Learning Interaction Kernels for Nonlocal Bacterial Growth Models

Jacob W. Murri*[1], Cameron Dudiak[2,3], James N. Wilking[3,4,5], Scott G. McCalla[2,3], and Hayden Schaeffer[1,6]

[1]Department of Mathematics, University of California, Los Angeles, Los Angeles, CA 90095
[2]Department of Mathematical Sciences, Montana State University, Bozeman, MT 59717
[3]Center for Biofilm Engineering, Montana State University, Bozeman, MT 59717
[4]Chemical and Biological Engineering Department, Montana State University, Bozeman, MT 59717
[5]Department of Physiology & Biomedical Engineering, Mayo Clinic, Rochester, MN 55905
[6]California NanoSystems Institute (CNSI), University of California, Los Angeles, Los Angeles, CA 90095

Abstract

A previously developed system of geometric nonlocal differential equations models the growth of *Paenibacillus dendritiformis* bacterial colonies by tracking their boundaries using a Lagrangian approach. This model contains self-interaction and cross-interaction kernels which are difficult to derive from first principles. We consider the problem of using data to learn these interaction kernels. Image processing and segmentation techniques are employed to extract contours from videos of growing bacterial colonies, and interaction kernels are modeled using simple parametrized functions. Model parameters are trained using a constrained optimization approach with the Nelder–Mead simplex method and L-BFGS-B to minimize the error between simulated and target contours. The fitted interaction kernels accurately predict the evolution of *P. dendritiformis* colonies and reproduce the "stationary fronts" phenomenon whereby two nearby colonies inhibit each other's growth. We compare models using shared interaction kernels and colony growth rates with those using distinct self- and cross-interaction kernels and varying per-colony growth rates. We also introduce a nonlocal transport mechanism directed along the intercolony center-of-mass axis.

Keywords: interaction kernels, parameter estimation, nonlocal transport, nonlocal forcing, moving interface, Paenibacillus dendritiformis
ORCID iDs: JWM, 0009-0007-9273-4309; CD, 0009-0006-2307-6376; JNW, 0000-0002-7255-7209; SGM, 0000-0003-0093-4658; HS, 0009-0002-9015-1357

1 Introduction

Biofilms are spatially structured microbial communities embedded within an extracellular matrix whose growth and organization arise from the interaction of cellular proliferation, nutrient transport, mechanical forces, and biochemical signaling across multiple spatial and temporal scales [1–4]. These systems play an important role in a broad range of biological and engineering settings, including chronic infections, antibiotic tolerance, and industrial biofouling [1, 5, 6]. Despite extensive experimental investigation, the development of predictive mathematical models capable of reproducing the full spatiotemporal evolution of biofilm and bacterial colony morphologies remains a significant challenge.

*Corresponding author: jwmurri@math.ucla.edu

A defining feature of many microbial growth systems is the emergence of coherent interfaces separating occupied and unoccupied regions. The evolution of these interfaces reflects the collective influence of local growth processes, geometric regularization, competition for resources, and long-range communication mechanisms such as quorum sensing and chemically mediated inhibition. Consequently, interfacial dynamics provide a natural reduced description of colony-scale behavior, allowing complex biological interactions to be encoded through the motion of evolving geometric objects. Such reduced descriptions are particularly attractive when many relevant variables, including chemical concentrations and intracellular signaling states, are either unobserved or only partially observed experimentally; this is typical for biofilm imaging studies.

A wide range of mathematical frameworks have been developed to study microbial growth and biological pattern formation. Agent-based models resolve individual cells and their interactions, providing detailed descriptions of collective phenomena such as differentiation, quorum sensing, and branching morphologies [7, 8]. Such models are most informative when the relevant cellular mechanisms are known, but they become computationally expensive for colonies of millions of cells and contain many parameters that are difficult to measure. At the continuum level, systems of ordinary and partial differential equations have been used to model population densities, nutrient fields, signaling molecules, and mechanical effects, often incorporating reaction–diffusion dynamics, fluid transport, elasticity, and free-boundary evolution [9–11]. More sophisticated formulations employ immersed-boundary, phase-field, and level-set methodologies to capture evolving interfaces and mechanical interactions [12–14]. While these approaches have yielded substantial biological insight, they often involve large numbers of parameters and latent variables, making quantitative studies difficult in the low-data and high-noise regimes typical of biological experiments.

An alternative perspective is provided by geometric evolution equations, in which the state of each biofilm colony is represented directly by one or more evolving interfaces. Such formulations connect naturally to the extensive mathematical literature on curvature-driven flows, front propagation, and free-boundary problems [15–17]. Classical examples include mean curvature flow and its variants, where interfacial motion is governed by local geometric quantities. Curvature-driven flows of this kind have long been used in materials science and metallurgy, for example to describe the motion of grain boundaries and the stability of growing interfaces [18, 19], and they arise as sharp-interface limits of phase-field models of phase separation and coarsening such as the Allen–Cahn and Cahn–Hilliard equations [20–23]. Interfacial motion of this type also emerges from reaction–diffusion systems with fast reaction and slow diffusion [24]. However, many multiscale and biological systems exhibit interactions mediated by signaling molecules, mechanical stresses, or other processes acting over finite spatial scales. These mechanisms naturally motivate the incorporation of nonlocal effects into geometric evolution laws. In multiscale reaction–diffusion systems, for example, interfaces interact through slowly varying chemical fields, and eliminating these fields yields effective nonlocal interactions between fronts; in singularly perturbed three-component systems, such front interactions can be derived rigorously in one spatial dimension [25] and observed in numerical simulations in two [26].

Nonlocal geometric partial differential equations (PDEs) have emerged in a variety of contexts, including materials science, aggregation dynamics, phase separation, image processing, and collective behavior [27–31]. In these models, interface motion depends both on local geometric information and on spatially distributed interactions encoded through integral operators or interaction kernels. Such formulations provide a mathematically natural framework for describing quorum sensing, toxin-mediated inhibition, competition between neighboring colonies, and mechanically mediated communication through the extracellular matrix. Moreover, nonlocal models can often be interpreted as effective descriptions of unresolved or latent dynamics, a perspective emphasized in the work of Du and collaborators and developed further in recent approaches to learning nonlocal

operators from partially observed systems [32–35].

Our motivating application is the growth of colonies of the bacterium *Paenibacillus dendritiformis*. When two colonies of the same strain grow on the same agar plate, they do not merge: their fronts stop advancing at a finite distance from each other, leaving a stable gap between the colonies [36]. This mutual inhibition is mediated by a protein, sibling lethal factor (SLF), which the colonies secrete in response to competition [37]. Such stationary, or arrested, fronts are a natural target for interfacial models with nonlocal interactions, since the colonies inhibit each other's growth before they come into contact.

The model considered in this work builds upon the nonlocal interfacial framework introduced in [38, 39], in which colony boundaries evolve under the combined influence of growth, curvature regularization, and nonlocal inhibitory interactions. Some previous numerical work fit a simplified model with shared colony growth rates and interaction kernels [38, 40], but we find that using a more general form of the model introduced in [38, 39], with varying per-colony growth rates and distinct self- and cross-interaction kernels, improves model fits. This extension greatly enlarges the class of admissible dynamics while remaining amenable to both analysis and numerical computation.

We also consider an extension of the model: a nonlocal transport mechanism which permits attractive or repulsive colony-scale drift induced by interactions between neighboring colonies. The resulting dynamics combine curvature-driven regularization, nonlocal normal forcing, and global transport effects within a unified geometric framework. However, we find that the effect of the nonlocal transport mechanism is minimal at best (losses improve by only 0.1–2.8%), indicating that the standard model from [38, 39] already captures the separation between colonies well.

A secondary contribution is a comparison of two parametrized classes of interaction kernels: Gaussian kernels, used in earlier work, and a new class of Ramp kernels. Learning the kernels nonparametrically, as has been done for systems of interacting particles and agents [41–45], is a natural direction for future work, although the limited contour data make it challenging.

A central objective of this work is to infer the parameters governing this nonlocal geometric PDE directly from experimental observations. This problem lies at the intersection of inverse problems, scientific machine learning, and data-driven model discovery. Recent advances have produced powerful techniques for learning dynamical systems from data, including sparse equation-discovery methods [46–48], neural differential equations [49], weak-form PDE identification approaches [50], and operator-learning frameworks such as DeepONets and Fourier Neural Operators [51, 52], as well as foundation models trained across many differential equations [53–55]. While highly successful in many settings, these approaches are generally designed for function-valued states evolving on fixed spatial domains and often rely on large volumes of high-quality observations.

The setting considered here presents several challenges that fall outside the assumptions underlying many existing learning frameworks. First, the state variable is not a function on a fixed domain but rather a collection of evolving interfaces represented as closed curves. Second, the available data are limited, noisy, and only partially observed, with important physical variables remaining latent. Third, the governing dynamics involve geometric constraints and nonlocal interactions that are difficult to encode within generic neural architectures. As a result, purely data-driven methods frequently suffer from poor interpretability, limited extrapolation capability, and difficulties in enforcing geometric consistency.

These considerations motivate the use of structured mechanistic models that incorporate nonlocal interactions explicitly. By restricting attention to a parametrized family of biologically motivated geometric evolution laws, one obtains a model class that remains sufficiently expressive to capture experimentally observed phenomena while retaining interpretability and enabling parameter estimation from limited data. In this sense, the present work contributes to a growing body of research demonstrating that carefully structured nonlocal models can provide a powerful

alternative to purely black-box learning approaches in partially observed scientific systems.

In addition to model development, we address a major practical bottleneck in quantitative biological modeling: the extraction of reliable geometric information from experimental imagery. We develop an improved segmentation and contour-processing pipeline for *P. dendritiformis* colony growth, producing smooth and temporally consistent interface representations suitable for downstream analysis. These data enable the formulation of a PDE-constrained optimization problem in which model parameters are inferred by minimizing discrepancies between observed and simulated interface evolution.

Our contributions are therefore as follows. First, we develop a robust image-processing pipeline that produces high-quality geometric representations of experimental growth dynamics. Second, we fit a nonlocal geometric model of bacterial colony evolution to experimental data, using its full generality by incorporating distinct per-colony growth rates and self- and cross-interaction kernels. Third, we compare this model with simpler parametrizations of earlier work by testing how sharing colony growth rates and interaction kernels affects model fit. Fourth, we demonstrate the robustness of the model by fitting an additional nonlocal transport mechanism and showing that the improvement is minimal at best. Fifth, we produce a comparison of two parametrized classes of interaction kernels: Gaussian kernels, used in earlier work, and a new class of Ramp kernels. Overall, our work demonstrates that the resulting model is identifiable, trainable, and quantitatively predictive in a low-data, high-noise regime, providing evidence that nonlocal geometric models constitute an effective framework for scientific machine learning in systems with substantial latent structure.

The remainder of the paper is organized as follows. Section 2 introduces the mathematical formulation of the model and associated representations of planar contours. Section 3 describes the parameter estimation framework and training methodology. Section 4 presents numerical results and comparisons with experimental data. Finally, Section 5 discusses the implications of the model, its limitations, and directions for future work.

2 Mathematical Preliminaries

2.1 Notation

This section will clarify and emphasize some of the notation used in this work. All of the contours will be functions defined on $\mathbb{T}^1 = [-\pi, \pi]$, where $\mathbb{T}^1$ is used instead of $[-\pi, \pi]$ to connote periodic boundary conditions. Variables in $\mathbb{T}^1$ will typically be denoted u, v, or w.

Vector-valued quantities will be bolded (for example, $\boldsymbol{\tau}$ for a tangent vector or $\mathbf{n}$ for a normal vector). The quantities in this work often depend on several indices. Vector indices will always be denoted with a subscript, while the contour index uses a superscript with parentheses (typically using the letter i as a variable). Likewise, spatial indices will typically use the letter j and temporal indices will use the letter k. These latter two indices can appear as subscripts if referring to discrete points. For example,

$$(\boldsymbol{\gamma}^{(i)}_{j,k})_2 \quad \text{or} \quad \boldsymbol{\gamma}^{(i)}(u_j, t_k)_2$$

refers to the 2nd vector component of the ith contour at the jth spatial location and the kth time. The latter usage emphasizes evaluating a function at a point, while the former usage connotes that the contour is being represented by a discrete set of points. The notation $[n]$, where $n \in \mathbb{Z}$, will be used to refer to the set $\{1, \ldots, n\}$.

Partial derivatives are represented in a few ways. They can be explicitly written out (e.g. $\frac{\partial \boldsymbol{\gamma}}{\partial u}(u, t)$) when clarity is required and the function arguments are emphasized. In this case the

variable in the partial derivative refers to the slot of the function which is being differentiated with respect to. To compress notation, $\partial_u \boldsymbol{\gamma}(u,t)$ or $\boldsymbol{\gamma}_u$ may be used (the latter only if the function arguments are also suppressed).

The notation $\arctan(y,x)$ refers to the two-argument arctangent function which returns the angle θ between the positive x-axis and the ray from the origin to the point (x,y), confined to $(-\pi,\pi]$.

2.2 Modeling using Planar Contours

In [38], the authors studied a class of nonlocal partial differential equations which model the interfacial dynamics related to growth of *P. dendritiformis* colonies. That model, in which every force acting on a colony boundary is directed normal to the boundary, is the model whose interaction kernels we learn in this work. An extension which also allows a rigid, non-normal drift of each colony is described in Section 2.6; it is compared against the standard model in Section 4.3.

Suppose the colony boundaries are modeled by a collection of m planar counterclockwise-oriented contours denoted as $\mathcal{C} = \{\boldsymbol{\gamma}^{(1)}, \dots, \boldsymbol{\gamma}^{(m)}\}$. These parametrized curves are defined as functions on $\mathbb{T}^1 \times [0,T]$ where $[0,T]$ is a certain interval of time:

$$\boldsymbol{\gamma}^{(i)}(u,t) = \begin{bmatrix} (\boldsymbol{\gamma}^{(i)}(u,t))_1 \\ (\boldsymbol{\gamma}^{(i)}(u,t))_2 \end{bmatrix}, \qquad \boldsymbol{\gamma}^{(i)} : \mathbb{T}^1 \times [0,T] \to \mathbb{R}^2,$$

where $\mathbb{T}^1$ is as defined in Section 2.1. Each $\boldsymbol{\gamma}^{(i)}$ is assumed to be a simple closed contour with counterclockwise orientation. We also assume that the parametrizations are smooth and regular in the sense that $|\partial_u \boldsymbol{\gamma}^{(i)}(u,t)| \neq 0$ for any $(u,t) \in \mathbb{T}^1 \times [0,T]$ and $i \in [m]$.

Let $\boldsymbol{\tau}^{(i)}(u,t)$ and $\mathbf{n}^{(i)}(u,t)$ represent the unit tangent and outward normal vectors to the curve $\boldsymbol{\gamma}^{(i)}$ at the point $u \in \mathbb{T}^1$ at time t, i.e.

$$\boldsymbol{\tau}^{(i)}(u,t) = \frac{\partial_u \boldsymbol{\gamma}^{(i)}(u,t)}{|\partial_u \boldsymbol{\gamma}^{(i)}(u,t)|} \tag{2.1}$$

$$\mathbf{n}^{(i)}(u,t) = \begin{bmatrix} (\boldsymbol{\tau}^{(i)}(u,t))_2 \\ -(\boldsymbol{\tau}^{(i)}(u,t))_1 \end{bmatrix} \tag{2.2}$$

Defining $(a,b)^\perp := (b,-a)$, we have that $\mathbf{n}^{(i)} = \boldsymbol{\tau}^{(i)\perp}$ and $\boldsymbol{\tau}^{(i)} = -\mathbf{n}^{(i)\perp}$. Furthermore, let $\kappa_{\boldsymbol{\gamma}^{(i)}}(u,t)$ represent the signed curvature of the contour $\boldsymbol{\gamma}^{(i)}$ at a point $u \in \mathbb{T}^1$ at time t:

$$\kappa_{\boldsymbol{\gamma}^{(i)}}(u,t) = \frac{\partial_u \boldsymbol{\tau}^{(i)}(u,t)}{|\partial_u \boldsymbol{\gamma}^{(i)}(u,t)|} \cdot \mathbf{n}_{\gamma^{(i)}}(u,t),$$

so that the identity

$$\kappa_{\boldsymbol{\gamma}^{(i)}}(u,t)\mathbf{n}^{(i)}(u,t) = \frac{\partial_u \boldsymbol{\tau}^{(i)}(u,t)}{|\partial_u \boldsymbol{\gamma}^{(i)}(u,t)|} \tag{2.3}$$

holds.

The evolution of these planar contours is given by

$$\partial_t \boldsymbol{\gamma}^{(i)}(u,t) = \left(\mu \kappa_{\boldsymbol{\gamma}^{(i)}}(u,t) + c^{(i)} - v_{\mathrm{NL}}^{(i)}(u,t)\right) \mathbf{n}^{(i)}(u,t), \tag{2.4}$$

where μ and $\{c^{(1)}, \dots, c^{(m)}\}$ are constants which describe (respectively) the dependence of colony growth on signed curvature, and a per-colony constant growth or death rate. Per-colony constant

growth rates as in [38], rather than the single shared growth rate of [40], are used to address varying initial size or vigor of distinct colonies. Nonlocal forces are described by $v_{\mathrm{NL}}^{(i)}$, the normal-direction nonlocal force modeled by interaction kernels.

Nonlocal Force from Interaction Kernels. The nonlocal force $v_{\mathrm{NL}}^{(i)}(u,t)$ modeling interactions between contours is described by interaction kernels f and g:

$$\begin{aligned} &v_{\mathrm{NL}}^{(i)}(u,t) \\ &:= \int_{\mathbb{T}^1} f\left(\frac{1}{2}\left|\boldsymbol{\gamma}^{(i)}(u,t)-\boldsymbol{\gamma}^{(i)}(v,t)\right|^2\right)\left|\partial_u\boldsymbol{\gamma}^{(i)}(v,t)\right|\mathrm{d}v \\ &+\sum_{j\neq i}\int_{\mathbb{T}^1} g\left(\frac{1}{2}\left|\boldsymbol{\gamma}^{(i)}(u,t)-\boldsymbol{\gamma}^{(j)}(v,t)\right|^2\right)\left|\partial_u\boldsymbol{\gamma}^{(j)}(v,t)\right|\mathrm{d}v. \end{aligned}$$

To align with the framework in [38], the inputs to interaction kernels are one-half of the squared distance between two points. For f, this pair of points lies on a single contour; thus f describes self-interaction. The input to g is one-half of the squared distance between a pair of points lying on two separate contours; hence g describes the interaction between colonies. We note that while [38] allow f and g to be distinct, the numerical study of [40] trains a model with $f = g$. We take the former approach, and in fact find that the best-fitting interaction kernels for self- and cross-interactions are very different, and requiring $f = g$ raises the best loss by 42–323% (see Section 4.2 for more details). Within the context of a *P. dendritiformis* bacterial growth model, f and g could describe the effect of a toxic protein, known as sibling lethal factor (SLF), which colonies release and which inhibits growth.

The interaction kernels are required to have several properties:

1. Interaction kernels should be functions from $\mathbb{R}^+$ to $\mathbb{R}^+$, i.e. only take nonnegative values. We assume that the interactions only serve to inhibit growth, not increase it (as in the example of SLF protein inhibiting growth of bacterial colonies).

2. Interaction kernels are the same for each pair of colonies (excepting self-interaction).

3. Interaction kernels should be continuous, monotone decreasing functions (the strength of interaction should be continuous in the distance, as well as decrease with increasing distance).

4. Interaction kernels should tend to zero as their input tends to infinity (colonies which are far enough away should have arbitrarily small interaction).

Substituting the nonlocal force from interaction kernels into (2.4) yields the full model

$$\begin{aligned} \frac{\partial}{\partial t}\boldsymbol{\gamma}^{(i)}(u,t) = \mathbf{n}^{(i)}(u,t)\Big[&\mu\,\kappa_{\boldsymbol{\gamma}^{(i)}}(u,t) + c^{(i)} - \\ &\int_{\mathbb{T}^1} f\left(\frac{1}{2}\left|\boldsymbol{\gamma}^{(i)}(u,t)-\boldsymbol{\gamma}^{(i)}(v,t)\right|^2\right)\left|\partial_u\boldsymbol{\gamma}^{(i)}(v,t)\right|\mathrm{d}v \\ &-\sum_{j\neq i}\int_{\mathbb{T}^1} g\left(\frac{1}{2}\left|\boldsymbol{\gamma}^{(i)}(u,t)-\boldsymbol{\gamma}^{(j)}(v,t)\right|^2\right)\left|\partial_u\boldsymbol{\gamma}^{(j)}(v,t)\right|\mathrm{d}v\Big], \\ &i\in[m]. \end{aligned} \tag{2.5}$$

2.3 Nondimensionalization

By introducing a time scale $\tau > 0$ and nondimensional time $t' := t/\tau$, the model equations can be rewritten to eliminate the curvature coefficient μ, at the cost of introducing τ itself as a new model parameter:

$$\begin{aligned}\frac{\partial}{\partial t'}\boldsymbol{\gamma}^{(i)}(u,t') = \mathbf{n}^{(i)}(u,t')\Big[&\tau\mu\,\kappa_{\boldsymbol{\gamma}^{(i)}}(u,t') + \tau c^{(i)} - \\ &\tau\int_{\mathbb{T}^1} f\left(\frac{1}{2}\left|\boldsymbol{\gamma}^{(i)}(u,t') - \boldsymbol{\gamma}^{(i)}(v,t')\right|^2\right)\left|\partial_u\boldsymbol{\gamma}^{(i)}(v,t')\right|\mathrm{d}v - \\ &\sum_{j\neq i}\tau\int_{\mathbb{T}^1} g\left(\frac{1}{2}\left|\boldsymbol{\gamma}^{(i)}(u,t') - \boldsymbol{\gamma}^{(j)}(v,t')\right|^2\right) \\ &\qquad\times\left|\partial_u\boldsymbol{\gamma}^{(j)}(v,t')\right|\mathrm{d}v\Big], \\ &\qquad\qquad i\in[m].\end{aligned} \tag{2.6}$$

After renaming t' to t, setting $\tau = 1/\mu$, and renaming $c^{(i)\prime} = \tau c^{(i)}$ to $c^{(i)}$, the coefficient on the curvature term is eliminated:

$$\begin{aligned}\frac{\partial}{\partial t}\boldsymbol{\gamma}^{(i)}(u,t) = \mathbf{n}^{(i)}(u,t)\Big[&\kappa_{\boldsymbol{\gamma}^{(i)}}(u,t) + c^{(i)} - \\ &\int_{\mathbb{T}^1}\tau f\left(\frac{1}{2}\left|\boldsymbol{\gamma}^{(i)}(u,t) - \boldsymbol{\gamma}^{(i)}(v,t)\right|^2\right)\left|\partial_u\boldsymbol{\gamma}^{(i)}(v,t)\right|\mathrm{d}v - \\ &\sum_{j\neq i}\int_{\mathbb{T}^1}\tau g\left(\frac{1}{2}\left|\boldsymbol{\gamma}^{(i)}(u,t) - \boldsymbol{\gamma}^{(j)}(v,t)\right|^2\right)\left|\partial_u\boldsymbol{\gamma}^{(j)}(v,t)\right|\mathrm{d}v\Big], \\ &\qquad\qquad i\in[m].\end{aligned} \tag{2.7}$$

Even though the total number of parameters stays the same, this representation is easier to work with numerically because the governing equation's diffusion term (and the associated time step restriction) does not depend on a model parameter. The disadvantage is that the interval between images, measured in nondimensional time, now depends on τ; Section 3.4 explains how the time step is chosen so that the discretized model remains smooth in τ. The interaction kernels now appear only through the rescaled kernels τf and τg, which we train directly (Section 3.1).

Interpretation of τ. Throughout this work, physical time is measured in units of the interval between successive images, so consecutive images are separated by $1/\tau$ units of nondimensional time. Since the time scale that sets the curvature coefficient to one is $\tau = 1/\mu$, the parameter τ is the reciprocal of the physical curvature coefficient, and in physical units the growth rate of colony i is $c^{(i)}/\tau$. A smaller value of τ therefore means that curvature-driven smoothing acts more strongly over each interval between images. Also, notably, curvature-driven smoothing is inversely proportional to radius for circular colonies, so it affects smaller colonies more heavily and may help to differentiate growth rates between colonies. The optimization finds a large range of values of τ appropriate for different settings and datasets; Section 4.1 shows that the best fits with the standard model have $\tau \approx 0.88$–0.89 (Dataset 1), 0.19–0.40 (Dataset 2) and 0.028–0.095 (Dataset 3).

2.4 Constant-Speed and Tangent-Angle Representations

Rather than evolving the parametrized contours $\boldsymbol{\gamma}^{(i)}$ directly, we follow [39] and represent each contour by its average speed, its centroid, and its tangent-angle function. This leads to a form of

the governing equations that is well suited to stable numerical computation (Section 2.5). This section summarizes the definitions; Supplementary Section S1 gives the details and derives the governing equations.

Given a regular planar contour $\boldsymbol{\gamma} : \mathbb{T}^1 \to \mathbb{R}^2$, its average speed

$$\sigma = \frac{1}{2\pi} \int_{\mathbb{T}^1} |\boldsymbol{\gamma}'(u)| \, \mathrm{d}u$$

is its arc length divided by 2π. Reparametrizing by the solution $\xi : \mathbb{T}^1 \to \mathbb{T}^1$ of the initial value problem

$$\xi'(u) = \frac{\sigma}{|\boldsymbol{\gamma}'(\xi(u))|}, \qquad \xi(-\pi) = -\pi$$

gives the *constant-speed representation* $\boldsymbol{\psi} := \boldsymbol{\gamma} \circ \xi$ of the same curve, which satisfies $|\boldsymbol{\psi}'(u)| = \sigma$ for all $u \in \mathbb{T}^1$ (Algorithm S8 describes the numerical implementation). Its *shape* $\boldsymbol{\phi} := \boldsymbol{\psi}/\sigma$ is a unit-speed curve, and the pair $(\boldsymbol{\phi}, \sigma)$ is called the *shape-scaling representation*. In [38] the scale is described instead by $\varepsilon := 1/\sigma^2$.

The unit tangent $\boldsymbol{\tau} = \boldsymbol{\phi}' = \boldsymbol{\psi}'/\sigma$ and the outward normal $\mathbf{n} = \boldsymbol{\tau}^\perp$ can be written as

$$\boldsymbol{\tau}(u) = \begin{bmatrix} -\sin\theta(u) \\ \cos\theta(u) \end{bmatrix}, \qquad \mathbf{n}(u) = \begin{bmatrix} \cos\theta(u) \\ \sin\theta(u) \end{bmatrix},$$

which defines the *tangent-angle function* $\theta(u) := \arctan(-\boldsymbol{\tau}(u)_1, \boldsymbol{\tau}(u)_2)$, chosen to be continuous in u. Since $\boldsymbol{\tau}$ is periodic, $\theta(\pi) - \theta(-\pi) = 2\pi k$ for an integer k, the winding number. The tangent-angle function θ, the average speed σ, and the centroid $\boldsymbol{\mu} = \frac{1}{2\pi}\int_{\mathbb{T}^1} \boldsymbol{\psi}(u)\,\mathrm{d}u$ together form the *tangent-angle representation* $(\theta, \sigma, \boldsymbol{\mu})$ of the contour.

For an integrable function f on $\mathbb{T}^1$, let $\bar{f} = \mu_f = \fint_{\mathbb{T}^1} f(u)\,\mathrm{d}u = \frac{1}{2\pi}\int_{\mathbb{T}^1} f(u)\,\mathrm{d}u$ denote its mean. We use two primitive operators: the mean-zero primitive

$$P_c[f](u) := \int_{-\pi}^{u} f(v)\,\mathrm{d}v + \frac{1}{2\pi}\int_{\mathbb{T}^1} (v-\pi) f(v)\,\mathrm{d}v,$$

which satisfies $\overline{P_c[f]} = 0$ (and is a periodic function on $\mathbb{T}^1$ only if $\mu_f = 0$), and the zero-Dirichlet primitive of $f - \mu_f$,

$$P_0[f](u) := \int_{-\pi}^{u} f(v)\,\mathrm{d}v - (u+\pi)\mu_f.$$

The constant-speed contour is recovered from its tangent-angle representation by

$$\boldsymbol{\psi}(u) = \boldsymbol{\mu} + \sigma P_c[\boldsymbol{\tau}](u), \qquad \boldsymbol{\tau}(u) = \begin{bmatrix} -\sin\theta(u) \\ \cos\theta(u) \end{bmatrix}. \tag{2.8}$$

2.5 Governing Equations in Tangent-Angle Representation

In the tangent-angle representation, the nondimensionalized governing equations (2.7) become (see Supplementary Section S1 for the derivation)

$$\begin{aligned}
\theta_t^{(i)} &= \frac{1}{{\sigma^{(i)}}^2}\left[\theta_{uu}^{(i)} + P_0[a^{(i)}]\theta_u^{(i)} - \mathcal{F}_u^{(i)}\right] \\
\partial_t\left[{\sigma^{(i)}}^2\right] &= -2\mu_{a^{(i)}} \\
\partial_t \boldsymbol{\mu}^{(i)} &= \frac{1}{2\pi\sigma^{(i)}}\int_{\mathbb{T}^1} P_0[a^{(i)}]\boldsymbol{\tau}^{(i)} + \mathcal{F}^{(i)}\mathbf{n}^{(i)}\,\mathrm{d}v \\
a^{(i)} &= (\theta_u^{(i)})^2 - \mathcal{F}^{(i)}\theta_u^{(i)} \\
\mathcal{F}^{(i)}(u,t) &= \sigma^{(i)}\Big(c^{(i)} - \\
&\quad \tau\int_{\mathbb{T}^1} f\left(\frac{1}{2}\left|\boldsymbol{\psi}^{(i)}(u,t) - \boldsymbol{\psi}^{(i)}(v,t)\right|^2\right)\sigma^{(i)}\,\mathrm{d}v \\
&\quad -\tau\sum_{j\neq i}\int_{\mathbb{T}^1} g\left(\frac{1}{2}\left|\boldsymbol{\psi}^{(i)}(u,t) - \boldsymbol{\psi}^{(j)}(v,t)\right|^2\right)\sigma^{(j)}\,\mathrm{d}v\Big) \\
\boldsymbol{\psi}^{(i)} &= \boldsymbol{\mu}^{(i)} + \sigma^{(i)}P_c[\boldsymbol{\tau}^{(i)}],
\end{aligned} \tag{2.9}$$

along with the winding number and periodicity constraints for the tangent-angle function:

$$\theta^{(i)}(\pi,t) - \theta^{(i)}(-\pi,t) = 2\pi k^{(i)}, \qquad k^{(i)} \in \mathbb{Z}$$

$$\theta_u^{(i)}(\pi,t) = \theta_u^{(i)}(-\pi,t).$$

Note that the centroid equation in (2.9) carries the factor $1/\sigma^{(i)} = \varepsilon^{(i)\,1/2}$ (in the notation $\varepsilon = 1/\sigma^2$ of [38]), whereas the equation for $\theta^{(i)}$ carries $1/{\sigma^{(i)}}^2 = \varepsilon^{(i)}$; the numerical scheme (Algorithm S15) must respect this difference. Here $\mathcal{F}^{(i)}$ is the normal forcing (growth minus nonlocal inhibition) scaled by $\sigma^{(i)}$, and $P_0[a^{(i)}]\theta_u^{(i)}$ is a nonlocal tangential (advection) term, which only reparametrizes the contour so as to keep its speed constant; it should not be confused with the nonlocal transport mechanism of Section 2.6. This form of the equations is well suited to stable numerical methods: its highest-order term is a linear diffusion term whose coefficient $1/{\sigma^{(i)}}^2$ depends only on time, and which can therefore be integrated exactly (Algorithm S10), while the nonlinear terms involve at most first derivatives of $\theta^{(i)}$.

