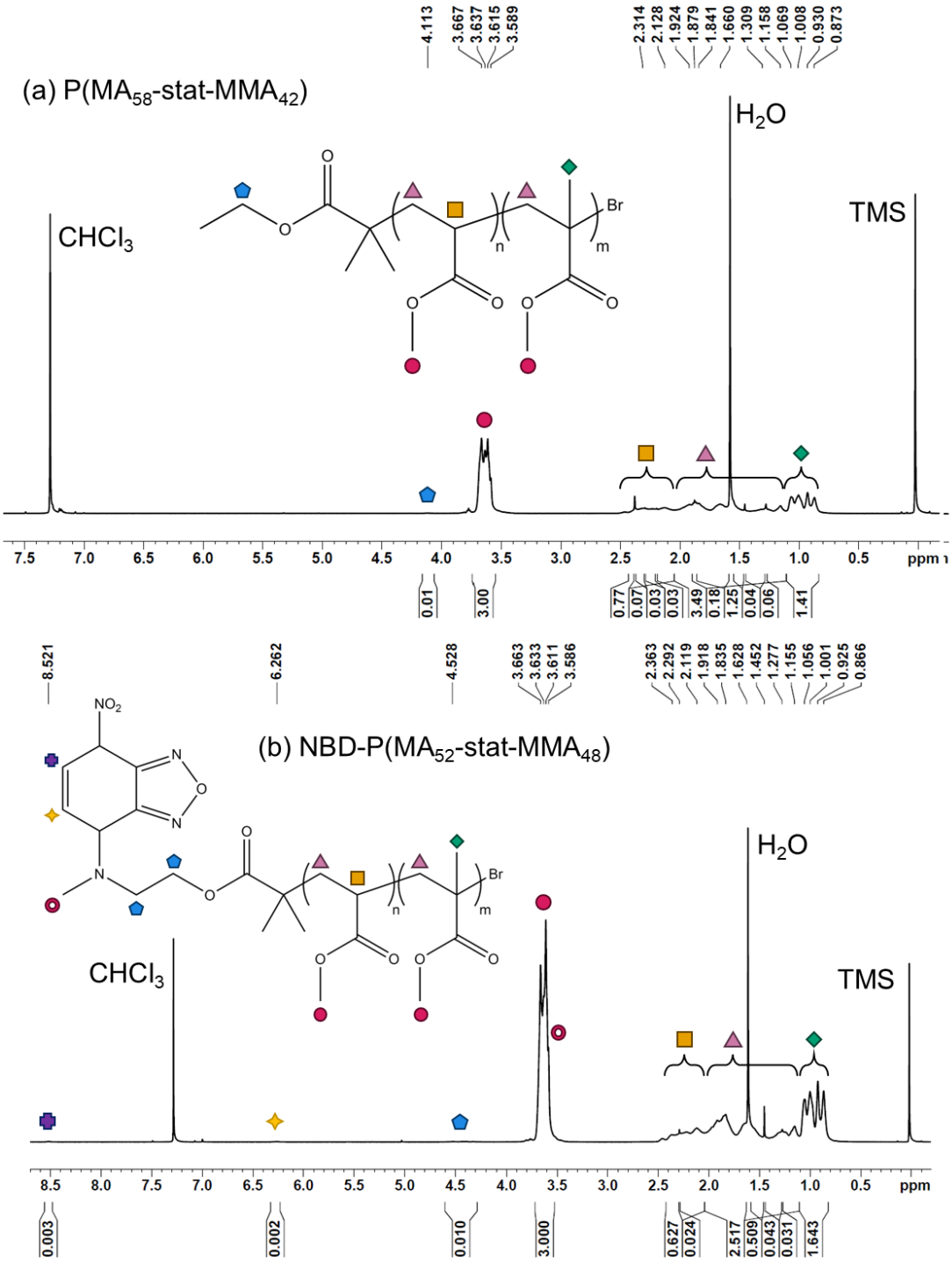
with dry $N_2$. The molar ratio of Total Monomer: PMDETA: Cu(I)Br: EBiB: Toluene was 325:1.5:1:1:540 (MA:MMA was 1:1). The reaction vessel underwent a minimum of five freeze-pump-thaw cycles before equilibrating at room temperature, heating to 65 °C, and reacting over the weekend. The resulting polymers were purified and precipitated by passing through basic alumina columns and 0.45 μm PTFE filters into hot hexanes twice before being collected under reduced pressure.




**Scheme S1**. Reaction scheme showing the synthetic pathway to poly(methyl acrylate – stat – methyl methacrylate) (P(MA-stat-MMA)) and Nitrobenzofurazan (NBD) labelled P(MA-stat-MMA) (NBD-P(MA-stat-MMA)): Synthesis of (a) labelled initiator NBD-BiB modified from Katzenstein et al.[1] and (b) statistical random copolymer via ATRP.

***Nuclear magnetic resonance (NMR) spectroscopy***

Proton NMR measurements were taken using a Bruker 500 MHz NMR instrument in ~4 mg/mL chloroform-*d* (Thermo Scientific, Waltham, MA) solutions.



