PS Poly: A chain tracing algorithm to determine persistence length and categorize complex polymers by shape

Elizabeth A. Conley[1], Creighton M. Lisowski[1], Katherine G. Schaefer[1], Harrison C. Davison[1], Julie Baguio[1], Ioan Kosztin[1] and Gavin M. King[1,2,3]

[1] Department of Physics and Astronomy, University of Missouri-Columbia, Columbia, MO 65211 USA

[2] Department of Biochemistry, University of Missouri-Columbia, Columbia, MO 65211 USA

[3] Materials Science and Engineering Institute, University of Missouri-Columbia, Columbia, MO 65211 USA

Abstract

The fundamental molecules of life are polymers. Prominent examples include nucleic acids and proteins, both of which exhibit a large array of mechanical properties and three-dimensional shapes. The bending rigidity of individual polymers is quantified by the persistence length. The shape of a polymer, dictated by the topology of the polymer backbone - a line trace through the center of the polymer along the contour path – is also an important characteristic. Common biomolecular architectures include linear, cyclic (ring-like), and branched structures; combinations of these can also exist, as in complex polymer networks. Determination of persistence length and shape are largely informative to polymer function and stability in biological environments. Here we demonstrate Persistence length Shape Polymer (PS Poly), a near-fully automated algorithm designed to obtain polymer persistence length and shape from single molecule images obtained in physiologically relevant fluid conditions via atomic force microscopy. The algorithm, which involves image reduction via skeletonization followed by end point and branch point detection, is capable of rapidly analyzing thousands of polymers with subpixel precision. Algorithm outputs were verified by analysis of deoxyribonucleic acid, a very well characterized macromolecule. The method was further demonstrated by application to candidalysin, a recently discovered and complex virulence factor from *Candida albicans*. Candidalysin forms polymers of highly variable shape and contour length and represents the first peptide toxin identified in a human fungal pathogen. PS Poly is a robust and general algorithm. It can be used to extract fundamental information about polymer backbone stiffness, shape, and more generally, polymerization mechanisms.

1. INTRODUCTION

Knowledge of cellular function and dysfunction (disease) has advanced through developing a detailed understanding of many semi-flexible polymeric molecules. A prime example is the recently discovered peptide toxin candidalysin (CL), which is the virulence factor secreted by the fungus *C. albicans* [1]. CL forms loops in solution which can then embed into membranes and form pores that damage host cells [2]. When establishing the molecular basis of a disease, such as invasive candidiasis which emanates from *C. albicans* and has a high mortality rate [3], characterizing polymer mechanical properties and topology (shape) provides significant insight. For example, bending rigidity, quantified by the persistence length, sheds light on the expected diameter of loops formed by CL. This geometric information can be used to predict what size molecules might be able to pass through the CL pores in host cell membranes during *C. albicans* infection. Additionally, when studying the kinetics of polymer loop formation (cyclization) or branching, which usually involves secondary polymerization interfaces [4,5], it is informative to separate and quantify polymers by shape so that distinct reactions can be isolated. Such analyses can be used to construct kinetic models of looping and branching, to determine under what conditions polymer cyclization occurs, and to explore how conversion of linear polymers to looped polymers can be controlled.

Atomic Force Microscopy (AFM) is a powerful single molecule imaging technique employed in micro/nanoscale biophysical investigations and has been used to shed light on polymer persistence length, l_P [6–9]. Analysis typically requires a stack of AFM image data containing many individual polymers. To perform l_P calculations, it is necessary to extract the coordinates along the chain contour, or "backbone", for each polymer in the analysis. Once these coordinates are

obtained, a polymer physics model such as the worm-like chain (WLC) can be used to deduce the persistence length through calculating mean-square end-to-end distances or correlations of backbone tangent vectors [10].

Existing software for polymer detection and characterization in AFM image data suffer from limitations [11–21]. For example, popular tools including EasyWorm [11], Skan [21], and AutoSmarTrace [19], do not simultaneously provide robust feature extraction, skeletonization, shape, and mechanical property calculations such as persistence length. While EasyWorm is suitable for analysis of simple, unbranched polymer chains, it does not determine polymer shape, requires significant manual input, and is only available to the MATLAB community; thus limiting its accessibility and adaptability for open-source workflows. AutoSmartTrace, also MATLAB-dependent, requires manual intervention for thresholding and feature identification, reducing reproducibility across diverse polymer morphologies. While Skan provides Python-based skeletonization and network analysis, its focus on general image processing necessitates extensive customization for polymer-specific applications, lacking built-in modules for persistence length quantification. These limitations underscore the need for an open-source, flexible, modular, and easily extensible solution for the automated detection and analysis of polymer structures recorded through AFM imaging.

PS Poly, introduced here, attempts to fill these gaps and is well suited for studies of complex polymerization processes such as those underlying the host cell attack mechanism of *Candida albicans* [2,22,23]. PS Poly is designed for polymer backbone isolation with sub-pixel precision, automated persistence length calculation, and shape categorization (e.g., linear, looped,

branched, branched-looped, or overlapping chains). A workflow of the algorithm is shown (**Fig. 1**). The program is open-source with code written in both Python and Igor Pro 7 (WaveMetrics, Inc.) [24] and is near-fully automatic, requiring only basic information from the user such as the pixel resolution (nm/pixel) of the source images. PS Poly eliminates manual input post-thresholding, and outputs quantitative metrics such as total polymerized length and branch point coordinates. Persistence length results were verified by comparison to established values and were robust to moderate levels of added noise. The use of a Python framework may enhance accessibility, automation, open-source development, and modularity. Thus, the algorithm represents a step toward a standardized tool for AFM-based polymer analysis.

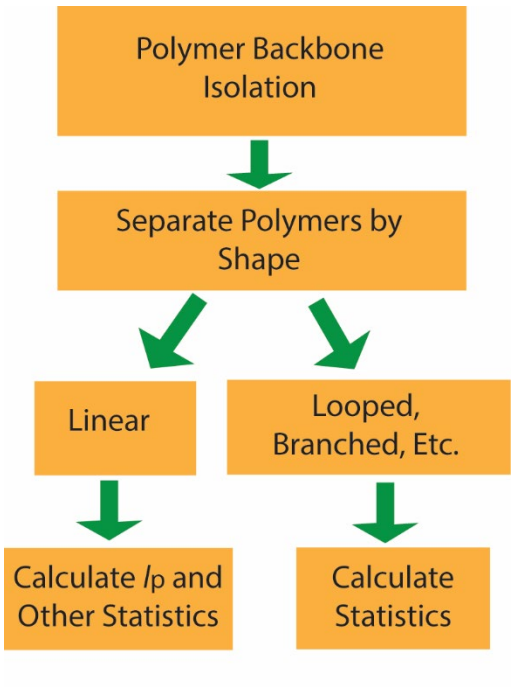


FIGURE 1. Program Overview. Images are processed and reduced to polymer backbone coordinates, then individual features are separated based upon shape. Usually, only measurements of linear polymers are considered when calculating persistence length, l_p.

2. METHODS

In this section we first describe the algorithm as it was originally written in Igor Pro 7, subsequently referred to as PS-Poly. We then turn attention to the Python implementation, PsPolypy. The methods used for calculating the polymer persistence length and uncertainty are also discussed. Finally, we describe the techniques used for acquiring AFM images of CL in near-native conditions.

2.1 Igor Implementation: PS-Poly

Here we describe the Igor Pro 7 implementation, PS-Poly. Briefly, in this algorithm the images are skeletonized, a convolutional filter is used to identify endpoints, and a pathfinding algorithm is used to determine the coordinates of all linear particles used for persistence length analysis. To separate features by shape, filters were developed to identify branch points and distinguish branches from cyclic or looped polymers. These steps are described below. We assume that image preprocessing such as background subtraction has already been applied to the raw AFM image data prior to PS-Poly analysis.

Particle segmentation & polymer backbone isolation

The program begins by loading full-field images. Isolating the polymer backbone is the next step, illustrated in **Figure 2**. This is achieved automatically, by employing Otsu's method [25], or manually, with user defined threshold corresponding to pixel intensity, which is proportional to topographical height, z, of the polymer in the AFM image. In either case, a binary "mask" image is created where all values above and below the threshold are set to 1 and 0, respectively. Then, if upscaling is desired, a copy of this mask is made with a higher pixel density by creating a new

image with a specified scaling factor. The size of the expanded mask has dimensions scaled by the specified scaling factor, and each pixel in the original mask is taken up by a block of *x* scaling pixels in the expanded mask. This allows the result to be obtained with a subpixel level of accuracy which can be valuable for characterizing short polymers. A "skeleton" of the mask is created through a surface thinning algorithm which eliminates layers of the image until only single pixel linewidth traces remain (**Fig. 2D**) [26].