2.6 Model Extension: Nonlocal Transport Mechanism

In this subsection we introduce a nonlocal transport mechanism which permits direct attractive or repulsive colony-scale drift induced by interactions between neighboring colonies when intercolony distances decrease below a specified threshold. While the standard model (2.5) is already capable of expressing colony-scale transport via the nonlocal interaction force, the nonlocal transport mechanism introduced here directly parametrizes this type of transport, thus decoupling it from the self- and cross-interactions of the nonlocal interaction force. Additionally, it can capture differences in growth rate between the front and back edges of colonies which the interaction kernels may not capture easily: an attractive transport mechanism acts like an increased growth rate on the "front" edge of a colony (facing its neighbors) and a decreased growth rate on its "back" edge, regardless of the precise distance of each edge from the other colonies. Section 4.3 describes the performance of this extension quantitatively.

Denote the nonlocal transport force by $\mathbf{h}^{(i)}(t) \in \mathbb{R}^2$, $i \in [m]$. We apply this nonlocal transport force to the whole of each colony i as a rigid translation, and take $\mathbf{h}^{(i)}(t)$ to be the sum of pairwise forces acting in the direction of the displacement between centroids of the contours. Let $\boldsymbol{\mu}^{(i)}(t) \in \mathbb{R}^2$ be the location of the centroid of contour i at time t, which for a constant-speed parametrization may be calculated using the formula

$$\boldsymbol{\mu}^{(i)}(t) = \frac{1}{2\pi} \int_{\mathbb{T}^1} \boldsymbol{\gamma}^{(i)}(u,t)\, \mathrm{d}u.$$

Then the nonlocal transport mechanism is given by a sum of pairwise forces

$$\begin{aligned} \mathbf{h}^{(i)}(t) &= \alpha \sum_{j \neq i} \frac{\boldsymbol{\mu}^{(i)}(t) - \boldsymbol{\mu}^{(j)}(t)}{|\boldsymbol{\mu}^{(i)}(t) - \boldsymbol{\mu}^{(j)}(t)|}\, \chi_w\Big(s^{(ij)}(t) - \beta\Big), \\ s^{(ij)}(t) &:= \min_{u,v \in \mathbb{T}^1} \Big|\boldsymbol{\gamma}^{(i)}(u,t) - \boldsymbol{\gamma}^{(j)}(v,t)\Big|, \end{aligned} \tag{2.10}$$

where $s^{(ij)}(t)$ is the minimum distance between colonies i and j, and the gate $\chi_w : \mathbb{R} \to [0,1]$ is the smoothstep function of width $w > 0$

$$\chi_w(r) := \begin{cases} 1, & r \leq -w/2, \\ 1 - 3\left(\frac{1}{2} + \frac{r}{w}\right)^2 + 2\left(\frac{1}{2} + \frac{r}{w}\right)^3, & |r| < w/2, \\ 0, & r \geq w/2. \end{cases} \tag{2.11}$$

Thus the pairwise force between colonies i and j has full strength $|\alpha|$ when the colonies are closer than $\beta - w/2$, vanishes when they are farther apart than $\beta + w/2$, and has half strength when $s^{(ij)} = \beta$; the threshold β is the distance at which the force switches on. The gate χ_w is continuously differentiable, and as $w \to 0$ it converges to the indicator function $\mathbf{1}_{\{s^{(ij)} < \beta\}}$. We use the smooth gate rather than the indicator because the indicator makes the model's right-hand side, and therefore the training loss, discontinuous in β and in the state; with the indicator the derivative of the loss with respect to β is zero almost everywhere (see Section 3.4). The width w is held fixed and is not a trainable parameter. A positive value of α pushes colonies apart and a negative value pulls them together; we do not constrain its sign.

Adding the nonlocal transport mechanism to (2.5) yields the model with transport

$$\begin{aligned} \frac{\partial}{\partial t}\boldsymbol{\gamma}^{(i)}(u,t) = {} & \mathbf{n}^{(i)}(u,t) \Big[\mu\, \kappa_{\boldsymbol{\gamma}^{(i)}}(u,t) + c^{(i)} - \\ & \int_{\mathbb{T}^1} f\left(\frac{1}{2}\left|\boldsymbol{\gamma}^{(i)}(u,t) - \boldsymbol{\gamma}^{(i)}(v,t)\right|^2\right) \left|\partial_u \boldsymbol{\gamma}^{(i)}(v,t)\right| \mathrm{d}v \\ & - \sum_{j \neq i} \int_{\mathbb{T}^1} g\left(\frac{1}{2}\left|\boldsymbol{\gamma}^{(i)}(u,t) - \boldsymbol{\gamma}^{(j)}(v,t)\right|^2\right) \left|\partial_u \boldsymbol{\gamma}^{(j)}(v,t)\right| \mathrm{d}v \Big] \\ & + \alpha \sum_{j \neq i} \frac{\boldsymbol{\mu}^{(i)}(t) - \boldsymbol{\mu}^{(j)}(t)}{|\boldsymbol{\mu}^{(i)}(t) - \boldsymbol{\mu}^{(j)}(t)|}\, \chi_w\Big(s^{(ij)}(t) - \beta\Big), \qquad i \in [m]; \\ s^{(ij)}(t) = {} & \min_{u,v \in \mathbb{T}^1} \Big|\boldsymbol{\gamma}^{(i)}(u,t) - \boldsymbol{\gamma}^{(j)}(v,t)\Big|, \qquad i,j \in [m]. \end{aligned} \tag{2.12}$$

Setting $\alpha = 0$ in (2.12) recovers the model (2.5) of [38], which we refer to as the *standard model*. Note that the standard model carries per-colony growth rates $c^{(i)}$ and allows $f \neq g$ as in [38].

Section 4.2 analyzes the cases where $c^{(i)} = c$ (shared growth rates) or $f = g$ (shared interaction kernels).

Effect on the governing equations. Since $\mathbf{h}^{(i)}(t)$ is a rigid translation, it changes neither the arc length nor the shape of a colony directly, and the evolution of the curve traced by $\boldsymbol{\gamma}^{(i)}(\cdot, t)$ still does not depend on its parametrization. In the tangent-angle representation, the equations for $\theta^{(i)}$ and $\sigma^{(i)}$ are therefore unchanged, and the transport mechanism enters only through the centroid equation (Supplementary Section S1.5). Nondimensionalizing as in Section 2.3 multiplies $\mathbf{h}^{(i)}$ by τ; renaming $\alpha' = \tau\alpha$ to α (the threshold β and width w are lengths and are unaffected), the governing equations with transport are

$$\begin{aligned}
\theta_t^{(i)} &= \frac{1}{{\sigma^{(i)}}^2}\left[\theta_{uu}^{(i)} + P_0[a^{(i)}]\theta_u^{(i)} - \mathcal{F}_u^{(i)}\right] \\
\partial_t\left[{\sigma^{(i)}}^2\right] &= -2\mu_{a^{(i)}} \\
\partial_t \boldsymbol{\mu}^{(i)} &= \frac{1}{2\pi\sigma^{(i)}}\int_{\mathbb{T}^1} P_0[a^{(i)}]\boldsymbol{\tau}^{(i)} + \mathcal{F}^{(i)}\mathbf{n}^{(i)}\,\mathrm{d}v + \mathbf{h}^{(i)} \\
\mathbf{h}^{(i)} &= \alpha\sum_{j\neq i}\frac{\boldsymbol{\mu}^{(i)} - \boldsymbol{\mu}^{(j)}}{|\boldsymbol{\mu}^{(i)} - \boldsymbol{\mu}^{(j)}|}\chi_w\Big(s^{(ij)} - \beta\Big) \\
s^{(ij)} &= \min_{u,v\in\mathbb{T}^1}\left|\boldsymbol{\psi}^{(i)}(u,t) - \boldsymbol{\psi}^{(j)}(v,t)\right| \\
a^{(i)} &= (\theta_u^{(i)})^2 - \mathcal{F}^{(i)}\theta_u^{(i)} \\
\mathcal{F}^{(i)}(u,t) &= \sigma^{(i)}\Big(c^{(i)} - \\
&\quad \tau\int_{\mathbb{T}^1} f\left(\frac{1}{2}\left|\boldsymbol{\psi}^{(i)}(u,t) - \boldsymbol{\psi}^{(i)}(v,t)\right|^2\right)\sigma^{(i)}\,\mathrm{d}v \\
&\quad -\tau\sum_{j\neq i}\int_{\mathbb{T}^1} g\left(\frac{1}{2}\left|\boldsymbol{\psi}^{(i)}(u,t) - \boldsymbol{\psi}^{(j)}(v,t)\right|^2\right)\sigma^{(j)}\,\mathrm{d}v\Big) \\
\boldsymbol{\psi}^{(i)} &= \boldsymbol{\mu}^{(i)} + \sigma^{(i)}P_c[\boldsymbol{\tau}^{(i)}],
\end{aligned} \tag{2.13}$$

with the same winding number and periodicity constraints as before. The transport mechanism translates each colony rigidly, but it still affects the shapes and growth of the colonies indirectly, because the cross-interaction term in $\mathcal{F}^{(i)}$ depends on the distances between colonies. As before, setting $\alpha = 0$ recovers (2.9).

3 Model Training

3.1 Trainable Model Parameters

Let $\mathcal{C} = \{\boldsymbol{\gamma}^{(i)}\}_{i\in[m]}$ be a collection of m parametrized smooth, simple, counterclockwise oriented contours $\boldsymbol{\gamma}^{(i)} : \mathbb{T}^1 \times \mathbb{R}^+ \to \mathbb{R}^2$ representing boundaries of colonies of *Paenibacillus dendritiformis* bacteria. The dynamics of these interfaces over time are described by the nonlocal equations (2.12). Since these reduce to the standard model (2.5) when $\alpha = 0$, we work with the more general form throughout this section, so that one set of equations covers both cases. These governing equations contain $\mu, c^{(1)}, \ldots, c^{(m)} \in \mathbb{R}^+$ as model parameters, together with $\alpha \in \mathbb{R}$ and $\beta \in \mathbb{R}^+$ when the transport mechanism is included (the gate width w is held fixed). Furthermore, the functions f and g are assumed to be functions from $\mathbb{R}^+$ to $\mathbb{R}^+$ which are members of a chosen parametrized

class of interaction kernels. In this work, two classes of interaction kernels, each parametrized using two parameters, are considered:

$$\text{Gaussian} := \left\{ f : \mathbb{R}^+ \to \mathbb{R} \;\middle|\; f(s) = \frac{c}{\sigma\sqrt{2\pi}} \exp\left(-\frac{s}{\sigma^2}\right), \; (c, \sigma) \in (\mathbb{R}^+)^2 \right\}$$

$$\text{Ramp} := \left\{ f : \mathbb{R}^+ \to \mathbb{R} \;\middle|\; f(s) = \max(a - b\sqrt{s}, 0), \; (a, b) \in (\mathbb{R}^+)^2 \right\}$$

Note that because the interaction kernels f and g use one-half of the squared distance as their argument, a function $f \in$ Ramp will have the shape of a downward piecewise linear ramp in terms of distance, hence the name.

Since both the Ramp and Gaussian classes of interaction kernels are closed under nonnegative scalar multiplication, the rescaled kernels τf and τg in the nondimensionalized equations (Section 2.3) remain within the same class for any $\tau > 0$, so we train them directly. In the same way, $c^{(i)}$ and α denote the nondimensional growth rates and transport strength of Sections 2.3 and 2.6.

Altogether, the standard model ($\alpha = 0$) is parametrized by a set of $m + 5$ parameters: the scaling parameters $\{c^{(i)}\}_{i\in[m]}$ and τ, along with two parameters for each of the interaction kernels f and g. In the case of the Gaussian kernels, these parameters may be written as

$$\{c^{(1)}, \ldots, c^{(m)}, \tau, c_f, \sigma_f, c_g, \sigma_g\} \in (\mathbb{R}^+)^{m+5},$$

and in the case of the Ramp kernels, these parameters may be written as

$$\{c^{(1)}, \ldots, c^{(m)}, \tau, a_f, b_f, a_g, b_g\} \in (\mathbb{R}^+)^{m+5}.$$

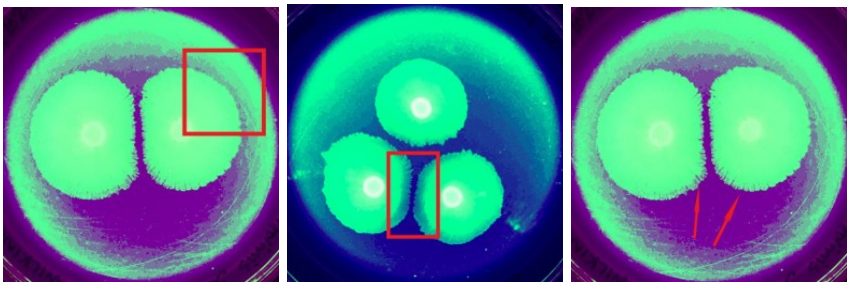
If a shared growth rate is used, then the number of parameters decreases to six in each case, and if shared interaction kernels are used, then the number of parameters decreases by two (to $m + 3$ if using varying growth rates, and to four if using a constant growth rate).

Including the nonlocal transport mechanism adds two parameters, the strength $\alpha \in \mathbb{R}$ and the threshold $\beta \in \mathbb{R}^+$, for a total of $m+7$ (the width w of the gate (2.11) is held fixed) if the rest of the model is standard. Because the gate is continuously differentiable, the loss depends continuously and differentiably on β (see Section 3.4); this would not be the case if the gate were replaced by the indicator function $\mathbf{1}_{\{s^{(ij)}<\beta\}}$.

3.2 Target Data

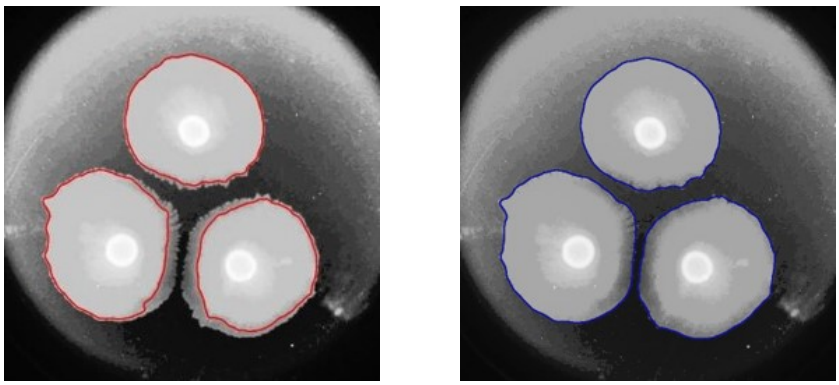
This work uses data from *Paenibacillus dendritiformis* bacterial colony growth experiments in [40]. Details about the experiments can be found in Appendix 6.1 and in [40]. The experiments yielded seven image sequences of Petri dishes containing growing collections of bacterial colonies.

Next, an image segmentation procedure was used to identify the regions in each image containing the colonies and the background, after which the boundaries between regions were extracted as contours containing 100 points each. Since a simple thresholding algorithm was insufficient to accurately capture the boundaries of the colonies due to the presence of false edges, the image segmentation procedure was based on seeded region growing (see Figure 2).



(a) Similar colors in background (b) False edges within the colony (c) Feathering effect on colony edges

Figure 1: Difficulties in Contour Extraction



(a) Thresholding (b) Seeded region growing

Figure 2: This figure compares two methods of contour extraction (thresholding and seeded region growing) on the 130$^{\text{th}}$ image in sequence 4. Here, thresholding is tricked by false edges within the colony and does not correctly capture the boundaries.

Another crucial goal of the image processing was to extract smooth contours. In the images of *P. dendritiformis* colonies, fine structures sometimes called "feathering" appear on the edges of the colonies (see Figure 1c). This is undesirable since the nonlocal model equations do not capture this level of intricate detail, but instead treat the contours as smooth curves. To smooth out the segmented regions, a bilateral filter and median blur were applied to the images before segmentation.

After contour extraction, three of the highest-quality sequences of contours were chosen to form the target data to train the model parameters. Dataset 1 was taken from the seventh sequence, Dataset 2 was taken from the fifth sequence, and Dataset 3 was taken from the third sequence. Datasets 1 and 2 are two-colony datasets, the former having two colonies of similar size and the latter having two colonies of differing size, while Dataset 3 is a three-colony dataset. The target data for each contour consists of 100 points in $\mathbb{R}^2$, oriented counterclockwise, which represent locations of pixels corresponding to the associated contour. If we let $u_j = -\pi + \frac{2\pi}{n}(j-1)$ for $j \in [n]$ with $n = 100$, then to define the ith target contour at time t_k we let

$$\tilde{\boldsymbol{\gamma}}_{j,k}^{(i)} := \tilde{\boldsymbol{\gamma}}^{(i)}(u_j, t_k) \in \mathbb{R}^2$$

represent the location of the jth extracted point in the contour.

To form a continuous representation of the target data, $\tilde{\boldsymbol{\gamma}}^{(i)}(u, t_k)$ can be defined at any other position $u \in \mathbb{T}^1$ using a WENO (Weighted Essentially Non-Oscillatory) interpolation scheme based on quadratic Lagrange polynomials (see Algorithm S5).

The extracted contour points are resampled so that the contour they represent has constant speed (see Algorithm S8). Since the sampling points in $\mathbb{T}^1$ are equally spaced, this means that the distance between successive contour points, along the contour (in arc length), should be equal. This enables the speed of the contour and the distance between contour points to be expressed as constants, rather than varying functions on $\mathbb{T}^1$, simplifying the modeling and allowing the use of the tangent-angle representation.

The resampling is done by computing a change of variables function $\tilde{\xi}^{(i)} : \mathbb{T}^1 \times \mathbb{R} \to \mathbb{T}^1$ which satisfies the ODE initial value problem

$$\partial_u \tilde{\xi}^{(i)}(u) = \frac{\tilde{\sigma}^{(i)}}{|\partial_u \tilde{\boldsymbol{\gamma}}^{(i)}(u)|}, \qquad \tilde{\xi}^{(i)}(-\pi) = -\pi.$$

The initial value problem is solved using the explicit fourth-order Runge–Kutta scheme, and then a new constant-speed representation $\tilde{\boldsymbol{\psi}}^{(i)} : \mathbb{T}^1 \times \mathbb{R} \to \mathbb{R}^2$ is created by interpolating at the new sample locations given through mapping by $\tilde{\xi}^{(i)}$ (see Algorithm S8 for more detail).

Lastly, the contour points are recalibrated so that the first point is oriented in the $(1, 0)$ direction from the centroid of the contour (see Algorithm S9). Once again, we assume that the contours are oriented in a counterclockwise direction, and are star-shaped with respect to their centroids so that this operation is well-defined. The recalibration is done in order to facilitate easy calculation of training metrics by comparing distances between corresponding points on target and simulated contours.

3.3 Training Metrics

Continuous Training Metrics. Let $\{\boldsymbol{\psi}^{(i)}\}_{i\in[m]}$ be a set of m continuous constant-speed simulated contours, and let $\{\tilde{\boldsymbol{\psi}}^{(i)}\}_{i\in[m]}$ be the continuous constant-speed target contours, with average speed $\tilde{\sigma}^{(i)}(t)$ and centroids $\boldsymbol{\mu}^{(i)}(t)$. The absolute loss at each time step is defined as

$$L_{\text{abs}}(t) = \frac{1}{m} \sum_{i=1}^{m} \int_{\mathbb{T}^1} |\boldsymbol{\psi}^{(i)}(u, t) - \tilde{\boldsymbol{\psi}}^{(i)}(u, t)|^2 \, \tilde{\sigma}^{(i)}(t) \, \mathrm{d}u$$

and the relative loss is defined as

$$L_{\text{rel}}(t) = \frac{\sum_{i=1}^{m} \int_{\mathbb{T}^1} |\boldsymbol{\psi}^{(i)}(u, t) - \tilde{\boldsymbol{\psi}}^{(i)}(u, t)|^2 \, \tilde{\sigma}^{(i)}(t) \, \mathrm{d}u}{\sum_{i=1}^{m} \int_{\mathbb{T}^1} |\tilde{\boldsymbol{\psi}}^{(i)}(u, t) - \boldsymbol{\mu}^{(i)}(t)|^2 \, \tilde{\sigma}^{(i)}(t) \, \mathrm{d}u}.$$

Note that in this metric, the simulated contour at a point $u \in \mathbb{T}^1$, $\boldsymbol{\psi}^{(i)}(u, t)$, is compared to $\tilde{\boldsymbol{\psi}}^{(i)}(u, t)$, not $\tilde{\boldsymbol{\psi}}^{(i)}(v, t)$ for some $v \neq u$, so the metric requires that $\boldsymbol{\psi}^{(i)}(u, t)$ and $\tilde{\boldsymbol{\psi}}^{(i)}(u, t)$ correspond to the same part of their respective contours. Ideally,

$$u = \operatorname*{argmin}_{v\in\mathbb{T}^1} |\boldsymbol{\psi}^{(i)}(v, t) - \tilde{\boldsymbol{\psi}}^{(i)}(u, t)|$$

but this may be impossible to accomplish if the contours have different shapes. Instead, we ensure comparability of the parametrizations of the contours by calibrating each so that $\boldsymbol{\psi}^{(i)}(-\pi, t)$ and $\tilde{\boldsymbol{\psi}}^{(i)}(-\pi, t)$ correspond to the points directly to the right of the respective centroids (assuming the contours are star-shaped with respect to each centroid). See Algorithm S9 for further details.

The (absolute or relative) loss over an interval $[0, T]$ is calculated as the mean of the (absolute or relative) loss at each time step. For instance,

$$L_{\text{abs}} = \frac{1}{T}\int_0^T \frac{1}{m}\sum_{i=1}^{m}\int_{\mathbb{T}^1} |\boldsymbol{\psi}^{(i)}(u,t) - \tilde{\boldsymbol{\psi}}^{(i)}(u,t)|^2\, \tilde{\sigma}^{(i)}(t)\,\mathrm{d}u\,\mathrm{d}t$$

$$L_{\text{rel}} = \frac{1}{T}\int_0^T \frac{\sum_{i=1}^{m}\int_{\mathbb{T}^1} |\boldsymbol{\psi}^{(i)}(u,t) - \tilde{\boldsymbol{\psi}}^{(i)}(u,t)|^2\, \tilde{\sigma}^{(i)}(t)\,\mathrm{d}u}{\sum_{i=1}^{m}\int_{\mathbb{T}^1} |\tilde{\boldsymbol{\psi}}^{(i)}(u,t) - \boldsymbol{\mu}^{(i)}(t)|^2\, \tilde{\sigma}^{(i)}(t)\,\mathrm{d}u}\,\mathrm{d}t$$

Discrete Training Metrics. As before, let $\{\boldsymbol{\psi}^{(i)}\}_{i\in[m]}$ be a set of m continuous constant-speed simulated contours, and let $\{\tilde{\boldsymbol{\psi}}^{(i)}\}_{i\in[m]}$ be the continuous constant-speed target contours, with average speed $\tilde{\sigma}^{(i)}(t)$ and centroids $\boldsymbol{\mu}^{(i)}$. Now if we let

$$u_j = -\pi + \frac{2\pi}{n}(j-1), \qquad j \in [n],$$

we may define

$$\boldsymbol{\psi}_j^{(i)} = \boldsymbol{\psi}^{(i)}(u_j), \qquad \tilde{\boldsymbol{\psi}}_j^{(i)} = \tilde{\boldsymbol{\psi}}^{(i)}(u_j) \qquad i \in [m],\ j \in [n].$$

When running a numerical simulation, these collections of n points constitute the full representation of the contour, so it is worth explaining the implementation of the training metrics in this discrete case. The absolute loss is defined as

$$L_{\text{abs}} = \frac{1}{mn}\sum_{i=1}^{m}\sum_{j=1}^{n} |\boldsymbol{\psi}_j^{(i)} - \tilde{\boldsymbol{\psi}}_j^{(i)}|^2\, \tilde{\sigma}^{(i)}$$

and the relative loss is defined as

$$L_{\text{rel}} = \frac{\sum_{i=1}^{m}\sum_{j=1}^{n} |\boldsymbol{\psi}_j^{(i)} - \tilde{\boldsymbol{\psi}}_j^{(i)}|^2\, \tilde{\sigma}^{(i)}}{\sum_{i=1}^{m}\sum_{j=1}^{n} |\tilde{\boldsymbol{\psi}}_j^{(i)} - \boldsymbol{\mu}^{(i)}|^2\, \tilde{\sigma}^{(i)}}$$

Now suppose we have K snapshots of a set of m target and simulated constant-speed contours, consisting of n points each, at times $\{t_1, \ldots, t_K\}$. We may write the associated simulated points as

$$\{\boldsymbol{\psi}_{j,k}^{(i)}, \quad i \in [m], \quad j \in [n], \quad k \in [K]\} \subseteq \mathbb{R}^2$$

and the associated target points as

$$\{\tilde{\boldsymbol{\psi}}_{j,k}^{(i)}, \quad i \in [m], \quad j \in [n], \quad k \in [K]\} \subseteq \mathbb{R}^2.$$

Suppose that the average speeds and centroids of the target contours are given by

$$\{\tilde{\sigma}_k^{(i)}, \quad i \in [m], \quad k \in [K]\}$$

and

$$\{\boldsymbol{\mu}_k^{(i)}, \quad i \in [m], \quad k \in [K]\}.$$

Then we can define the absolute and relative loss over the time sequence $\{t_1, \ldots, t_K\}$ by

$$L_{\text{abs}} = \frac{1}{Kmn}\sum_{k=1}^{K}\sum_{i=1}^{m}\sum_{j=1}^{n} |\boldsymbol{\psi}_{j,k}^{(i)} - \tilde{\boldsymbol{\psi}}_{j,k}^{(i)}|^2\, \tilde{\sigma}_k^{(i)}$$

$$L_{\mathrm{rel}} = \frac{1}{K} \sum_{k=1}^{K} \frac{\sum_{i=1}^{m} \sum_{j=1}^{n} |\boldsymbol{\psi}_{j,k}^{(i)} - \tilde{\boldsymbol{\psi}}_{j,k}^{(i)}|^2 \, \tilde{\sigma}_k^{(i)}}{\sum_{i=1}^{m} \sum_{j=1}^{n} |\tilde{\boldsymbol{\psi}}_{j,k}^{(i)} - \boldsymbol{\mu}_k^{(i)}|^2 \, \tilde{\sigma}_k^{(i)}}. \tag{3.1}$$

Comparison with Prior Work. The training approach of [40] fit a version of the standard model with shared Gaussian self- and cross-interaction kernels and a shared growth rate, as described above. Results on that class of models are in Section 4.2.

The work of [40] also used a different loss: each boundary point on the target (observed) boundary contour was compared with the nearest point on the simulated contour, weighted by arc length and summed over the images. The loss used here instead compares points with the same parameter value u, after both contours have been given constant-speed parametrizations and calibrated to a common reference direction. This makes the loss at each image a smooth quadratic function of the simulated contour points, whereas a nearest-neighbor distance has kinks whenever the nearest neighbor of a point changes. The relative loss is also normalized by the spread of the target contours at each image, so that losses can be compared across datasets whose colonies differ in size.

3.4 Optimization Problems

Training consists of an iterative optimization method. Each evaluation of the loss requires simulating discrete contours over an interval $[0, T]$ and comparing them with target discrete contours extracted from images at a series of times $\{0, t_1, \ldots, t_K\} \subseteq [0, T]$. Model parameters are then updated based on an optimization scheme. The objective is the relative loss L_{rel} (see Eq. (3.1)) of the previous section.

This is a PDE-constrained optimization problem. Computing the gradient of the objective exactly would require either the adjoint of the nonlinear, nonlocal governing equations or automatic differentiation through the specialized numerical scheme used to solve them. Since the model has at most ten parameters, it is instead practical to approximate the gradient by central finite differences, at a cost of $2d+1$ forward simulations per gradient for d parameters. These simulations are independent of each other and can be run in parallel. For this to work, the discrete objective must be differentiable in the parameters, at least on the scale of the finite-difference step; the end of this section explains why this holds after two small modifications to the numerical scheme. We use two optimization methods: the derivative-free Nelder–Mead method (Section 3.5) and the quasi-Newton method L-BFGS-B with finite-difference gradients (Section 3.6).