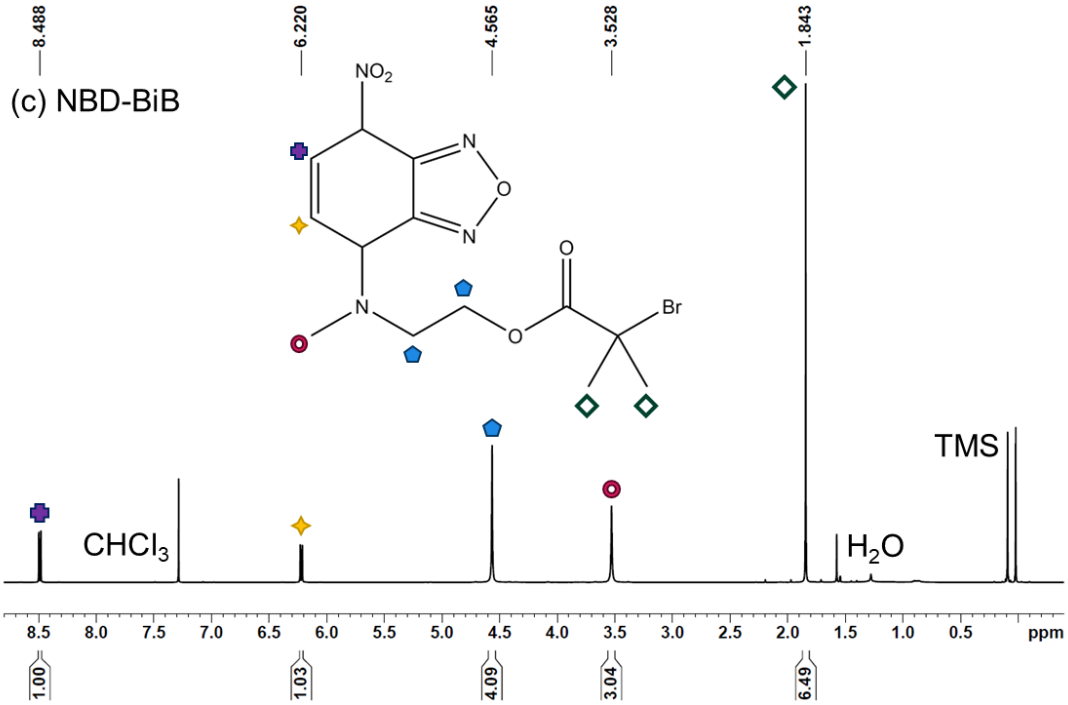



**Figure S1**. 500 MHz $^{1}$H NMR spectra of (a) unlabelled P(MA-stat-MMA), (b) labelled NBD-P(MA-stat-MMA), and (c) NBD-labelled initiator NBD-BiB. Actual monomer ratios were determined by calibration of the methyl peaks at δ ~3.6 ppm (hot pink circles) to an integration of 3, verification with the backbone $CH_2$ peaks at δ ~1-1.9 (pink triangles) ppm to an integration of ~2, and then subsequent calculations with the ratio between the peaks at δ ~2.3 ppm (yellow squares) and δ ~1.0 ppm (green diamonds) (which are the peaks unique to the monomers MA and MMA, respectively). Water and other impurities were subtracted out from the integration values. For D. P. and number-averaged molecular weight ($M_n$) calculations, the $CH_3$ calibration peak ( ~ 3.6 ppm, integration of 3) was compared to either the unlabelled initiator peak at δ ~ 4.5 ppm or the labelled initiator peak at δ ~ 6.3 ppm. Calculations and table containing obtained values are below.

$$MA\% + MMA\% = 100\% \rightarrow MMA\% = 100 - MA\%$$

$$MA\%: MMA\% = \frac{MA\ CH\ integration}{1} : \frac{MMA\ CH_3\ integration}{3} \rightarrow \frac{MA\%}{MMA\%} = \frac{\frac{MA\ CH}{1}}{\frac{MMA\ CH_3}{3}}$$

$$MA\% \times \frac{CH_3}{3} = MMA\% \times \frac{CH}{1} \rightarrow MA\% \times CH_3 = 3 \times (100 - MA\%) \times CH$$

$$MA\% \times (CH_3 + 3 \times CH) = 300 \times CH \rightarrow MA\% = \frac{300 \times CH}{CH_3 + 3 \times CH}$$

$$\frac{\frac{CH_3\ integration}{3}}{\frac{Initiator\ integration}{[2\ (unlabelled)\ or\ 1\ (labelled)]}} = D.P.$$

| | Unlabelled | Labelled |
|---|---|---|
| $CH_3$ calibration | 3 | 3 |
| Backbone $CH_2$ | 1.9577 | 1.9342 |
| MA CH | 0.6454 | 0.6028 |
| MMA $CH_3$ | 1.4081 | 1.6433 |
| %MA | 57.9 | 52.39 |
| %MMA | 42.1 | 47.61 |
| Initiator | 0.0104 | 0.0025 |
| D. P. | 200 | 400 |
| $M_n$ (kDa) | 18.4 | 20 |
| Sample Label | P($MA_{58}$-stat-$MMA_{42}$) | NBD-P($MA_{52}$-stat-$MMA_{48}$) |

***Gel permeation chromatography (GPC)***

GPC was performed using an Agilent Technologies 1260 Infinity, fitted with a Gel 5 µm guard column, a PL Gel 5 µm mix D 1° column, and a PL Gel 5 µm Mix C 1° column. Both unlabelled P(MA-stat-MMA) and labelled NBD-P(MA-stat-MMA) were run in ~3 mg/mL THF solutions at 1 mL/min with a toluene flow marker.