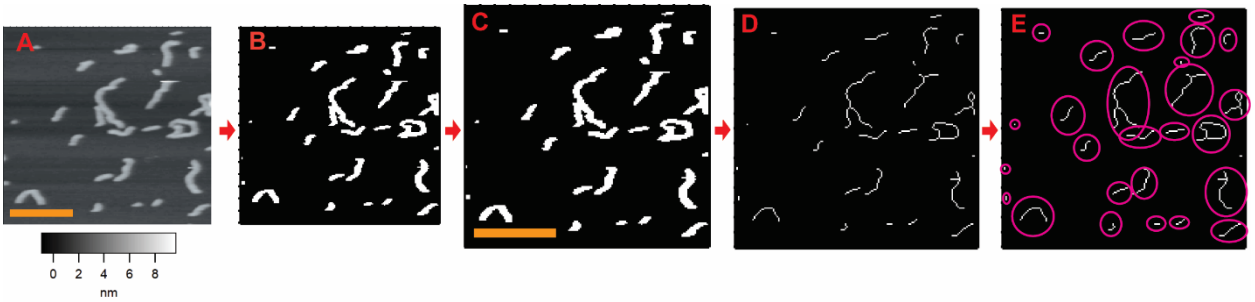


FIGURE 2. Polymer backbone isolation procedure. The first steps to perform PS-Poly calculations on an AFM image of candidalysin are shown. Scale bars (yellow) are 100nm. (**A**) Raw AFM image with topographic height, *z,* shown in greyscale. (**B**) Mask created from the AFM data. (**C**) New mask that has been expanded to a higher pixel density. (**D**) Skeleton created from the expanded mask. (**E**) Each molecule is registered as a unique object (pink circles).

Acquiring polymer coordinates

To obtain separate lists of coordinates for each molecule, we begin by looping through each pixel on a duplicate of the thinned image. Once a 1-valued pixel is found, that coordinate is stored. Then a flood-fill algorithm fills-in all 1-valued pixels which are continuous with that coordinate. This process continues in a loop until the duplicated image is entirely 0-valued, and the resulting list of coordinates correspond to exactly one "seed" pixel per molecule. Then, depth-first search

149 (DFS) is applied in a square around the seed pixel [27]. DFS is used to identify connected pixels
150 and provides detailed information about each connected component. The DFS algorithm explores
151 all possible paths stemming from one input coordinate until a path is found to another input
152 coordinate. The implementation of DFS in this program returns 1 if a path is found between the
153 two coordinates, and 0 if there is no possible path. The search radius is incremented with each
154 loop iteration, and the loop breaks once all locations continuous with the seed pixel are found.
155 We found that applying DFS, as opposed to checking for continuity with all of the one-valued
156 pixels in the image, reduces computational time significantly.
157
158 ***Sorting polymers by shape***
159 Polymers are sorted based on their shape: Linear, Looped, Branched, and Branched-Looped.
160 Examples of these four primary polymer classes are shown in AFM images of CL (**Fig. 3**). This
161 sorting first requires polymer termination point identification, achieved through convolutional
162 filtering.
163

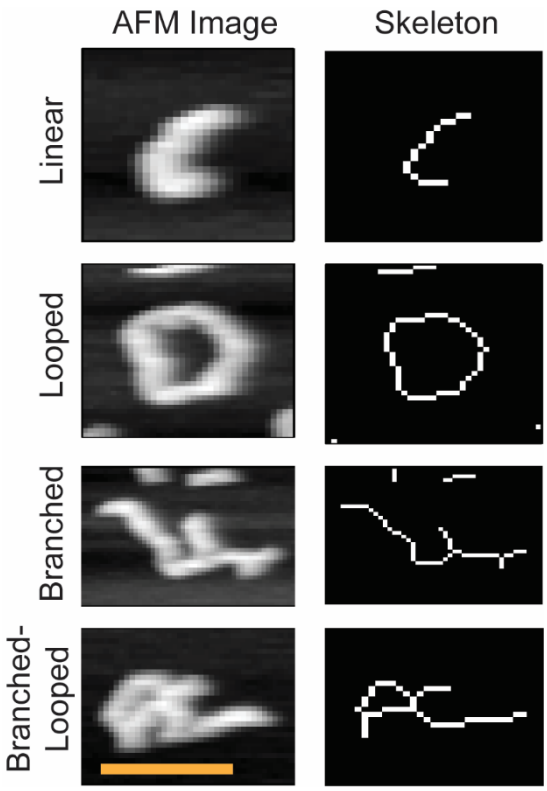


FIGURE 3. Primary polymer types. Examples of the different particle types identified through PS-Poly. Raw AFM image data of CL is shown next to processed skeleton images for Linear, Looped, Branched, and Branched-Looped polymers. The scale bar spans 50 nm and applies to all images.

Polymer end point determination

The algorithm loops through the image, cropping 9 × 9 pixel areas surrounding each central pixel test point. **Figure 4A** demonstrates the pixel grids that are created for every 1-valued pixel in the image. There are 16 possible endpoint configurations because there are 8 possible neighboring pixels, and so for one neighbor, there are 8 possible combinations. For 2 neighbors, there are 16 possible combinations, but only half of them are endpoints because the 2 neighboring pixels must also be adjacent to each other in order to be an endpoint. The pixel grids corresponding to the 16 possible endpoint configurations are shown in **Figure 4B**. Each pixel grid is compared

with each of the 16 endpoint grids and if any one of them is an exact match, then that coordinate is considered to be an endpoint.

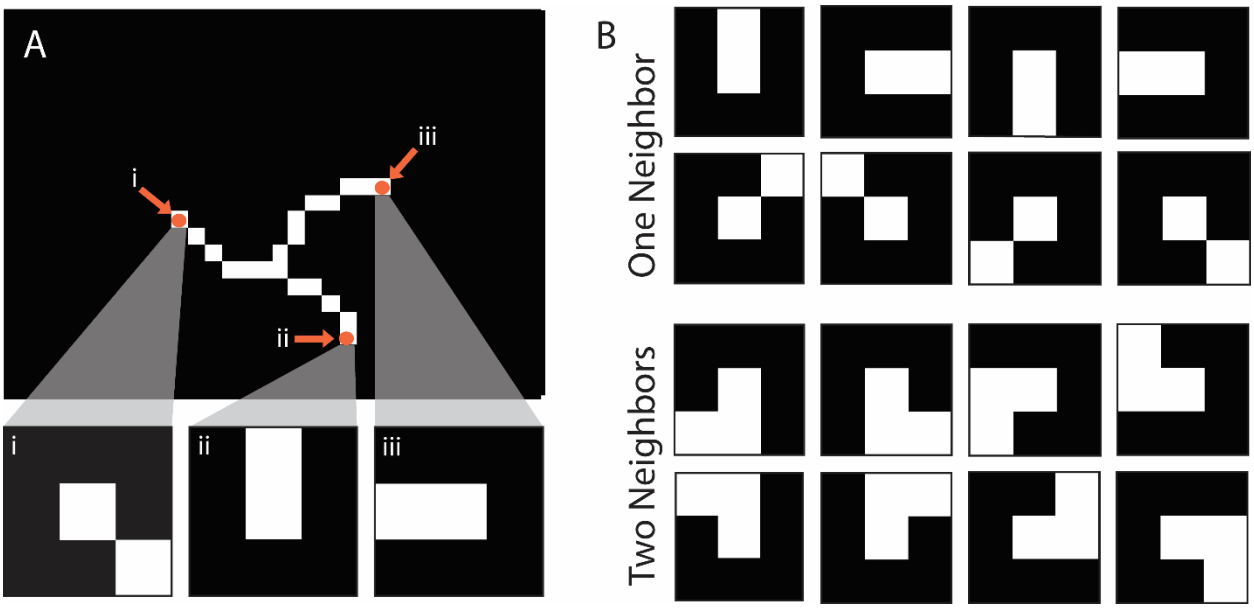


FIGURE 4. **Convolutional filtering for end point determination**. (**A**) shows 9 x 9 pixel grids that are created from points *i*, *ii*, and *iii* on a skeletonized polymer. (**B**) shows the sixteen different shapes that were used in the convolutional filter that finds endpoint coordinates. Shapes are clustered via number of neighbors.