Simply computing the simulated evolution of the contours on an interval $[0, T]$ with a prescribed initial condition is already expensive, so each evaluation of the objective is costly. The wall-clock time for an evaluation of the objective in the standard model can be as low as 15.5 seconds if τ is large, but as long as 423 seconds if τ is small and more time steps are required.

This section describes the constrained optimization problem at three levels of complexity, starting with the optimization problem in the continuous planar contour formulation, then moving to the continuous tangent-angle formulation, and then moving to the discrete tangent-angle formulation (which is the one actually implemented in code to train the model). Details on this formulation are further explained in Supplementary Sections S2 and S3.

All of the contours are assumed to have constant speed, in the sense that the derivative of the parametrization they define has a constant magnitude. Hence they are denoted by $\boldsymbol{\psi}$ and $\tilde{\boldsymbol{\psi}}$ instead of $\boldsymbol{\gamma}$ and $\tilde{\boldsymbol{\gamma}}$. Furthermore, in each formulation, the class of kernels $\mathcal{F}$ may be chosen to be Ramp or Gaussian, which are each two-parameter families. The simulated contours are calibrated as described in the previous section before they are compared with the target contours. To shorten

the notation, let

$$D(t) := \sum_{i=1}^{m} \int_{\mathbb{T}^1} |\tilde{\boldsymbol{\psi}}^{(i)}(u,t) - \boldsymbol{\mu}^{(i)}(t)|^2 \, \tilde{\sigma}^{(i)}(t) \, \mathrm{d}u$$

denote the spread of the target contours at time t (the denominator of the relative loss), which does not depend on the model parameters. The problems are stated for the model with transport; for the standard model, $\alpha = 0$ is fixed and the minimization is over the remaining $m + 5$ parameters.

Optimization Problem for Continuous Planar Contours. The optimization problem for continuous planar contours may be formulated as follows:

$$\begin{aligned}
&\min_{\substack{f,g\in\mathcal{F};\, c^{(1)},\ldots,c^{(m)},\tau\in\mathbb{R}^+ \\ \alpha\in\mathbb{R};\, \beta\in\mathbb{R}^+}} \frac{1}{T}\int_0^T \frac{1}{D(t)} \\
&\qquad\qquad \times \sum_{i=1}^{m} \int_{\mathbb{T}^1} |\boldsymbol{\psi}^{(i)}(u,t) - \tilde{\boldsymbol{\psi}}^{(i)}(u,t)|^2 \, \tilde{\sigma}^{(i)}(t) \, \mathrm{d}u \, \mathrm{d}t \\
&\text{subject to} \quad \frac{\partial}{\partial t}\boldsymbol{\psi}^{(i)}(u,t) = \mathbf{n}_{\boldsymbol{\psi}^{(i)}}(u,t) \Big[(\kappa_{\boldsymbol{\psi}^{(i)}}(u,t) + c^{(i)}) \\
&- \int_{\mathbb{T}^1} \tau f\left(\frac{1}{2}\left|\boldsymbol{\psi}^{(i)}(u,t) - \boldsymbol{\psi}^{(i)}(v,t)\right|^2\right) \left|\partial_u \boldsymbol{\psi}^{(i)}(v,t)\right| \mathrm{d}v \\
&- \sum_{j\neq i} \int_{\mathbb{T}^1} \tau g\left(\frac{1}{2}\left|\boldsymbol{\psi}^{(i)}(u,t) - \boldsymbol{\psi}^{(j)}(v,t)\right|^2\right) \\
&\qquad \times \left|\partial_u \boldsymbol{\psi}^{(j)}(v,t)\right| \mathrm{d}v \Big] \\
&+ \alpha \sum_{j\neq i} \frac{\boldsymbol{\mu}^{(i)}(t) - \boldsymbol{\mu}^{(j)}(t)}{\|\boldsymbol{\mu}^{(i)}(t) - \boldsymbol{\mu}^{(j)}(t)\|_2} \, \chi_w\Big(s^{(ij)}(t) - \beta\Big), \\
&\boldsymbol{\psi}^{(i)}(u,0) = \tilde{\boldsymbol{\psi}}^{(i)}(u,0) \qquad i \in [m]
\end{aligned}$$

Optimization Problem for Continuous Tangent-Angle Representation of Contours. The optimization problem using the equivalent governing equations expressed in the tangent-angle representation may be formulated as follows:

$$\begin{aligned}
&\min_{\substack{f,g\in\mathcal{F};\, c^{(1)},\ldots,c^{(m)},\tau\in\mathbb{R}^+ \\ \alpha\in\mathbb{R};\, \beta\in\mathbb{R}^+}} \frac{1}{T}\int_0^T \frac{1}{D(t)} \\
&\qquad\qquad \times \sum_{i=1}^{m} \int_{\mathbb{T}^1} |\boldsymbol{\psi}^{(i)}(u,t) - \tilde{\boldsymbol{\psi}}^{(i)}(u,t)|^2 \, \tilde{\sigma}^{(i)}(t) \, \mathrm{d}u \, \mathrm{d}t \\
&\text{subject to } \partial_t \theta^{(i)}(u,t) = \varepsilon^{(i)}(t) \left(\partial_u^2 \theta^{(i)} - \partial_u \mathcal{F}^{(i)}\right) \\
&+ \varepsilon^{(i)}(t) \left[\int_{-\pi}^{u} a^{(i)}(v) \, \mathrm{d}v - (u+\pi) \fint_{\mathbb{T}^1} a^{(i)}(v) \, \mathrm{d}v\right] \partial_u \theta^{(i)}, \\
&\partial_t \varepsilon^{(i)} = 2\varepsilon^{(i)}(t)^2 \fint_{\mathbb{T}^1} a^{(i)}(v) \, \mathrm{d}v \\
&\partial_t \boldsymbol{\mu}^{(i)}(t) = \varepsilon^{(i)}(t)^{1/2} \fint_{\mathbb{T}^1} \mathbf{v}^{(i)}(u) \, \mathrm{d}u + \mathbf{h}^{(i)}(t) \\
&\mathcal{F}^{(i)}(u,t) = \sigma^{(i)}(t) \Big[c^{(i)}
\end{aligned}$$

$$
\begin{aligned}
&-\tau \int_{\mathbb{T}^1} f\left(\frac{1}{2}\left|\boldsymbol{\psi}^{(i)}(u,t)-\boldsymbol{\psi}^{(i)}(v,t)\right|^2\right)\sigma^{(i)}(t)\,\mathrm{d}v \\
&-\tau \sum_{j\neq i}\int_{\mathbb{T}^1} g\left(\frac{1}{2}\left|\boldsymbol{\psi}^{(i)}(u,t)-\boldsymbol{\psi}^{(j)}(v,t)\right|^2\right)\sigma^{(j)}(t)\,\mathrm{d}v\Bigg] \\
&a^{(i)}(u,t) = (\partial_u\theta^{(i)}(u,t))^2 - \mathcal{F}^{(i)}(u,t)\partial_u\theta^{(i)}(u,t) \\
&\mathbf{v}^{(i)}(u,t) = \left[\int_{-\pi}^{u} a^{(i)}(v)\,\mathrm{d}v - (u+\pi)\fint_{\mathbb{T}^1} a^{(i)}(v)\,\mathrm{d}v\right]\boldsymbol{\tau}^{(i)} \\
&\qquad + \mathcal{F}^{(i)}\mathbf{n}^{(i)} \\
&\boldsymbol{\psi}^{(i)}(u,t) = \boldsymbol{\mu}^{(i)}(t) + \sigma^{(i)}(t)P_c[\boldsymbol{\tau}^{(i)}(\cdot,t)](u) \\
&\boldsymbol{\psi}^{(i)}(u,0) = \tilde{\boldsymbol{\psi}}^{(i)}(u,0) \qquad i\in[m]
\end{aligned}
$$

where $\varepsilon^{(i)} = 1/{\sigma^{(i)}}^2$ and $\mathbf{h}^{(i)}$ is the nonlocal transport mechanism (2.10).

Optimization Problem for Discrete Tangent-Angle Representation of Contours. Let $\{0, t_1, \ldots, t_K\} \subseteq [0,T]$ be the sequence of times at which the images are taken, and let

$$
D_k := \sum_{i=1}^{m}\sum_{j=1}^{n} |\tilde{\boldsymbol{\psi}}_{j,k}^{(i)} - \boldsymbol{\mu}_k^{(i)}|^2\,\tilde{\sigma}_k^{(i)}.
$$

Then the governing equations may be simulated numerically (see Algorithms S15 and S1 for the implementation used in this work), and the optimization problem using the discrete formulation is as follows:

$$
\begin{aligned}
&\min_{\substack{f,g\in\mathcal{F};\ c^{(1)},\ldots,c^{(m)},\tau\in\mathbb{R}^+ \\ \alpha\in\mathbb{R};\ \beta\in\mathbb{R}^+}} \sum_{k=1}^{K}\frac{1}{D_k}\sum_{i=1}^{m}\sum_{j=1}^{n}|\boldsymbol{\psi}_{j,k}^{(i)} - \tilde{\boldsymbol{\psi}}_{j,k}^{(i)}|^2\,\tilde{\sigma}_k^{(i)} \\
&\text{subject to } \{\boldsymbol{\psi}_{j,k}^{(i)}, i\in[m], j\in[n], k\in[K]\} \text{ generated by} \\
&\text{Algorithm S1 with } N_{\text{sub}} \text{ substeps per image and} \\
&\boldsymbol{\psi}_{j,0}^{(i)} = \tilde{\boldsymbol{\psi}}_{j,0}^{(i)}, \qquad i\in[m], j\in[n]
\end{aligned}
$$

Time Discretization. Algorithm S1 (Supplementary Section S2) advances the solution from one image to the next in a fixed number N_{sub} of equal substeps. Since consecutive images are $1/\tau$ units of nondimensional time apart (Section 2.3), each substep has length

$$
\Delta t = \frac{1}{\tau N_{\text{sub}}}.
$$

The number N_{sub} is chosen once, before an optimization run begins, and is held fixed during the run, so that a change in τ changes the length of each substep but not the number of substeps. We choose N_{sub} so that $\Delta t/\Delta x \le 0.75$ at the initial value of τ, where $\Delta x = 2\pi/n$ is the grid spacing on $\mathbb{T}^1$. As τ changes during an optimization run, $\Delta t/\Delta x$ may drift; in runs where it drifted too high, we restarted the run from its best point with a larger value of N_{sub}.

Optimization runs with small τ values, which need many substeps, were sometimes run with coarser steps (up to $\Delta t/\Delta x \approx 3$) to reduce their cost; but every loss and parameter set reported in Section 4 was re-evaluated with $\Delta t/\Delta x \le 0.75$.

The time-stepping scheme is second-order accurate in Δt (see Supplementary Section S3), so the discretization error falls by a factor of about four each time N_{sub} is doubled. For example, at the optimized parameters of the standard model for Dataset 1 with Gaussian kernels, the relative difference between the objective and its value computed with 400 substeps per image was 3.6×10^{-6}, 1.4×10^{-5}, 5.8×10^{-5}, and 2.7×10^{-4} at $\Delta t/\Delta x = 0.18$, 0.36, 0.71, and 1.48 respectively. The error is therefore negligible when $\Delta t/\Delta x \le 0.75$, and the fitted parameters are insensitive to the precise choice of N_{sub} as long as it is not too small.

The time-stepping scheme also contains a time step restriction which keeps the average speeds $\sigma^{(i)}$ positive (Algorithm S15). With a fixed number of substeps, this restriction is only a safeguard, and it never became active in any of our runs: at the optimized parameters of the configurations we checked, the maximum allowed step exceeded Δt by a factor of at least 43 (Supplementary Section S3).

Differentiability of the Objective. Gradient-based optimization requires the objective to be differentiable in the model parameters, at least on the scale of the finite-difference step used to estimate its gradient (Section 3.6). Two features of earlier versions of our numerical scheme made the discrete objective discontinuous, and both have been removed:

1. *The transport gate.* Using the smoothstep χ_w in place of the indicator function $\mathbf{1}_{\{s^{(ij)}<\beta\}}$ makes the objective continuously differentiable in both $s^{(ij)}$ and β. Using the indicator function makes the objective piecewise constant in β: its derivative with respect to β is zero wherever it exists, and jumps occur whenever a change in the parameters moves the time at which a pair of colonies crosses the threshold β past a time step.

2. *The number of time steps.* Earlier versions used a fixed time step Δt and took a shorter final step onto each image time. The number of steps between images, $\lceil 1/(\tau\Delta t)\rceil$, then changes by one whenever $1/\tau$ crosses a multiple of Δt, and the objective jumps. Near the optimum for Dataset 1 with Gaussian kernels, the jump across one such boundary was 1.6×10^{-4} of the objective, and the forward difference quotient with respect to $1/\tau$ across the jump was about 7.8, compared with values of order 0.1 on either side. With a fixed number of substeps per image, the time grid scales with $1/\tau$ and the objective is smooth.

The remaining operations in the numerical scheme are continuous and piecewise smooth. The FFT-based differentiation and integration (Algorithms S6 and S7) are linear, and the exponential time-differencing update (Algorithm S10) and the Newton iteration used for recentering (Algorithm S13) are smooth. The WENO interpolation (Algorithm S5) branches on the ratio of its two smoothness indicators for numerical stability, but both branches evaluate the same rational function of the indicators, which are bounded away from zero; the interpolation weights are therefore smooth functions of the data. Resampling (Algorithm S8) is applied only to the target data, since the tangent-angle representation yields a constant-speed contour at every step. Recalibration (Algorithm S9) is applied to the simulated contours only when they are compared with the target contours to calculate the loss, and does not feed back into the dynamics; it depends on the parameters through the location of a zero crossing found by linear interpolation, which moves continuously, although its derivative jumps when the crossing moves from one grid interval to the next. Finally, the Ramp kernels $\max(a - b\sqrt{s}, 0)$ and the discrete minimum distance $s^{(ij)}$ in the transport gate are Lipschitz but not differentiable everywhere: the former has a kink when the distance between a pair of grid points crosses the end of the ramp, and the latter when the closest pair of grid points changes.

The discrete objective is therefore continuous and piecewise smooth, and each of its kinks comes from a single pair of grid points (or a single grid interval) and changes the slope of the objective

only slightly. In practice the objective behaves as a smooth function on the scale of our finite-difference step. We checked this directly by computing central-difference estimates of the gradient with steps h ranging from 10^{-6} to 10^{-2} (relative steps, in the log parametrization of Section 3.6). In a check done with the shared growth rate model with Gaussian kernels at its initial point for Dataset 1, the estimates of all six partial derivatives agreed to within 0.01% for $h \in [10^{-5}, 10^{-3}]$. With the transport mechanism and the smoothstep gate, the estimates of the partial derivatives with respect to α and β agreed to within 0.1% and 0.3% respectively. We therefore use $h = 10^{-4}$. Differentiability does not rule out very steep slopes, however: near the small-τ shared growth rate optima of Section 4.2, the objective is steep enough in some directions that refining them with L-BFGS-B was ineffective.

The Ramp kernels are only Lipschitz, and for them the nonlocal forces are continuously differentiable only under additional conditions on the set of pairs of points at distance $\sqrt{2}\,a/b$; we do not pursue this here. We do not attempt a proof for the discrete model, whose objective is only piecewise smooth for the reasons given above; the numerical checks above show that it behaves as a smooth function at the scale that matters for our optimization.

3.5 Nelder–Mead Method

We use the Nelder–Mead method (sometimes called the amoeba method), a direct search method based on comparing function values at points on a simplex, for optimizing the objective function [56]. This method works with a simplex (a polytope of $d+1$ vertices in $\mathbb{R}^d$) by iteratively updating the point with the worst value in the simplex via reflection, expansion, contraction, or shrinkage of the simplex. Algorithm S3 (Supplementary Section S2) describes `scipy`'s implementation of the Nelder–Mead method [57]. While the Nelder–Mead method has descent guarantees in the case of a uniformly convex function, our problem is not uniformly convex and these guarantees do not apply.

We apply the Nelder–Mead method to the model parameters directly (not to the log parametrization of Section 3.6), with `scipy`'s default convergence tolerances (10^{-4} on the size of the simplex and on the spread of the function values across its vertices). When the simplex converges, we restart the method from the best point found with a fresh simplex, and we stop once a restart improves the best loss by less than a relative amount of 10^{-4}, or when the time budget for the run is exhausted.

Because the Nelder–Mead method moves a simplex of parameters, the loss is evaluated many times. The corresponding loss curve appears somewhat oscillatory as the algorithm tests points and finds that some do not have as low an objective function value as those previously tested.

3.6 L-BFGS-B with Finite-Difference Gradients

Since the objective is differentiable on the scale of our finite-difference step (Section 3.4), it can also be minimized with a gradient-based quasi-Newton method. We use L-BFGS-B [58, 59], a limited-memory BFGS method which handles bound constraints, as implemented in `scipy` [57], with the gradient approximated by central differences [60]. The only bounds are the wide box constraints described below, which serve as a safeguard and are not active at any of the solutions we report. The limited memory usage is not strictly necessary for an optimization problem with few parameters; nevertheless, L-BFGS-B was chosen for its modern implementation in `scipy`. Algorithm S4 (Supplementary Section S2) summarizes the method.

Log Parametrization. The model parameters differ in magnitude by several orders of magnitude, and all of them except α must be positive. We therefore optimize over variables $\mathbf{u}$ with $x_i = e^{u_i}$ for every parameter x_i except the transport parameters α and β, which are optimized

directly (α because it may change sign; β is bounded below by 10^{-3}). In these variables positivity is automatic, a step in u_i is a relative change in x_i, and a single finite-difference step size is appropriate for all of the log-parametrized parameters. We also bound each log-parametrized variable to within 5 of its initial value, i.e. to within a factor of $e^5 \approx 150$ of the initial parameter.

Central Differences. The partial derivative with respect to u_i is approximated by

$$\hat{g}_i(\mathbf{u}) = \frac{L(\mathbf{u} + h_i \mathbf{e}_i) - L(\mathbf{u} - h_i \mathbf{e}_i)}{2h_i},$$

with $h_i = h$ for the log-parametrized variables and $h_i = h\,|x_{0,i}|$ for α and β, where $\mathbf{x}_0$ is the initial point. One gradient therefore costs $2d$ evaluations of the objective in addition to the evaluation at $\mathbf{u}$ itself (13 in the six-parameter case), all of which are independent and are run in parallel. We use $h = 10^{-4}$, which lies well inside the range of steps over which the estimates agree (Section 3.4): it is large enough that rounding errors in the difference of two nearly equal losses are negligible, and small enough that the $O(h^2)$ truncation error is not visible.

Scaling of the First Step. On its first iteration, L-BFGS-B has no curvature information and uses the identity as its approximation of the Hessian, so its first trial step is the negative gradient itself. For our objective, the gradient at the initial point has a norm of order 10^2–10^3 in the log variables, so an unscaled first step would change the parameters by enormous factors and drive every variable to its bound. We therefore multiply the objective by the constant $\kappa = \rho_0 / \|\hat{\mathbf{g}}(\mathbf{u}_0)\|$, which does not change its minimizers, so that the first trial step has length ρ_0 in the variables $\mathbf{u}$. We use $\rho_0 = 0.05$, i.e. a first step which changes the parameters by about 5%. Later iterations use the curvature information accumulated by L-BFGS-B.

Failed Simulations. Occasionally a trial point causes the forward simulation to fail, for example when an interface degenerates. At such points we replace the objective by the best value found so far plus the penalty $10^3(1 + \|\mathbf{u} - \mathbf{u}_{\text{best}}\|^2)$. Because the penalty grows with the distance from the best point, the finite-difference gradient points back toward feasible parameters and the line search backtracks, rather than seeing a flat region that it would mistake for a minimum.

Combining the Two Methods. Nelder–Mead and L-BFGS-B have complementary strengths. Because the Nelder–Mead simplex takes large exploratory steps, it is more flexible and less likely to get stuck. L-BFGS-B descends to the bottom of a basin with far fewer evaluations of the objective, even though each gradient costs $2d+1$ evaluations when there are d parameters. Additionally, these evaluations can be run in parallel, so L-BFGS-B can also be much faster in wall-clock time. We therefore mainly rely upon Nelder–Mead to find a basin, and use L-BFGS-B to converge quickly to its minimum. For example, in the last round of optimization for Dataset 1 with Gaussian kernels in a shared growth rate model, Nelder–Mead took about 2298 function evaluations to converge, with 39 evaluations from L-BFGS-B used to descend to a true local minimum. Starting from the same initial point as Nelder–Mead, L-BFGS-B took 767 evaluations to descend to the same minimum. These loss curves are shown in Figure 3.

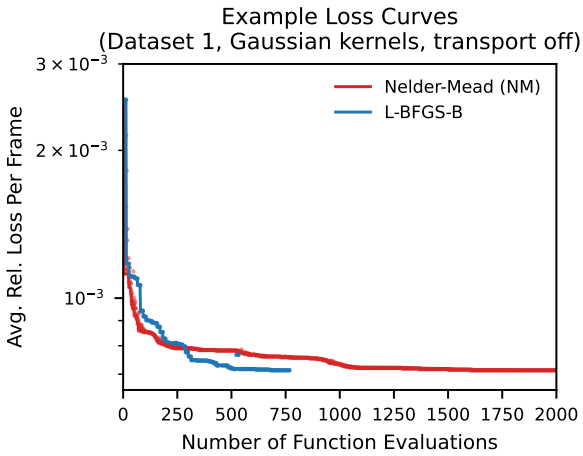


Figure 3: This figure shows the loss at each function evaluation in the Nelder–Mead (NM) algorithm and the L-BFGS-B algorithm with finite differences during training on Dataset 1 with Gaussian kernels, a single growth rate c shared by both colonies, and transport turned off. This run illustrates the general behavior of the two methods; the fits reported in Section 4 use per-colony growth rates. The x-axis represents the number of function evaluations; small dots show the actual function evaluations and the solid lines show the cumulatively smallest function evaluation. For this dataset, both NM and L-BFGS-B converge to the same minimum, but L-BFGS-B reaches it more quickly.

4 Results

In this section we give quantitative results for fitting models to target contours extracted from experimental data. We fit models to three datasets: Dataset 1 (two colonies, similar size), Dataset 2 (two colonies, different size), and Dataset 3 (three colonies), using both the Ramp and the Gaussian classes of interaction kernels. Dataset 1, where the two colonies are approximately the same size, is the least challenging dataset, while Dataset 3, with three colonies of different sizes, is the most challenging. In every case the objective is the relative loss (Algorithm S2), computed with a fixed number of substeps per image chosen as in Section 3.4. Section 4.1 reports the fits of the standard model, Section 4.2 reports results for models with shared growth rate and/or shared interaction kernels, and Section 4.3 reports results for models including the nonlocal transport mechanism.

Throughout this section, colonies are numbered as follows: in Datasets 1 and 2, colony 1 is the left colony and colony 2 the right colony; in Dataset 3, colony 1 is the left colony, colony 2 the upper colony, and colony 3 the lower-right colony, which is also the largest.

4.1 Standard Model

For each configuration we fit the $m+5$ parameters of the standard model (2.5), following the strategy of Section 3.6 using both Nelder–Mead and L-BFGS-B. The best results for each configuration along with the associated parameters are displayed in Table 1. Notably, the best fits have a large range in τ, and the values of τ in different datasets do not overlap: $\tau = 0.88$–0.89 (Dataset 1), 0.19–0.40 (Dataset 2), 0.027–0.029 (Dataset 3).

Next we visually compare predictions made by numerically simulating contours with the target contours extracted from experimental data. The simulation begins from the initial extracted

	Dataset 1		Dataset 2		Dataset 3	
	Gaussian	Ramp	Gaussian	Ramp	Gaussian	Ramp
c	$c_1 = 1.93 \times 10^{0}$ $c_2 = 1.87 \times 10^{0}$	$c_1 = 1.84 \times 10^{0}$ $c_2 = 1.79 \times 10^{0}$	$c_1 = 2.80 \times 10^{-1}$ $c_2 = 3.02 \times 10^{-1}$	$c_1 = 5.97 \times 10^{-1}$ $c_2 = 6.44 \times 10^{-1}$	$c_1 = 6.62 \times 10^{-2}$ $c_2 = 6.43 \times 10^{-2}$ $c_3 = 6.96 \times 10^{-2}$	$c_1 = 6.22 \times 10^{-2}$ $c_2 = 6.05 \times 10^{-2}$ $c_3 = 6.53 \times 10^{-2}$
τ	8.94×10^{-1}	8.77×10^{-1}	1.87×10^{-1}	4.03×10^{-1}	2.86×10^{-2}	2.66×10^{-2}
Self-interaction	$c_f = 6.01 \times 10^{-1}$ $\sigma_f = 1.87 \times 10^{0}$	$a_f = 1.21 \times 10^{-1}$ $b_f = 4.03 \times 10^{-2}$	$c_f = 7.61 \times 10^{-3}$ $\sigma_f = 5.42 \times 10^{-2}$	$a_f = 5.84 \times 10^{-2}$ $b_f = 2.44 \times 10^{-2}$	$c_f = 4.90 \times 10^{-1}$ $\sigma_f = 1.13 \times 10^{0}$	$a_f = 1.72 \times 10^{-1}$ $b_f = 1.04 \times 10^{-1}$
Cross-interaction	$c_g = 1.16 \times 10^{0}$ $\sigma_g = 3.31 \times 10^{1}$	$a_g = 1.37 \times 10^{-2}$ $b_g = 2.39 \times 10^{-4}$	$c_g = 9.78 \times 10^{-1}$ $\sigma_g = 3.44 \times 10^{1}$	$a_g = 1.04 \times 10^{-2}$ $b_g = 1.67 \times 10^{-4}$	$c_g = 8.01 \times 10^{-1}$ $\sigma_g = 4.35 \times 10^{1}$	$a_g = 6.72 \times 10^{-3}$ $b_g = 8.69 \times 10^{-5}$
Avg. rel. loss	6.07×10^{-4}	$\mathbf{6.05 \times 10^{-4}}$	8.19×10^{-4}	$\mathbf{7.91 \times 10^{-4}}$	2.23×10^{-3}	$\mathbf{2.20 \times 10^{-3}}$

Table 1: Optimized parameters and average relative losses of the standard model (per-colony growth rates, $f \neq g$, no transport) for each dataset and kernel class, irrespective of basin. Losses are the average over images of the relative loss defined in Section 3.3. The lowest loss for each dataset is indicated in bold.

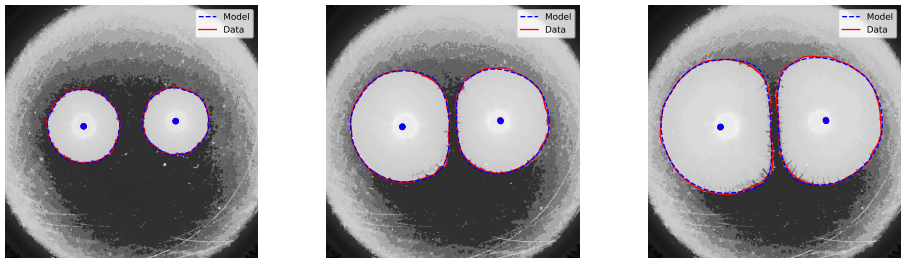
contours, and parameters are chosen based on the best fits given by Table 1. For each dataset and kernel class, a sequence of three images shows the evolution of the model predictions and the target data over time. Each image shows the raw grayscale image of the bacterial colonies growing in a Petri dish, overlaid with a solid red line and a dotted blue line. The solid red line depicts the contour extracted from the image using the segmentation procedure described in Section 6.2. The dotted blue line represents the simulated contour at the relevant time step. The centroids of the predicted and target contours are shown as blue and red dots, respectively. Figure 4 corresponds to Dataset 1 (two colonies, similar size), Figure 5 to Dataset 2 (two colonies, different size), and Figure 6 to Dataset 3 (three colonies). Overall, the Gaussian and Ramp kernels perform well and capture the dynamics of colony growth accurately, but they perform best on Dataset 1, where the two colonies are approximately the same size. Dataset 3, with three colonies of different sizes, is the most challenging dataset and has the worst performance.

Next we illustrate the relative error at each image of the best fits of the standard model for Datasets 1, 2, and 3 (Figure 7). Because the extracted contours are relatively noisy, each dataset contains frames where all best-fit models struggle. The figure shows that the performance of the Gaussian and Ramp kernels is very similar across datasets, with the Ramp kernels performing slightly better in all three datasets.

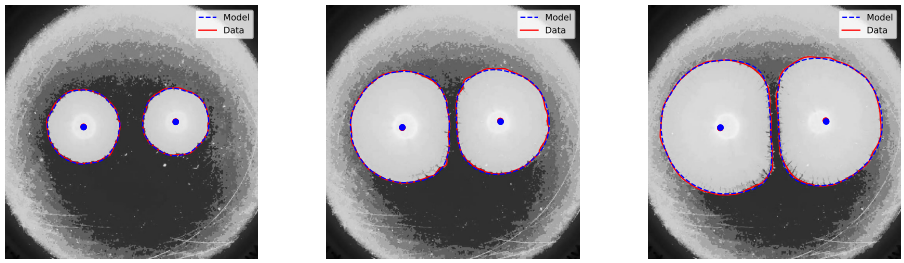
Lastly, we display the fitted interaction kernels in the best models for each configuration in Figure 8. The kernels are shown in physical units (in contrast to Table 1) as functions of the distance d in pixels between two boundary points; the logarithmic horizontal axis makes both short- and long-range kernels visible but distorts their shapes. Across datasets, the self-interaction kernels have a higher peak but have a much shorter range, reducing their effective strength. Cross-interaction kernels reach over a much larger range, meaning that inhibition between colonies is active from very early times, while self-interaction is much more local and weak. Notably, the best-fit Gaussian self-interaction kernel for Dataset 2 is barely distinguishable from zero.