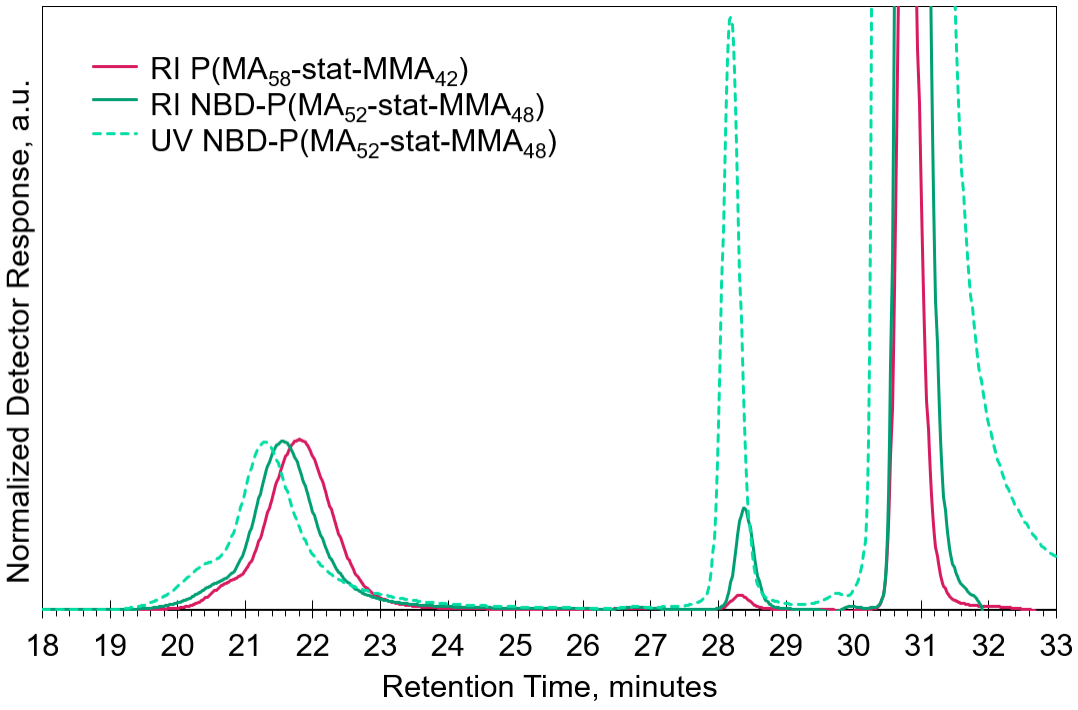



**Figure S2**. GPC traces for both copolymers used in this investigation. Refractive Index (RI) response (solid lines) was measured to calculate molecular weight (table below, PMMA standards) and UV response (dashed line, 488nm laser) was measured to verify the absence of unreacted initiator in the fluorescently labelled sample. Polymer peaks located at retention times 19-24 minutes; butylated hydroxytoluene (BHT) inhibitor (present in the THF solvent used with the instrument) represented by peaks at 29-30 minutes;

toluene flow marker peaks at 30.5-32 minutes. The UV trace displays heightened response of a higher molecular weight (lower retention time) shoulder, which we attribute to termination by combination resulting in two flurophores on a single polymer chain (one at each end). We do not see this as significant cause for concern as the dispersities by RI response remain relatively low, though it is worth noting.

| | $M_p$, kDa | $M_n$, kDa | $M_w$, kDa | Đ, kDa/kDa |
|---|---|---|---|---|
| P($MA_{58}$-stat-$MMA_{42}$) | 30.7 | 28.4 | 33.8 | 1.19 |
| NBD-P($MA_{52}$-stat-$MMA_{48}$) | 41.5 | 36.8 | 44.2 | 1.20 |

***Thermal Gravimetric Analysis (TGA)***

TGA was performed using a TA Instruments Q50 Thermal Gravimetric Analyzer at a heating rate of 10 °C/min. Error is within ± 2 °C. Mass loss was less than 1.5% for all samples up to 145°C.

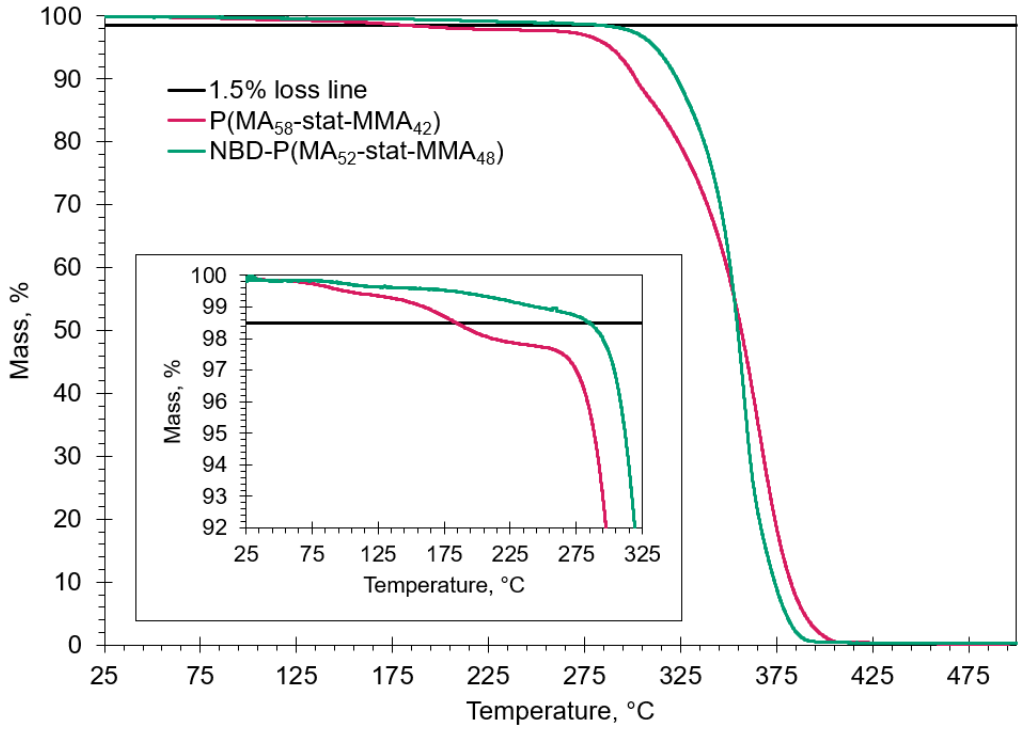



**Figure S3**. TGA curves for both copolymers used in this investigation, with 1.5% mass loss indicated by the solid black line.

***Differential scanning calorimetry (DSC)***

DSC was performed using a TA Instruments Q20 Differential Scanning Calorimeter at a heating/cooling rate of 10 °C/min for the first heat and cool cycles, and then a heating rate of 0.7 °C/min for the second heat cycle (shown). Instrument error is within ± 2 °C.

