`modelRxiv`: A platform for the distribution, computation and interactive display of models

Keith D. Harris[1], Guy Hadari[1], and Gili Greenbaum[1]

[1] *Department of Ecology, Evolution and Behavior, The Hebrew University of Jerusalem, Jerusalem, Israel*

February 17, 2022

Abstract

Modeling the dynamics of biological processes is ubiquitous across the ecological and evolutionary disciplines. However, the increasing complexity of these models poses a significant challenge to the dissemination of model-derived results. With the existing standards of scientific publishing, most often only a small subset of model results are generated, presented in static figures or tables, and made available to the scientific community. Further exploration of the parameter space of a model, investigation of possible variations of a model, and validation of the results in relation to model assumptions commonly rely on local deployment of code supplied by the authors. While releasing model code is a publication requirement for most scientific journals, there are currently no standardized protocols or coding-language requirements. Deploying models locally poses a technical challenge due to the specific framework and environment in which a model was developed, and can preclude model validation and exploration by readers and reviewers. To address this issue, we developed a platform that serves as an interactive repository of biological models, called `modelRxiv` (`https://modelrxiv.org`). The platform provides a unified interface for the analysis of models developed in multiple programming languages but does not require any technical understanding of the model implementation. To reduce model computation time, the platform allows users to pool computational resources available to them, including lab workstations, clusters and cloud services. By making published models accessible, this platform promises to significantly improve the accessibility, reproducibility and validation of ecological and evolutionary models.

1 Introduction

Modeling the dynamics of ecological and evolutionary processes, as well as of other biological processes, has been key to the development of ecology and evolution for more than a century [1]. Over the past few decades, the availability of computational power has paved the way for models of increasing complexity, involving numeric, stochastic, and individual-based modeling features, and exploration of multiple parameters spanning over large parameter spaces [2]. Detailed and high-resolution investigation of models can offer important insights on general phenomena as well as on specific systems; however, traditional scientific publishing standards—static papers depicting several model results representing select parameter values—are limited in demonstrating the full scope of the model results and their implications. This affects both the dissemination of potential insights from a model and the ability to assess the robustness of model results during the scientific peer-review process.

Currently, models in ecology and evolution, and across the biological disciplines, are implemented in a number of different programming languages, and there is no standard framework for publishing model code. Consequently, deploying models locally by a reviewer or a reader usually requires prior knowledge of the specific coding language or framework in which the model code was written, the installation of dependencies, and learning how to operate the model code and visualize results. This technical hurdle can make published models essentially inaccessible, where only the authors of the model are able to reproduce its results.

Several platforms have been developed to address some of these difficulties. These approaches can be categorized into two broad classes: (i) a 'model-centric' approach, where the emphasis is on model analysis

and visualization, and users are not required to interact with the underlying model code, and (ii) a 'code-centric' approach, where users interact with model code in order to analyze the model. Model-centric platforms provide a single programming language for modeling (e.g., *NetLogo* [3], *Mathematica* [4]), while code-centric platforms accommodate the deployment of multiple programming languages in a single interface (e.g., *Jupyter* [5], *CodeOcean* [6]). In both classes, models are accessed through a unified interface, reducing the investment needed by reviewers or readers to interact with the model. However, most published models are implemented in different programming languages, and translating model code so that it is compatible with a model-centric platform can be impractical. On the other hand, code-centric solutions that support multiple programming languages are focused on model implementation, and lack features that can make model analysis straightforward and accessible.

To bridge the gaps between these two approaches, we developed an interactive repository of biological models called `modelRxiv` (`https://modelrxiv.org`). Our browser-based platform provides a solution to the technical challenges involved in analyzing published models, and is designed to provide a model-centric solution while supporting multiple programming languages and frameworks. In addition, to support the increasing amount of computational resources needed to analyze models, the platform allows distribution of model computation across multiple deployment environments. `modelRxiv` is designed with a simple user interface that does not require a technical understanding of programming or modeling to operate. We envision this platform as a tool for model validation and exploration by reviewers and readers, and as a repository that can make published and unpublished models accessible to the scientific community.

2 Platform

`modelRxiv` is a browser-based application, with model visualization occurring in the browser, and model computation occurring in the browser or in additional environments (see section 2.3). The `modelRxiv` user interface has three main screens: (i) an index of models, which can be filtered and sorted according to model metadata such as title, authors, model categories, and keywords (Fig. 1); (ii) model analysis pages, which include model visualization, parameter manipulation and model analysis features (Figs. 2–4); (iii) a model builder page for developing and uploading models. Models can be either 'public', viewable by everyone, or in 'sandbox mode' where only the owner can see and interact with the model. Public models can be viewed and analyzed without logging in or creating a user, but registration is necessary to upload models and analyze models in sandbox mode (registration is free and anonymous). In this section we describe the main features of `modelRxiv`, which can be accessed through the model analysis page or the model builder page.