4.2 Comparison to Models from Prior Work

This subsection compares the standard model with per-colony growth rates and distinct self- and cross-interaction kernels with the model class of [40] (shared constant growth rate and shared interaction kernels), as well as with the two intermediate models (per-colony growth rates with shared interaction kernels, and shared growth rates with distinct self- and cross-interaction kernels). Optimization is done following the strategy of Section 3.6 using both Nelder–Mead and L-BFGS-B. The best parameters and losses are shown in Table 2. Forcing interaction kernels to be shared

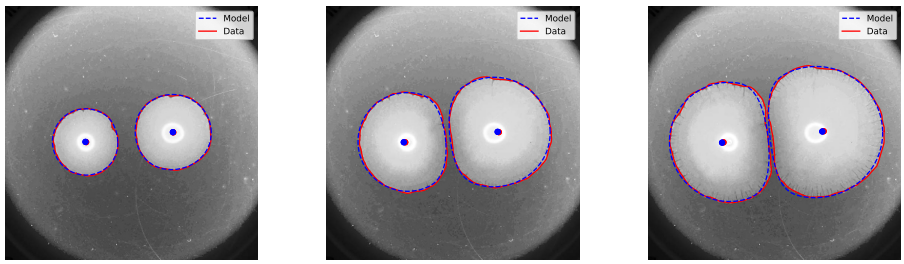


(a) Ramp kernel, step 30 (b) Ramp kernel, step 70 (c) Ramp kernel, step 101

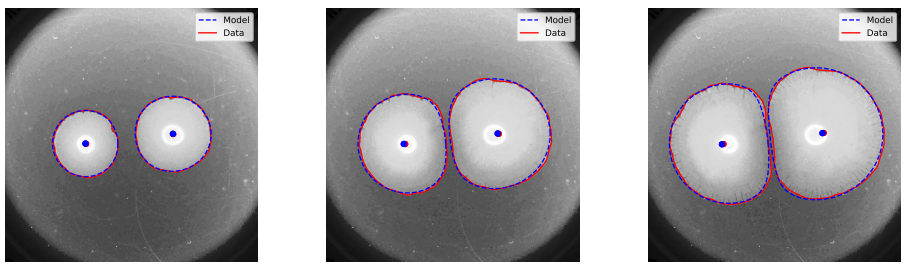


(d) Gaussian kernel, step 30 (e) Gaussian kernel, step 70 (f) Gaussian kernel, step 101

Figure 4: Dataset 1 (two colonies, similar size), standard model. Simulated contours of the best fits of the standard model compared with the target contours. Colony 1 is the left colony and colony 2 the right colony. Here Gaussian and Ramp kernels both perform very well, capturing the shape and positions of the colonies accurately.

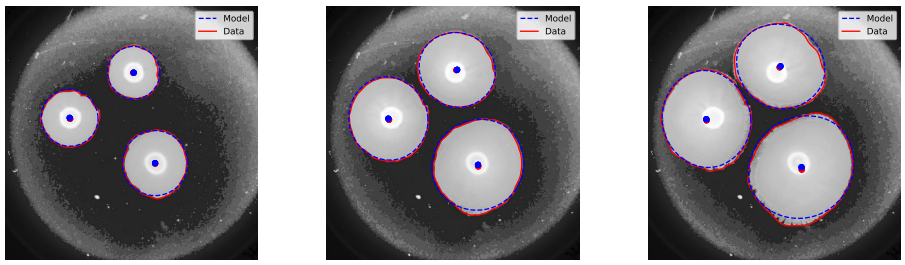


(a) Ramp kernel, step 50 (b) Ramp kernel, step 90 (c) Ramp kernel, step 118

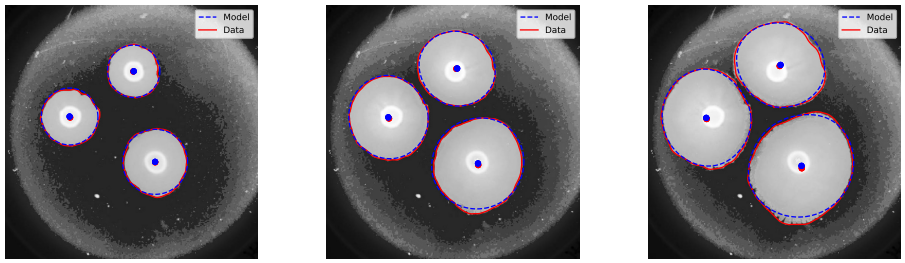


(d) Gaussian kernel, step 50 (e) Gaussian kernel, step 90 (f) Gaussian kernel, step 118

Figure 5: Dataset 2 (two colonies, different size), standard model. Simulated contours of the best fits of the standard model compared with the target contours. Colony 1 is the left colony and colony 2 the right colony. Here Gaussian and Ramp kernels both perform very well, capturing the shape and positions of the colonies accurately.



(a) Ramp kernel, step 50 (b) Ramp kernel, step 90 (c) Ramp kernel, step 118



(d) Gaussian kernel, step 50 (e) Gaussian kernel, step 90 (f) Gaussian kernel, step 118

Figure 6: Dataset 3 (three colonies), standard model. Simulated contours of the best fits of the standard model compared with the target contours. Colony 1 is the left colony, colony 2 the upper colony, and colony 3 the lower-right (largest) colony. Here, the model captures the stationary fronts and predicts the sizes of the colonies well. It struggles more in capturing the shapes of the colonies, especially colony 3.

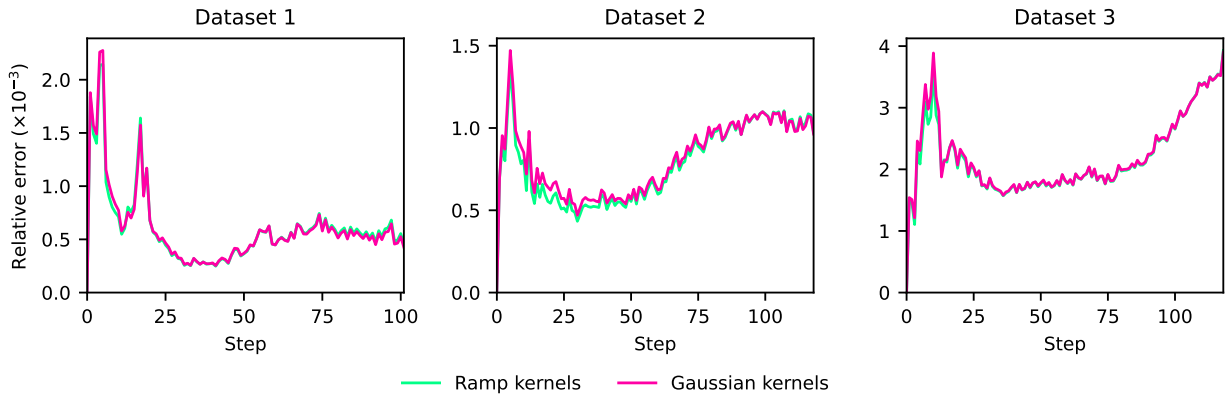


Figure 7: Relative error at each image of the best fits of the standard model for Dataset 1 (two colonies, similar size), Dataset 2 (two colonies, different size), and Dataset 3 (three colonies), for each kernel class.

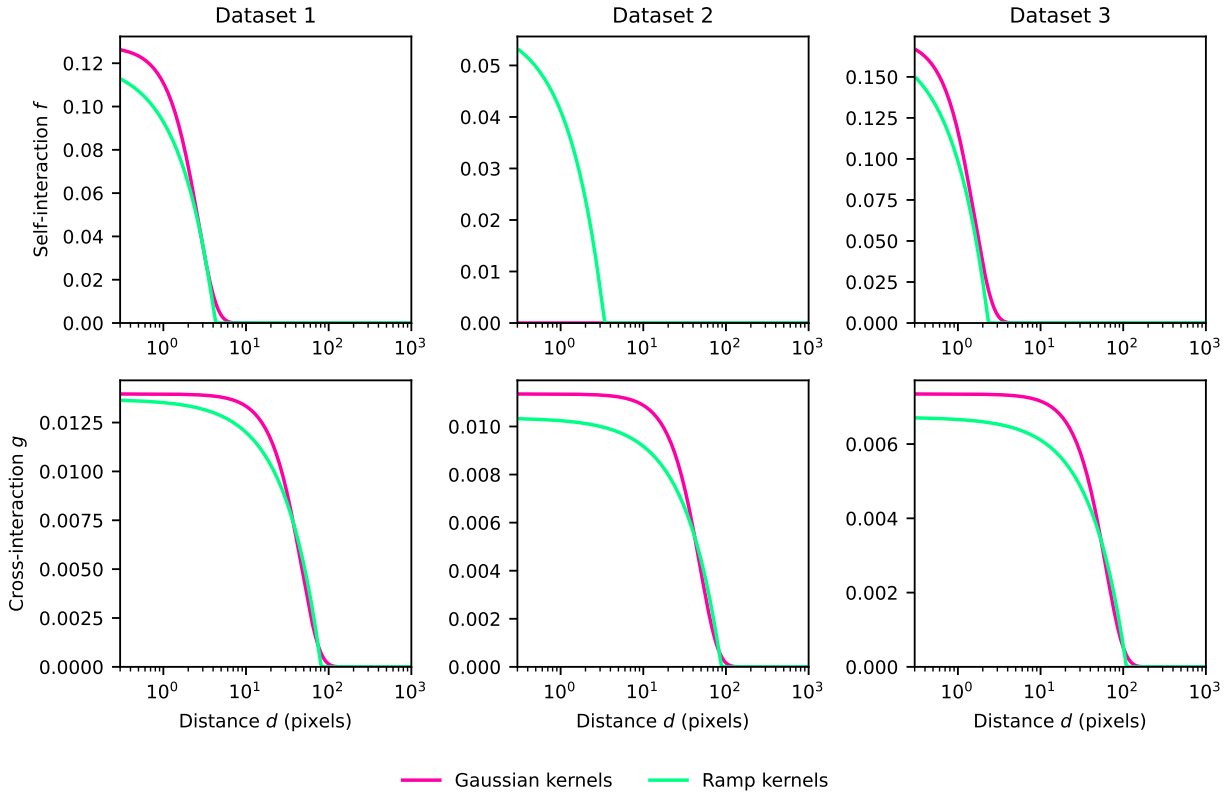


Figure 8: Fitted interaction kernels of the standard model for each dataset (columns): self-interaction kernels f (top row) and cross-interaction kernels g (bottom row), for the best fit of each kernel class. Note that the logarithmic horizontal axis distorts the shapes of Gaussian and Ramp kernels. In every fit, the self-interaction kernel has a higher peak but a much shorter range than the cross-interaction kernel. For Dataset 2 the Gaussian self-interaction kernel ($\sigma_f \approx 0.05$ pixels) is too narrow to be visible on this axis.

		per-colony c, $f \neq g$		shared c, $f \neq g$		per-colony c, $f = g$		shared c, $f = g$	
		Avg. rel. loss	τ	Avg. rel. loss	τ	Avg. rel. loss	τ	Avg. rel. loss	τ
Dataset 1	Gaussian	$\mathbf{6.07 \times 10^{-4}}$	8.94×10^{-1}	7.14×10^{-4} (+18%)	7.12×10^{-1}	2.44×10^{-3} (+302%)	1.35×10^{0}	2.57×10^{-3} (+323%)	6.39×10^{-1}
	Ramp	$\mathbf{6.05 \times 10^{-4}}$	8.77×10^{-1}	7.00×10^{-4} (+16%)	7.05×10^{-1}	2.40×10^{-3} (+296%)	1.33×10^{0}	2.52×10^{-3} (+317%)	6.33×10^{-1}
Dataset 2	Gaussian	$\mathbf{8.19 \times 10^{-4}}$	1.87×10^{-1}	1.50×10^{-3} (+83%)	3.83×10^{-2}	1.71×10^{-3} (+109%)	7.60×10^{-1}	2.36×10^{-3} (+188%)	7.76×10^{-1}
	Ramp	$\mathbf{7.91 \times 10^{-4}}$	4.03×10^{-1}	1.46×10^{-3} (+84%)	3.66×10^{-2}	1.80×10^{-3} (+128%)	9.67×10^{-1}	2.30×10^{-3} (+191%)	7.95×10^{-1}
Dataset 3	Gaussian	$\mathbf{2.23 \times 10^{-3}}$	2.86×10^{-2}	3.07×10^{-3} (+37%)	1.01×10^{-2}	3.17×10^{-3} (+42%)	5.45×10^{0}	3.20×10^{-3} (+43%)	1.76×10^{-2}
	Ramp	$\mathbf{2.20 \times 10^{-3}}$	2.66×10^{-2}	3.07×10^{-3} (+39%)	5.81×10^{-3}	3.13×10^{-3} (+42%)	2.90×10^{0}	5.28×10^{-3} (+140%)	8.11×10^{-1}

Table 2: Average relative losses and values of τ of the best fits of four model classes, without transport, for each dataset and kernel class: per-colony or shared growth rates, with separate ($f \neq g$) or identical ($f = g$) self- and cross-interaction kernels. The best loss in each configuration (row) is shown in bold, and percentages are changes relative to the first column. The last column, with Gaussian kernels, is the model class of [40].

($f = g$) raises the loss more than sharing growth rates, but both raise it substantially.

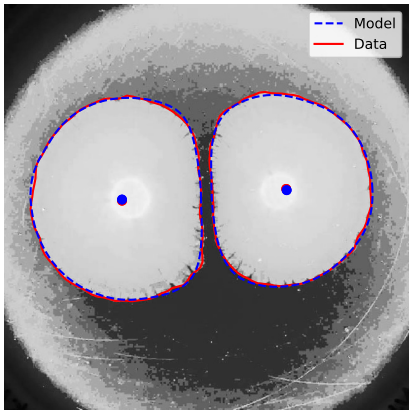
Next we visually compare predictions made by numerically simulating contours with the target contours extracted from experimental data, as in the previous subsection. Parameters are chosen based on the best fits given by Table 2. For each dataset, we compare the four model configurations in Table 2 (taking the best of either Ramp or Gaussian), displaying a single still image for each configuration. Figure 9 corresponds to Dataset 1 (two colonies, similar size), Figure 10 to Dataset 2 (two colonies, different size), and Figure 11 to Dataset 3 (three colonies). For each dataset, an image near the end of the sequence is chosen.

For Dataset 1, sharing the growth rate does not significantly impact the model fit since the two colonies are very close to the same size. However, sharing interaction kernels causes the best-fitting model to either overpredict the sizes of the colonies or introduce crossing contours, increasing the loss by 296–323%. For Dataset 2, sharing the growth rate leads to inaccurate predictions of colony size, increasing the loss by over 80%, and sharing interaction kernels distorts shapes even further, resulting in losses 109–191% larger than the standard model. For Dataset 3, sharing the growth rate leads to inaccurate predictions of colony size, and sharing interaction kernels worsens the fit, but the effect is smaller than in Datasets 1 and 2, only increasing the loss by around 40%.

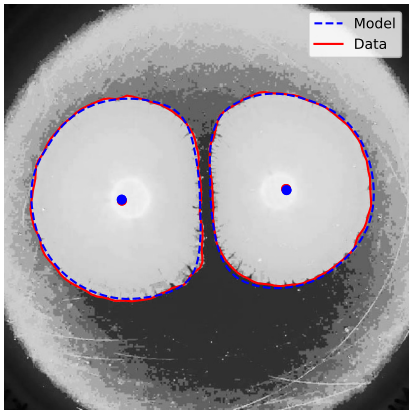
4.3 Nonlocal Transport Mechanism

This subsection quantitatively analyzes the performance of model configurations with and without the nonlocal transport mechanism introduced in Section 2.6. Optimization is done following the strategy of Section 3.6 using both Nelder–Mead and L-BFGS-B. In the Nelder–Mead runs, the initial simplex vertex for α was placed at $-\alpha_0$ rather than $1.05\,\alpha_0$, so that both repulsive and attractive transport were explored from the start. All runs with transport used the smoothstep gate with width $w = 10\sqrt{2}$ in pixel units.

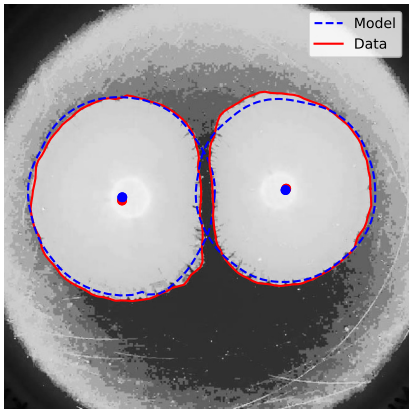
The best parameters and losses are shown in Table 3. The addition of the nonlocal transport mechanism does not significantly improve the standard model, indicating that the colony-scale transport induced by the nonlocal interaction force is already sufficiently captured by the standard model. However, in models with shared growth rates, the addition of the nonlocal transport mechanism can compensate somewhat for the missing degrees of freedom. (All models shown in this section use distinct self- and cross-interaction kernels, i.e. $f \neq g$). Notably, in the shared growth rate configurations where the nonlocal transport mechanism helps most, the β parameter is



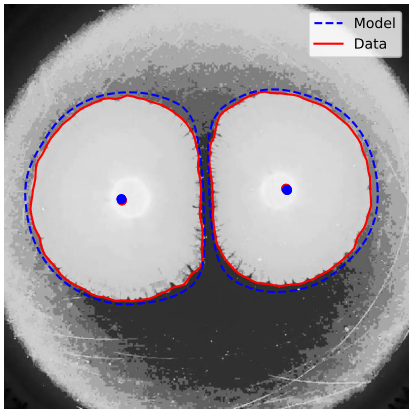
(a) Per-colony c, $f \neq g$ (Ramp kernels)



(b) Shared c, $f \neq g$ (Ramp kernels)

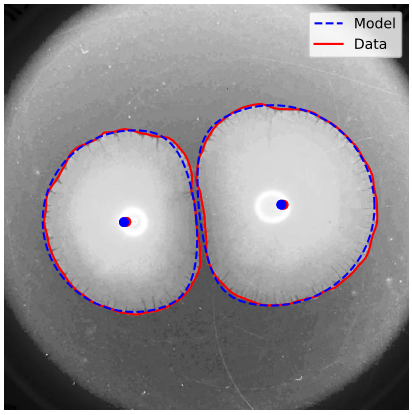


(c) Per-colony c, $f = g$ (Ramp kernels)

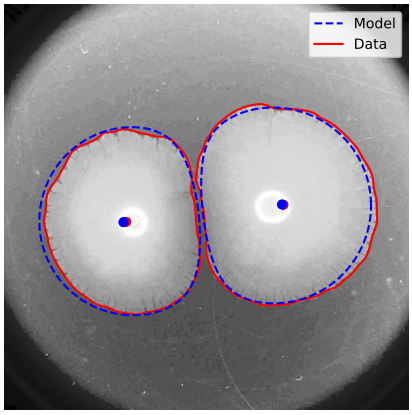


(d) Shared c, $f = g$ (Ramp kernels)

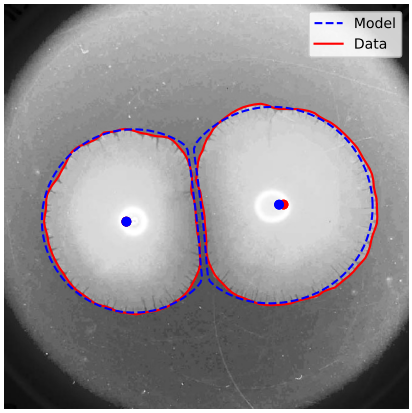
Figure 9: Dataset 1 (two colonies, similar size), step 90: simulated contours of the best fit of each model class without transport, compared with the target contours. In each case the better of the Gaussian and Ramp kernels is shown. In the model with $f = g$ and shared c, the growth rate of the colonies is overpredicted and the separation between colonies is underpredicted. When $f = g$ but c is allowed to vary, the best fit has contours that cross, so it does not reproduce the stationary fronts phenomenon.



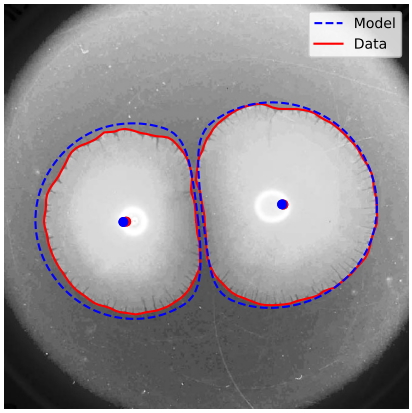
(a) Per-colony c, $f \neq g$ (Ramp kernels)



(b) Shared c, $f \neq g$ (Ramp kernels)

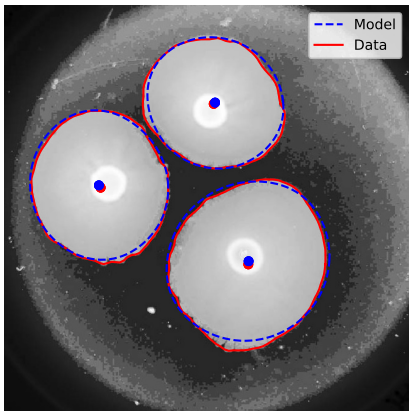


(c) Per-colony c, $f = g$ (Gaussian kernels)

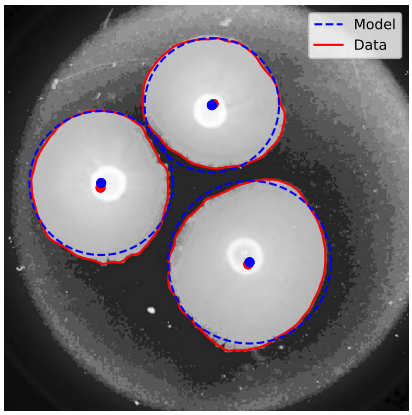


(d) Shared c, $f = g$ (Ramp kernels)

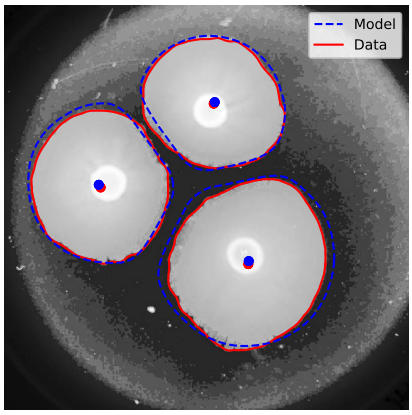
Figure 10: Dataset 2 (two colonies, different size), step 110: simulated contours of the best fit of each model class without transport, compared with the target contours. In each case the better of the Gaussian and Ramp kernels is shown. Sharing a growth rate (right column) causes the size of Colony 1 (left) to be overpredicted and the size of Colony 2 (right) to be underpredicted, and forcing $f = g$ worsens the fit. Notably, the lower left configuration ($f = g$ with varying c) shows corners in the colonies near the edge of the stationary front.



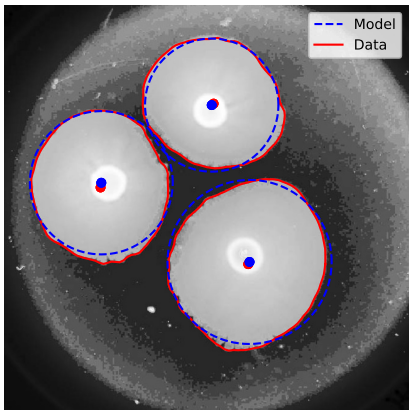
(a) Per-colony c, $f \neq g$ (Ramp kernels)



(b) Shared c, $f \neq g$ (Gaussian kernels)



(c) Per-colony c, $f = g$ (Ramp kernels)



(d) Shared c, $f = g$ (Gaussian kernels)

Figure 11: Dataset 3 (three colonies), step 110: simulated contours of the best fit of each model class without transport, compared with the target contours. In each case the better of the Gaussian and Ramp kernels is shown. Sharing a growth rate (right column) causes the sizes of the colonies to be off and causes the separation between colonies to be underpredicted, and forcing $f = g$ further worsens the fit. Notably, the lower left configuration ($f = g$ with varying c) shows the model overpredicting the separation between Colonies 1 and 2 (left side) and producing a straighter front than the data.

		per-colony c, transport off		per-colony c, transport on		shared c, transport off		shared c, transport on	
		Avg. rel. loss	τ	Avg. rel. loss	τ	Avg. rel. loss	τ	Avg. rel. loss	τ
Dataset 1	Gaussian	6.07×10^{-4}	8.94×10^{-1}	6.03×10^{-4} (-0.6%)	8.70×10^{-1}	7.14×10^{-4} ($+18\%$)	7.12×10^{-1}	7.04×10^{-4} ($+16\%$)	6.94×10^{-1}
	Ramp	6.05×10^{-4}	8.77×10^{-1}	6.05×10^{-4} (-0.1%)	8.69×10^{-1}	7.00×10^{-4} ($+16\%$)	7.05×10^{-1}	6.96×10^{-4} ($+15\%$)	6.92×10^{-1}
Dataset 2	Gaussian	8.19×10^{-4}	1.87×10^{-1}	7.96×10^{-4} (-2.8%)	2.27×10^{-1}	1.50×10^{-3} ($+83\%$)	3.83×10^{-2}	1.03×10^{-3} ($+25\%$)	7.35×10^{-1}
	Ramp	7.91×10^{-4}	4.03×10^{-1}	7.74×10^{-4} (-2.1%)	3.79×10^{-1}	1.46×10^{-3} ($+84\%$)	3.66×10^{-2}	8.97×10^{-4} ($+13\%$)	7.40×10^{-1}
Dataset 3	Gaussian	2.23×10^{-3}	2.86×10^{-2}	2.21×10^{-3} (-1.0%)	2.71×10^{-2}	3.07×10^{-3} ($+37\%$)	1.01×10^{-2}	2.66×10^{-3} ($+19\%$)	1.01×10^{-2}
	Ramp	2.20×10^{-3}	2.66×10^{-2}	2.18×10^{-3} (-0.9%)	2.59×10^{-2}	3.07×10^{-3} ($+39\%$)	5.81×10^{-3}	2.36×10^{-3} ($+7.0\%$)	9.81×10^{-3}

Table 3: Average relative losses and values of τ of the best fits with and without the nonlocal transport mechanism, for per-colony and shared growth rates, for each dataset and kernel class (note that $f \neq g$ throughout). Percentages are changes relative to the first column (standard model).

large enough that the transport gate is open for most of the frames, so it acts more like a constant attractive force than one that switches on when colonies get close enough.

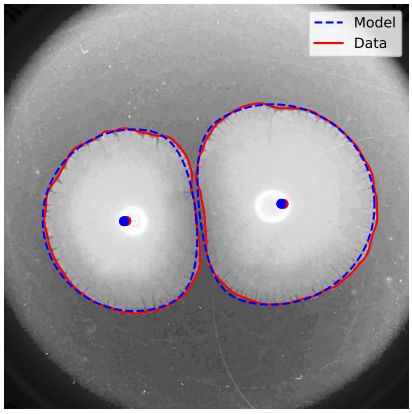
Almost all of the best transport fits for each dataset, in both shared growth rate and varying growth rate model configurations, have a negative α value, indicating that the nonlocal transport mechanism causes attraction rather than repulsion (with the only exception being the shared growth rate, transport-on, Dataset 3 Gaussian case). However, the magnitude of this effect is small for the standard (varying growth rate) model.

Next we visually compare predictions made by numerically simulating contours with the target contours extracted from experimental data, as in the previous subsection. Parameters are chosen based on the best fits given by Table 3. We compare the four model configurations in Table 3 (taking the best of either Ramp or Gaussian), displaying a single still image for each configuration. Because the fits are broadly similar and to save space, we just display Dataset 2 (Figure 12), which is emblematic of the other two datasets. In the shared growth rate configuration, transport lowers the error at intermediate times as well as the total loss, but the attraction pushes the two fronts past each other in the final images, so the stationary front is not reproduced. In the varying growth rate configuration, adding transport only improves the loss by 2%.

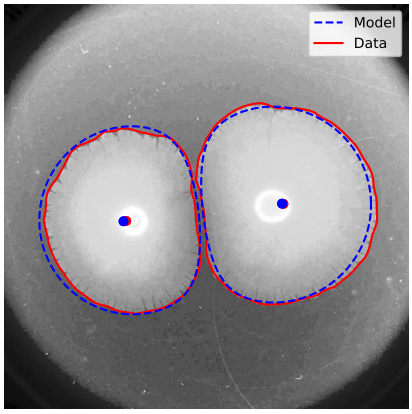
Next, we show the minimum intercolony distance and centroid errors (as before, for Dataset 2 only). The minimum intercolony distance is shown in Figure 13 and the centroid errors are shown in Figure 14. On Dataset 2 in the shared growth rate configuration, using the nonlocal transport mechanism allows the loss to decrease, the colony sizes to more closely match, and the centroid error to decrease, but does not as faithfully predict the minimum intercolony distance, which touches zero as the fronts cross.

5 Discussion and Future Work

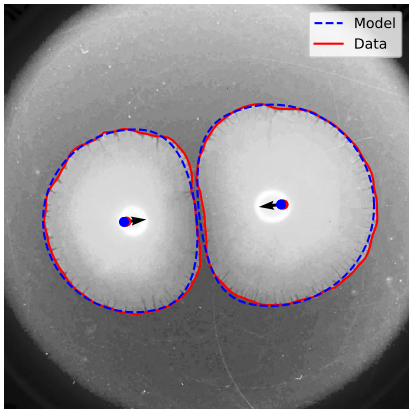
This work uses a PDE-constrained optimization procedure to learn interaction kernels for a nonlocal bacterial growth model from real experimental image sequences. Taken as a whole, this demonstrates an effective scientific machine learning framework for systems with a priori unknown governing dynamics and limited data. We consider two different forms of parametrized interaction kernels (Gaussian kernels, used in earlier work [38, 40], and Ramp kernels, introduced here), and demonstrate a small but notable difference in performance between these two classes, with Ramp kernels performing better (by 0.25–3.4% on two-colony image sequences and 1.4% on a three-colony



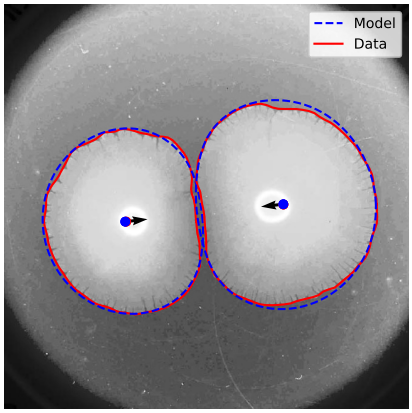
(a) Per-colony c, transport off (Ramp kernels)



(b) Shared c, transport off (Ramp kernels)



(c) Per-colony c, transport on (Ramp kernels)



(d) Shared c, transport on (Ramp kernels)

Figure 12: Dataset 2 (two colonies, different size), step 110: simulated contours of the best fits with and without the nonlocal transport mechanism, for per-colony and shared growth rates, compared with the target contours. In each case the better of the Gaussian and Ramp kernels is shown. In the shared c configuration, turning on transport somewhat compensates for the lack of per-colony growth rates, improving the loss at intermediate times. However, it pushes the two fronts past each other in the final frames (thus not reproducing the "stationary front" phenomenon).