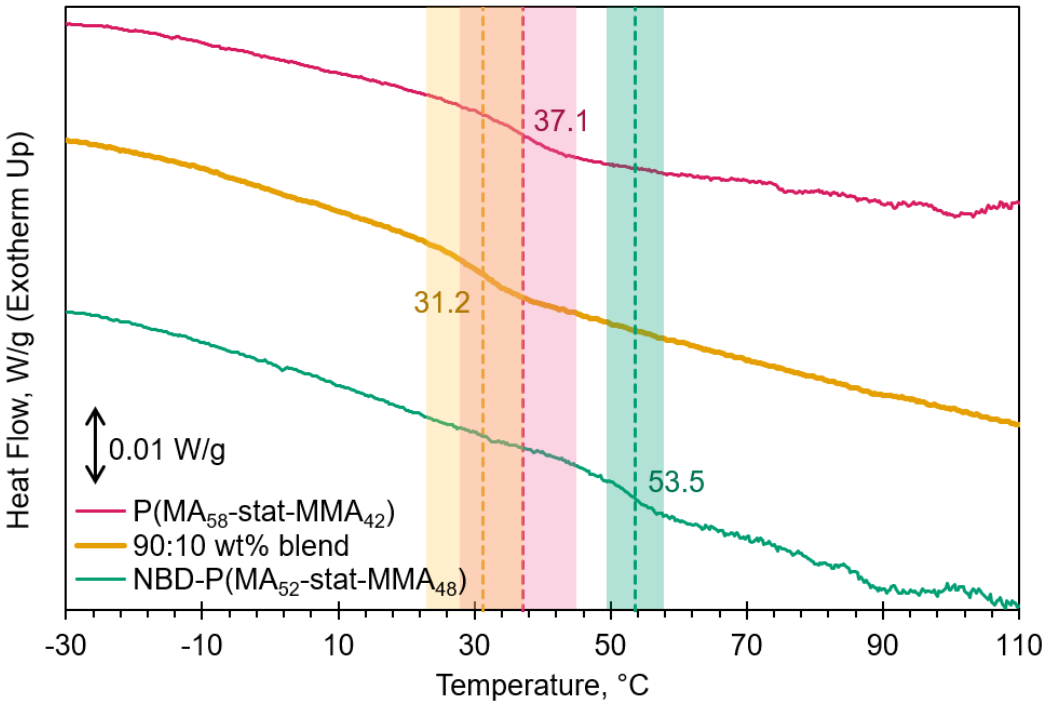



**Figure S4**. Second-heat DSC curves for both copolymers used in this investigation, as well as for a 90 wt% unlabelled to 10 wt% labelled blend, with $T_g$ indicated for each by the vertical dashed lines and the onset and offset (as measured by TA Universal Analysis Software) indicated by the shaded regions. Curves have been vertically shifted for clarity. There are possible secondary $T_g$s between ~80-110 °C for the component copolymers, but they have small magnitude combined with general noise in the higher-temperature region due to the low rate. Homopolymer PMA is reported to have a $T_g$ of -1 ± 1 °C,[5] and PMMA a $T_g$ of 112 °C.[6] There is no visible secondary $T_g$ in the blend system.

| | $T_g$, °C | Onset, °C | Offset, °C |
|---|---|---|---|
| P(MA58-stat-MMA42) | 37.1 | 27.8 | 44.8 |
| 90:10 wt% blend | 31.2 | 22.9 | 37.3 |
| NBD-P(MA52-stat-MMA48) | 53.5 | 49.6 | 57.7 |

***Zero-shear viscosity measurement***

Zero-shear viscosity for our copolymer blend system was measured via rheology using a TA Instruments Discovery HR 10. A frequency sweep was performed at 53, 73, 93, and 120 °C and the viscosity was fit to a single exponential function for extrapolation to zero shear.

$$\eta = A \cdot exp\left(\frac{-f}{t}\right) + y_0 \qquad \textbf{Eq. S1}$$

where $\eta$ is the viscosity, $f$ is the frequency (shear rate), and $A$, $t$, and $y_0$ are the fitting parameters for amplitude, time constant, and vertical offset respectively.

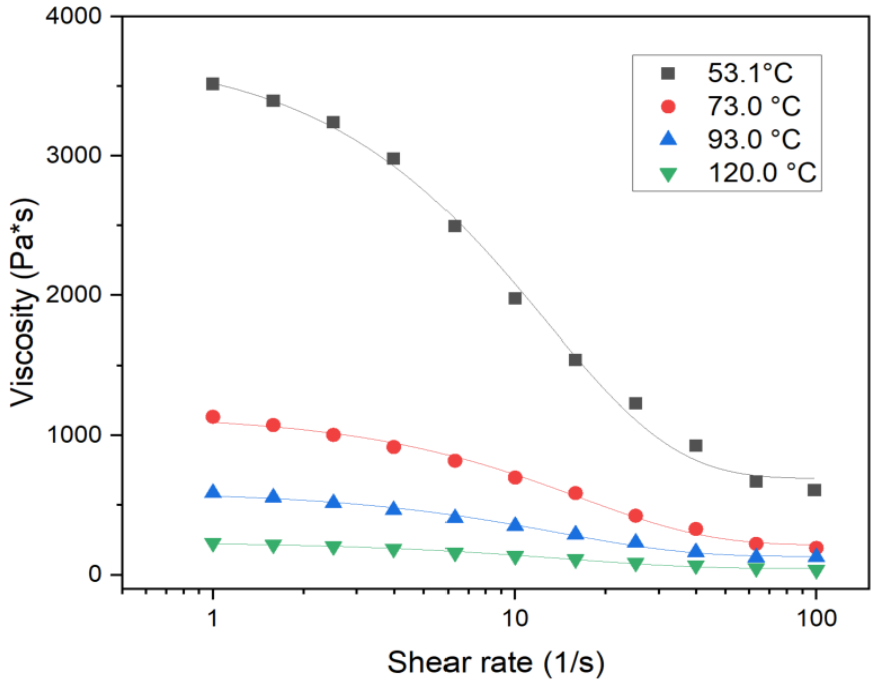



**Figure S5**. Viscosity as a function of shear rate and temperature for blended P($MA_{58}$-stat-$MMA_{42}$) (90 wt%) and NBD-P($MA_{52}$-stat-$MMA_{48}$). The data was fit to a single exponential function (fitting parameters below) from which the zero-shear viscosity $\eta_0$ (also below) was extrapolated.