Branch point identification

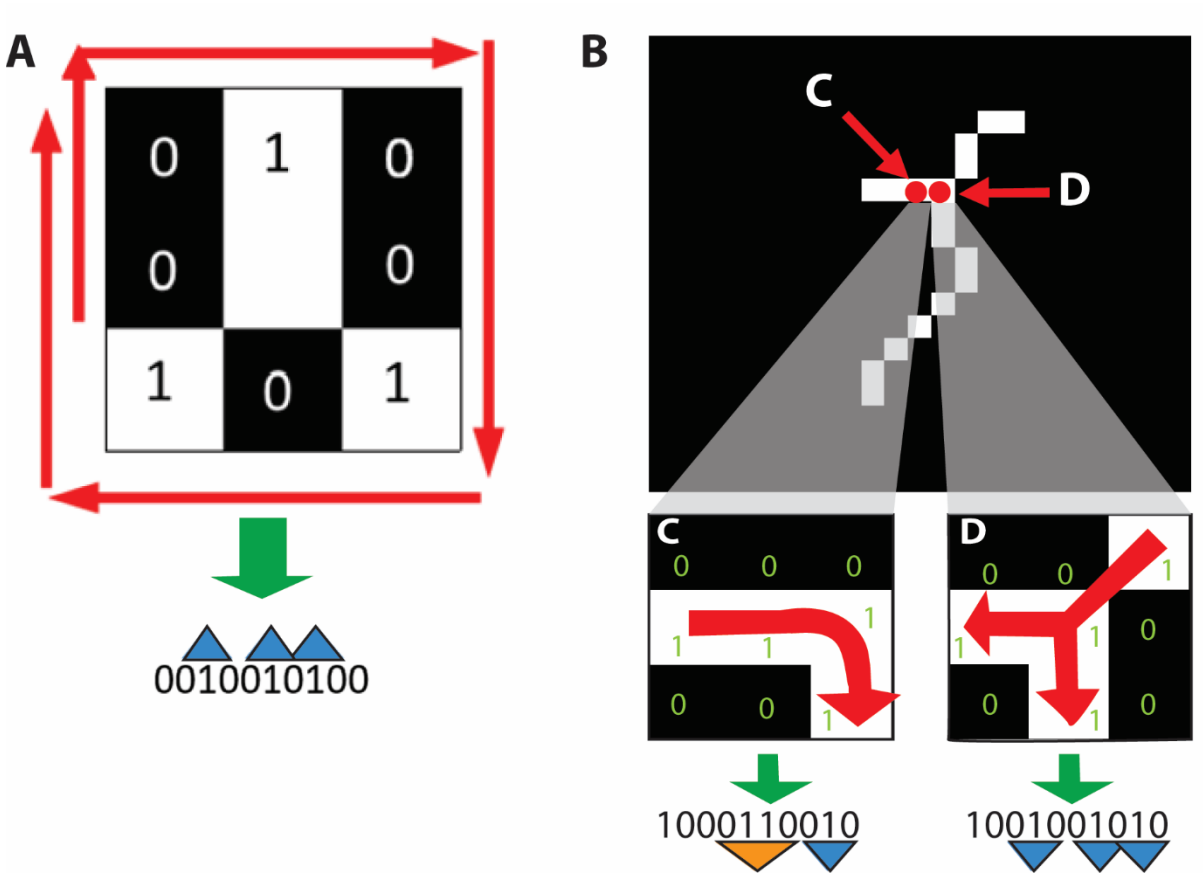
Following endpoint detection, branch points are identified through a filter that works by creating an array corresponding to neighboring pixel values that circle about a central pixel of interest (**Figure 5A**). Each path comprises a clockwise 10-pixel-long sweep. A unique path is counted for each 1 value that is abutted by 0 values (**Figure 5A**, blue triangles). We note that the last two pixels in the path are repeats of the first two, allowing evaluation of the starting point of the array. The array is examined for unique paths. If three or more unique paths are found, then the

193 pixel of interest is determined to be a branch point. The algorithm is also prevented from

194 overcounting branch points, as not all points with three neighbors are true branch points

195 (**Figure 5B**).

196



197

198 **FIGURE 5. Branch point determination**. (**A**) Each pixel in the image is centered in a 3 x 3 grid and a

199 string of 1's and 0's begins at any point neighboring the test pixel. In this example three unique paths

200 (blue triangles) are found, identifying the center pixel as a branch point. (**B**) Technique for prevention of

201 branch point overcounting. Test point *D* has 3 neighboring 1-valued pixels, each of which corresponds to

202 a unique path. Test point *C* also has 3 neighboring 1-valued pixels, but it does not exhibit three unique

203 paths and is thus not a branch point. The non-unique path is marked by the orange triangle.

204

205 ***Shape calling, overlapped identification, and total polymer length determination***

If a polymer has exactly two endpoints and no branch points, it is considered to be linear. If there are no endpoints and no branch points, then it is considered to be a loop. Further, polymers that branch and loop are separated from those which branch but do not loop. We further differentiated true branch points from segments where polymers drape over themselves as they adsorb to the imaging surface. These overlapped polymers were identified via co-localization of branch points with topographically high points along the polymer backbone (**Figure 6**).

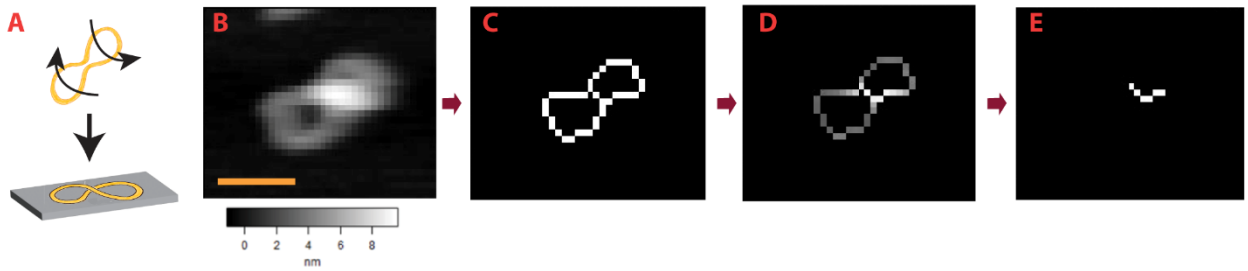


FIGURE 6. Process for determining overlapped polymers. (**A**) Cartoon showing an example of how an overlapped polymer may form. (**B**) AFM image of CL polymer. The height is shown in greyscale, the lateral scale bar (yellow) is 20 nm. (**C**) Skeleton created from the data in panel B shows a junction comprising the intersection of 4 branches. (**D**) New skeleton that incorporates height information. (**E**) Skeleton created at a threshold of 1.5 times the average height of the polymer distal from the junction identifies the polymer as overlapped.

The total polymerized length is found by applying a pathfinding algorithm which sums all pixels in each feature. For overlapped particles, the length is computed by adding the length of the original skeleton with the skeleton made at a threshold of 1.5 times the average height of all pixels on the skeleton with incorporated height information. If over 80% of the polymer is above 1.5 times the average height of the polymer backbone, then it is sorted separately as a noise

particle. Such features could be aggregates or other artifacts. Polymers with high points that are not overlaps are still sorted by their shape and stored in a separate folder.

2.2 Python Implementation: PsPolypy

Preprocessing, Image Loading, & Upscaling

We assume that any image preprocessing including background subtraction has already been applied. PsPolypy uses as input a list of full-field images (denoted as I_n) that contain polymer particles. The images I_n may have different pixel resolution ($N_n \times M_n$) but the same fixed real-space resolution (Res), measured in nanometers per pixel (nm/px). Each image is first converted to normalized grayscale, with pixel intensities $0 \leq I_n \leq 1$. Optionally, users can upscale the pixel resolution of the images using k-order interpolation, as implemented in scikit-image [28,29]. For a user-defined magnification factor γ, the pixel and the real-space resolution of the upscaled image becomes ($\gamma \cdot N_n \times \gamma \cdot M_n$) and Res/$\gamma$ (nm/px), respectively. This optional step allows for finer image details to be analyzed.

Particle Segmentation

Each image I_n in the set undergoes particle segmentation through a multi-step process. Initially, Otsu thresholding is applied to create a binary mask B_n. This mask then undergoes connected-component labeling, segmenting it into distinct regions, each corresponding to a unique polymer particle, with a well-defined bounding box. To ensure complete particle representation, any region whose bounding box touches the edge of the full-field image is discarded, as these particles may be partially cut off. The original image and binary mask are then cropped according to the bounding boxes of all remaining regions. Finally, a list of particle objects is

created, with each object containing the cropped image of an individual particle and its corresponding cropped binary mask.