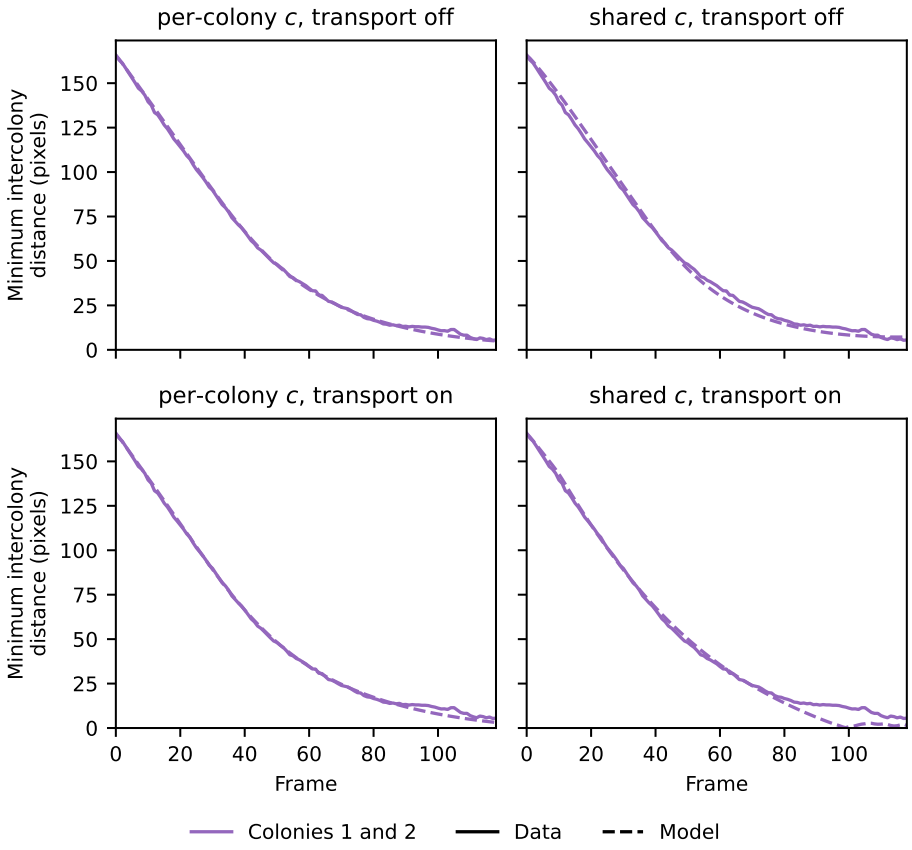


Figure 13: Minimum intercolony distance in pixels for Dataset 2 (two colonies, different size), for the best fits with and without the nonlocal transport mechanism, for per-colony and shared growth rates, compared to data. In each case, the single model with the best overall fit is chosen, whether Gaussian or Ramp. While adding transport improves the loss in the shared-c configuration by 38% (see Table 3), it results in the intercolony distance collapsing to zero (see lower right subfigure).

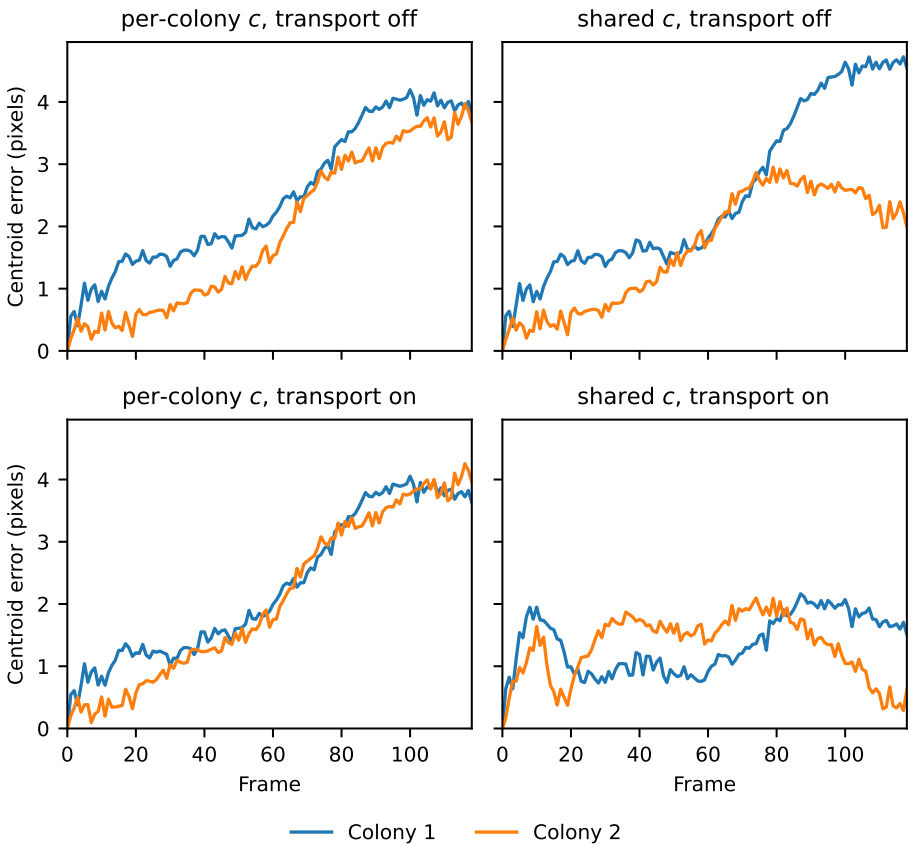


Figure 14: Centroid error (compared to data) in pixels for Dataset 2 (two colonies, different size), for the best fits with and without the nonlocal transport mechanism, for per-colony and shared growth rates. In each case, the single model with the best overall fit is chosen, whether Gaussian or Ramp. In the shared-c configuration, adding transport allows the centroid error to be kept under 2.2 pixels, a large improvement. However, adding transport to the varying-c configuration does not result in a large improvement.

image sequence).

The fitted interaction kernels differ considerably between datasets and between types (self- vs cross-interaction). In every fit, the cross-interaction kernel has a much longer range than the self-interaction kernel, and attempting to fit shared interaction kernels results in 42–323% higher losses, supporting the modeling choice to parametrize the kernels separately.

Another contribution of this work is the demonstration of the robustness of the model of [38] with respect to colony-scale drift. To test this, we compare the standard model with an extension including a nonlocal transport mechanism. Even though it includes two new parameters, the transport mechanism only reduces the loss of the best fits by 0.1–2.8%. On the other hand, in a model where growth rates are shared, transport reduces the loss of the best fits by 13–38% on Datasets 2 and 3, suggesting that the nonlocal transport mechanism is compensating for a missing degree of freedom in the model in this case, providing support for using distinct per-colony growth rates.

On the methodological side, we find that the objective becomes differentiable in the model parameters, at least on the scale relevant to optimization, after two small changes to the numerical scheme: a fixed number of substeps per image and a smooth transport gate. This allows L-BFGS-B with finite-difference gradients to complement Nelder–Mead. The main bottlenecks of the approach are the time required to evaluate the objective with the PDE solver (up to several minutes per evaluation when τ is small) and the difficult loss landscape.

There are several directions for future work. For one, all of the results reported here are fits to the training data. Further analyses could test the predictive ability of the model by held-out prediction (training on early images and predicting later ones) or by generalization between datasets. Because the fitted kernels differ significantly across datasets, fitting all image sequences jointly would test how well a single set of kernels describes all of the experiments. Learning the interaction kernels nonparametrically, rather than within the Ramp and Gaussian classes, is another natural extension, although the sparsity of the data makes this challenging.

Funding

We acknowledge support from the National Science Foundation (NSF) under grants 1813654, 2112085 and 2331100, and from the Army Research Office (ARO) under grant W911NF-19-1-0288.

Acknowledgements

We would like to acknowledge participants in the 2023 UCLA CAM REU (Yuxuan Liu, Kyle Alkire, Yuntong Ju, Kevin Phan, Ryan Schoenburg, Yue Yang, and Liangjiawa Yuan) for their help in generating and testing some of the early ideas and software in this direction, including the image processing software.

References

[1]J. W. Costerton, P. S. Stewart, and E. P. Greenberg, “Bacterial biofilms: a common cause of persistent infections”, Science **284**, 1318–1322 (1999).

[2]L. Hall-Stoodley, J. W. Costerton, and P. Stoodley, “Bacterial biofilms: from the natural environment to infectious diseases”, Nature Reviews Microbiology **2**, 95–108 (2004).

[3]H.-C. Flemming, J. Wingender, U. Szewzyk, P. Steinberg, S. A. Rice, and S. Kjelleberg, "Biofilms: an emergent form of bacterial life", Nature Reviews Microbiology **14**, 563–575 (2016).

[4]J. N. Wilking, T. E. Angelini, A. Seminara, M. P. Brenner, and D. A. Weitz, "Biofilms as complex fluids", MRS Bulletin **36**, 385–391 (2011).

[5]P. S. Stewart and J. William Costerton, "Antibiotic resistance of bacteria in biofilms", The Lancet **358**, 135–138 (2001).

[6]H.-C. Flemming, P. S. Murthy, R. Venkatesan, and K. Cooksey, editors, *Marine and industrial biofouling*, Springer Series on Biofilms (Springer, Berlin, Heidelberg, 2009).

[7]M. R. Warren, H. Sun, Y. Yan, J. Cremer, B. Li, and T. Hwa, "Spatiotemporal establishment of dense bacterial colonies growing on hard agar", eLife **8**, e41093 (2019).

[8]E. Ben-Jacob, I. Cohen, and H. Levine, "Cooperative self-organization of microorganisms", Advances in Physics **49**, 395–554 (2000).

[9]I. Klapper, C. J. Rupp, R. Cargo, B. Purvedorj, and P. Stoodley, "Viscoelastic fluid description of bacterial biofilm material properties", Biotechnology and Bioengineering **80**, 289–296 (2002).

[10]I. Klapper and J. Dockery, "Finger formation in biofilm layers", SIAM Journal on Applied Mathematics **62**, 853–869 (2002).

[11]I. Klapper and J. Dockery, "Mathematical description of microbial biofilms", SIAM Review **52**, 221–265 (2010).

[12]C. S. Peskin, "The immersed boundary method", Acta Numerica **11**, 479–517 (2002).

[13]S. Osher and J. A. Sethian, "Fronts propagating with curvature-dependent speed: algorithms based on Hamilton-Jacobi formulations", Journal of Computational Physics **79**, 12–49 (1988).

[14]L.-Q. Chen, "Phase-field models for microstructure evolution", Annual Review of Materials Research **32**, 113–140 (2002).

[15]L. C. Evans, H. M. Soner, and P. E. Souganidis, "Phase transitions and generalized motion by mean curvature", Communications on Pure and Applied Mathematics **45**, 1097–1123 (1992).

[16]Y. Giga, *Surface evolution equations: a level set approach*, Vol. 99, Monographs in Mathematics (Birkhäuser, Basel, 2006).

[17]J. A. Sethian, *Level set methods and fast marching methods: evolving interfaces in computational geometry, fluid mechanics, computer vision, and materials science*, 2nd edition, Vol. 3, Cambridge Monographs on Applied and Computational Mathematics (Cambridge University Press, 1999).

[18]W. W. Mullins, "Two-dimensional motion of idealized grain boundaries", in *Fundamental contributions to the continuum theory of evolving phase interfaces in solids: a collection of reprints of 14 seminal papers*, edited by J. M. Ball, D. Kinderlehrer, P. Podio-Guidugli, and M. Slemrod (Springer, Berlin, Heidelberg, 1999), pages 70–74.

[19]W. W. Mullins and R. F. Sekerka, "Morphological stability of a particle growing by diffusion or heat flow", Journal of Applied Physics **34**, 323–329 (1963).

[20]S. M. Allen and J. W. Cahn, "A microscopic theory for antiphase boundary motion and its application to antiphase domain coarsening", Acta Metallurgica **27**, 1085–1095 (1979).

[21]J. W. Cahn and J. E. Hilliard, "Free energy of a nonuniform system. I. Interfacial free energy", The Journal of Chemical Physics **28**, 258–267 (1958).

[22]R. L. Pego, "Front migration in the nonlinear Cahn-Hilliard equation", Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences **422**, 261–278 (1989).

[23]S. Dai and K. Promislow, "Geometric evolution of bilayers under the functionalized Cahn–Hilliard equation", Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences **469**, 20120505 (2013).

[24]J. Rubinstein, P. Sternberg, and J. B. Keller, "Fast reaction, slow diffusion, and curve shortening", SIAM Journal on Applied Mathematics **49**, 116–133 (1989).

[25]P. van Heijster, A. Doelman, T. J. Kaper, and K. Promislow, "Front interactions in a three-component system", SIAM Journal on Applied Dynamical Systems **9**, 292–332 (2010).

[26]P. van Heijster and B. Sandstede, "Coexistence of stable spots and fronts in a three-component FitzHugh–Nagumo system", RIMS Kokyuroku Bessatsu B **31**, 135–155 (2012).

[27]A. L. Bertozzi and T. Laurent, "Finite-time blow-up of solutions of an aggregation equation in $\mathbb{R}^n$", Communications in Mathematical Physics **274**, 717–735 (2007).

[28]J. A. Carrillo, M. Fornasier, J. Rosado, and G. Toscani, "Asymptotic flocking dynamics for the kinetic Cucker–Smale model", SIAM Journal on Mathematical Analysis **42**, 218–236 (2010).

[29]T. Kolokolnikov, H. Sun, D. Uminsky, and A. L. Bertozzi, "Stability of ring patterns arising from two-dimensional particle interactions", Physical Review E **84**, 015203 (2011).

[30]F. Otto, T. Rump, and D. Slepčev, "Coarsening rates for a droplet model: rigorous upper bounds", SIAM Journal on Mathematical Analysis **38**, 503–529 (2006).

[31]X. Tian and Q. Du, "Analysis and comparison of different approximations to nonlocal diffusion and linear peridynamic equations", SIAM Journal on Numerical Analysis **51**, 3458–3482 (2013).

[32]Q. Du, *Nonlocal modeling, analysis, and computation* (Society for Industrial and Applied Mathematics, Philadelphia, PA, 2019).

[33]S. G. McCalla and J. H. von Brecht, "Consistent dynamics of stripes formed by cell-type interfaces", SIAM Journal on Applied Dynamical Systems **17**, 2615–2633 (2018).

[34]G. Smith, N. Stouffer, S. G. McCalla, and D. P. Zosso, "A partial differential equation model for Janus particle swarming", Royal Society Open Science **13**, 251936 (2026).

[35]L. Lyu, X. Yu, and H. Schaeffer, "MVNN: a measure-valued neural network for learning McKean-Vlasov dynamics from particle data", Journal of Computational Physics **569**, 115467 (2027).

[36]A. Be'er, H. Zhang, E.-L. Florin, S. M. Payne, E. Ben-Jacob, and H. L. Swinney, "Deadly competition between sibling bacterial colonies", Proceedings of the National Academy of Sciences **106**, 428–433 (2009).

[37]A. Be'er, G. Ariel, O. Kalisman, Y. Helman, A. Sirota-Madi, H. Zhang, E.-L. Florin, S. M. Payne, E. Ben-Jacob, and H. L. Swinney, "Lethal protein produced in response to competition between sibling bacterial colonies", Proceedings of the National Academy of Sciences **107**, 6258–6263 (2010).

[38]S. G. McCalla and J. H. von Brecht, "Fronts under arrest: nonlocal boundary dynamics in biology", Physical Review E **94**, 060401 (2016).

[39]J. H. von Brecht, S. G. McCalla, and E. H. Kim, *Fronts under arrest II: analytical foundations*, 2023, arXiv:2311.01633 [math.AP].

[40]C. Dudiak, "Analysis of dynamic biological systems imagery", Chapter 3, *Paenibacillus dendritiformis* boundary tracking and optimized modeling, pp. 13–31, Master's thesis (Montana State University, Apr. 2022).

[41]F. Lu, M. Zhong, S. Tang, and M. Maggioni, "Nonparametric inference of interaction laws in systems of agents from trajectory data", Proceedings of the National Academy of Sciences **116**, 14424–14433 (2019).

[42]F. Lu, M. Maggioni, and S. Tang, "Learning interaction kernels in heterogeneous systems of agents from multiple trajectories", Journal of Machine Learning Research **22**, 1–67 (2021).

[43]F. Lu, M. Maggioni, and S. Tang, "Learning interaction kernels in stochastic systems of interacting particles from multiple trajectories", Foundations of Computational Mathematics **22**, 1013–1067 (2022).

[44]Z. Li, F. Lu, M. Maggioni, S. Tang, and C. Zhang, "On the identifiability of interaction functions in systems of interacting particles", Stochastic Processes and their Applications **132**, 135–163 (2021).

[45]Y. Liu, S. G. McCalla, and H. Schaeffer, "Random feature models for learning interacting dynamical systems", Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences **479**, 20220835 (2023).

[46]M. Schmidt and H. Lipson, "Distilling free-form natural laws from experimental data", Science **324**, 81–85 (2009).

[47]S. L. Brunton, J. L. Proctor, and J. N. Kutz, "Discovering governing equations from data by sparse identification of nonlinear dynamical systems", Proceedings of the National Academy of Sciences **113**, 3932–3937 (2016).

[48]H. Schaeffer and S. G. McCalla, "Sparse model selection via integral terms", Physical Review E **96**, 023302 (2017).

[49]R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud, "Neural ordinary differential equations", in Advances in Neural Information Processing Systems, Vol. 31 (2018).

[50]D. A. Messenger and D. M. Bortz, "Weak SINDy for partial differential equations", Journal of Computational Physics **443**, 110525 (2021).

[51]L. Lu, P. Jin, G. Pang, Z. Zhang, and G. E. Karniadakis, "Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators", Nature Machine Intelligence **3**, 218–229 (2021).

[52]Z. Li, N. B. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar, "Fourier neural operator for parametric partial differential equations", in International Conference on Learning Representations (2021).

[53]Y. Liu, Z. Zhang, and H. Schaeffer, "PROSE: predicting multiple operators and symbolic expressions using multimodal transformers", Neural Networks **180**, 106707 (2024).

[54]L. Yang, S. Liu, T. Meng, and S. J. Osher, "In-context operator learning with data prompts for differential equation problems", Proceedings of the National Academy of Sciences **120**, e2310142120 (2023).

[55]Y. Liu, J. Sun, and H. Schaeffer, "BCAT: a block causal transformer for PDE foundation models for fluid dynamics", Journal of Artificial Intelligence Research **87**, 10.1613/jair.1.22089 (2026).

[56]F. Gao and L. Han, "Implementing the Nelder-Mead simplex algorithm with adaptive parameters", Computational Optimization and Applications **51**, 259–277 (2012).

[57]P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, et al., "SciPy 1.0: fundamental algorithms for scientific computing in Python", Nature Methods **17**, 261–272 (2020).

[58]R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu, "A limited memory algorithm for bound constrained optimization", SIAM Journal on Scientific Computing **16**, 1190–1208 (1995).

[59]C. Zhu, R. H. Byrd, P. Lu, and J. Nocedal, "Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization", ACM Transactions on Mathematical Software **23**, 550–560 (1997).

[60]J. Nocedal and S. J. Wright, *Numerical optimization*, 2nd edition (Springer, New York, 2006).

[61]G. Bradski, "The OpenCV library", Dr. Dobb's Journal of Software Tools (2000).

[62]H. Zhang and K. Wu, "A vehicle detection algorithm based on three-frame differencing and background subtraction", in 2012 Fifth International Symposium on Computational Intelligence and Design, Vol. 1 (2012), pages 148–151.

[63]R. Adams and L. Bischof, "Seeded region growing", IEEE Transactions on Pattern Analysis and Machine Intelligence **16**, 641–647 (1994).

[64]A. Mehnert and P. Jackway, "An improved seeded region growing algorithm", Pattern Recognition Letters **18**, 1065–1071 (1997).

[65]T. Holy, *ImageSegmentation.jl*, `https://github.com/JuliaImages/ImageSegmentation.jl`, 2023.

[66]G.-S. Jiang and C.-W. Shu, "Efficient implementation of weighted ENO schemes", Journal of Computational Physics **126**, 202–228 (1996).

6 Appendix

6.1 Experiments

This work uses data from *Paenibacillus dendritiformis* bacterial colony growth experiments in [40]. Experiments were conducted using sets of six Petri dishes of bacterial colonies placed into a single roundtable in order to ensure a consistent temperature of 29.5 °C and 30% humidity for each set. Rotation of the roundtable allowed a camera to take a picture of a different Petri dish every five minutes; thus for a single colony, images are taken at 30-minute intervals. To further improve the pictures taken, lighting conditions were adjusted so that only the plate was illuminated. Because the bacterial colonies often did not grow during an initial period of 24 hours, further processing of the images was done to discard images that did not show bacterial growth.

The experiments yielded seven image sequences which were amenable to segmentation, which were numbered 1–7. Image sequences 1–6 have 120 images each, while image sequence 7 has a sequence of 103 images. The size of each image in pixels is the same within each sequence but differs across different sequences, ranging from 1080 pixels to 1110 pixels in both height and width. The time between images in each sequence is long enough that the size of the bacterial colonies can be considered non-decreasing.

6.2 Image Processing

In order to model the behavior of the bacterial colonies, boundaries of the colonies were extracted using image segmentation and used as the target data in an optimization routine. The aim of the image segmentation was to capture accurate and smooth contours so that the data can be used to train the unknown parameters and kernels of the nonlocal PDE model. Smooth contours are preferred because the nonlocal PDE model does not account for fine-scale structures and intricate details on the colony boundaries.

However, difficulties arose from the similarity between the color of the colonies and the color of the background, making it challenging for contour extraction algorithms to distinguish between the two. In Figure 1a, the color of the colonies is too close to the color of the background, which can result in over-segmentation (where the extracted contours include regions which are not within the bacterial colony). In contrast, Figure 1b shows false edges where the color of the edges of the colonies is more similar to the color of the background than the color of the interior of the colonies. This can lead to under-segmentation (where the extracted contours do not include these edge regions which are part of the colony). A third challenge is the "feathering", or intricate fractal-like structures present along the contours of bacterial colonies (see Figure 1c). This can lead to a lack of smoothness in the extracted contours.

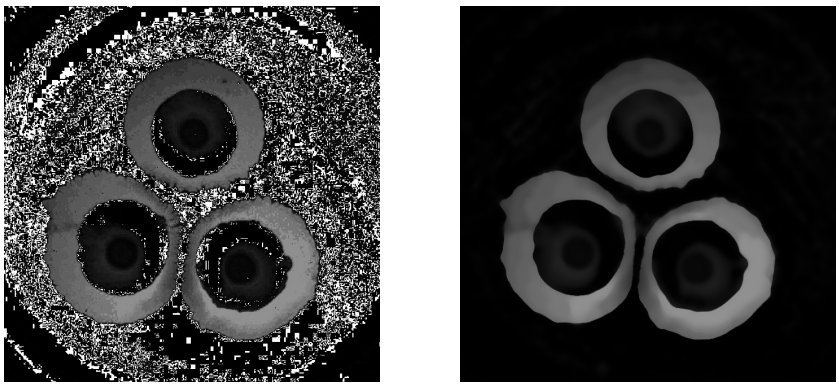
6.2.1 Pre-processing

Previous work used a simple thresholding method to perform the contour extraction. However, due to the difficulties stated above, the extracted contours displayed both over-segmentation and under-segmentation [40]. The approach taken in this work first uses various preprocessing strategies to handle these difficulties: cropping, bilateral and median blurring, and frame differencing. The preprocessing makes it easier for the image segmentation algorithm to correctly identify the contours of the bacterial colonies.

Cropping was done to eliminate unnecessary information and to speed up the rest of the pre-processing by reducing the amount of work that needs to be done. In particular, 40% of each side of each image is cropped to retain the center of the original image.

Next, bilateral and median blurring are applied in order to obtain smooth contours. Filtering was done by using `OpenCV`'s `bilateralFilter` and `medianBlur` in Python [61]. For bilateral blurring, a value of 15 is used for the pixel neighborhood, filter sigma in the color space, and filter sigma in the coordinate space. For median blurring, a value of 15 is used for the size of the kernel.

Lastly, frame differencing was done to isolate only the growth of the bacterial colonies. Frame differencing is a method used to detect the motion of objects [62]. Two images are converted into grayscale and the difference between the two images is taken pixel by pixel. Afterward, thresholding is applied to remove any unwanted artifacts. In Figure 15a, the image is produced by taking the difference between the 120^{th} image and the 60^{th} image. Thresholding is applied to get rid of any white pixels as seen in Figure 15b. In grayscale, any pixel with a value of 200 or less is converted into a black pixel. Any non-black pixels in Figure 15b represent the growth of the colony from the 60^{th} image in the sequence to the 120^{th} image in the sequence.



(a) Before thresholding (b) After thresholding

Figure 15: Frame differences, before and after thresholding

For each image sequence, the initial 10–15 images are preprocessed using only cropping and a bilateral filter. Since issues with under-segmentation, over-segmentation, and rough edges were much more prevalent with the later images than the earlier images, the rest of the images were preprocessed by a combination of cropping, bilateral and median blurring, and frame differencing.

6.2.2 Segmentation

Seeded region growing was chosen for segmenting the preprocessed image since it is fast and parameter-free [63, 64]. To segment an image using seeded region growing, seeds, or labels attached to specific pixels, must first be initialized. Based on a similarity criterion, these seeds iteratively expand into regions which form the segments of the image. In particular, for this application, the mean color of the region is used as the similarity criterion.

The number of desired segments in each image is controlled by the number of seed initializations. For instance, in Figure 15b, a minimum of four seeds would be placed: one seed for each of the bacterial colonies and one seed for the background. This ensures that the number of regions in the segmented image will be four. Furthermore, since the size of the bacterial colonies is non-decreasing, for every image except the first in a dataset, new seeds can be initialized from the previous segmented image in the sequence. This is done by using the pixels that are labeled as being part of the bacterial colonies as new seeds. This also ensures that the size of the bacterial colonies in the segmented image is non-decreasing.

However, if an image is not properly segmented, this often means that the segmentation will not be successful for the rest of the images in the sequence. In most cases, if under-segmentation occurs, the segmented images will be sufficient to be used for training data. If over-segmentation occurs, the error in segmentation will propagate as most of the background will be considered as part of the bacterial colonies. One approach to handle this is to place more seeds for labeling the background to ensure that the seeded region growing algorithm does not label them as part of the bacterial colonies.

The algorithm to extract the contours from the images is written in Python and Julia. The preprocessing and contour extraction use Python's `OpenCV` package [61] and the seeded region growing is in Julia's `ImageSegmentation.jl` package [65]. The final step of extracting the contours from a segmented image uses `findContours` from OpenCV, although any contour extraction algorithm should be sufficient for this last step.

Algorithm 1 Extracting Contours from Images of Bacterial Colonies

Input: Sequence of K images, each containing m colonies
Output: Sequence of K sets of m contours
  for $k = 1$ to K **do**
    Preprocess image k by cropping, blurring, and frame differencing as needed
  end for
  Initialize m seeds for image 1 in the sequence
  for $k = 1$ to K **do**
    Apply seeded region growing to image k to get segmented image k
    if $k < K$ **then**
      Initialize m new seeds for image $k + 1$ from segmented image k
    end if
  end for
  for $k = 1$ to K **do**
    Extract m contours from segmented image k
  end for

6.2.3 Results

Figure 2a features contour extraction using a threshold-based method [40]. Figure 2b, in contrast, shows contour extraction using seeded region growing. The combination of frame differencing and seeded region growing successfully captures the false edges of the bottom two colonies. The extracted contours in Figure 2b generally have smoother contours than the contours in Figure 2a due to the bilateral and median filters used. For a single image sequence, the algorithm takes about 37 seconds on an AMD Ryzen$^{\text{TM}}$ 7 5825U with 24 gigabytes of RAM.

Supplementary Material

Section, equation, and algorithm numbers with the prefix S refer to this supplementary material; all other numbers refer to the main text.

S1 Derivation of the Governing Equations

This section derives the tangent-angle form (2.9) of the governing equations, stated in Section 2.5, from the nondimensionalized model (2.7), using the shape-scaling representation as an intermediate step, and then describes how the nonlocal transport mechanism modifies the result. Much of the derivation is similar to that in [39].

S1.1 Constant-Speed and Shape-Scaling Representations

This section gives the details of the constant-speed and shape-scaling representations summarized in Section 2.4.