| Parameter: | **A** | **t** | **$y_0$** | **$\eta_0$** |
|---|---|---|---|---|
| Units: | Pa·s | 1/s | Pa·s | Pa·s |
| **Temperature** | **53.1 °C** | | | |
| Value | 3063.19 | 12.63 | 691.84 | 3755.03 |
| Standard Error | 70.60 | 0.84 | 52.44 | 87.95 |
| **Temperature** | **73.0 °C** | | | |
| Value | 936.47 | 16.21 | 213.89 | 1150.36 |
| Standard Error | 24.44 | 1.26 | 20.63 | 31.98 |
| **Temperature** | **93.0 °C** | | | |
| Value | 466.27 | 13.85 | 132.03 | 598.30 |
| Standard Error | 14.40 | 1.25 | 11.23 | 18.26 |
| **Temperature** | **120.0 °C** | | | |
| Value | 189.37 | 14.65 | 45.94 | 235.31 |
| Standard Error | 6.36 | 1.45 | 5.11 | 8.16 |

***Sample film preparation***

Two films were spin-coated from THF solution (~10 wt% polymer) onto UV-Ozone treated (Jelight, Model 18 UVO Cleaner, 5 minutes) clean glass slides. The polymer components of the solution were composed of 10:90 wt% labelled:unlabelled P(MA-stat-MMA) based on self-quenching measurements below. Samples were spun at a rate of 1000 rpm for 1 minute, and dried and annealed under vacuum (<1 mTorr) at 60°C overnight.

### *Self-quenching test*

To assess the degree of self-quenching, fluorescence intensity was measured using a Nikon Eclipse Ti A1 scanning confocal laser microscope equipped with a 488 nm excitation laser and GaAsP PMTs and hybrid TCSPC detectors to ensure fluorophore concentration stayed below the self-quenching limit. **Figure S6** shows no evidence of self-quenching in blended samples up to ~12.5 wt% labelled polymer (~0.07 wt% NBD); the relationship between intensity and concentration begins to plateau approaching ~25 wt% labelled polymer (~0.14 wt% NBD), which indicates the onset of self-quenching. The intensity detection limit was reached in the 25wt% samples, meaning the severity of the plateau may be less in reality. However, based on this data we hold that samples with less than 12.5 wt% labelled polymer will not self-quench.

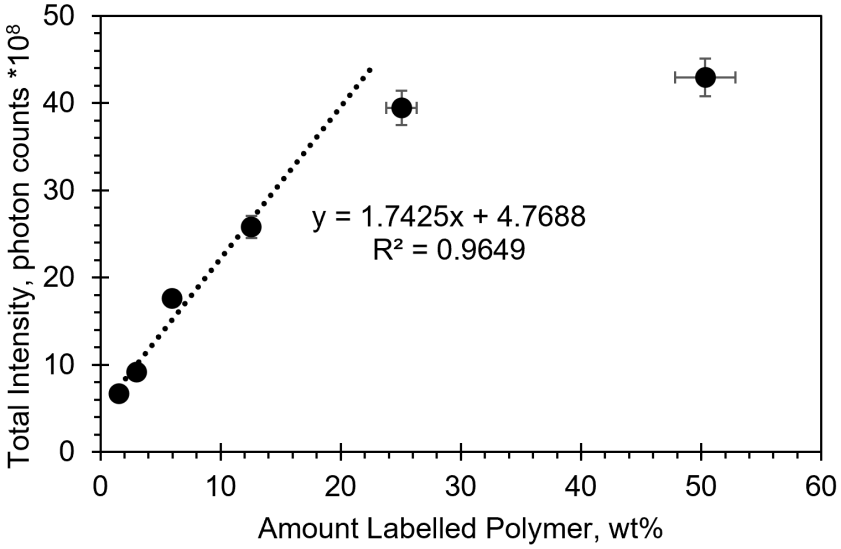



**Figure S6**. Fluorescence intensity as a function of wt% labelled polymer in a series of thin films. Error bars represent 5% error.

### *Ellipsometry*

A single-wavelength (6328Å HeNe Gas Laser) Gaertner Scientific Corporation LSE ellipsometer using Gaertner Ellipsometer Measurement Program (LGEMP) software was used to determine film thicknesses via the Cauchy model.[7] The thicknesses of the two films used in this study were found to be 166.2 ± 5.7 nm and 169.5 ± 8.2 nm, from an average and standard deviation of 5-6 points in the centre of the film.

### *FLIM-FRAPP*

FLIM and FRAPP measurements were conducted on Nikon Eclipse Ti A1 scanning confocal laser microscope using Nikon NIS Elements software. The microscope was equipped with a 488 nm excitation laser, GaAsP PMTs and hybrid TCSPC detectors, and a Tokai Hit TP-CHSQ-C temperature-controlled stage (Thermo Plate Cooling and Heating Controller, CBU Water Cooling Controller). The heating stage temperature was calibrated with a VWR Traceable 2-Channel Thermometer in direct physical contact with the sample, with error ± (0.3% + 1) °C.

### *Tg determination from FLIM*

For FLIM, the field of view was scanned multiple times at each temperature within the full range (~10-50°C) to build a color-coded map of averaged lifetime values; each pixel in the 512×512 image corresponds to one lifetime value for that area on the film.

The temperature was increased stepwise over the course of the FLIM investigations. After every increase, the film was held at temperature and a color-coded map of the sample field of view was generated, where each pixel corresponded to the average lifetime of all photons in that space. Subsequently averaging over the entire area provided one lifetime value per temperature with minimal error (95% confidence interval). Rather than approximating two separate trends with linear fits, we accurately determined the glass transition temperature using the logarithmic hyperbolic cosine equation used by Dalnoki-Veress et al.[8]

$$\tau_{\text{lifetime}} = w\left(\frac{M-G}{2}\right)\cdot \ln\left(\cosh\left(\frac{T-T_g}{w}\right)\right) + \left(T-T_g\right)\cdot\left(\frac{M+G}{2}\right) + c \qquad \textbf{Eq. S2}$$

where the fitting parameters *w*, *M*, *G*, and *c* (**Table S1**) correspond to the width of the transition region, the slope of the rubbery region, the slope of the glassy region, and the y-axis (lifetime) value at $T_{g,\text{lifetime}}$, respectively.[8] To aid in proper fitting, *w* is set to 10 °C. "Initial guesses" and reasonable order-of-magnitude limits were input for each variable, and the built-in Origin Pro fitting program converged the fit with $R^2$ values above 0.980 for all datasets.