Skeletonization

The skeleton of each polymer particle (**Fig. 7**, *green*) is obtained by applying the skeletonize method from scikit-image to the corresponding binary mask. The resulting skeleton is then analyzed using Skan [21], which automatically determines the topology (e.g., liner, branched, looped or cyclic) and geometric features (e.g., end-to-end distance, contour length) of the particle. We denote the set of all paths as P, and the m-th path as P_m. This skeletal analysis captures the main features of the particle's structure and morphology.

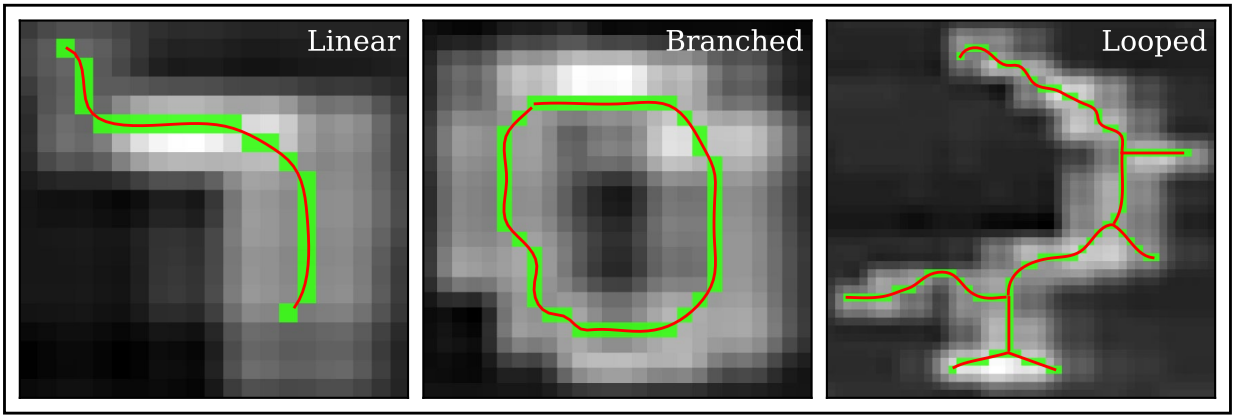


FIGURE 7. Skeletonization and interpolation of different polymer topologies. Examples of linear, looped, and branched polymers are shown along with the skeleton (*green*) and interpolation (*red*).

Classification

Polymers are classified into one of six categories based on their skeleton's structure: Linear, Branched, Looped, Branched-Looped, Overlapped, or Unknown. If the skeleton contains a single

path with distinct endpoints, the particle is classified as Linear (see **Fig. 7**). If the skeleton has a single path where the start and end points coincide, the particle is classified as Looped. For skeletons with multiple paths, each crossing is checked whether it is a branch junction or an overlap, defined by having a height 1.5 larger than the average polymer backbone height distal to the crossing. Polymers with at least one overlap are classified as Overlapped (as shown in **Fig. 6**). For the rest of the skeletons, if no combination of paths forms a cycle (as determined using the NetworkX graph representation of the skeleton) [30], the particle is classified as Branched. Conversely, if multiple paths are present and at least one cycle exists, the particle is classified as Branched-Looped. If none of the above criteria are met, the particle is categorized as Unknown. After classification, users may select only the types of particles they wish for further analysis.

Interpolate Skeletons

Each path (longer than 3 pixels) within each particle (digitized skeleton) undergoes cubic B-spline interpolation by employing the SciPy's splev function [31]. The coordinates along the interpolated skeleton path are given by $x(t)$ and $y(t)$, where t is the distance along the contour. The interpolated paths are sampled at user-defined intervals dt along the contour. The interpolated skeleton provides a more precise representation of the particle compared to the original (digitized) skeleton (**Fig. 7**, compare *red curve* with *green pixels*). For each sampling point along the interpolated path, both the position $\vec{r}(t) = (x(t), y(t))$ of the point and the tangent unit vector $\vec{\tau} = (\dot{x}(t), \dot{y}(t))$ to the path are recorded.

Mean end-to-end distance

The mean end-to-end distance of the particles, $\langle R^2 \rangle$, as a function of contour length L is determined as follows. First, for the m-th path (P_m), a symmetric distance matrix $D_{i,i'}^m$ is constructed by calculating the Euclidian distance between all pairs of points i and $i' \geq i$, along the contour. Thus, the distances with the same lag $l = t_{i'} - t_i$ correspond to the l-th superdiagonal of $D_{i,i'}^m$. Finally, $\langle R^2 \rangle$ for $L = l$ is calculated as the mean of the squares of all l-th superdiagonal elements of $D_{i,i'}^m$ for all paths P_m. The uncertainty of $\langle R^2 \rangle(L)$ is estimated by calculating the standard error of the mean (SEM).

Mean orientation correlation function

The orientation (or tangent-tangent) correlation of polymer particles is defined by $\cos\theta_{i,i'} = \vec{\tau}_i \cdot \vec{\tau}_{i'}$, where $\vec{\tau}_i$ is the unit tangent vector to the interpolated path at point i. The mean orientation correlation $\langle \cos\theta \rangle(l)$, as a function of path length l, is calculated by first constructing the correlation matrix $D_{i,i'}^m = \cos\theta_{i,i'}$, where m is the path index. For $l = t_{i'} - t_i$, $\langle \cos\theta \rangle(l)$ is calculated as the mean of all l-th superdiagonal elements of $D_{i,i'}^m$ for all paths P_m. Similarly to $\langle R^2 \rangle$, the uncertainty of $\langle \cos\theta \rangle$ is estimated through the corresponding SEM.

2.3 <u>Persistence Length (l_p) Calculations</u>

In both Python and Igor Pro implementations, the persistence length, l_p, of the polymer particles was estimated by fitting either $\langle R^2 \rangle(L)$ or $\langle \cos\theta \rangle(l)$ to their expressions from the worm-like-chain (WLC) model which is commonly used to describe semi-flexible polymers [10], i.e.,

$$\langle R^2 \rangle = 4Ll_p \left(1 - \frac{2l_p}{L} \left(1 - e^{-L/2l_p} \right) \right),$$

and

$$\langle \cos\theta \rangle = \exp\left(-l/2l_p\right) .$$

In PsPolypy, these nonlinear fits are performed using the feature rich lmfit python library [32].

2.4 Model Fitting and Uncertainty Analysis

In each data set $\{x_i, y_i, \Delta y_i\}$, where x_i represents the length along the paths, y_i the mean of either R_i^2 or $\cos\theta_i$, and Δy_i the corresponding standard error of the mean (SEM), we calculated the mean $\langle y \rangle$ and the uncertainty Δy as a function of $x = l$ or $x = L$. Data was fitted to the one parameter, l_p, WLC model (discussed above) using the lmfit python package. The fitting process employed weighted least squares minimization, with weights calculated as $w_i = 1/\Delta y_i^2$. The best-fit parameter l_{p0} was determined by minimizing the chi-square statistic, with its uncertainty Δl_p derived from the covariance matrix of the fit, scaled by the reduced chi-squared, $\sqrt{\chi_\nu^2}$. The 95% confidence interval (CI) for l_p was calculated as $l_{p0} \pm 1.96\Delta l_p$, representing the range within which the true parameter value is likely to lie with 95% probability. To visualize the model's predictive capability, 95% prediction bands were computed as $\mathrm{f}(\mathrm{x}; \mathrm{l_{p0}}) \pm 1.96 * \sigma_{\mathrm{total}}(\mathrm{x})$, where $\mathrm{f}(\mathrm{x}; \mathrm{l_{p0}})$ is the WLC model of either R^2or $\cos\theta$ and $\sigma_{\mathrm{total}}^2(x) = \sigma_{\mathrm{model}}^2(x) + \sigma_{\mathrm{residual}}^2 + \Delta \mathrm{y}^2$. Here, $\sigma_{model}(x)$ represents the model uncertainty propagated from Δl_p, $\sigma_{residual}$ is the standard deviation of the residuals, and Δy accounts for the measurement uncertainty. The resulting plots display the original data points with error bars $\pm\Delta y_i$, the best-fit curve $f(x; l_{p0})$, and the 95% prediction bands, illustrating both the uncertainty in the model and the expected range for new observations.