2.1 Visualization of model dynamics

To demonstrate the visualization options available on `modelRxiv`, we implemented and analyzed a well-known set of models in ecology [7]. The original manuscript [7] included four different approaches to modeling predator-prey ('hawks' and 'doves') competition: (i) a dynamical system with infinite and continuous population sizes and no spatial component, (ii) a reaction-diffusion system with continuous space, (iii) a random patch model with discrete individuals and no spatial structure, and (iv) an interacting particle system with discrete individuals and spatial structure. The paper aimed to demonstrate the importance of modeling assumptions of spatialness and discreteness in ecological modeling [7]. We implemented (i), (iii), and (iv) in *JavaScript* and uploaded them to `modelRxiv` as a single model.

In the model analysis page, model dynamics can be produced "on-the-fly", where visualization of the model occurs in parallel to the computation. This can be done when the model provides a recurrence function that returns the step $t+1$ of the model given step t, and when the model uses a framework that is browser compatible (Table 3). This type of visualization allows users to pause and restart model computation while the model is running, as dynamics are generated. Alternatively, models can return the dynamics of the model upon completion. The platform can display any parameters returned by the model functions (see *Methods*), according to the definition of the parameters by the model owner (see section 2.5 for an explanation on how parameters are defined).

For the hawk-dove models we implemented, the visualization includes four panels, three depicting the average population size of hawks and doves in the three models, and one depicting the state of the interacting particle system model in a 2-dimensional grid (Fig. 2). To the right of the four panels is a menu for