**Table S1**. Fitting parameters for all FLIM runs reported.

| | Parameter: | **w** | **M** | **G** | **T** | **c** |
|---|---|---|---|---|---|---|
| ID | Units: | °C | 1/°C | 1/°C | °C | ns/ns |
| 1 | **Run 1** | | | | | |
| | **Value** | 10.00 | -2.46E-04 | -9.27E-05 | 38.21 | 1.00 |
| | **Standard Error** | 0.00 | 1.56E-05 | 5.31E-06 | 2.03 | 3.23E-04 |
| | **Run 2** | | | | | |
| | **Value** | 10.00 | -1.86E-04 | -2.87E-05 | 34.79 | 1.00 |
| | **Standard Error** | 0.00 | 1.61E-05 | 8.88E-06 | 2.52 | 2.59E-04 |
| | **Run 3** | | | | | |
| | **Value** | 10.00 | -1.63E-04 | -2.63E-05 | 37.89 | 1.00 |
| | **Standard Error** | 0.00 | 2.21E-05 | 8.36E-06 | 3.36 | 2.91E-04 |
| 2 | **Run 1** | | | | | |
| | **Value** | 10.00 | -2.44E-04 | -8.77E-05 | 31.55 | 1.00 |
| | **Standard Error** | 0.00 | 1.81E-05 | 1.54E-05 | 3.16 | 5.21E-04 |
| | **Run 2** | | | | | |
| | **Value** | 10.00 | -2.48E-04 | -4.46E-05 | 37.45 | 1.00 |
| | **Standard Error** | 0.00 | 2.99E-05 | 1.15E-05 | 2.98 | 4.03E-04 |

### *D determination from FRAPP*

The self-diffusion coefficient (*D*) was obtained from FRAPP following analysis procedures outlined in Katzenstein et al.[1] An initial region of interest (ROI) in the form of a regular series of lines ~6 μm wide and ~330 μm long was photobleached in the centre of the field of view by the 488 nm excitation laser, generating a sinusoidal intensity profile. Successive fluorescence scans were taken every 15 minutes following the

initial bleach to create an image sequence documenting two-dimensional diffusion over multiple hours; over time as polymer diffusion occurs, the amplitude of the sinusoid decreases. A vertically averaged linear intensity profile was extracted from each image using ImageJ, and input into a Python script (see **Appendix**) for normalization, peak fitting, and calculation of diffusion coefficient.

The Python script first stored and sorted the input .csv files with the raw intensity data. To account for photobleaching over time due to repeated exposure to the excitation laser during imaging, the intensity values for each timestamp were normalized by the average intensity of a ~30000 μm$^2$ "background" area outside of the bleached region (**Figure 2a** in the main manuscript). To account for potential artifacts or bright spots in the film, the self-normalized intensity values for each image were also normalized via subtraction of the intensity values from a pre-bleach scan of the same area (which was similarly self-normalized). Normalized intensity was then plotted versus x-distance, and peaks and valleys in the sinusoid profile were extracted and stored. From these values, the amplitude and wavelength (pitch size) were calculated. The average amplitude associated with each timestamp was then divided by the initial amplitude $A_0$ of the bleached pattern at time $t$=0, and plotted against time. A linear regression was performed, fitting the data to **Equation S3** and obtaining $D$ by way of the slope of the line:

$$\ln\left(\frac{A(t)}{A_0}\right) = \frac{-4\pi^2 D}{\lambda^2} t \qquad \textbf{Eq. S3}$$

where $A(t)$ is the amplitude at time $t$, $A_0$ is the amplitude at time $t$=0, $D$ is the self-diffusion coefficient, and $\lambda$ is the wavelength of the bleached pattern.

***D calculations from literature***

$D$ of entangled P(MA-stat-MMA) has not been experimentally reported in the literature for our molecular weight and temperature. To validate our results, we performed calculations from theory using literature values for associated parameters. Graessley[9] and Pearson et al.[10] propose a calculation method using rheological data by

$$D = \frac{kT\rho N_A R_g^2}{6\eta_0 M_e} \qquad \textbf{Eq. S4}$$

where $D$ is the self-diffusion coefficient, $k$ is the Boltzmann constant, $T$ is the temperature, $\rho$ is the density, $N_A$ is Avogadro's number, $R_g^2$ is the squared radius of gyration of the polymer, $\eta_0$ is the zero-shear viscosity, and $M_e$ is the entanglement molecular weight.

The temperature, $T$, was ~53 °C (326 K) for all FRAPP runs. Fetters et al. provide a density of 1.11 g/cm$^3$ for PMA and 1.14 g/cm$^3$ for PMMA, a mean-squared end-to-end distance over molecular weight ($\langle r^2\rangle/M$, used to calculate $R_g^2$) of 0.436 Å$^2$·mol/g for PMA and 0.425 Å$^2$·mol/g for atactic PMMA, and an entanglement molecular weight of 11.0 kDa for PMA and 10.1 kDa for PMMA.[11] Wu reports $M_e$ to be 9.07 kDa for PMA and 9.2 kDa for PMMA,[12] so the final value used is an average between the two (10.04 kDa for PMA and 9.65 kDa for PMMA). To obtain copolymer estimates, all homopolymer values were averaged according to the sample's monomer composition (for example, the density of a P(MA$_{58}$-co-MMA$_{42}$) sample was estimated as (1.11·0.58) + (1.14·0.42) = 1.12 g/cm$^3$).

Thus, we estimate $D$ for our system to be $3.4\times10^{-12} \pm 8.0\times10^{-18}$ cm$^2$/s.