2.5 Atomic force microscopy imaging of CL

333 CL was purchased from Peptide 2.0 in powder, then hydrated to 100 μM in MilliQ water. The
334 stock solution was stored in 10 μL aliquots at -80 °C. Imaging was performed as previously
335 described [2]. Briefly, a CL aliquot was thawed and diluted to 330 nM in the imaging buffer (10
336 mM Hepes, 150 mM NaCl, pH 7.3). Ninety microliters was added to freshly cleaved mica disks
337 and incubated for 10 minutes at room temperature (~25 °C). The samples were washed by
338 exchanging 90 μL of imaging buffer five times to remove any particles in solution or loosely
339 bound particles. Samples were imaged in the imaging buffer using biolever mini dips (Olympus,
340 k~0.1 N/m, f_o~30 kHz in fluid) in tapping mode (Cypher, Asylum Research). Throughout
341 imaging, the tip sample force magnitude was kept to ≤100 pN, a regime in which minimal
342 protein distortion is expected. Prior to algorithm implementation, images were flattened using
343 commercial AFM software (Asylum Research).
344

345 **3. RESULTS AND DISCUSSION**

346 Double stranded DNA represents a convenient benchmark for polymer chain tracing algorithms
347 as its persistence length has been well characterized. In an analysis of four AFM images
348 containing 206 linear strands of DNA with data from Hennan *et al* [8], PS-Poly persistence length
349 results for DNA were found to be 48 ± 3 nm. This is within the margin of error for the widely
350 accepted value for double stranded DNA persistence length of around 50 nm (**Table 1**). The
351 persistence length for the polymer Candidalysin was determined in an analogous manner and
352 found to be 12.1 ± 0.3 nm using seven images containing 670 linear polymers. Using the
353 Easyworm software [11], the results for Candidalysin were found to be 12 ± 3 nm [2], which is in
354 good agreement with our algorithm.
355

356 **Table 1.** Comparison of PS-Poly to previous work. Results were obtained via the end-to-end distance

357 method for persistence length.

Sample	Persistence Length, Previous work	Persistence Length using PS-Poly
DNA	46.8 +/- .6 nm N = 126 (Ref 8)	47.8 +/- 2.6 nm N = 206
Candidalysin	12 +/- 3 nm N = 191 (Ref 2)	12.1 +/- .3 nm N = 752

358

359

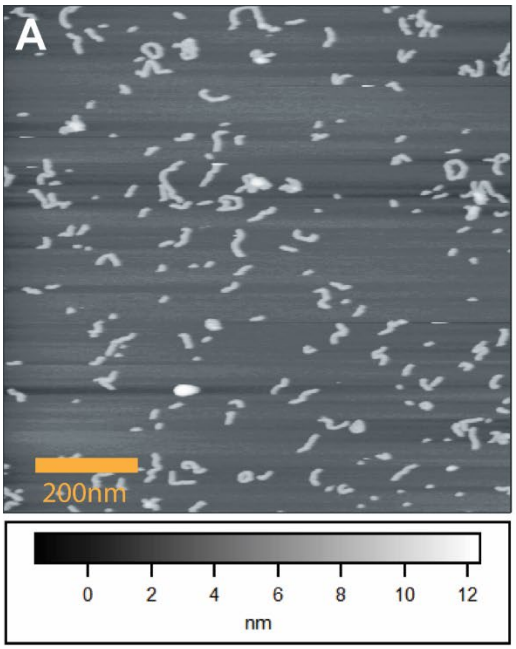
360 In the process of categorizing polymer feature shapes, the coordinates of all endpoints, branch

361 points, and three-dimensional overlaps are stored in the output as well as the total length of each

362 feature, total polymerized length for each image, and total polymerized length for all images. An

363 example AFM image of CL and the resulting PS-Poly shape output are shown (**Fig. 8**).

364



B

Particle Type	Number of Features	Average Length (nm)	Percentage of Total Polymerized Length
Linear	164	25.5	62.1
Looped	2	76.7	2.3
Branched (no looping)	18	78.0	20.8
Branched-Looped	8	125.1	14.8
Overlapped	0	N/A	0
Number of High Points			17
Number of Noise Particles			1

FIGURE 8. The output of PS-Poly for shape categorization. (**A**) Input AFM image of Candidalysin. The scale bar is 200 nm and the greyscale spans 12 nm. (**B**) Output for shape categorization shown in tabular format. Two types of artifact are identified, high points, defined as any point on a particle that is above 1.5 times the average height of the polymer backbone in the image and noise particles, defined as any particle in which 80% or more of the pixels are above 1.5 times the average height of the polymers in the image.

After establishing the overall agreement between our algorithm output and previous work, we analyzed the potential errors and robustness of the persistence length calculations. CL was employed as this polymer represents a general and complex test case. AFM image data revealed contour lengths ranging from individual CL subunits that appear as punctate features of dimension roughly equivalent to the AFM tip radius (~5 nm) to well over 100 nm. However, CL polymers that are both Linear and long were rare. This is because the longer a CL polymer becomes the more likely it is to become Looped or Branched or both (i.e, Branched-Looped) [22]. To handle these variations, in our CL l_p analysis we restricted the fitting window to contour

lengths between 10 and 30 nm. The justification for excluding data with (a) $L < 10$nm, and (b) $L > 30$nm, is that the WLC model best describes semi-flexible polymers with L comparable or larger than l_p, which in our case is larger than 10 nm; and (b) at higher contour lengths, the poor sampling of long CL polymers in the images makes the data analysis less robust. Only linear particles were used when fitting $\langle R^2 \rangle$ the WLC model. When considering only one end-to-end distance per particle, PsPolypy returned $l_p = 12.0 \pm 0.9$ nm (**Fig. 9**). On the other hand, when considering multiple segments per particle, as described in the Methods section (which is the default), we obtained a more precise result, $l_p = 12.5 \pm 0.2$ nm (**Fig. 10B**). In both cases, the fits were good quality, as indicated by $R^2 > 0.9$ and the 95% prediction bands.

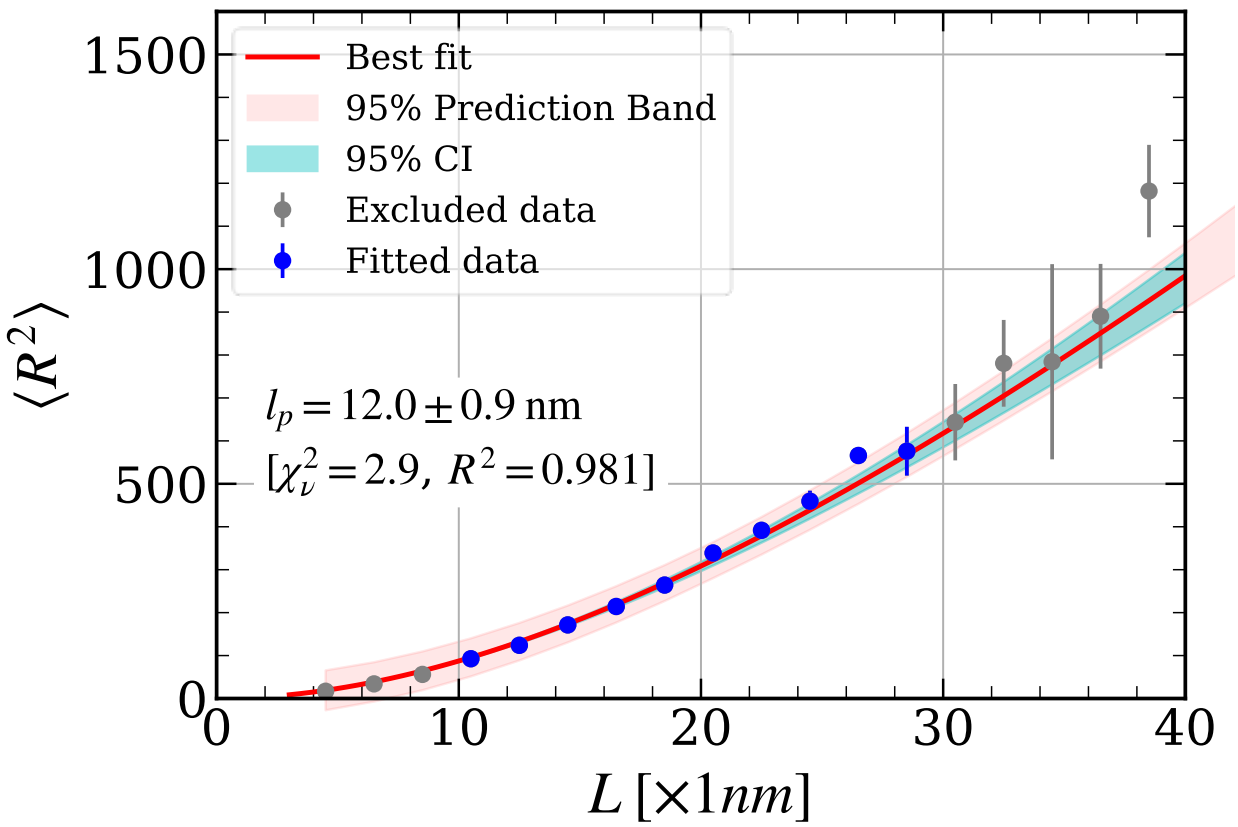


Figure 9. Limiting the contour length fitting window provides robust persistence length calculations

for CL. Plot of mean-square end-to-end distance versus contour length for CL polymers. The persistence length l_p was found to be 12.0 ± 0.9 nm. A 95% confidence interval (CI, *teal shaded region*) and 95% prediction band (*pink shaded region*) are shown.