Given a planar contour $\boldsymbol{\gamma} : \mathbb{T}^1 \to \mathbb{R}^2$, define its average speed σ as

$$\sigma = \frac{1}{2\pi} \int_{\mathbb{T}^1} |\boldsymbol{\gamma}'(u)| \, \mathrm{d}u,$$

the arc length of $\boldsymbol{\gamma}$ divided by 2π. Then we may construct a change of variables $\xi : \mathbb{T}^1 \to \mathbb{T}^1$ to convert $\boldsymbol{\gamma}$ to a constant-speed representation. Let $\eta : \mathbb{T}^1 \to \mathbb{T}^1$ be defined by

$$\eta(u) := -\pi + \frac{1}{\sigma} \int_{-\pi}^{u} |\boldsymbol{\gamma}'(v)| \, \mathrm{d}v.$$

Note that η is well defined as a function from $\mathbb{T}^1$ to $\mathbb{T}^1$ since $\eta(-\pi) = -\pi$ and $\eta(\pi) = \pi$. As long as the contour $\boldsymbol{\gamma}$ is regular, i.e. $|\boldsymbol{\gamma}'(u)| \neq 0$ for any $u \in \mathbb{T}^1$, we have $\eta'(u) > 0$. Using this fact along with the definition of σ we can see that η is injective and surjective, so it is an invertible function on $\mathbb{T}^1$ and we may let $\xi = \eta^{-1}$. Then $\boldsymbol{\psi} := \boldsymbol{\gamma} \circ \xi$ is the constant speed representation of the contour $\boldsymbol{\gamma}$, since

$$|\boldsymbol{\psi}'(u)| = |\boldsymbol{\gamma}'(\xi(u))||\xi'(u)| = |\boldsymbol{\gamma}'(\xi(u))| \frac{1}{|\eta'(\xi(u))|} = \sigma.$$

Alternatively, ξ may be computed as the solution to the initial value problem

$$\xi'(u) = \frac{\sigma}{|\boldsymbol{\gamma}'(\xi(u))|}, \qquad \xi(-\pi) = -\pi$$

(see also Eq. (S3.1)). This procedure is more suitable for numerical algorithms and is the implementation actually used to convert discrete contours to a constant-speed representation (see Algorithm S8 for details on the numerical implementation of this algorithm).

Dividing by σ yields the unit-speed representation $\boldsymbol{\phi}$, otherwise called the "shape" of the interface:

$$\boldsymbol{\phi}(u) := \frac{\boldsymbol{\psi}(u)}{\sigma} = \frac{\boldsymbol{\gamma}(\xi(u))}{\sigma}. \tag{S1.1}$$

Thus, if one has access to the shape $\boldsymbol{\phi}$ and the average speed σ (the "shape-scaling representation"), one may recover $\boldsymbol{\gamma}$ using

$$\boldsymbol{\gamma}(u) = \sigma \boldsymbol{\phi}(\eta(u)).$$

Note that in [38] the scale is defined as $\varepsilon := 1/\sigma^2$, which gives another equivalent representation.

Recall that the unit tangent is $\boldsymbol{\tau} = \boldsymbol{\phi}' = (-\sin\theta, \cos\theta)^T$ and the outward normal is $\mathbf{n} = \boldsymbol{\tau}^\perp = (\cos\theta, \sin\theta)^T$ (Section 2.4). Differentiation relates the shape function $\boldsymbol{\phi}$ to the tangent-angle function θ as follows:

$$\begin{aligned}
\frac{\partial^2 \boldsymbol{\phi}}{\partial u^2}(u,t) &= -\frac{\partial \theta}{\partial u}(u,t)\mathbf{n}(u,t) \\
\frac{\partial^3 \boldsymbol{\phi}}{\partial u^3}(u,t) &= -\frac{\partial^2 \theta}{\partial u^2}(u,t)\mathbf{n}(u,t) - \left(\frac{\partial \theta}{\partial u}(u,t)\right)^2 \boldsymbol{\tau}(u,t) \\
\frac{\partial^2 \boldsymbol{\phi}}{\partial u \partial t}(u,t) &= -\frac{\partial \theta}{\partial t}(u,t)\mathbf{n}(u,t)
\end{aligned} \tag{S1.2}$$

S1.2 Identities for Evolving Planar Contours

Consider an evolving regular planar contour $\boldsymbol{\gamma}(u,t) \in C^2(\mathbb{T}^1 \times [0,T], \mathbb{R}^2)$ which is twice differentiable in u and t. Let $\sigma(t)$ be the average speed of $\boldsymbol{\gamma}(\cdot,t)$, given by

$$\sigma(t) := \frac{1}{2\pi}\int_{\mathbb{T}^1} \left|\frac{\partial \boldsymbol{\gamma}}{\partial u}(v,t)\right| \mathrm{d}v. \tag{S1.3}$$

Then, as in the previous section, define $\eta : \mathbb{T}^1 \times [0,T] \to \mathbb{T}^1$ and $\xi : \mathbb{T}^1 \times [0,T] \to \mathbb{T}^1$ by

$$\eta(u,t) = -\pi + \frac{1}{\sigma(t)}\int_{-\pi}^{u} \left|\frac{\partial \boldsymbol{\gamma}}{\partial u}(v,t)\right| \mathrm{d}v \tag{S1.4}$$

$$\eta(\xi(u,t),t) = \xi(\eta(u,t),t) = u \tag{S1.5}$$

for all $(u,t) \in \mathbb{T}^1 \times [0,T]$. Thus η satisfies

$$\eta(-\pi,t) = -\pi, \qquad \frac{\partial \eta}{\partial u}(u,t) = \frac{1}{\sigma(t)}\left|\frac{\partial \boldsymbol{\gamma}}{\partial u}(u,t)\right|. \tag{S1.6}$$

Thus

$$1 = \frac{\partial}{\partial u}\left[\eta(\xi(u,t),t)\right] = \frac{\partial \eta}{\partial u}(\xi(u,t),t)\frac{\partial \xi}{\partial u}(u,t)$$

so ξ satisfies

$$\xi(-\pi,t) = -\pi, \qquad \frac{\partial \xi}{\partial u}(u,t) = \frac{\sigma(t)}{\left|\frac{\partial \boldsymbol{\gamma}}{\partial u}(\xi(u,t),t)\right|}. \tag{S1.7}$$

As before, $\boldsymbol{\psi}(u,t) := \boldsymbol{\gamma}(\xi(u,t),t)$ is the constant-speed representation of this evolving contour. Using the chain rule we have

$$\frac{\partial \boldsymbol{\psi}}{\partial u}(u,t) = \frac{\partial \boldsymbol{\gamma}}{\partial u}(\xi(u,t),t)\frac{\partial \xi}{\partial u}(u,t) \tag{S1.8}$$

$$\frac{\partial \boldsymbol{\psi}}{\partial t}(u,t) = \frac{\partial \boldsymbol{\gamma}}{\partial u}(\xi(u,t),t)\frac{\partial \xi}{\partial t}(u,t) + \frac{\partial \boldsymbol{\gamma}}{\partial t}(\xi(u,t),t). \tag{S1.9}$$

Define $\mathbf{v}(u,t) = \frac{\partial \boldsymbol{\gamma}}{\partial t}(\xi(u,t),t)$ to be the rate of change of $\boldsymbol{\gamma}$ evaluated in the constant-speed coordinate system. Then we have

$$\frac{\partial \mathbf{v}}{\partial u}(u,t) = \frac{\partial^2 \boldsymbol{\gamma}}{\partial u \partial t}(\xi(u,t),t)\frac{\partial \xi}{\partial u}(u,t). \tag{S1.10}$$

Furthermore, by differentiating the average speed $\sigma(t)$ and applying (S1.7) we obtain

$$\begin{aligned}\sigma'(t) &= \frac{1}{2\pi}\int_{\mathbb{T}^1} \frac{\frac{\partial\boldsymbol{\gamma}}{\partial u}(v,t)}{\left|\frac{\partial\boldsymbol{\gamma}}{\partial u}(v,t)\right|}\cdot\frac{\partial^2\boldsymbol{\gamma}}{\partial u\partial t}(v,t)\,\mathrm{d}v \\ &= \frac{1}{2\pi\sigma(t)}\int_{\mathbb{T}^1}\frac{\partial\boldsymbol{\gamma}}{\partial u}(\xi(v,t),t)\cdot\frac{\partial^2\boldsymbol{\gamma}}{\partial u\partial t}(\xi(v,t),t)\left(\frac{\partial\xi}{\partial u}(v,t)\right)^2\mathrm{d}v \\ &= \frac{1}{2\pi\sigma(t)}\int_{\mathbb{T}^1}\frac{\partial\boldsymbol{\psi}}{\partial u}(v,t)\cdot\frac{\partial\mathbf{v}}{\partial u}(v,t)\,\mathrm{d}v\end{aligned} \tag{S1.11}$$

By differentiating $\xi(\eta(u,t),t)=u$ in time, we have

$$\frac{\partial\xi}{\partial t}(\eta(u,t),t) = -\frac{\partial\xi}{\partial u}(\eta(u,t),t)\frac{\partial\eta}{\partial t}(u,t),$$

or, after changing variables,

$$\frac{\partial\xi}{\partial t}(u,t) = -\frac{\partial\xi}{\partial u}(u,t)\frac{\partial\eta}{\partial t}(\xi(u,t),t). \tag{S1.12}$$

Thus, using (S1.9) and (S1.8), we have

$$\begin{aligned}\frac{\partial\boldsymbol{\psi}}{\partial t}(u,t)-\mathbf{v}(u,t) &= \frac{\partial\boldsymbol{\gamma}}{\partial u}(\xi(u,t),t)\frac{\partial\xi}{\partial t}(u,t) \\ &= -\frac{\partial\boldsymbol{\gamma}}{\partial u}(\xi(u,t),t)\frac{\partial\xi}{\partial u}(u,t)\frac{\partial\eta}{\partial t}(\xi(u,t),t) \\ &= -\frac{\partial\boldsymbol{\psi}}{\partial u}(u,t)\frac{\partial\eta}{\partial t}(\xi(u,t),t).\end{aligned} \tag{S1.13}$$

Using (S1.4) we may compute (by way of (S1.7), (S1.11), (S1.8), (S1.10))

$$\begin{aligned}&\frac{\partial\eta}{\partial t}(\xi(u,t),t) \\ &= \frac{1}{\sigma(t)}\int_{-\pi}^{\xi(u,t)}\frac{\frac{\partial\boldsymbol{\gamma}}{\partial u}(v,t)}{\left|\frac{\partial\boldsymbol{\gamma}}{\partial u}(v,t)\right|}\cdot\frac{\partial^2\boldsymbol{\gamma}}{\partial u\partial t}(v,t)\,\mathrm{d}v \\ &\quad -\frac{\sigma'(t)}{\sigma(t)^2}\int_{-\pi}^{\xi(u,t)}\left|\frac{\partial\boldsymbol{\gamma}}{\partial u}(v,t)\right|\mathrm{d}v \\ &= \frac{1}{\sigma(t)}\int_{-\pi}^{u}\frac{\frac{\partial\boldsymbol{\gamma}}{\partial u}(\xi(v,t),t)}{\left|\frac{\partial\boldsymbol{\gamma}}{\partial u}(\xi(v,t),t)\right|}\cdot\frac{\partial^2\boldsymbol{\gamma}}{\partial u\partial t}(\xi(v,t),t)\frac{\partial\xi}{\partial u}(v,t)\,\mathrm{d}v \\ &\quad -\frac{\sigma(t)}{\sigma(t)^2}\left(\eta(\xi(u,t),t)+\pi\right)\sigma'(t) \\ &= \frac{1}{\sigma(t)^2}\int_{-\pi}^{u}\frac{\partial\boldsymbol{\psi}}{\partial u}(v,t)\cdot\frac{\partial\mathbf{v}}{\partial u}(v,t)\,\mathrm{d}v \\ &\quad -\frac{1}{\sigma(t)^2}\frac{u+\pi}{2\pi}\int_{\mathbb{T}^1}\frac{\partial\boldsymbol{\psi}}{\partial u}(v,t)\cdot\frac{\partial\mathbf{v}}{\partial u}(v,t)\,\mathrm{d}v\end{aligned} \tag{S1.14}$$

Now, recall the operator $P_0 : C(\mathbb{T}^1,\mathbb{R}^d)\to C(\mathbb{T}^1,\mathbb{R}^d)$ of Section 2.4, which maps f to the zero-Dirichlet primitive of $f-\mu_f$:

$$P_0[f](u) := \int_{-\pi}^{u} f(v)\,\mathrm{d}v - (u+\pi)\mu_f.$$

Then we have

$$\frac{\partial \eta}{\partial t}(\xi(u,t),t) = \frac{1}{\sigma(t)^2} P_0 \left[\frac{\partial \boldsymbol{\psi}}{\partial u}(\cdot,t) \cdot \frac{\partial \mathbf{v}}{\partial u}(\cdot,t) \right](u), \tag{S1.15}$$

so that the transport relation

$$\frac{\partial \boldsymbol{\psi}}{\partial t}(u,t) + \frac{P_0 \left[\frac{\partial \boldsymbol{\psi}}{\partial u}(\cdot,t) \cdot \frac{\partial \mathbf{v}}{\partial u}(\cdot,t) \right](u)}{\sigma(t)^2} \frac{\partial \boldsymbol{\psi}}{\partial u}(u,t) = \mathbf{v}(u,t) \tag{S1.16}$$

holds.

Now, let $\boldsymbol{\phi}(u,t) = \boldsymbol{\psi}(u,t)/\sigma(t)$ be the shape of the interface as in (S1.1), where

$$\left| \frac{\partial \boldsymbol{\phi}}{\partial u}(u,t) \right| = \frac{\sigma(t)}{\sigma(t)} = 1. \tag{S1.17}$$

Using (2.1) and (2.2) we can derive that

$$\frac{\partial \boldsymbol{\phi}}{\partial u}(u,t) = \boldsymbol{\tau}(\xi(u,t),t) \tag{S1.18}$$

$$\left(\frac{\partial \boldsymbol{\phi}}{\partial u} \right)^{\perp} = \mathbf{n}(\xi(u,t),t) \tag{S1.19}$$

Recalling the identity (2.3), we have

$$\frac{\partial^2 \boldsymbol{\phi}}{\partial u^2}(u,t) = \sigma(t)\, \kappa_{\boldsymbol{\gamma}}(\xi(u,t),t)\, \mathbf{n}(\xi(u,t),t). \tag{S1.20}$$

Conversely, at each fixed t the recovery formula of the previous section gives

$$\boldsymbol{\gamma}(u,t) = \sigma(t)\, \boldsymbol{\phi}(\eta(u,t),t). \tag{S1.21}$$

Since η depends on the parametrization of $\boldsymbol{\gamma}$ through (S1.4), the evolving pair $(\boldsymbol{\phi}, \sigma)$ determines the evolving contour only up to a time-dependent reparametrization, namely $\boldsymbol{\psi}(u,t) = \sigma(t)\boldsymbol{\phi}(u,t)$. This is sufficient for our purposes: the governing equations (2.5) prescribe only the normal velocity of each contour, so the evolution of the curve traced by $\boldsymbol{\gamma}^{(i)}(\cdot,t)$ does not depend on its parametrization. Note also that, unlike the tangent-angle representation, the shape-scaling representation loses no centroid information, since $\boldsymbol{\psi}$ is a position rather than a derivative.

S1.3 Governing Equations in Shape-Scaling Representation

The goal of this section is to use the shape-scaling representation to derive an equivalent form of the governing equations. To avoid carrying the curvature coefficient μ through the derivation only to eliminate it later (see Section 2.3), we derive directly from the nondimensionalized governing equations (2.7), in which the curvature coefficient has already been set to 1 and the interaction kernels f, g carry an explicit factor of τ. Much of the derivation is similar to that found in [39]. The nonlocal transport mechanism is treated separately in Section S1.5.

Consider a family of evolving contours $\{\boldsymbol{\gamma}^{(i)}(u,t), i \in [m]\} \subseteq C^2(\mathbb{T}^1 \times [0,T], \mathbb{R}^2)$ which are twice differentiable in u and t. Let $\sigma^{(i)}(t)$ be the average speed of the ith contour, given by (S1.3), and define

$$\eta^{(i)}(u,t), \qquad \xi^{(i)}(u,t), \qquad \boldsymbol{\psi}^{(i)}(u,t), \qquad \boldsymbol{\phi}^{(i)}(u,t), \qquad i \in [m], u \in \mathbb{T}^1, t \in [0,T]$$

by (S1.4), (S1.5), $\boldsymbol{\psi}^{(i)}(u,t) = \boldsymbol{\gamma}^{(i)}(\xi^{(i)}(u,t),t)$, and (S1.1), respectively.

Now consider the constant growth/decay and normal-direction nonlocal force terms. These may be encapsulated in

$$F^{(i)}(u,t) = c^{(i)} - \tau v_{\mathrm{NL}}^{(i)}(u,t).$$

Once again considering the identity (2.3), the governing equations can be written as

$$\partial_t \boldsymbol{\gamma}^{(i)}(u,t) = \frac{\partial_u \boldsymbol{\tau}^{(i)}(u,t)}{|\partial_u \boldsymbol{\gamma}^{(i)}(u,t)|} + F^{(i)}(u,t)\mathbf{n}^{(i)}(u,t) \tag{S1.22}$$

Let $\mathcal{F}^{(i)}$ be a scaled and shifted version of $F^{(i)}$:

$$\begin{aligned}
\frac{\mathcal{F}^{(i)}(u,t)}{\sigma^{(i)}(t)} &= c^{(i)} - \tau v_{\mathrm{NL}}^{(i)}(\xi^{(i)}(u,t),t) \\
&= c^{(i)} - \tau \int_{\mathbb{T}^1} f\left(\frac{1}{2}\left|\boldsymbol{\psi}^{(i)}(u,t) - \boldsymbol{\psi}^{(i)}(v,t)\right|^2\right)\sigma^{(i)}(t)\,\mathrm{d}v \\
&\quad - \tau \sum_{j\neq i} \int_{\mathbb{T}^1} g\left(\frac{1}{2}\left|\boldsymbol{\psi}^{(i)}(u,t) - \boldsymbol{\psi}^{(j)}(v,t)\right|^2\right)\sigma^{(j)}(t)\,\mathrm{d}v.
\end{aligned}$$

Thus, applying the change of variables $\xi^{(i)}$ and the identity (S1.20) to the equation (S1.22) yields

$$\begin{aligned}
\partial_t \boldsymbol{\gamma}^{(i)}(\xi^{(i)}(u,t),t) = \frac{1}{\sigma^{(i)}(t)}\Big[&\partial_u^2 \boldsymbol{\phi}^{(i)}(u,t) \\
&+ \mathcal{F}^{(i)}(u,t)\left(\partial_u \boldsymbol{\phi}^{(i)}(u,t)\right)^{\perp}\Big].
\end{aligned}$$

Following the previous section, define the left-hand side above as

$$\mathbf{v}^{(i)}(u,t) = \partial_t \boldsymbol{\gamma}^{(i)}(\xi^{(i)}(u,t),t).$$

Recalling (S1.17), we know that $\partial_u^2 \boldsymbol{\phi}^{(i)}(u,t) \perp \partial_u \boldsymbol{\phi}^{(i)}(u,t)$. Hence $\mathbf{v}^{(i)} \perp \partial_u \boldsymbol{\phi}^{(i)}(u,t)$. Suppressing function arguments, we can write

$$\begin{aligned}
\mathbf{v}^{(i)} &= \langle \mathbf{v}^{(i)}, {\boldsymbol{\phi}_u^{(i)}}^{\perp}\rangle {\boldsymbol{\phi}_u^{(i)}}^{\perp} \\
\mathbf{v}_u^{(i)} &= \left(\langle \mathbf{v}_u^{(i)}, {\boldsymbol{\phi}_u^{(i)}}^{\perp}\rangle + \langle \mathbf{v}^{(i)}, {\boldsymbol{\phi}_{uu}^{(i)}}^{\perp}\rangle\right){\boldsymbol{\phi}_u^{(i)}}^{\perp} \\
&\quad + \langle \mathbf{v}^{(i)}, {\boldsymbol{\phi}_u^{(i)}}^{\perp}\rangle {\boldsymbol{\phi}_{uu}^{(i)}}^{\perp} \\
\langle \mathbf{v}_u^{(i)}, \boldsymbol{\phi}_u^{(i)}\rangle &= \langle \mathbf{v}^{(i)}, {\boldsymbol{\phi}_u^{(i)}}^{\perp}\rangle \langle {\boldsymbol{\phi}_{uu}^{(i)}}^{\perp}, \boldsymbol{\phi}_u^{(i)}\rangle \\
&= -\langle \mathbf{v}^{(i)}, {\boldsymbol{\phi}_u^{(i)}}^{\perp}\rangle \langle \boldsymbol{\phi}_{uu}^{(i)}, {\boldsymbol{\phi}_u^{(i)}}^{\perp}\rangle
\end{aligned}$$

Thus

$$\begin{aligned}
\langle \mathbf{v}_u^{(i)}, \boldsymbol{\phi}_u^{(i)}\rangle &= -\langle \mathbf{v}^{(i)}, {\boldsymbol{\phi}_u^{(i)}}^{\perp}\rangle \langle \boldsymbol{\phi}_{uu}^{(i)}, {\boldsymbol{\phi}_u^{(i)}}^{\perp}\rangle \\
&= -\frac{1}{\sigma^{(i)}}\langle \boldsymbol{\phi}_{uu}^{(i)} + \mathcal{F}^{(i)}{\boldsymbol{\phi}_u^{(i)}}^{\perp}, {\boldsymbol{\phi}_u^{(i)}}^{\perp}\rangle \langle \boldsymbol{\phi}_{uu}^{(i)}, {\boldsymbol{\phi}_u^{(i)}}^{\perp}\rangle \\
&= -\frac{1}{\sigma^{(i)}}\left(\langle \boldsymbol{\phi}_{uu}^{(i)}, {\boldsymbol{\phi}_u^{(i)}}^{\perp}\rangle^2 + \mathcal{F}^{(i)}\langle \boldsymbol{\phi}_{uu}^{(i)}, {\boldsymbol{\phi}_u^{(i)}}^{\perp}\rangle\right) \\
&= -\frac{1}{\sigma^{(i)}}\left(|\boldsymbol{\phi}_{uu}^{(i)}|^2 + \mathcal{F}^{(i)}\langle \boldsymbol{\phi}_{uu}^{(i)}, {\boldsymbol{\phi}_u^{(i)}}^{\perp}\rangle\right)
\end{aligned}$$

Let $a^{(i)}(u,t)$ be the term in parentheses on the right. Then

$$a^{(i)} = -\sigma^{(i)}\langle \mathbf{v}_u^{(i)}, \boldsymbol{\phi}_u^{(i)}\rangle = -\langle \mathbf{v}_u^{(i)}, \boldsymbol{\psi}_u^{(i)}\rangle$$

so the transport relation (S1.16) becomes

$$\boldsymbol{\psi}_t^{(i)} = \mathbf{v}^{(i)} + \frac{P_0\left[a^{(i)}\right]}{{\sigma^{(i)}}^2}\boldsymbol{\psi}_u^{(i)}(u,t).$$

Furthermore, recalling (S1.11), we have

$$\sigma^{(i)\prime}(t) = -\frac{1}{2\pi\sigma^{(i)}(t)}\int_{\mathbb{T}^1} a^{(i)}(v,t)\,\mathrm{d}v = -\frac{1}{\sigma^{(i)}(t)}\mu_{a^{(i)}}$$

Using the product rule, we have

$$\begin{aligned}\partial_t\boldsymbol{\phi}^{(i)}(u,t) &= \frac{\partial_t\boldsymbol{\psi}^{(i)}(u,t)}{\sigma^{(i)}(t)} - \boldsymbol{\psi}^{(i)}(u,t)\frac{\sigma^{(i)\prime}(t)}{\sigma^{(i)}(t)^2}\\ &= \frac{\mathbf{v}^{(i)}(u,t)}{\sigma^{(i)}(t)} + \frac{P_0\left[a^{(i)}(\cdot,t)\right](u)}{\sigma^{(i)}(t)^2}\boldsymbol{\phi}_u^{(i)}(u,t)\\ &\quad + \frac{\boldsymbol{\phi}^{(i)}(u,t)}{\sigma^{(i)}(t)^2}\mu_{a^{(i)}}\end{aligned} \tag{S1.23}$$

Thus, again suppressing function arguments,

$$\begin{aligned}\boldsymbol{\phi}_t^{(i)} &= \frac{1}{{\sigma^{(i)}}^2}\Big[\boldsymbol{\phi}_{uu}^{(i)} + P_0[a^{(i)}]\boldsymbol{\phi}_u^{(i)}\\ &\quad + \mathcal{F}^{(i)}{\boldsymbol{\phi}_u^{(i)}}^{\perp} + \mu_{a^{(i)}}\boldsymbol{\phi}^{(i)}\Big]\end{aligned}$$

Thus the full dynamics can be described as follows (with the equation for $\sigma^{(i)}$ written in terms of ${\sigma^{(i)}}^2$ for convenience):

$$\begin{aligned}\boldsymbol{\phi}_t^{(i)} &= \frac{1}{{\sigma^{(i)}}^2}\Big[\boldsymbol{\phi}_{uu}^{(i)} + P_0[a^{(i)}]\boldsymbol{\phi}_u^{(i)}\\ &\quad + \mathcal{F}^{(i)}{\boldsymbol{\phi}_u^{(i)}}^{\perp} + \mu_{a^{(i)}}\boldsymbol{\phi}^{(i)}\Big]\\ \partial_t\left[{\sigma^{(i)}}^2\right] &= -2\mu_{a^{(i)}}\\ a^{(i)} &= |\boldsymbol{\phi}_{uu}^{(i)}|^2 + \langle\boldsymbol{\phi}_{uu}^{(i)}, {\boldsymbol{\phi}_u^{(i)}}^{\perp}\rangle\mathcal{F}^{(i)},\\ \mathcal{F}^{(i)}(u,t) &= \sigma^{(i)}\Big(c^{(i)} -\\ &\quad \tau\int_{\mathbb{T}^1} f\left(\frac{1}{2}\left|\boldsymbol{\psi}^{(i)}(u,t) - \boldsymbol{\psi}^{(i)}(v,t)\right|^2\right)\sigma^{(i)}\,\mathrm{d}v\\ &\quad -\tau\sum_{j\neq i}\int_{\mathbb{T}^1} g\left(\frac{1}{2}\left|\boldsymbol{\psi}^{(i)}(u,t) - \boldsymbol{\psi}^{(j)}(v,t)\right|^2\right)\sigma^{(j)}\,\mathrm{d}v\Big)\\ \boldsymbol{\psi}^{(i)} &= \sigma^{(i)}\boldsymbol{\phi}^{(i)}\end{aligned} \tag{S1.24}$$

This form of the equations has the advantage of having a constant in front of the diffusion term in the evolution for $\boldsymbol{\phi}^{(i)}$, but also introduces a nonlocal tangential (advection) term $P_0[a^{(i)}]\boldsymbol{\phi}_u^{(i)}$ which was not present before. This term only reparametrizes the contour, keeping its speed constant; it should not be confused with the nonlocal transport mechanism of Section 2.6.

S1.4 Governing Equations in Tangent-Angle Representation

The tangent-angle representation may also be used to obtain an equivalent form of the governing equations (2.7).

As in a previous section, let

$$\boldsymbol{\tau}^{(i)}(u,t) = \frac{\partial \boldsymbol{\phi}^{(i)}}{\partial u}(u,t)$$

be the unit tangent to the ith contour, and let $\mathbf{n}^{(i)}$ be the corresponding unit normal. Using the properties described by (S1.2) and differentiating the evolution equation (S1.24), we obtain

$$\begin{aligned}
\partial_t \boldsymbol{\phi}_u^{(i)} &= \frac{1}{{\sigma^{(i)}}^2}\Big[\boldsymbol{\phi}_{uuu}^{(i)} + P_0[a^{(i)}]\boldsymbol{\phi}_{uu}^{(i)} + (a^{(i)} - \mu_{a^{(i)}})\boldsymbol{\phi}_u^{(i)} \\
&\quad + \mu_{a^{(i)}}\boldsymbol{\phi}_u^{(i)} + \mathcal{F}_u^{(i)}{\boldsymbol{\phi}_u^{(i)}}^{\perp} + \mathcal{F}^{(i)}{\boldsymbol{\phi}_{uu}^{(i)}}^{\perp}\Big] \\
&= \frac{1}{{\sigma^{(i)}}^2}\Big[-\theta_{uu}^{(i)}\mathbf{n}^{(i)} - \left(\theta_u^{(i)}\right)^2\boldsymbol{\tau}^{(i)} - P_0[a^{(i)}]\theta_u^{(i)}\mathbf{n}^{(i)} \\
&\quad + a^{(i)}\boldsymbol{\tau}^{(i)} + \mathcal{F}_u^{(i)}\mathbf{n}^{(i)} + \mathcal{F}^{(i)}\theta_u^{(i)}\boldsymbol{\tau}^{(i)}\Big] \\
&= \frac{1}{{\sigma^{(i)}}^2}\left[-\theta_{uu}^{(i)} - P_0[a^{(i)}]\theta_u^{(i)} + \mathcal{F}_u^{(i)}\right]\mathbf{n}^{(i)}
\end{aligned}$$

Here we have used the fact that

$$a^{(i)} = |\boldsymbol{\phi}_{uu}^{(i)}|^2 + \langle \boldsymbol{\phi}_{uu}^{(i)}, {\boldsymbol{\phi}_u^{(i)}}^{\perp}\rangle\mathcal{F}^{(i)} = {\theta_u^{(i)}}^2 - \mathcal{F}^{(i)}\theta_u^{(i)}.$$

Taking a dot product with $-\mathbf{n}^{(i)}$ yields $\theta_t^{(i)}$:

$$\theta_t^{(i)} = \frac{1}{{\sigma^{(i)}}^2}\left[\theta_{uu}^{(i)} + P_0[a^{(i)}]\theta_u^{(i)} - \mathcal{F}_u^{(i)}\right]$$

Furthermore, since the centroids are given by $\boldsymbol{\mu}^{(i)} = \mu_{\boldsymbol{\psi}^{(i)}}$, by integrating (S1.24) multiplied by $\sigma^{(i)}$ we obtain

$$\partial_t \boldsymbol{\mu}^{(i)} = \frac{1}{2\pi\sigma^{(i)}}\int_{\mathbb{T}^1} P_0[a^{(i)}]\boldsymbol{\tau}^{(i)} + \mathcal{F}^{(i)}\mathbf{n}^{(i)}\,\mathrm{d}v.$$

The evolution equation for $\sigma^{(i)}$ remains the same. Together, these give the system (2.9) stated in Section 2.5, along with the winding number and periodicity constraints for the tangent-angle function.