**References**


1 J. M. Katzenstein, D. W. Janes, H. E. Hocker, J. K. Chandler and C. J. Ellison, *Macromolecules*, 2012, **45**, 1544–1552.
2 M. Semsarzadeh, M. Rostami Daronkola and M. Abdollahi, *J. Macromol. Sci. Pure Appl. Chem.*, 2007, **44**, 953–961.
3 C.-F. J. Kuo, J.-B. Chen, P.-Y. Chen and G. R. S. Dewangga, *Text. Res. J.*, 2019, **89**, 5177–5186.
4 A. Limer and D. M. Haddleton, *Macromolecules*, 2006, **39**, 1353–1358.
5 L. Andreozzi, C. Autiero, M. Faetti, M. Giordano and F. Zulli, *J. Non-Cryst. Solids*, 2006, **352**, 5050–5054.
6 B. Lu, K. Lamnawar, A. Maazouz and H. Zhang, *Soft Matter*, 2016, **12**, 3252–3264.
7 J. N. Hilfiker and J. A. Woollam, in *Encyclopedia of Modern Optics*, ed. R. D. Guenther, Elsevier, Oxford, 2005, pp. 297–307.
8 K. Dalnoki-Veress, J. A. Forrest, C. Murray, C. Gigault and J. R. Dutcher, *Phys. Rev. E*, 2001, **63**, 031801.
9 W. W. Graessley, *J. Polym. Sci. Polym. Phys. Ed.*, 1980, **18**, 27–34.
10 D. S. Pearson, G. Ver Strate, E. Von Meerwall and F. C. Schilling, *Macromolecules*, 1987, **20**, 1133–1141.
11 L. J. Fetters, D. J. Lohse and W. W. Graessley, *J. Polym. Sci. Part B Polym. Phys.*, 1999, **37**, 1023–1033.
12 S. Wu, *J. Polym. Sci. Part B Polym. Phys.*, 1989, **27**, 723–741.