We next challenged the algorithm by repeating persistence length calculations with noise added to the raw image data. **Figure 10A** shows an AFM image of CL with an increasing amount of white gaussian noise added, quantified by the standard deviation (std). Analysis of the polymer persistence length from this data is shown using both the end-to-end distance (**R^2**) method (**Fig. 10B**) and the tangent-tangent correlation (**TTC**) method (**Fig. 10C**). Despite this added noise, the calculated l_p value remains close to the nominal value of 12 nm (14 nm) using the **R^2** (**TTC**) method. Note that the slightly larger l_p value returned by the TTC method is most likely due to its sensitivity to the precise form of the interpolated skeletons. This conclusion is consistent with the relatively large 95% prediction band in **Fig. 10C**. It appears, however, that the algorithm is robust to a moderate amount of random pixel noise as is typically encountered in experimental settings.

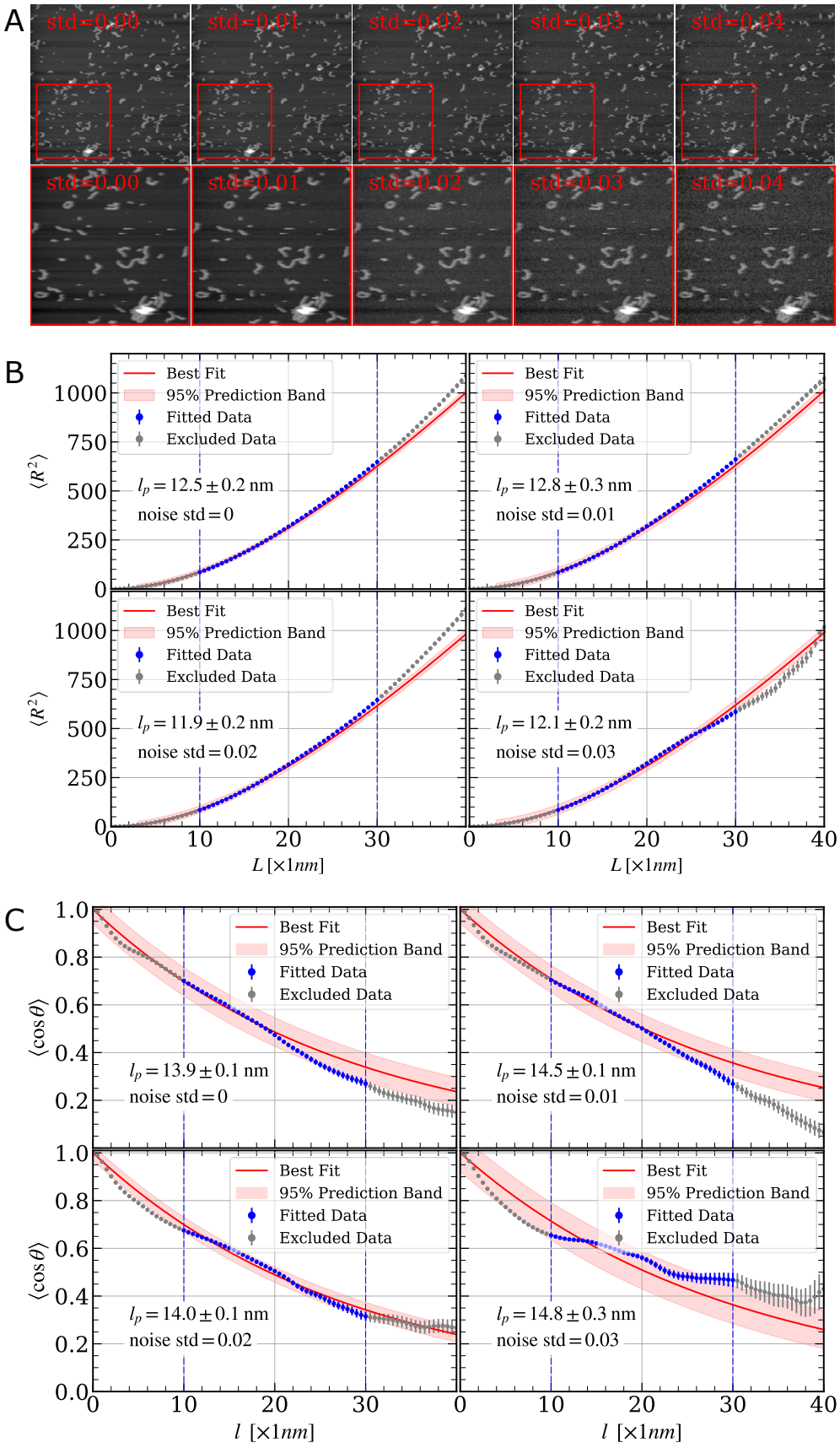
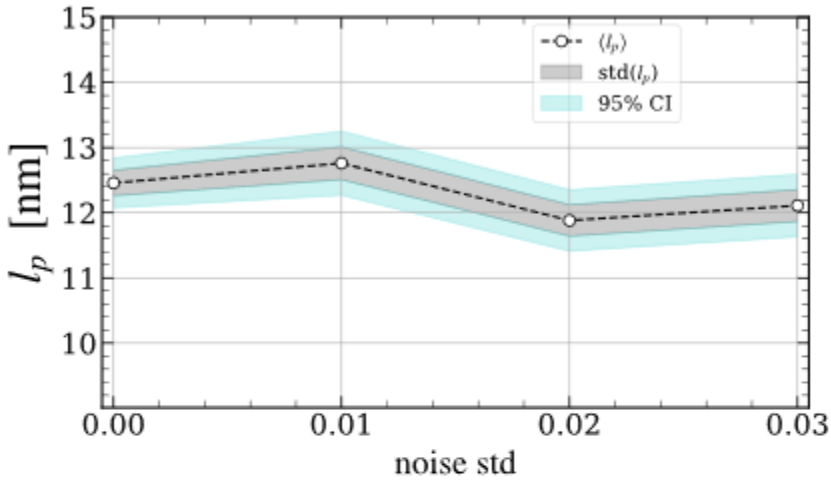

A
std=0.00
std=0.01
std=0.02
std=0.03
std=0.04
B
Best Fit
95% Prediction Band
Fitted Data
Excluded Data
$l_p = 12.5 \pm 0.2$ nm
noise std = 0
$l_p = 12.8 \pm 0.3$ nm
noise std = 0.01
$l_p = 11.9 \pm 0.2$ nm
noise std = 0.02
$l_p = 12.1 \pm 0.2$ nm
noise std = 0.03
$\langle R^2 \rangle$
L [×1nm]
C
$l_p = 13.9 \pm 0.1$ nm
noise std = 0
$l_p = 14.5 \pm 0.1$ nm
noise std = 0.01
$l_p = 14.0 \pm 0.1$ nm
noise std = 0.02
$l_p = 14.8 \pm 0.3$ nm
noise std = 0.03
$\langle \cos\theta \rangle$
l [×1nm]

Figure 10. Persistence length calculations appear stable in the face of noise. (**A**) Image sequence showing the addition of noise. The second-row images are detailed views of the region indicated in upper images (*red rectangles*). The added noise levels for each plot are indicated and range between 0 and 0.03 (std). Calculations of persistence length using different methods: (**B**) end-to-end distance and (**C**) tangent-tangent correlation. As before, only data within 10 – 30 nm contour windows were included in the fits. The expected range for new observations are indicated by the 95% prediction bands.

A summary plot displaying l_p outputs is provided in **Fig. 11**. Results from the $\mathbf{R^2}$ and **TTC** methods are shown in **Fig. 11A** and **B,** respectively. The tables contain additional information. We observe that as the image noise increases, the number of features detected as Linear decreases. This not only results in lower statistical weight of the l_p calculation but also in reduced quality of the fits.