S1.5 Effect of the Nonlocal Transport Mechanism

Since $\mathbf{h}^{(i)}(t)$ is a rigid translation, the evolution of the curve traced by $\boldsymbol{\gamma}^{(i)}(\cdot,t)$ still does not depend on its parametrization, and the derivations of Sections S1.3 and S1.4 carry over with one change. The velocity $\mathbf{v}^{(i)}$ gains the u-independent term $\mathbf{h}^{(i)}$, so it is $\mathbf{v}^{(i)} - \mathbf{h}^{(i)}$, rather than $\mathbf{v}^{(i)}$, that is orthogonal to $\boldsymbol{\phi}_u^{(i)}$. Because $\partial_u \mathbf{h}^{(i)} = 0$, however, $\mathbf{v}_u^{(i)}$ is unchanged, and hence so are $a^{(i)}$, $P_0[a^{(i)}]$, and the evolution of $\sigma^{(i)}$. In the shape-scaling representation (S1.24), the equation for $\boldsymbol{\phi}^{(i)}$ gains the term $\mathbf{h}^{(i)}/\sigma^{(i)}$. In the tangent-angle representation, this term is annihilated by differentiation in u, so the equations for $\theta^{(i)}$ and $\sigma^{(i)}$ are unchanged. Integrating the equation for $\boldsymbol{\phi}^{(i)}$ multiplied by $\sigma^{(i)}$, as in Section S1.4, shows that the centroid velocity gains the term $\mathbf{h}^{(i)}$. In the nondimensionalized variables, this gives the system (2.13).

S2 Optimization Algorithms

This section collects the algorithms used to evaluate the training objective and to minimize it (Sections 3.4–3.6). Algorithm S1 simulates the contours over the sequence of images with a fixed number of substeps per image, and Algorithm S2 uses it to evaluate the relative loss. Algorithms S3 and S4 are the two optimization methods: `scipy`'s implementation of the Nelder–Mead method, and L-BFGS-B with central-difference gradients in the log parametrization of Section 3.6.

Algorithm S1 Simulate Contours Numerically

Input: Starting contour points $\{\boldsymbol{\psi}_{j,0}^{(i)}, i \in [m], j \in [n]\}$
Number of images K (images are one unit of physical time apart), Number of substeps per image N_{sub}
Interaction kernels $f, g : \mathbb{R}^+ \to \mathbb{R}^+$
Time scaling factor $\tau \in \mathbb{R}^+$, Model parameters $c^{(1)}, \dots, c^{(m)}, \beta \in \mathbb{R}^+$, $\alpha \in \mathbb{R}$ ($\alpha = 0$ for the standard model)

Output: Calibrated contour points $\{\boldsymbol{\psi}_{j,k}^{(i)}, i \in [m], j \in [n], k \in [K]\}$ at the image times
Convert starting contour points to tangent-angle representation using Algorithm S11, and recenter using Algorithm S13
Store tangent-angle representation $\{\theta_j^{(i)}, \boldsymbol{\mu}^{(i)}, \sigma^{(i)},\ i \in [m], j \in [n]\}$ and set the substep length

$$\Delta t \leftarrow \frac{1}{\tau N_{\text{sub}}}$$

for $k = 1, \dots, K$ **do**
  for $\ell = 1, \dots, N_{\text{sub}}$ **do**
    Advance $\theta_j^{(i)}, \boldsymbol{\mu}^{(i)}, \sigma^{(i)}$ by Δt using Algorithm S15, which also recenters the tangent-angle function and reconstructs the contour points $\boldsymbol{\psi}_j^{(i)}$
    If the time step restriction in Algorithm S15 shortened the step, stop with an error (see Section 3.4)
  end for
  Recalibrate the contour points using Algorithm S9 and store them:

$$\boldsymbol{\psi}_{j,k}^{(i)} \leftarrow \boldsymbol{\psi}_j^{(i)}, \qquad i \in [m], j \in [n]$$

  (Recalibration only changes the parametrization used for comparison with the target contours; it does not affect the subsequent evolution.)
end for

Algorithm S2 Evaluating the Relative Loss over Time Sequence

Input: Starting contour points $\{\tilde{\boldsymbol{\psi}}_{j,0}^{(i)}, i \in [m], j \in [n]\}$
Number of images K, Number of substeps per image N_{sub}
Target contour points $\{\tilde{\boldsymbol{\psi}}_{j,k}^{(i)}, i \in [m], j \in [n], k \in [K]\}$, target average speeds $\{\tilde{\sigma}_k^{(i)}, i \in [m], k \in [K]\}$, and target centroids $\{\boldsymbol{\mu}_k^{(i)}, i \in [m], k \in [K]\}$
Interaction kernels $f, g : \mathbb{R}^+ \to \mathbb{R}^+$,
Time scaling factor $\tau \in \mathbb{R}^+$, Model parameters $c^{(1)}, \ldots, c^{(m)}, \beta \in \mathbb{R}^+$, $\alpha \in \mathbb{R}$
Output: Loss L_{rel}
Simulate contours numerically using Algorithm S1 to obtain calibrated simulated contour points

$$\{\boldsymbol{\psi}_{j,k}^{(i)}, i \in [m], j \in [n], k \in [K]\}.$$

Compute relative loss (assuming target and simulated contour points are calibrated appropriately using Algorithm S9):

$$L_{\text{rel}} \leftarrow \frac{1}{K} \sum_{k=1}^{K} \frac{\sum_{i=1}^{m} \sum_{j=1}^{n} |\boldsymbol{\psi}_{j,k}^{(i)} - \tilde{\boldsymbol{\psi}}_{j,k}^{(i)}|^2 \, \tilde{\sigma}_k^{(i)}}{\sum_{i=1}^{m} \sum_{j=1}^{n} |\tilde{\boldsymbol{\psi}}_{j,k}^{(i)} - \boldsymbol{\mu}_k^{(i)}|^2 \, \tilde{\sigma}_k^{(i)}}$$

Algorithm S3 Nelder–Mead method for Derivative-Free Optimization

Input: Objective function to minimize, $f : \mathbb{R}^d \to \mathbb{R}$
Number of iterations N
Reflection coefficient $\rho > 0$ (default 1), expansion multiplier $\chi > 1$ (default 2), contraction coefficient $\psi \in (0, 0.5]$ (default 0.5), shrink factor $\sigma \in (0, 1)$ (default 0.5)
Initial guess $\mathbf{x}_0 \in \mathbb{R}^d$
(Optional): Initial (additional) simplex vertices $\mathbf{x}_1, \ldots, \mathbf{x}_d \in \mathbb{R}^d$
Output: Current point $\mathbf{x}^* \in \mathbb{R}^d$ with minimal objective function value
if simplex vertices $\mathbf{x}_1, \ldots, \mathbf{x}_d \in \mathbb{R}^d$ not provided **then**
generate them by perturbing one coordinate of $\mathbf{x}_0$ at a time:

$$\mathbf{x}_i \leftarrow \begin{cases} \mathbf{x}_0 + 0.05\,(\mathbf{x}_0)_i\,\mathbf{e}_i & \text{if } (\mathbf{x}_0)_i \neq 0 \\ \mathbf{x}_0 + 0.00025\,\mathbf{e}_i & \text{otherwise,} \end{cases} \qquad i \in [d].$$

end if
Compute function values at vertices:

$$f_0, \ldots, f_d \leftarrow f(\mathbf{x}_0), \ldots, f(\mathbf{x}_d)$$

for $j = 1$ to N **do**
Reindex vertices $\mathbf{x}_0, \ldots, \mathbf{x}_d$ and associated function values $f_0, \ldots, f_d$ to maintain ordering

$$f_0 \leq f_1 \leq \cdots \leq f_d$$

Compute centroid of the d best vertices and reflected point

$$\mathbf{x}_c \leftarrow \frac{1}{d}\sum_{i=0}^{d-1} \mathbf{x}_i, \quad \mathbf{x}_r \leftarrow \mathbf{x}_c + \rho\,(\mathbf{x}_c - \mathbf{x}_d), \quad f_r \leftarrow f(\mathbf{x}_r)$$

if $f_r < f_0$, compute expansion point:

$$\mathbf{x}_e \leftarrow \mathbf{x}_c + \rho\,\chi\,(\mathbf{x}_c - \mathbf{x}_d), \qquad f_e \leftarrow f(\mathbf{x}_e)$$

$$\mathbf{x}_d \leftarrow \begin{cases} \mathbf{x}_e & \text{if } f_e < f_r \\ \mathbf{x}_r & \text{otherwise} \end{cases}, \qquad f_d \leftarrow \begin{cases} f_e & \text{if } f_e < f_r \\ f_r & \text{otherwise} \end{cases}$$

else if $f_0 \leq f_r < f_{d-1}$, use reflection

$$\mathbf{x}_d \leftarrow \mathbf{x}_r, \qquad f_d \leftarrow f_r$$

else ($f_r \geq f_{d-1}$) use contraction, outside if $f_r < f_d$ and inside otherwise:

$$\mathbf{x}_k \leftarrow \begin{cases} \mathbf{x}_c + \psi\rho(\mathbf{x}_c - \mathbf{x}_d) & \text{if } f_r < f_d \\ \mathbf{x}_c - \psi(\mathbf{x}_c - \mathbf{x}_d) & \text{if } f_r \geq f_d, \end{cases} \qquad f_k \leftarrow f(\mathbf{x}_k)$$

if $f_k \leq f_r$ (outside) or $f_k < f_d$ (inside), accept:

$$\mathbf{x}_d \leftarrow \mathbf{x}_k, \qquad f_d \leftarrow f_k$$

else perform shrink and re-evaluate:

$$\mathbf{x}_i \leftarrow \mathbf{x}_0 + \sigma(\mathbf{x}_i - \mathbf{x}_0), \qquad f_i \leftarrow f(\mathbf{x}_i), \qquad i = 1, \ldots, d$$

end if
end for
return best point $\mathbf{x}_0$

Algorithm S4 L-BFGS-B with Central-Difference Gradients

Input: Objective L (Algorithm S2) as a function of the parameters $\mathbf{x} \in \mathbb{R}^d$
Initial parameters $\mathbf{x}_0 \in \mathbb{R}^d$, set $\Lambda \subseteq [d]$ of log-parametrized coordinates
Finite-difference step $h > 0$ (default 10^{-4}), first-step length $\rho_0 > 0$ (default 0.05), bound radius $B > 0$ (default 5)
Output: Best parameters found, $\mathbf{x}^* \in \mathbb{R}^d$

1: Change variables:

$$u_{0,i} \leftarrow \begin{cases} \log x_{0,i} & i \in \Lambda \\ x_{0,i} & i \notin \Lambda \end{cases}, \qquad x_i(\mathbf{u}) := \begin{cases} e^{u_i} & i \in \Lambda \\ u_i & i \notin \Lambda \end{cases}$$

$$\ell(\mathbf{u}) := L(\mathbf{x}(\mathbf{u}))$$

2: Set finite-difference steps: $h_i \leftarrow h$ for $i \in \Lambda$, and $h_i \leftarrow h\,|x_{0,i}|$ otherwise
3: Define the gradient estimate, computing the $2d + 1$ values of ℓ in parallel:

$$\hat{g}_i(\mathbf{u}) \leftarrow \frac{\ell(\mathbf{u} + h_i \mathbf{e}_i) - \ell(\mathbf{u} - h_i \mathbf{e}_i)}{2h_i}, \qquad i \in [d]$$

If a simulation fails at a point $\mathbf{u}$, use $\ell(\mathbf{u}) \leftarrow \ell_{\text{best}} + 10^3(1 + \|\mathbf{u} - \mathbf{u}_{\text{best}}\|^2)$, where $\mathbf{u}_{\text{best}}$ is the best point evaluated so far
4: Scale the objective so that the first step has length ρ_0:

$$\kappa \leftarrow \rho_0 / \|\hat{\mathbf{g}}(\mathbf{u}_0)\|$$

5: Set bounds $|u_i - u_{0,i}| \le B$ for $i \in \Lambda$ (and $\beta \ge 10^{-3}$ if β is optimized)
6: Run L-BFGS-B on $\kappa\,\ell$ with gradient $\kappa\,\hat{\mathbf{g}}$ from $\mathbf{u}_0$, subject to the bounds, until its convergence criteria are met or the time budget is exhausted
7: **return** $\mathbf{x}^* \leftarrow \mathbf{x}(\mathbf{u}_{\text{best}})$

S3 Numerical Algorithms

This section describes in detail the numerical procedures used to model the evolution of contours.

Algorithm S5 describes a 3rd-order WENO interpolation procedure for functions defined on $\mathbb{T}^1$. It assumes that values of a function are known at evenly spaced points $v_j := -\pi + \frac{2\pi}{n}(j - 1) \in \mathbb{T}^1$, $j \in [n]$. Given $u \in \mathbb{T}^1$, the interpolation procedure uses a four-point stencil with indices $\{\ell', \ell, r, r'\} \subseteq [n]$ with $u \in [\ell, r)$. The procedure estimates the value of the function at u using two quadratic Lagrange polynomials on the three-point stencils $\{\ell', \ell, r\}$ and $\{\ell, r, r'\}$, which are combined using Jiang–Shu smoothness measures [66]. A WENO procedure is used so that higher-order curvature information is preserved while not inducing spurious oscillations near discontinuities.

Algorithm S5 WENO Interpolation on $\mathbb{T}^1$ using Quadratic Lagrange Polynomials

Input: Interpolation locations $u_1, \dots, u_m \in \mathbb{T}^1$

Function values $f_1, \dots, f_n \in V$ at evenly spaced points $v_j := -\pi + \frac{2\pi}{n}(j-1) \in \mathbb{T}^1, \quad j \in [n]$

Output: Interpolated values $\tilde{f}_1, \dots, \tilde{f}_m \in V$ at locations $u_1, \dots, u_m$

1: Define interval length and machine epsilon:

$$\Delta \leftarrow 2\pi/n, \qquad \varepsilon \leftarrow \text{machine epsilon}$$

2: Compute stencil indices $\{(\ell_i', \ell_i, r_i, r_i')\}_{i\in[m]} \subseteq [n]$:

$$\ell_i \leftarrow 1 + \lfloor (u_i + \pi)/\Delta \rfloor, \quad r_i \leftarrow (\ell_i + 1) \bmod n$$

$$\ell_i' \leftarrow (\ell_i - 1) \bmod n, \quad r_i' \leftarrow (r_i + 1) \bmod n$$

3: Map u_i from $[v_{\ell_i}, v_{r_i})$ to $[0, 1)$:

$$\theta_i \leftarrow \frac{u_i - (-\pi + (\ell_i - 1)\,\Delta)}{\Delta}, \qquad i \in [m]$$

4: Initialize linear blending weight:

$$w_i \leftarrow \frac{2}{3} - \frac{1}{3}\theta_i, \qquad i \in [m]$$

5: Compute Jiang–Shu smoothness measures ($i \in [m]$):

$$(\beta_L)_i \leftarrow \varepsilon + (f_{r_i} - f_{\ell_i})^2 + \frac{13}{12}(f_{r_i} + f_{\ell_i'} - 2f_{\ell_i})^2,$$
$$(\beta_R)_i \leftarrow \varepsilon + (f_{r_i} - f_{\ell_i})^2 + \frac{13}{12}(f_{r_i'} + f_{\ell_i} - 2f_{r_i})^2$$

6: Adjust weights using smoothness measures ($i \in [m]$):

$$w_i \leftarrow \begin{cases} \dfrac{w_i}{w_i + \left(\frac{(\beta_L)_i}{(\beta_R)_i}\right)^2 (1 - w_i)}, & \text{if } (\beta_R)_i > (\beta_L)_i, \\ 1 - \dfrac{(1 - w_i)}{(1 - w_i) + \left(\frac{(\beta_R)_i}{(\beta_L)_i}\right)^2 w_i}, & \text{otherwise.} \end{cases}$$

7: Evaluate left and right quadratic polynomials ($i \in [m]$):

$$(p_L)_i \leftarrow f_{\ell_i} + \tfrac{1}{2}(f_{r_i} - f_{\ell_i'})\theta_i + \tfrac{1}{2}(f_{r_i} + f_{\ell_i'} - 2f_{\ell_i})\theta_i^2,$$
$$(p_R)_i \leftarrow f_{r_i} + \tfrac{1}{2}(f_{r_i'} - f_{\ell_i})(\theta_i - 1) + \tfrac{1}{2}(f_{r_i'} + f_{\ell_i} - 2f_{r_i})(\theta_i - 1)^2$$

8: Compute final interpolated values ($i \in [m]$):

$$\tilde{f}_i \leftarrow w_i(p_L)_i + (1 - w_i)(p_R)_i$$

Algorithms S6 and S7 show how derivatives and antiderivatives of functions defined on $\mathbb{T}^1$ can be computed using the Fast Fourier Transform (FFT).

Algorithm S6 Computing Derivative of Discrete Functions on $\mathbb{T}^1$ using FFT

Input: Function values $f_1, \ldots, f_n \in \mathbb{R}$ at n evenly spaced points in $\mathbb{T}^1$
Output: Derivative values $f'_1, \ldots, f'_n \in \mathbb{R}$ at n evenly spaced points in $\mathbb{T}^1$

1: Vectorize:

$$\mathbf{f} \leftarrow (f_1, \ldots, f_n) \in \mathbb{R}^n$$

2: Compute FFT:

$$\hat{\mathbf{f}} \leftarrow \texttt{FFT}(\mathbf{f})$$

3: Define wavenumbers:

$$\mathbf{k} \leftarrow [0, 1, \ldots, \lfloor (n-1)/2 \rfloor, -\lfloor n/2 \rfloor, \ldots, -1]$$

4: Multiply by $i\mathbf{k}$ and compute inverse FFT:

$$\mathbf{f}' = \texttt{IFFT}(i\mathbf{k} \odot \hat{\mathbf{f}})$$

5: Extract components:

$$(f'_1, \ldots, f'_n) \leftarrow \mathbf{f}' \in \mathbb{R}^n$$

A discrete function's antiderivative is periodic, and hence expressible via Fourier synthesis, if and only if the function has zero mean; a function with nonzero mean instead has an antiderivative containing a linear drift term, which is not periodic. To handle both cases, Algorithm S7 first separates out the mean c of the input, computes a periodic (mean-zero) primitive of the remaining zero-mean part using the FFT, and then reintroduces the linear drift $c \cdot x_j$ corresponding to the removed mean. An optional final adjustment anchors the result at x_1 to produce a zero-Dirichlet primitive instead.

Algorithm S7 Computing Primitive of Discrete Functions on $\mathbb{T}^1$ using FFT

Input: Function values $f_1, \dots, f_n \in \mathbb{R}$ at n evenly spaced points $x_j := -\pi + \frac{2\pi}{n}(j-1) \in \mathbb{T}^1$
Type of primitive (mean-zero or zero-Dirichlet)
Output: Primitive values $Pf_1, \dots, Pf_n \in \mathbb{R}$ at n evenly spaced points in $\mathbb{T}^1$

1: Compute the mean of f and the mean-zero remainder:

$$c \leftarrow \frac{1}{n}\sum_{j=1}^{n} f_j, \qquad \tilde{f}_j \leftarrow f_j - c, \quad j \in [n]$$

2: Vectorize:

$$\tilde{\mathbf{f}} \leftarrow (\tilde{f}_1, \dots, \tilde{f}_n) \in \mathbb{R}^n$$

3: Compute FFT:

$$\hat{\tilde{\mathbf{f}}} \leftarrow \texttt{FFT}(\tilde{\mathbf{f}})$$

4: Define reciprocal of wavenumbers:

$$\boldsymbol{\xi} \leftarrow \left[1, 1, \frac{1}{2}, \frac{1}{3}, \dots, \frac{1}{\lfloor (n-1)/2 \rfloor}, -\frac{1}{\lfloor n/2 \rfloor}, \dots, -1\right]$$

5: Multiply by $-i\boldsymbol{\xi}$ and compute inverse FFT:

$$P\tilde{\mathbf{f}} = \texttt{IFFT}(-i\boldsymbol{\xi} \odot \hat{\tilde{\mathbf{f}}})$$

6: Reintroduce the linear drift corresponding to the removed mean, giving the mean-zero primitive:

$$Pf_j \leftarrow \left(P\tilde{\mathbf{f}}\right)_j + c \cdot x_j, \qquad j \in [n]$$

7: For zero-Dirichlet primitive, additionally:

$$Pf_j \leftarrow Pf_j - Pf_1, \qquad j \in [n]$$

Algorithm S8 is a procedure for obtaining a constant-speed representation of a contour $\boldsymbol{\gamma} : \mathbb{T}^1 \to \mathbb{R}^2$, i.e. an alternate parametrization $\boldsymbol{\psi} : \mathbb{T}^1 \to \mathbb{R}^2$ defined so that $|\boldsymbol{\psi}'(u)|$ is constant for all $u \in \mathbb{T}^1$. This is done by computing a change of variables map $\xi : \mathbb{T}^1 \to \mathbb{T}^1$ which solves the initial value problem

$$\xi'(u) = \frac{\sigma}{|\boldsymbol{\gamma}'(\xi(u))|}, \qquad \xi(-\pi) = -\pi \tag{S3.1}$$

where

$$\sigma := \frac{1}{2\pi}\int_{\mathbb{T}^1} |\boldsymbol{\gamma}'(u)| \, \mathrm{d}u$$

is the average speed of the contour. Then $\boldsymbol{\psi}$ may be defined as the composition $\boldsymbol{\psi} := \boldsymbol{\gamma} \circ \xi$, and such a $\boldsymbol{\psi}$ will have the property that $|\boldsymbol{\psi}'(u)| = \sigma$ for all $u \in \mathbb{T}^1$ (i.e. $\boldsymbol{\psi}$ is a constant-speed representation of the same contour). Algorithm S8 demonstrates this procedure using discrete representations of the contours, by using the FFT to compute $|\boldsymbol{\gamma}'(u)|$ and then a 4th-order Runge–Kutta method to solve the IVP (S3.1). Lastly, interpolation (see Algorithm S5) is needed to compute the composition $\boldsymbol{\gamma} \circ \xi$.

Algorithm S8 Resampling for Contours in $\mathbb{R}^2$

Input: Contour points $\boldsymbol{\gamma}_1, \ldots, \boldsymbol{\gamma}_n \in \mathbb{R}^2$
Output: Contour points $\boldsymbol{\psi}_1, \ldots, \boldsymbol{\psi}_n \in \mathbb{R}^2$ describing a constant-speed representation of the original contour

1: Compute directional derivatives of $\boldsymbol{\gamma}$ (see Algorithm S6) ($i \in [2]$):

$$(\boldsymbol{\gamma}_1)'_i, \ldots, (\boldsymbol{\gamma}_n)'_i \leftarrow \texttt{Derivative}((\boldsymbol{\gamma}_1)_i, \ldots, (\boldsymbol{\gamma}_n)_i)$$

2: Compute speed ($j \in [n]$) and average speed

$$\sigma_j \leftarrow \sqrt{\sum_{i=1}^{2} ((\boldsymbol{\gamma}_j)'_i)^2}, \qquad \sigma \leftarrow \frac{1}{n} \sum_{j=1}^{n} \sigma_j$$

3: Compute change of variables $\xi_1, \ldots, \xi_n \in \mathbb{T}^1$ to satisfy the IVP (S3.1). See Algorithm S5

$$\xi_1 \leftarrow -\pi,$$

for $j = 2$ to n **do**

$$\begin{aligned}
F_1 &\leftarrow \texttt{Interpolate}(\xi_{j-1}, (\sigma/\sigma_1, \ldots, \sigma/\sigma_n)) \\
F_2 &\leftarrow \texttt{Interpolate}(\xi_{j-1} + \tfrac{1}{2}\tfrac{2\pi}{n} F_1, (\sigma/\sigma_1, \ldots, \sigma/\sigma_n)) \\
F_3 &\leftarrow \texttt{Interpolate}(\xi_{j-1} + \tfrac{1}{2}\tfrac{2\pi}{n} F_2, (\sigma/\sigma_1, \ldots, \sigma/\sigma_n)) \\
F_4 &\leftarrow \texttt{Interpolate}(\xi_{j-1} + \tfrac{2\pi}{n} F_3, (\sigma/\sigma_1, \ldots, \sigma/\sigma_n)) \\
\xi_j &\leftarrow \xi_{j-1} + \tfrac{2\pi}{n}(\tfrac{1}{6}F_1 + \tfrac{1}{3}F_2 + \tfrac{1}{3}F_3 + \tfrac{1}{6}F_4)
\end{aligned}$$

endfor

4: Compute constant-speed representation $\boldsymbol{\psi} : \mathbb{T}^1 \to \mathbb{R}^2$:

$$\boldsymbol{\psi}_1, \ldots, \boldsymbol{\psi}_n \leftarrow \texttt{Interpolate}((\xi_1, \ldots, \xi_n), (\boldsymbol{\gamma}_1, \ldots, \boldsymbol{\gamma}_n))$$

Algorithm S9 describes the procedure used to recalibrate contour points so that the first point is oriented in the $(1, 0)$ direction from the centroid of the contour. This is done primarily because the training metric used to measure the discrepancy between a target contour $\tilde{\boldsymbol{\psi}}$ and a simulated contour $\boldsymbol{\psi}$ measures the distance between $\tilde{\boldsymbol{\psi}}(u)$ and $\boldsymbol{\psi}(u)$ for the same point $u \in \mathbb{T}^1$. Thus, rotating the parametrizations so that $u = -\pi$ points to the $(1, 0)$ direction in each case ensures that similar parameter values match up with corresponding portions of each contour.

This procedure assumes that the interior of the contour $C \subseteq \mathbb{R}^2$ is star-shaped with respect to the centroid $\boldsymbol{\mu}$ of the contour, i.e. that for all $\mathbf{x} \in C$ the line segment between $\boldsymbol{\mu}$ and $\mathbf{x}$ is contained within C.

Algorithm S9 Contour Recalibration

Input: Constant-speed contour points $\boldsymbol{\psi}_1, \ldots, \boldsymbol{\psi}_n \in \mathbb{R}^2$

Output: Recalibrated constant-speed contour points $\boldsymbol{\psi}_1, \ldots, \boldsymbol{\psi}_n \in \mathbb{R}^2$ which describe the same contour, but with the first point oriented in the $(1, 0)$ direction from the centroid

1: Compute centroid:

$$\boldsymbol{\mu} \leftarrow \frac{1}{n} \sum_{i=1}^{n} \boldsymbol{\psi}_i$$

2: Compute angles of points from centroid ($j \in [n]$):

$$\theta_j \leftarrow \arctan((\boldsymbol{\psi}_j)_2 - (\boldsymbol{\mu})_2, (\boldsymbol{\psi}_j)_1 - (\boldsymbol{\mu})_1)$$

3: Find indices near zero crossing (assuming counterclockwise orientation):

$$k \leftarrow \min\{j : \theta_j > 0\}, \qquad \ell \leftarrow (k-1) \bmod n$$

4: Use linear interpolation to find the approximate fractional index of zero crossing:

$$r \leftarrow \frac{k\theta_\ell - \ell\theta_k}{\theta_\ell - \theta_k}$$

5: Get shifted angles in $\mathbb{T}^1$, $(i = 1, \ldots, n)$:

$$\theta'_i \leftarrow (\tfrac{2\pi}{n} r + \theta_1) - 2\pi \lfloor (\tfrac{2\pi}{n} r + \theta_1 + \pi)/(2\pi) \rfloor$$

6: Interpolate points at shifted indices (see Algorithm S5):

$$\boldsymbol{\psi}_1, \ldots, \boldsymbol{\psi}_n \leftarrow \texttt{Interpolate}((\theta'_1, \ldots, \theta'_n), (\boldsymbol{\psi}_1, \ldots, \boldsymbol{\psi}_n))$$

Algorithm S10 shows a 2nd-order Exponential Time Differencing (ETD2) scheme for time-stepping the forced diffusion equation

$$q_t(u,t) = q_{uu}(u,t) + G(u,t)$$

where $q(\cdot, t) : \mathbb{T}^1 \to \mathbb{R}$. In the Fourier domain, the equation reads

$$\hat{q}_t(k,t) = -k^2 \hat{q}(k,t) + \hat{G}(k,t)$$

which has the exact solution

$$\hat{q}(k, t+\Delta t) = e^{-k^2 \Delta t} \hat{q}(k,t) + \int_0^{\Delta t} e^{-k^2(\Delta t - s)} \hat{G}(k, t+s)\, \mathrm{d}s.$$

If we approximate $\hat{G}(k, t+s)$ using linear interpolation

$$\hat{G}(\cdot, t+s) \approx \hat{G}(\cdot, t) + \frac{s}{\Delta t} \left(\hat{G}(\cdot, t+\Delta t) - \hat{G}(\cdot, t) \right)$$

then substituting into the exact formula yields

$$\begin{aligned} \hat{q}(k, t+\Delta t) = e^{-k^2 \Delta t} \hat{q}(k,t) &+ \Delta t \varphi_1(-k^2 \Delta t) \hat{G}(k,t) \\ &+ \Delta t \varphi_2(-k^2 \Delta t) \left(\hat{G}(k, t+\Delta t) - \hat{G}(k,t) \right) \end{aligned}$$

where $\varphi_1(z) = \frac{e^z - 1}{z}$ and

$$\varphi_2(z) = \frac{\varphi_1(z) - 1}{z} = \frac{e^z - z - 1}{z^2}.$$

This yields a second-order approximation for $q(u, t + \Delta t)$ after the inverse FFT is performed. Note that for small inputs, the functions φ_1 and φ_2 are computed using a series approximation to avoid small terms in the denominator.