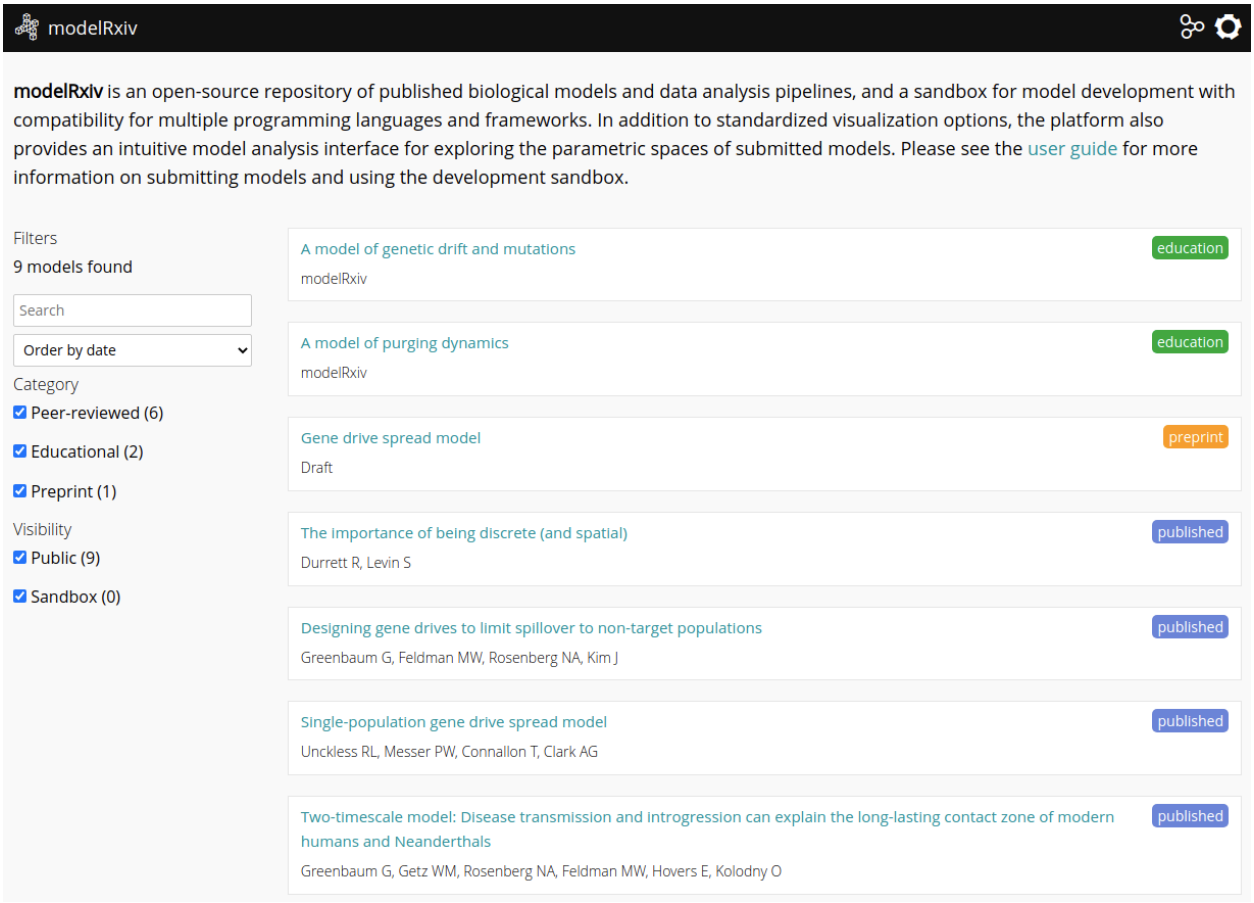

Figure 1: **'Index of models' page of `modelRxiv`, with list of models belonging to different categories**. On this page users can search and sort models, and select a model to view its 'model analysis page'. When authenticated, the index will list models that are publicly available, as well as private models uploaded by the user.

manipulating the model parameters (box on the far right of Figure 2). This menu offers two options, either changing the model parameters manually (here we display only a, b, c and d, which are the parameters determining the predator-prey dynamics), or choosing parameter presets (discussed in the 4 section below).

2.2 Analysis of models

Visualizing dynamics with fixed parameter values, such as in Figure 2, can be informative for understanding the dynamics in a given scenario, but often there is a need to explore the behavior of models over a large part of the parameter space. We added a feature in `modelRxiv` to support such investigations, which visualizes model outputs for a range of input parameters in the form of a grid. We refer to these plots as 'output summary plots', as they visualize the outputs of multiple model runs. For the above hawk-dove model, we define an output parameter, termed the 'outcome', as the agreement or disagreement between the dynamical system and the random patches or interacting particle system. To demonstrate the generation of an output summary plot for this model, we define four possible values of the outcome parameter in terms of potential predator-prey co-existence (Fig. 3): all approaches agree (in blue), the interacting particle system is different and the other two approaches agree (in green), discrete and infinite population approaches differ (in cyan) and all approaches disagree (in red); these modeling approach-agreement outcomes were the focus of the original manuscript ([7]). The values and colors of the outcomes can be defined in the model builder page (see section 2.5). With these outcomes defined by the model owner, users can select a range of parameter values for investigation for two parameters, and set the values of the remaining parameters. By selecting the 'Grid' button, an output summary plot of the selected parameter space is computed and displayed (Fig. 3). These

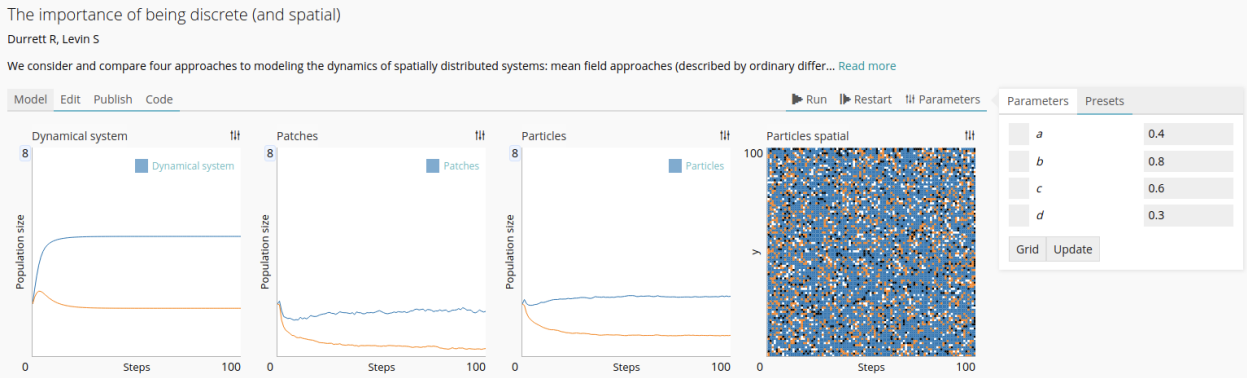

Figure 2: **The 'model analysis page' of a hawk-dove model, demonstrating different display types and parameter manipulation**. This page shows the basic display of a model while running. On the left are four panels that show the dynamics in three different approaches to the hawk-dove model (the fourth panel shows the spatial structure of the interacting particle system). These dynamics relate to the parameters selected in the parameter panel on the right.

summary plots are an important aspect of model analysis, as they allow more comprehensive exploration of the parameter space of the model compared to running dynamics for specific parameters.