A

noise std	All	Linear	$10 < L < 30$	l_p	95% CI	R^2
0	607	376	217 (57.7%)	12.5 ± 0.2	0.4	0.996
0.01	604	350	203 (58.0%)	12.8 ± 0.3	0.5	0.994
0.02	614	312	192 (61.5%)	11.9 ± 0.2	0.5	0.994
0.03	605	244	142 (58.2%)	12.1 ± 0.2	0.5	0.993



B

noise std	All	Linear	$10 < L < 30$	l_p	95% CI	R^2
0	607	376	575	13.9 ± 0.1	0.2	0.920
0.01	604	350	584	14.5 ± 0.1	0.2	0.928
0.02	614	312	676	14.0 ± 0.1	0.2	0.980
0.03	605	244	770	14.8 ± 0.3	0.6	0.386

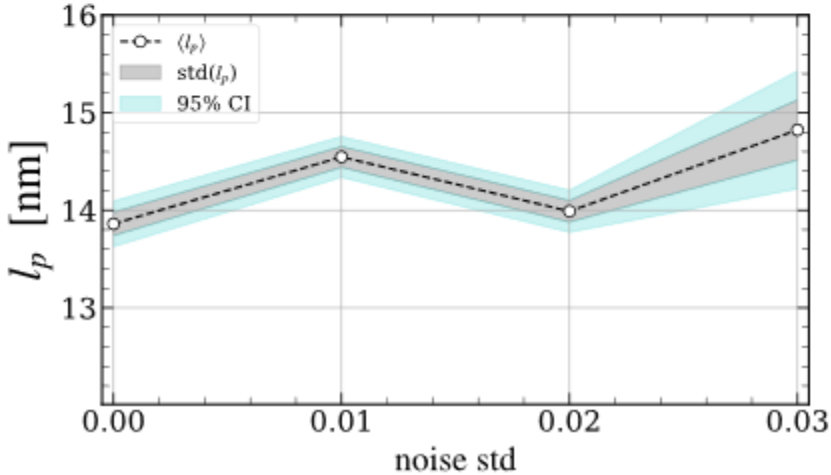


Figure 11. Persistence length output in the presence of noise. Persistence length results, as a function of added image noise, obtained by using the (**A**) **R²** or (**B**) **TTC** methods. Rows in the tables are defined: ***noise std*** = noise level added to each image; ***All*** = total number of polymer features detected; ***Linear*** = number of features identified as Linear; ***10 < L < 30*** = number of Linear features exhibiting contour lengths within the 10 – 30 nm fitting window; $\boldsymbol{l_p}$ = value of persistence length; ***95% CI*** = confidence interval; R^2 statistical goodness of fit parameter.

430

431 **Computational Performance Evaluation**

432 The runtime of PS-Poly in *Igor Pro 7* (64-bit) was compared to PsPolypy by processing a dataset
433 of nine AFM images containing 779 particles using similar hardware. PS-Poly processed the
434 dataset in 892 seconds on a CPU with an average frequency of approximately 4.2 GHz, while
435 PsPolypy processed the dataset in 3.2 seconds on a CPU with an average frequency of
436 approximately 4.4 GHz. While direct speedup comparisons are complicated by the variation of
437 CPU utilization, these results suggest that the Python implementation achieves a significant
438 reduction in runtime, on the order of two orders of magnitude. In addition to its superior
439 calculation speed, Python's open source and object-oriented design provides users with greater
440 flexibility to fine-tune the algorithm to specific needs.

441

442

443 **4. CONCLUSIONS AND FUTURE OUTLOOK**

444 PS Poly is an algorithm designed to calculate both persistence length and shape from single
445 molecule image data of complex polymers. When implemented as a python package the
446 algorithm is modular and object oriented, making it straightforward to maintain and to extend its
447 features and scope. While it can batch process stacks of AFM images, PsPolypy can also be
448 easily used for customized workflows because most of the attributes and methods are directly
449 available to the user. PS Poly currently provides two WLC model-based methods for calculating
450 persistence length (**R^2** and **TTC**) from the skeletonized representation of the polymer particles;
451 additional methods can be added in the future, as needed. Furthermore, its automated shape
452 detection can be used to provide insight into polymerization mechanisms, as we have recently
453 shown [22]. The benefits of automating this process include reduced human bias as well as time

saved by the user and the related improvement in statistical weight of the data set. The current implementations of PS Poly use the Otsu's method for image thresholding which works effectively for high signal-to-noise images such as those typically acquired in AFM. However, other segmentation methods such as watershed algorithms and machine/deep learning features can also be added to future versions. For best results, it is important to reduce noise in the input images before running the program. As we showed, the algorithm is robust to moderate levels of random pixel noise, but it will break down if higher noise levels are encountered. While PS Poly was developed to analyze AFM image data, it has the potential to generalize to other imaging modalities.

References

(1) Moyes, D. L.; Wilson, D.; Richardson, J. P.; Mogavero, S.; Tang, S. X.; Wernecke, J.; Höfs, S.; Gratacap, R. L.; Robbins, J.; Runglall, M.; Murciano, C.; Blagojevic, M.; Thavaraj, S.; Förster, T. M.; Hebecker, B.; Kasper, L.; Vizcay, G.; Iancu, S. I.; Kichik, N.; Häder, A.; Kurzai, O.; Luo, T.; Krüger, T.; Kniemeyer, O.; Cota, E.; Bader, O.; Wheeler, R. T.; Gutsmann, T.; Hube, B.; Naglik, J. R. Candidalysin Is a Fungal Peptide Toxin Critical for Mucosal Infection. *Nature* **2016**, *532* (7597), 64–68. https://doi.org/10.1038/nature17625.

(2) Russell, C. M.; Schaefer, K. G.; Dixson, A.; Gray, A. L.; Pyron, R. J.; Alves, D. S.; Moore, N.; Conley, E. A.; Schuck, R. J.; White, T. A.; Do, T. D.; King, G. M.; Barrera, F. N. The Candida Albicans Virulence Factor Candidalysin Polymerizes in Solution to Form Membrane Pores and Damage Epithelial Cells. *eLife* **2022**, *11*, e75490. https://doi.org/10.7554/eLife.75490.

(3) Mayer, F. L.; Wilson, D.; Hube, B. Candida Albicans Pathogenicity Mechanisms. *Virulence* **2013**, *4* (2), 119–128. https://doi.org/10.4161/viru.22913.

(4) Haque, F.; Grayson, S. The Synthesis, Properties and Potential Applications of Cyclic Polymers. *Nat. Chem.* **2020**, *12*, 433–444. https://doi.org/10.1038/s41557-020-0440-5.

(5) Voit, B. I.; Lederer, A. Hyperbranched and Highly Branched Polymer Architectures—Synthetic Strategies and Major Characterization Aspects. *Chem. Rev.* **2009**, *109* (11), 5924–5973. https://doi.org/10.1021/cr900068q.

(6) Müller, D. J.; Dumitru, A. C.; Lo Giudice, C.; Gaub, H. E.; Hinterdorfer, P.; Hummer, G.; De Yoreo, J. J.; Dufrêne, Y. F.; Alsteens, D. Atomic Force Microscopy-Based Force Spectroscopy and Multiparametric Imaging of Biomolecular and Cellular Systems. *Chem. Rev.* **2021**, *121* (19), 11701–11725. https://doi.org/10.1021/acs.chemrev.0c00617.