Algorithm S10 Timestepping Forced Diffusion Equation on $\mathbb{T}^1$

Input: Function values $q_{1,k}, \ldots, q_{n,k} \in \mathbb{R}$ at time t_k

Time step $\Delta t > 0$

Current-time forcing $G_{1,k}, \ldots, G_{n,k} \in \mathbb{R}$ and one-step-ahead forcing $G_{1,k+1}, \ldots, G_{n,k+1}$ at time $t_{k+1} := t_k + \Delta t$

Output: Function values $q_{1,k+1}, \ldots, q_{n,k+1} \in \mathbb{R}$ at time $t_{k+1} := t_k + \Delta t$

1: Vectorize:

$$\mathbf{q}_k \leftarrow (q_{1,k}, \ldots, q_{n,k}) \in \mathbb{R}^n$$

$$\mathbf{G}_\ell \leftarrow (G_{1,\ell}, \ldots, G_{n,\ell}) \in \mathbb{R}^n, \qquad \ell = k, k+1$$

2: Define wavenumbers:

$$\mathbf{k} \leftarrow [0, 1, \ldots, \lfloor (n-1)/2 \rfloor, -\lfloor n/2 \rfloor, \ldots, -1] \in \mathbb{Z}^n$$

3: Compute FFTs

$$\hat{\mathbf{q}}_k \leftarrow \texttt{FFT}(\mathbf{q}_k); \qquad \hat{\mathbf{G}}_\ell \leftarrow \texttt{FFT}(\mathbf{G}_\ell), \quad \ell = k, k+1$$

4: Compute exponential of Fourier-space diffusion operator (using vectorized operations):

$$\mathbf{E}^{(0)} \leftarrow \exp(-\mathbf{k}^2 \Delta t) \in \mathbb{R}^n$$

5: Compute coefficients for time-stepping, using vectorized operations (note: revert to series approximation for $|k_i^2 \Delta t| \ll 1$):

$$\mathbf{E}^{(1)} \leftarrow \frac{\mathbf{E}^{(0)} - 1}{-\Delta t \mathbf{k}^2} \in \mathbb{R}^n$$

$$\mathbf{E}^{(2)} \leftarrow \frac{\mathbf{E}^{(1)} - 1}{-\Delta t \mathbf{k}^2} \in \mathbb{R}^n$$

6: Evaluate approximation to exact time step using vectorized operations:

$$\hat{\mathbf{q}}_{k+1} \leftarrow \mathbf{E}^{(0)} \hat{\mathbf{q}}_k + \Delta t \left(\mathbf{E}^{(1)} \hat{\mathbf{G}}_k + \mathbf{E}^{(2)} (\hat{\mathbf{G}}_{k+1} - \hat{\mathbf{G}}_k) \right)$$

7: Compute inverse FFT:

$$\mathbf{q}_{k+1} \leftarrow \texttt{IFFT}(\hat{\mathbf{q}}_{k+1})$$

8: Extract components:

$$q_{1,k+1}, \ldots, q_{n,k+1} \leftarrow \mathbf{q}_{k+1} \in \mathbb{R}^n$$

Algorithm S11 shows how to transform a contour from a representation as a parametrized function $\boldsymbol{\gamma} : \mathbb{T}^1 \to \mathbb{R}^2$ to the tangent-angle representation, consisting of a tangent-angle function $\theta : \mathbb{T}^1 \to \mathbb{R}$, a speed σ, and a centroid $\boldsymbol{\mu}$. The discrete values $\theta_1, \ldots, \theta_n$ of the tangent-angle function are only defined $\bmod 2\pi$, but because simulating the evolution of the contour requires simulating a diffusion equation for θ, we must ensure that there are no unnecessarily large jumps in multiples of 2π. To accomplish this, the discrete values are modified so that successive angles differ by less than π (for example, if $\theta_2 - \theta_1 > \pi$, then we subtract 2π from θ_2). Note that this may mean that θ_n differs from θ_1 by approximately an integer multiple of 2π, which is the winding number of the contour.

By comparison, Algorithm S12 (which converts back to the parametrized representation) is much more straightforward and utilizes mean-zero primitives (see Algorithm S7).

Algorithm S11 Transforming a Contour $\boldsymbol{\gamma} : \mathbb{T}^1 \to \mathbb{R}^2$ to Tangent-Angle Representation

Input: Contour points $\boldsymbol{\gamma}_1, \ldots, \boldsymbol{\gamma}_n \in \mathbb{R}^2$
Output: Tangent-angle function values $\theta_1, \ldots, \theta_n \in \mathbb{R}$,
centroid $\boldsymbol{\mu} \in \mathbb{R}^2$, average speed $\sigma \in \mathbb{R}^+$

1: Compute directional derivatives of $\boldsymbol{\gamma}$ (see Algorithm S6) ($i \in [2]$):

$$(\boldsymbol{\gamma}_1)'_i, \ldots, (\boldsymbol{\gamma}_n)'_i \leftarrow \texttt{Derivative}((\boldsymbol{\gamma}_1)_i, \ldots, (\boldsymbol{\gamma}_n)_i)$$

2: Compute speed ($j \in [n]$), average speed, and centroid

$$\sigma_j \leftarrow \sqrt{\sum_{i=1}^{2} ((\boldsymbol{\gamma}_j)'_i)^2}, \qquad \sigma \leftarrow \frac{1}{n} \sum_{j=1}^{n} \sigma_j, \qquad \boldsymbol{\mu} \leftarrow \frac{1}{n} \sum_{j=1}^{n} \boldsymbol{\gamma}_j$$

3: Compute tangent-angle function values ($j \in [n]$):

$$\theta_j \leftarrow \arctan\left(-\frac{(\boldsymbol{\gamma}_j)'_1}{\sigma_j}, \frac{(\boldsymbol{\gamma}_j)'_2}{\sigma_j}\right)$$

4: Clean tangent-angle function values so that $|\theta_j - \theta_{j+1}| \leq \pi$ for all $j \in [n-1]$:

$$\delta_j \leftarrow \theta_{j+1} - \theta_j, \qquad j \in [n-1]$$

$$J_j \leftarrow 2\pi \left(H(\delta_j - \pi) - H(-\delta_j - \pi)\right), \qquad j \in [n-1]$$

$$\theta_j \leftarrow \theta_1 + \sum_{i=1}^{j-1} (\delta_i - J_i)$$

Algorithm S12 Transforming a Contour from Tangent-Angle Representation to $\boldsymbol{\gamma} : \mathbb{T}^1 \to \mathbb{R}^2$

Input: Tangent-angle function values $\theta_1, \ldots, \theta_n \in \mathbb{R}$
centroid $\boldsymbol{\mu} \in \mathbb{R}^2$, average speed $\sigma \in \mathbb{R}^+$
Output: Contour points $\boldsymbol{\gamma}_1, \ldots, \boldsymbol{\gamma}_n \in \mathbb{R}^2$

1: Compute mean-zero primitives componentwise ($j \in [n]$) (see Algorithm S7)

$$(\boldsymbol{\gamma}_j)_1 \leftarrow \sigma \cdot \texttt{Primitive}(-\sin(\theta_1), \ldots, -\sin(\theta_n)) + \mu_1$$

$$(\boldsymbol{\gamma}_j)_2 \leftarrow \sigma \cdot \texttt{Primitive}(\cos(\theta_1), \ldots, \cos(\theta_n)) + \mu_2$$

Due to numerical error, the discrete values $\theta_1, \ldots, \theta_n$ of the tangent-angle function may drift so that they no longer represent a contour centered at $(0, 0)$. This will cause the stored centroid value to be inaccurate. Instead of compounding the error by recalculating the centroid, Algorithm S13 demonstrates how to recenter the tangent-angle function by using several Newton steps. After these steps are performed, the discrete values are "cleaned" in a procedure similar to Algorithm S11 to ensure that $|\theta_{j+1} - \theta_j| \leq \pi$ for all $j \in [n-1]$.

Algorithm S13 Recenter Tangent-Angle Function

Input: Tangent-angle function values $\theta_1, \ldots, \theta_n \in \mathbb{T}^1$
Output: New tangent-angle function values $\theta_1, \ldots, \theta_n \in \mathbb{T}^1$

1: Save first tangent angle: $\varphi \leftarrow \theta_1$

2: Compute negative sines and cosines ($j \in [n]$):

$$s_j = -\sin(\theta_j), \qquad c_j = \cos(\theta_j)$$

3: Newton iteration on $\mathbf{F}(x, y) := \sum_{j=1}^{n} \frac{(s_j + x, c_j + y)}{\sqrt{s_j^2 + c_j^2}}$:

for $i = 1$ to 4 **do**

$$\mathbf{F} \leftarrow \sum_{j=1}^{n} \frac{(s_j, c_j)}{\sqrt{s_j^2 + c_j^2}}$$

$$D\mathbf{F} \leftarrow \sum_{j=1}^{n} \frac{1}{(s_j^2 + c_j^2)^{3/2}} \begin{bmatrix} c_j^2 & -s_j c_j \\ -s_j c_j & s_j^2 \end{bmatrix}$$

$$(s_j, c_j) \leftarrow (s_j, c_j) - (D\mathbf{F})^{-1}\mathbf{F}, \qquad j \in [n]$$

4: Recompute tangent-angle function values ($j \in [n]$):

$$\theta_j \leftarrow \arctan\left(-\frac{s_j}{\sqrt{s_j^2 + c_j^2}}, \frac{c_j}{\sqrt{s_j^2 + c_j^2}} \right), \qquad j \in [n]$$

5: Clean tangent-angle function values so that $|\theta_j - \theta_{j+1}| \leq \pi$ for all $j \in [n-1]$:

$$\delta_j \leftarrow \theta_{j+1} - \theta_j, \qquad j \in [n-1]$$

$$J_j \leftarrow 2\pi \left(H(\delta_j - \pi) - H(-\delta_j - \pi) \right), \qquad j \in [n-1]$$

$$\theta_j \leftarrow \theta_1 + \sum_{i=1}^{j-1} (\delta_i - J_i)$$

6: Re-anchor the tangent-angle values so that the first value is unchanged from its input value φ, preserving the reference direction of the contour:

$$\theta_j \leftarrow \theta_j - \theta_1 + \varphi, \qquad j \in [n]$$

Algorithm S14 computes the nonlocal forces between each pair of colonies. For self-interactions ($i \in [m]$)

$$\texttt{NLF}^{(ii)}(u) = \int_{\mathbb{T}^1} f\left(\frac{1}{2} \left| \boldsymbol{\gamma}^{(i)}(u,t) - \boldsymbol{\gamma}^{(i)}(v,t) \right|^2 \right) \left| \partial_u \boldsymbol{\gamma}^{(i)}(v,t) \right| \mathrm{d}v$$

and for cross-interactions ($i, j \in [m]; i \neq j$)

$$\texttt{NLF}^{(ij)}(u) = \int_{\mathbb{T}^1} g\left(\frac{1}{2} \left| \boldsymbol{\gamma}^{(i)}(u,t) - \boldsymbol{\gamma}^{(j)}(v,t) \right|^2 \right) \left| \partial_u \boldsymbol{\gamma}^{(j)}(v,t) \right| \mathrm{d}v.$$

Assuming that the parametrizations $\{\boldsymbol{\gamma}^{(i)}\}_{i \in [m]}$ each have constant speed $\sigma^{(i)}$, the computation of the integrals reduces to sums over the discrete contour points. Lastly, recall that the forces are multiplied by the time scaling $\tau > 0$, which is included as a model parameter.

Algorithm S14 Compute Nonlocal Forces

Input: Contour points $\{\boldsymbol{\gamma}_\ell^{(i)} : i \in [m], \ell \in [n]\} \subset \mathbb{R}^2$ at n evenly spaced points in $\mathbb{T}^1$, for m colonies
Average speeds $\{\sigma^{(i)}\}_{i=1}^m \subset \mathbb{R}$
Interaction kernels $f, g : \mathbb{R}^+ \to \mathbb{R}^+$
Time scaling factor $\tau \in \mathbb{R}^+$

Output: Nonlocal forces $\{\mathtt{NLF}_\ell^{(ij)} : i, j \in [m], \ell \in [n]\}$ on colony i from colony j, at n evenly spaced points in $\mathbb{T}^1$

1: Compute self-interactions ($i \in [m]; \ell \in [n]$):

$$\mathtt{NLF}_\ell^{(ii)} \leftarrow \tau\sigma^{(i)} \frac{2\pi}{n} \sum_{k=1}^{n} f\left(\frac{1}{2}\left\|\boldsymbol{\gamma}_\ell^{(i)} - \boldsymbol{\gamma}_k^{(i)}\right\|^2\right)$$

2: Compute cross-interactions ($i, j \in [m]; i \neq j; \ell \in [n]$):

$$\mathtt{NLF}_\ell^{(ij)} \leftarrow \tau\sigma^{(j)} \frac{2\pi}{n} \sum_{k=1}^{n} g\left(\frac{1}{2}\left\|\boldsymbol{\gamma}_\ell^{(i)} - \boldsymbol{\gamma}_k^{(j)}\right\|^2\right)$$

Algorithm S15 (continued in Algorithm S17) illustrates the procedure for computing a time step of the governing equations in the tangent-angle representation (see (2.9), or (2.13) when the transport mechanism is included). First, the curvature $\mathcal{K}^{(i)}$, the scaled forcing $\mathcal{F}^{(i)}$, and the bending parameter $\mathbf{a}^{(i)}$ are computed at the current time, together with the minimum distances between colonies, which come from the same pairwise distances used to compute the nonlocal forces. The time step restriction which guarantees that $\sigma^{(i)}$ stays positive (derived below) is then checked; with a fixed number of substeps per image it does not become active in practice (Section 3.4), but the check is retained as a safeguard. Estimates of $(\mathcal{K}^{(i)}, \mathcal{F}^{(i)}, \mathbf{a}^{(i)})$ at the next time are computed using linear extrapolation, and $\sigma^{(i)}$ is advanced to the next time step. The total diffusion force for the tangent-angle function is computed, and the tangent-angle function is advanced using a pseudo-exponential integrator. The centroids are then advanced by the trapezoidal rule. Note that the centroid velocity carries the factor $1/\sigma^{(i)}$, not the factor $1/{\sigma^{(i)}}^2$ which appears in the equation for $\theta^{(i)}$ (compare (2.9)). Finally, the tangent-angle function is recentered, the contour points are reconstructed, and, if the transport mechanism is included, each colony is translated rigidly by the transport displacement. The transport step is an explicit Euler step, which uses the minimum distances from the start of the step and the directions between the centroids after they have been advanced.

For the standard model, each stage of the update is second-order accurate in Δt: the exponential time-differencing update for $\theta^{(i)}$ (Algorithm S10), the second-order Adams–Bashforth update for ${\sigma^{(i)}}^2$, and the trapezoidal rule for the centroids, with the forces at the next time estimated by linear extrapolation. The convergence study in Section 3.4 confirms that the loss converges at second order.

Time Step Restriction. The average speeds evolve according to $\partial_t[{\sigma^{(i)}}^2] = -2\mu_{a^{(i)}}$ (see (2.9)). Writing $\mathcal{M} = \mathcal{M}_k^{(i)}$ for the mean of $a^{(i)}$ at time t_k, $\dot{\mathcal{M}} = \partial_t \tilde{\mathcal{M}}_k^{(i)}$ for the estimate of its time derivative, and $\sigma = \sigma_k^{(i)}$, Algorithm S15 advances the squared speed by the second-order step

$$\left(\sigma_{k+1}^{(i)}\right)^2 = \sigma^2 - 2\Delta t\, \mathcal{M} - (\Delta t)^2 \dot{\mathcal{M}},$$

whose right-hand side must be positive for $\sigma_{k+1}^{(i)}$ to be defined. This holds whenever $2\Delta t|\mathcal{M}| +$

$(\Delta t)^2|\dot{\mathcal{M}}| < \sigma^2$, and a simple sufficient condition is

$$\Delta t < \frac{1}{2}\frac{\sigma^2}{\sqrt{\mathcal{M}^2 + \frac{1}{2}\sigma^2|\dot{\mathcal{M}}|}}. \tag{S3.2}$$

To see this, let $D = \mathcal{M}^2 + \frac{1}{2}\sigma^2|\dot{\mathcal{M}}|$ and $x = |\mathcal{M}|/\sqrt{D} \in [0,1]$, so that $\frac{1}{2}\sigma^2|\dot{\mathcal{M}}|/D = 1 - x^2$. If (S3.2) holds, then $2\Delta t|\mathcal{M}| < \sigma^2 x$ and $(\Delta t)^2|\dot{\mathcal{M}}| < \sigma^4|\dot{\mathcal{M}}|/(4D) = \frac{1}{2}\sigma^2(1-x^2)$, so

$$\begin{aligned} 2\Delta t|\mathcal{M}| + (\Delta t)^2|\dot{\mathcal{M}}| &< \sigma^2\left(x + \tfrac{1}{2}(1-x^2)\right) \\ &= \sigma^2\left(1 - \tfrac{1}{2}(1-x)^2\right) \le \sigma^2. \end{aligned}$$

The bound (S3.2) is never smaller than $1/\sqrt{2}$ times the largest step allowed by the condition $2\Delta t|\mathcal{M}| + (\Delta t)^2|\dot{\mathcal{M}}| < \sigma^2$, namely $\sigma^2/(|\mathcal{M}| + \sqrt{\mathcal{M}^2 + \sigma^2|\dot{\mathcal{M}}|})$, and it equals that step when $\dot{\mathcal{M}} = 0$. At the optimized parameters of all 19 per-colony fits (both kernel classes and both basins, with and without transport), the right-hand side of (S3.2) exceeded the substep Δt by a factor of at least 43 throughout the simulation.

Algorithm S15 Advance to Next Time Step

Input: Tangent-angle values $\{\theta_{\ell,k}^{(i)} : i \in [m], \ell \in [n]\} \subset \mathbb{R}$ at time t_k, Average speeds $\{\sigma_k^{(i)}\}_{i\in[m]} \subset \mathbb{R}$ at time t_k,
centroids $\{\boldsymbol{\mu}_k^{(i)}\}_{i\in[m]} \subset \mathbb{R}^2$ at time t_k, Interaction kernels $f, g : \mathbb{R}^+ \to \mathbb{R}^+$, Time scaling factor $\tau \in \mathbb{R}^+$,
Model parameters $c^{(1)}, \ldots, c^{(m)}, \beta \in \mathbb{R}^+$, $\alpha \in \mathbb{R}$ ($\alpha = 0$ for the standard model), Gate width $w > 0$,
Time step $\Delta t \in \mathbb{R}^+$, Current time t_k, Past time t_{k-1}
Past values of $\boldsymbol{\mathcal{K}}, \boldsymbol{\mathcal{F}}, \mathbf{a}$: $\{(\boldsymbol{\mathcal{K}}_{k-1}^{(i)}, \boldsymbol{\mathcal{F}}_{k-1}^{(i)}, \mathbf{a}_{k-1}^{(i)}), i \in [m]\}$

Output: Time $t_{k+1} \in \mathbb{R}$, Tangent-angle values $\{\theta_{\ell,k+1}^{(i)} : i \in [m], \ell \in [n]\} \subset \mathbb{R}$ at time t_{k+1},
Average speeds $\{\sigma_{k+1}^{(i)}\}_{i\in[m]} \subset \mathbb{R}$ at time t_{k+1}, centroids $\{\boldsymbol{\mu}_{k+1}^{(i)}\}_{i\in[m]} \subset \mathbb{R}^2$ at time t_{k+1}, contour points $\{\boldsymbol{\gamma}_{\ell,k+1}^{(i)} : i \in [m], \ell \in [n]\}$ at time t_{k+1}

1: Update forces (see Algorithms S6, S14, S12), $(i \in [m])$:

$$\boldsymbol{\mathcal{K}}_k^{(i)} \leftarrow \texttt{Derivative}(\theta_{1,k}^{(i)}, \ldots, \theta_{n,k}^{(i)}) \in \mathbb{R}^n, \qquad \boldsymbol{\gamma}_k^{(i)} \leftarrow \texttt{Contour}\left(\boldsymbol{\theta}_k^{(i)}, \sigma_k^{(i)}, \boldsymbol{\mu}_k^{(i)}\right)$$

$$\boldsymbol{\mathcal{F}}_k^{(i)} \leftarrow c^{(i)}\sigma_k^{(i)}\mathbf{1} - \texttt{NLForces}(\{\boldsymbol{\gamma}_k^{(i)}\}_{i\in[m]}, \{\sigma_k^{(i)}\}_{i\in[m]}, f, g, \tau) \in \mathbb{R}^n, \qquad \mathbf{a}_k^{(i)} \leftarrow \boldsymbol{\mathcal{K}}_k^{(i)} \odot \left(\boldsymbol{\mathcal{K}}_k^{(i)} - \boldsymbol{\mathcal{F}}_k^{(i)}\right)$$

2: If the transport mechanism is included, record the minimum distances between colonies from the pairwise distances computed in Algorithm S14, $(i, j \in [m]; i \neq j)$:

$$s_k^{(ij)} \leftarrow \min_{\ell,\ell' \in [n]} \left|\boldsymbol{\gamma}_{\ell,k}^{(i)} - \boldsymbol{\gamma}_{\ell',k}^{(j)}\right|$$

3: Compute mean of $\mathbf{a}$ and estimate its derivative over $[t_{k-1}, t_k]$, $(i \in [m])$

$$\mathcal{M}_k^{(i)} \leftarrow \frac{1}{n}\sum_{\ell=1}^{n} a_{\ell,k}^{(i)}, \qquad \partial_t \tilde{\mathcal{M}}_k^{(i)} \leftarrow \frac{1}{n}\sum_{\ell=1}^{n} \frac{a_{\ell,k}^{(i)} - a_{\ell,k-1}^{(i)}}{t_k - t_{k-1}}$$

4: Check the time step restriction (S3.2), which guarantees that every $\sigma_{k+1}^{(i)}$ is well defined and positive (a safeguard; with a fixed number of substeps per image it does not become active in practice, and Algorithm S1 stops with an error if it does):

$$\Delta t < \min_{i\in[m]} \frac{1}{2} \frac{\left(\sigma_k^{(i)}\right)^2}{\sqrt{\left(\mathcal{M}_k^{(i)}\right)^2 + \frac{1}{2}\left(\sigma_k^{(i)}\right)^2 \left|\partial_t \tilde{\mathcal{M}}_k^{(i)}\right|}}$$

5: Set next time: $t_{k+1} \leftarrow t_k + \Delta t$

6: Use linear extrapolation to estimate next values for $\mathbf{a}, \boldsymbol{\mathcal{F}}, \boldsymbol{\mathcal{K}}$, for $(i \in [m]; \ell \in [n])$:

$$\tilde{a}_{\ell,k+1}^{(i)} \leftarrow a_{\ell,k}^{(i)} + \Delta t \frac{a_{\ell,k}^{(i)} - a_{\ell,k-1}^{(i)}}{t_k - t_{k-1}}, \quad \tilde{\mathcal{K}}_{\ell,k+1}^{(i)} \leftarrow \mathcal{K}_{\ell,k}^{(i)} + \Delta t \frac{\mathcal{K}_{\ell,k}^{(i)} - \mathcal{K}_{\ell,k-1}^{(i)}}{t_k - t_{k-1}}, \quad \tilde{\mathcal{F}}_{\ell,k+1}^{(i)} \leftarrow \mathcal{F}_{\ell,k}^{(i)} + \Delta t \frac{\mathcal{F}_{\ell,k}^{(i)} - \mathcal{F}_{\ell,k-1}^{(i)}}{t_k - t_{k-1}}$$

7: Advance average speeds using 2nd-order Adams–Bashforth explicit method $(i \in [m])$:

$$\sigma_{k+1}^{(i)} \leftarrow \sqrt{\left(\sigma_k^{(i)}\right)^2 - 2(\Delta t)\left(\mathcal{M}_k^{(i)} + \tfrac{1}{2}\Delta t \cdot \partial_t \tilde{\mathcal{M}}_k^{(i)}\right)}$$

Algorithm S16 Advance to Next Time Step (continued from Algorithm S15)

8: Compute zero-Dirichlet primitives of a (see Algorithm S7), $(i \in [m])$:

$$\mathbf{p}_k^{(i)} \leftarrow \texttt{Primitive}(\mathbf{a}_k^{(i)}), \qquad \tilde{\mathbf{p}}_{k+1}^{(i)} \leftarrow \texttt{Primitive}(\tilde{\mathbf{a}}_{k+1}^{(i)})$$

9: Compute diffusion force at time t_k and estimate at time t_{k+1} (see Algorithm S6), $(i \in [m])$:

$$\mathbf{G}_k^{(i)} \leftarrow \mathbf{p}_k^{(i)} \odot \boldsymbol{\mathcal{K}}_k^{(i)} - \texttt{Derivative}\left(\mathcal{F}_{1,k}^{(i)}, \ldots, \mathcal{F}_{n,k}^{(i)}\right)$$

$$\tilde{\mathbf{G}}_{k+1}^{(i)} \leftarrow \tilde{\mathbf{p}}_{k+1}^{(i)} \odot \tilde{\boldsymbol{\mathcal{K}}}_{k+1}^{(i)} - \texttt{Derivative}\left(\tilde{\mathcal{F}}_{1,k+1}^{(i)}, \ldots, \tilde{\mathcal{F}}_{n,k+1}^{(i)}\right)$$

10: Advance tangent-angle function using pseudo-exponential integrator with effective time step (see Algorithm S10), $(i \in [m])$:

$$\omega^{(i)} \leftarrow \left\lceil \frac{\theta_{n,k}^{(i)} - \theta_{1,k}^{(i)}}{2\pi} \right\rceil, \qquad \eta_{\ell,k}^{(i)} \leftarrow \theta_{\ell,k}^{(i)} - \omega^{(i)} \left(-\pi + \tfrac{2\pi}{n}(\ell - 1)\right), \qquad \ell \in [n]$$

$$\Delta t_{\text{eff}}^{(i)} = \Delta t \cdot \frac{1}{2}\left(\frac{1}{\left(\sigma_{k+1}^{(i)}\right)^2} + \frac{1}{\left(\sigma_k^{(i)}\right)^2}\right), \qquad \boldsymbol{\eta}_{k+1}^{(i)} \leftarrow \texttt{ForcedDiffusionTimeStep}\left(\boldsymbol{\eta}_k^{(i)}, \Delta t_{\text{eff}}^{(i)}, \mathbf{G}_k^{(i)}, \tilde{\mathbf{G}}_{k+1}^{(i)}\right)$$

$$\theta_{\ell,k+1}^{(i)} \leftarrow \eta_{\ell,k+1}^{(i)} + \omega^{(i)} \left(-\pi + \tfrac{2\pi}{n}(\ell - 1)\right), \qquad \ell \in [n]$$

Algorithm S17 Advance to Next Time Step (continued from Algorithm S15)

11: Advance centroids using the trapezoidal rule, $(i \in [m])$:

$$\mathbf{v}_k^{(i)} \leftarrow \frac{1}{n}\sum_{\ell=1}^{n}\left(p_{\ell,k}^{(i)}\begin{bmatrix}-\sin(\theta_{\ell,k}^{(i)})\\ \cos(\theta_{\ell,k}^{(i)})\end{bmatrix} + \mathcal{F}_{\ell,k}^{(i)}\begin{bmatrix}\cos(\theta_{\ell,k}^{(i)})\\ \sin(\theta_{\ell,k}^{(i)})\end{bmatrix}\right)$$

$$\tilde{\mathbf{v}}_{k+1}^{(i)} \leftarrow \frac{1}{n}\sum_{\ell=1}^{n}\left(\tilde{p}_{\ell,k+1}^{(i)}\begin{bmatrix}-\sin(\theta_{\ell,k+1}^{(i)})\\ \cos(\theta_{\ell,k+1}^{(i)})\end{bmatrix} + \tilde{\mathcal{F}}_{\ell,k+1}^{(i)}\begin{bmatrix}\cos(\theta_{\ell,k+1}^{(i)})\\ \sin(\theta_{\ell,k+1}^{(i)})\end{bmatrix}\right)$$

$$\hat{\boldsymbol{\mu}}_{k+1}^{(i)} \leftarrow \boldsymbol{\mu}_k^{(i)} + \Delta t \cdot \frac{1}{2}\left(\frac{\mathbf{v}_k^{(i)}}{\sigma_k^{(i)}} + \frac{\tilde{\mathbf{v}}_{k+1}^{(i)}}{\sigma_{k+1}^{(i)}}\right)$$

12: Recenter the tangent-angle function (Algorithm S13) and reconstruct the contour points (Algorithm S12), $(i \in [m])$:

$$\boldsymbol{\theta}_{k+1}^{(i)} \leftarrow \texttt{Recenter}\left(\boldsymbol{\theta}_{k+1}^{(i)}\right), \qquad \boldsymbol{\gamma}_{k+1}^{(i)} \leftarrow \texttt{Contour}\left(\boldsymbol{\theta}_{k+1}^{(i)}, \sigma_{k+1}^{(i)}, \hat{\boldsymbol{\mu}}_{k+1}^{(i)}\right)$$

13: Apply the nonlocal transport mechanism as a rigid translation of each colony (explicit Euler step; skip if $\alpha = 0$), $(i \in [m])$:

$$\mathbf{d}^{(i)} \leftarrow \Delta t \cdot \alpha \sum_{j \neq i} \frac{\hat{\boldsymbol{\mu}}_{k+1}^{(i)} - \hat{\boldsymbol{\mu}}_{k+1}^{(j)}}{\left|\hat{\boldsymbol{\mu}}_{k+1}^{(i)} - \hat{\boldsymbol{\mu}}_{k+1}^{(j)}\right|} \chi_w\left(s_k^{(ij)} - \beta\right), \quad \boldsymbol{\mu}_{k+1}^{(i)} \leftarrow \hat{\boldsymbol{\mu}}_{k+1}^{(i)} + \mathbf{d}^{(i)}, \quad \boldsymbol{\gamma}_{\ell,k+1}^{(i)} \leftarrow \boldsymbol{\gamma}_{\ell,k+1}^{(i)} + \mathbf{d}^{(i)}, \quad \ell \in [n]$$