**APPENDIX**: Python Script for Calculation of Self-Diffusion Coefficient

```
# -*- coding: utf-8 -*-
"""
Created on Mon Dec 16 17:41:30 2024
Last updated: Apr 23 2025
"""

import numpy as np
import matplotlib.pyplot as plt
from scipy import stats, signal
import pandas as pd
import sys
import os

# ----------------------------
# Set up directory and filenames
# ----------------------------
output_dir = r # Choose where to store output plots (copy as path -> paste after
"r" - path must be in "")
if not os.path.exists(output_dir):
    os.makedirs(output_dir)

df = pd.read_csv(r)  # Paste file path for intensity values after "r" - path
must be in ""
df_background = pd.read_csv(r)  # Paste file path for background values after
"r" - path must be in ""

# ----------------------------
# Detect which columns to treat as time points
# ----------------------------
# We'll look for columns that can be converted to an integer (e.g. "0", "15",
"30")
# and are present in both data frames. We also skip "prebleach".
time_candidates = []
for col in df.columns:
    if col.lower() == "prebleach":
        continue
    try:
        # If col is numeric (like "0", "15", "30", ...), convert to int
        t = int(col)
        # Only accept this time if it also exists in df_background
        if col in df_background.columns:
            time_candidates.append(t)
    except ValueError:
        # This column name isn't numeric, so ignore it
        pass

# Sort the resulting list of time points
time_points = sorted(time_candidates)

if len(time_points) == 0:
    print("No numeric time columns found in both data files. Please check your
CSVs.")
    sys.exit(1)

print("Detected time points:", time_points)
```

```
# ----------------------------
# Extract "prebleach" arrays
# ----------------------------
if "prebleach" not in df.columns or "prebleach" not in df_background.columns:
    print("'prebleach' column not found in one or both data frames.")
    sys.exit(1)

background = np.array(df["prebleach"])
average_background = np.average(background)

# Build corresponding arrays for each time point,
# making sure to include all you want to analyze
D_arrays = [np.array(df[str(t)]) for t in time_points]            # intensities
D_b_arrays  =  [np.array(df_background[str(t)])  for  t  in  time_points]  #
background

# -------------
# Normalization
# -------------
# Step 1: self-normalized prebleach
avg_b_prebleach = np.average(df_background["prebleach"])
S_N_prebleach = background / avg_b_prebleach

# Step 2: self-normalize each time array by its background
S_N_time_series = []
for i in range(len(D_arrays)):
    avg_b = np.average(D_b_arrays[i])
    S_N_array = D_arrays[i] / avg_b
    S_N_time_series.append(S_N_array)

# Step 3: full normalization: subtract S_N(prebleach) from S_N_time
normalized_intensities = []
for i in range(len(S_N_time_series)):
    # Important to ensure indexing is consistent
    # If the number of cells/traces changed over time, you may need a length
check
    norm_i = S_N_time_series[i] - S_N_prebleach
    normalized_intensities.append(norm_i)

# -------------
# Parameters
# -------------
scale_factor = 1 / 1.61  # for x-axis distance, microns, etc. - 1.61 microns =
1 px

# Lists to store results
amplitude_fp_mean = []
average_wavelengths = []
all_data_for_combined = []

# -------------
# Peak detection
# -------------
for i, y_data in enumerate(normalized_intensities):
    current_time = time_points[i]
```

```
# Because we sometimes get incomplete columns or other issues,
# protect against empty arrays:
if len(y_data) == 0:
    amplitude_fp_mean.append(np.nan)
    average_wavelengths.append(np.nan)
    all_data_for_combined.append(y_data)
    continue

# Improved peak detection:
# 'prominence' is often more robust than a fixed 'height' threshold.
# Adjust these values as needed for your data.
peaks, _ = signal.find_peaks(
    y_data,
    prominence=0.05,   # tries to ensure only "significant" peaks are found
    distance=10       # minimum distance between peaks (tweak as necessary)
)
# For valleys, invert y_data and do the same
valleys, _ = signal.find_peaks(
    -y_data,
    prominence=0.05,
    distance=10
)

# Optionally, ignore endpoints if you wish:
# e.g. drop any peak/valley at the first or last few data points
# to avoid false detection near boundaries.
# Example: require that peaks be at indices 2:-2, etc.
valid_peaks = [p for p in peaks if p > 2 and p < (len(y_data) - 2)]
valid_valleys = [v for v in valleys if v > 2 and v < (len(y_data) - 2)]
valid_peaks = np.array(valid_peaks)
valid_valleys = np.array(valid_valleys)

if len(valid_peaks) == 0 or len(valid_valleys) == 0:
    print(f"No peaks or valleys detected at time {current_time} minutes.")
    amplitude_fp_mean.append(np.nan)
    average_wavelengths.append(np.nan)
    all_data_for_combined.append(y_data)
    continue

# Extract the intensities at those indices
max_intensities = y_data[valid_peaks]
min_intensities = y_data[valid_valleys]

# If you still need the "paired" approach (peak-valley-peak-valley...),
# you can do additional logic here. A simpler approach is to measure
# average amplitude from all recognized peaks & valleys.
# For example:
#   amplitude ~ average(peak) - average(valley)
# But if your analysis requires a strict pairing, you'll need more logic
# (e.g., interleave them, ensure same length, etc.)

mean_peak = np.mean(max_intensities)
mean_valley = np.mean(min_intensities)
amplitude_this_time = mean_peak - mean_valley

# If you want to measure wavelength, you can measure peak-to-peak or
# valley-to-valley distances. The simplest approach is average difference:
```

```
    if len(valid_peaks) > 1:
        peak_dists = np.diff(valid_peaks)
        valley_dists = np.diff(valid_valleys)
        # Example of a naive "average wavelength" from mean of peak-peak and
valley-valley
        avg_wavelength = np.mean([np.mean(peak_dists), np.mean(valley_dists)])
    else:
        avg_wavelength = np.nan

    amplitude_fp_mean.append(amplitude_this_time)
    average_wavelengths.append(avg_wavelength)
    all_data_for_combined.append(y_data)

    # ----------------------
    # Create individual plots
    # ----------------------
    plt.figure(figsize=(8, 4), dpi=150)
    x_axis = np.arange(len(y_data)) * scale_factor
    plt.plot(x_axis, y_data, label='Data')
    plt.scatter(valid_peaks * scale_factor, max_intensities, color='red',
marker='^', label='Peaks')
    plt.scatter(valid_valleys * scale_factor, min_intensities, color='green',
marker='v', label='Valleys')
    plt.title(f"Peak/Valley Plot for Timepoint {current_time} min")
    plt.xlabel("Distance (µm)")
    plt.ylabel("Normalized Intensity (a.u.)")
    plt.legend(loc='best')
    plt.tight_layout()
    outname = os.path.join(output_dir, f"dataset_{current_time}.png")
    plt.savefig(outname, dpi=300)
    plt.show()

# -------------
# Amplitude vs. Time
# -------------
# amplitude_fp_mean[i] corresponds to time_points[i].
# If the first amplitude corresponds to time=0, that is often A0.
# If your time=0 amplitude is indeed A0, do:
A0 = amplitude_fp_mean[0] if not np.isnan(amplitude_fp_mean[0]) else np.nan

# Build an array of ln(A(t)/A0)
log_normalized_amplitude = []
time_for_regression = []

for i, A_t in enumerate(amplitude_fp_mean):
    if i == 0:
        # skip the first point to exclude ln(A(0)/A0)
        continue
    # Protect against NaN or zero
    if A0 is not None and A0 != 0 and not np.isnan(A_t) and not np.isnan(A0):
        log_normalized_amplitude.append(np.log(A_t / A0))
        time_for_regression.append(time_points[i])

if len(log_normalized_amplitude) > 2:
    # Linear regression
    slope, intercept, r_value, p_value, std_err = stats.linregress(
        time_for_regression, log_normalized_amplitude
```

```
    )
    r_squared = r_value**2
else:
    slope, intercept, r_squared = np.nan, np.nan, np.nan

#   D = (((avg wavelength * 10^-4)^2) * slope) / (-4 pi^2)
mean_wavelength_in_cm = ((np.nanmean(average_wavelengths) * scale_factor)* 1e-
4)  # convert from px to microns to cm
stdv_wavelength = (np.std(average_wavelengths) * scale_factor) * 1e-4
Diffusion_coefficient  =  ((mean_wavelength_in_cm**2)  *  (slope/60))  /  (-4  *
np.pi**2)

print("Wavelength (cm):", mean_wavelength_in_cm)
print("Wavelength Std. Dev. (cm):", stdv_wavelength)
print("Slope (1/s):", slope/60)
print("Slope Error (1/s):", std_err/60)
print("Intercept:", intercept)
print("R²:", r_squared)
print("Diffusion coefficient (cm²/s):", Diffusion_coefficient)
print("Plot Data (ln(A(t)/A0) vs t)", log_normalized_amplitude,
time_for_regression)

# Prepare data for plotting the fit line
y_fit = []
for t in time_for_regression:
    y_fit.append(slope * t + intercept)

# -------------
# Plot the decay
# -------------
plt.figure(figsize=(8, 4), dpi=150)
plt.scatter(time_for_regression, log_normalized_amplitude, label='Experimental
data')
if len(time_for_regression) == len(y_fit):
    plt.plot(time_for_regression, y_fit, label='Linear fit', linestyle='--')

plt.title("Exponential Decay of Normalized Amplitude Over Time")
plt.xlabel("Time, minutes")
plt.ylabel("ln(A(t)/A\u2080), a. u.")
equation_text = (
    f"ln(A(t)/A\u2080) = {slope:.4e}·t + {intercept:.4e}\n"
    f"R² = {r_squared:.4f}"
)
plt.text(
    0.05, 0.2, equation_text,
    transform=plt.gca().transAxes,
    fontsize=10, verticalalignment='top',
    bbox=dict(boxstyle="round", fc="w")
)
plt.legend(loc='best')
plt.tight_layout()
plt.savefig(os.path.join(output_dir, "decay_plot.png"), dpi=300)
plt.show()

# -------------
# Combined Plot
# -------------
```

```
plt.figure(figsize=(10, 5), dpi=150)
for i, data_line in enumerate(all_data_for_combined):
    x_axis = np.arange(len(data_line)) * scale_factor
    plt.plot(x_axis, data_line, label=f"{time_points[i]} min")
plt.title("Combined Data Plot (No Peak/Valley Markers)")
plt.xlabel("Distance (μm)")
plt.ylabel("Normalized Intensity (a.u.)")
plt.legend(loc="upper right", fontsize="x-small")
plt.tight_layout()
plt.savefig(os.path.join(output_dir, "combined.png"), dpi=300)
plt.show()
```