(7) Weaver, D. R.; King, G. M. Atomic Force Microscopy Reveals Complexity Underlying General Secretory System Activity. *Int. J. Mol. Sci.* **2023**, *24* (1), 55. https://doi.org/10.3390/ijms24010055.
(8) Heenan, P.; Perkins, T. Imaging DNA Equilibrated onto Mica in Liquid Using Biochemically Relevant Deposition Conditions. *ACS Nano* **2019**, *13*, 4220–4229.
(9) Rivetti, C.; Guthold, M.; Bustamante, C. Scanning Force Microscopy of DNA Deposited onto Mica: EquilibrationversusKinetic Trapping Studied by Statistical Polymer Chain Analysis. *J. Mol. Biol.* **1996**, *264* (5), 919–932. https://doi.org/10.1006/jmbi.1996.0687.
(10) Rubinstein, M.; Colby, R. *Polymer Physics*; Oxford University Press Inc.: New York, 2003.
(11) Lamour, G.; Kirkegaard, J. B.; Li, H.; Knowles, T. P.; Gsponer, J. Easyworm: An Open-Source Software Tool to Determine the Mechanical Properties of Worm-like Chains. *Source Code Biol. Med.* **2014**, *9* (1), 16. https://doi.org/10.1186/1751-0473-9-16.
(12) Beton, J. G.; Moorehead, R.; Helfmann, L.; Gray, R.; Hoogenboom, B. W.; Joseph, A. P.; Topf, M.; Pyne, A. L. B. TopoStats – A Program for Automated Tracing of Biomolecules from AFM Images. *Methods* **2021**, *193*, 68–79. https://doi.org/10.1016/j.ymeth.2021.01.008.
(13) Usov, I.; Mezzenga, R. FiberApp: An Open-Source Software for Tracking and Analyzing Polymers, Filaments, Biomacromolecules, and Fibrous Objects. *Macromolecules* **2015**, *48* (5), 1269–1280. https://doi.org/10.1021/ma502264c.
(14) Konrad, S. F.; Vanderlinden, W.; Frederickx, W.; Brouns, T.; Menze, B. H.; Feyter, S. D.; Lipfert, J. High-Throughput AFM Analysis Reveals Unwrapping Pathways of H3 and CENP-A Nucleosomes. *Nanoscale* **2021**, *13* (10), 5435–5447. https://doi.org/10.1039/D0NR08564B.
(15) Wiggins, P. A.; van der Heijden, T.; Moreno-Herrero, F.; Spakowitz, A.; Phillips, R.; Widom, J.; Dekker, C.; Nelson, P. C. High Flexibility of DNA on Short Length Scales Probed by Atomic Force Microscopy. *Nat. Nanotechnol.* **2006**, *1* (2), 137–141. https://doi.org/10.1038/nnano.2006.63.
(16) Faas, F. G. A.; Rieger, B.; Vliet, L. J. van; Cherny, D. I. DNA Deformations near Charged Surfaces: Electron and Atomic Force Microscopy Views. *Biophys. J.* **2009**, *97* (4), 1148–1157. https://doi.org/10.1016/j.bpj.2009.06.015.
(17) Brangwynne, C. P.; Koenderink, G. H.; Barry, E.; Dogic, Z.; MacKintosh, F. C.; Weitz, D. A. Bending Dynamics of Fluctuating Biopolymers Probed by Automated High-Resolution Filament Tracking. *Biophys. J.* **2007**, *93* (1), 346–359. https://doi.org/10.1529/biophysj.106.096966.
(18) Zhang, H.; Söderholm, N.; Sandblad, L.; Wiklund, K.; Andersson, M. DSeg: A Dynamic Image Segmentation Program to Extract Backbone Patterns for Filamentous Bacteria and Hyphae Structures. *Microsc. Microanal.* **2019**, *25* (3), 711–719. https://doi.org/10.1017/S1431927619000308.
(19) Schneider, M.; Al-Shaer, A.; Forde, N. R. AutoSmarTrace: Automated Chain Tracing and Flexibility Analysis of Biological Filaments. *Biophys. J.* **2021**, *120* (13), 2599–2608. https://doi.org/10.1016/j.bpj.2021.05.011.
(20) Graham, J. S.; McCullough, B. R.; Kang, H.; Elam, W. A.; Cao, W.; De La Cruz, E. M. Multi-Platform Compatible Software for Analysis of Polymer Bending Mechanics. *PLoS ONE* **2014**, *9* (4), e94766. https://doi.org/10.1371/journal.pone.0094766.

(21) Nunez-Iglesias, J.; Blanch, A. J.; Looker, O.; Dixon, M. W.; Tilley, L. A New Python Library to Analyse Skeleton Images Confirms Malaria Parasite Remodelling of the Red Blood Cell Membrane Skeleton. *PeerJ* **2018**, *6*, e4312. https://doi.org/10.7717/peerj.4312.
(22) Schaefer, K. G.; Russell, C. M.; Pyron, R. J.; Conley, E. A.; Barrera, F. N.; King, G. M. Polymerization Mechanism of the Candida Albicans Virulence Factor Candidalysin. *J. Biol. Chem.* **2024**, *300* (6). https://doi.org/10.1016/j.jbc.2024.107370.
(23) Lin, J.; Miao, J.; Schaefer, K. G.; Russell, C. M.; Pyron, R. J.; Zhang, F.; Phan, Q. T.; Solis, N. V.; Liu, H.; Tashiro, M.; Dordick, J. S.; Linhardt, R. J.; Yeaman, M. R.; King, G. M.; Barrera, F. N.; Peters, B. M.; Filler, S. G. Sulfated Glycosaminoglycans Are Host Epithelial Cell Targets of the Candida Albicans Toxin Candidalysin. *Nat. Microbiol.* **2024**, *9* (10), 2553–2569. https://doi.org/10.1038/s41564-024-01794-8.
(24) Https://Github.Com/KingGavinM/
(25) Houssein, E. H.; Mohamed, G. M.; Ibrahim, I. A.; Wazery, Y. M. An Efficient Multilevel Image Thresholding Method Based on Improved Heap-Based Optimizer. *Sci. Rep.* **2023**, *13* (1), 9094. https://doi.org/10.1038/s41598-023-36066-8.
(26) Zhang, T. Y.; Suen, C. Y. A Fast Parallel Algorithm for Thinning Digital Patterns. *Commun ACM* **1984**, *27* (3), 236–239. https://doi.org/10.1145/357994.358023.
(27) Even, S.; Even, G. *Graph Algorithms*; Cambridge University Press, 2011.
(28) van der Walt, S.; Schönberger, J. L.; Nunez-Iglesias, J.; Boulogne, F.; Warner, J. D.; Yager, N.; Gouillart, E.; Yu, T. Scikit-Image: Image Processing in Python. *PeerJ* **2014**, *2*, e453. https://doi.org/10.7717/peerj.453.
(29) Van Rossum, G.; Drake, F. L. *Python 3 Reference Manual*; CreateSpace: Scotts Valley, CA, 2009.
(30) Hagberg, A. A.; Schult, D. A.; Swart, P. J. Exploring Network Structure, Dynamics, and Function Using NetworkX. In *Proceedings of the 7th Python in Science Conference*; Varoquaux, G., Vaught, T., Millman, J., Eds.; Pasadena, CA USA, 2008; pp 11–15.
(31) Virtanen, P.; Gommers, R.; Oliphant, T. E.; Haberland, M.; Reddy, T.; Cournapeau, D.; Burovski, E.; Peterson, P.; Weckesser, W.; Bright, J.; Van Der Walt, S. J.; Brett, M.; Wilson, J.; Millman, K. J.; Mayorov, N.; Nelson, A. R. J.; Jones, E.; Kern, R.; Larson, E.; Carey, C. J.; Polat, İ.; Feng, Y.; Moore, E. W.; VanderPlas, J.; Laxalde, D.; Perktold, J.; Cimrman, R.; Henriksen, I.; Quintero, E. A.; Harris, C. R.; Archibald, A. M.; Ribeiro, A. H.; Pedregosa, F.; Van Mulbregt, P.; SciPy 1.0 Contributors; Vijaykumar, A.; Bardelli, A. P.; Rothberg, A.; Hilboll, A.; Kloeckner, A.; Scopatz, A.; Lee, A.; Rokem, A.; Woods, C. N.; Fulton, C.; Masson, C.; Häggström, C.; Fitzgerald, C.; Nicholson, D. A.; Hagen, D. R.; Pasechnik, D. V.; Olivetti, E.; Martin, E.; Wieser, E.; Silva, F.; Lenders, F.; Wilhelm, F.; Young, G.; Price, G. A.; Ingold, G.-L.; Allen, G. E.; Lee, G. R.; Audren, H.; Probst, I.; Dietrich, J. P.; Silterra, J.; Webber, J. T.; Slavič, J.; Nothman, J.; Buchner, J.; Kulick, J.; Schönberger, J. L.; De Miranda Cardoso, J. V.; Reimer, J.; Harrington, J.; Rodríguez, J. L. C.; Nunez-Iglesias, J.; Kuczynski, J.; Tritz, K.; Thoma, M.; Newville, M.; Kümmerer, M.; Bolingbroke, M.; Tartre, M.; Pak, M.; Smith, N. J.; Nowaczyk, N.; Shebanov, N.; Pavlyk, O.; Brodtkorb, P. A.; Lee, P.; McGibbon, R. T.; Feldbauer, R.; Lewis, S.; Tygier, S.; Sievert, S.; Vigna, S.; Peterson, S.; More, S.; Pudlik, T.; Oshima, T.; Pingel, T. J.; Robitaille, T. P.; Spura, T.; Jones, T. R.; Cera, T.; Leslie, T.; Zito, T.; Krauss, T.; Upadhyay, U.; Halchenko, Y. O.; Vázquez-Baeza, Y. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nat. Methods* **2020**, *17* (3), 261–272. https://doi.org/10.1038/s41592-019-0686-2.

(32) Newville, M.; Stensitzki, T.; Allen, D. B.; Ingargiola, A. LMFIT: Non-Linear Least-Square Minimization and Curve-Fitting for Python, 2014. https://doi.org/10.5281/zenodo.11813.