2.3 Distribution of computation

For many modern models, a comprehensive analysis requires large computational resources that necessitate distribution of computation through multi-threading. However, multi-threading can present technical difficulties that prevent many readers and reviewers from attempting to investigate models. To address this difficulty, we integrated simple and user-friendly features for distribution of computation directly in `modelRxiv`. To achieve this, we use a feature for multi-threading available in most browsers (see *Methods*). This allows `modelRxiv` to distribute grid computation across multiple threads of the local machine as well as to pool computation from several connected machines. This can be useful, for example, when users wish to generate high-resolution output summary plots (e.g. Fig. 3), and requires little-to-no technical knowledge from the user.

Terms and definitions

To illustrate the meaning of the terms with respect to a specific model, we refer below to the model of gene drive spread described in section 2.4.

Model input parameters. A set of parameters that can be manipulated by the user, that are received by the model functions. Input parameters have default values that are defined by the model owner. In the gene drive spread model, an input parameter could be the migration rate between populations, or the initial frequency of the gene drive allele.

Model dynamics. A set of parameters returned for each step of the model that can be visualized while the model is running or when it has completed. A dynamics parameter in the gene drive spread model could track the frequency of the gene drive allele over time in each population.

Model output. A set of parameters returned for each model run. An output parameter in the gene drive spread model could be the deployment outcome, determined in the model code according to the frequency of the gene drive allele in each population in the final model step.

Dynamics plot. A plot that is produced either while the model is running, or after it has completed, using one or more dynamics parameters. The dynamics plots generated for model dynamics depend on the definition of dynamics parameters by the model owner (e.g. continuous or discrete).

Model output summary plot. A plot that is produced when analyzing a model using ranges of input parameters. The type of plot will depend on the input parameters selected and their definition by the model owner.

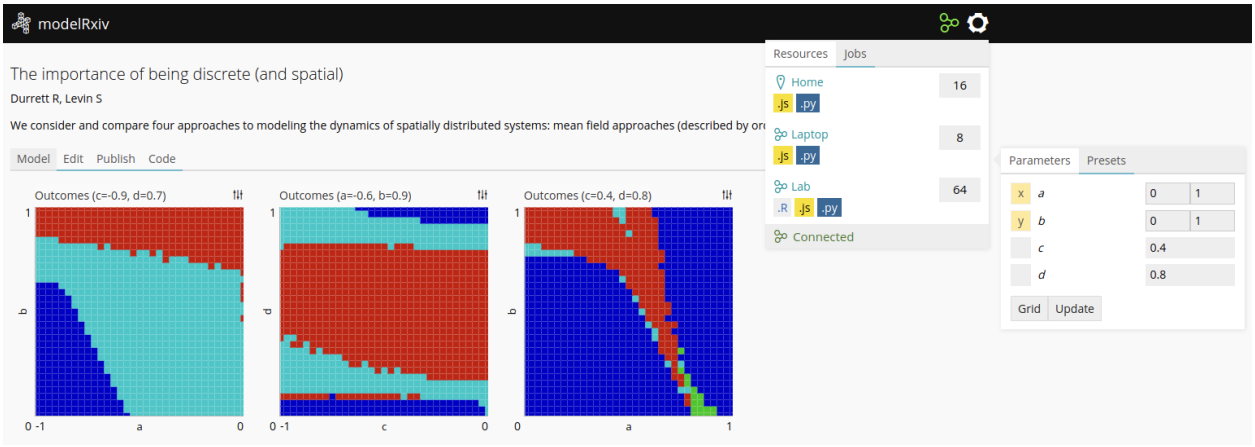

Figure 3: **An 'output summary plot' for the hawk-dove models**. The three panels on the left show output summary plots produced for different regions of the model parameter space. Colors in these panels relate to outcomes of the model for the specified input parameters: all approaches agree (blue), interacting particle system is different and the non-spatial approaches agree (green), discrete and infinite population approaches differ (cyan), and all approaches disagree (red). The parameter menu on the right has two parameters selected with a range of values (a and b in the range $[0, 1]$, c and d with fixed values); clicking the 'Grid' button produces the plots on the left. In this example, the intensive computation needed to generate the summary plots was conducted by distributed computation on the three different machines that appear in the resources menu on the top right, denoted by labels: a local machine connected via the browser (labeled 'Home'), a remote machine connected via the browser (labeled 'Laptop') and a lab machine connected via a back-end utility (labeled 'Lab'). For each machine, a list of supported languages is also given: the two machines connected via the browser ('Home' and 'Laptop') can process *JavaScript* ('.js' label) and *Python* ('.py' label) code; the machine connected via the back-end utility ('Lab') can also process *R* files. The resource menu also lists the number of computation threads that can be used for each machine. To control computation distribution, the computation threads can be adjusted by changing the number in the boxes.

In the multi-threading feature, individual machines connect to a messaging server that is part of the `modelRxiv` platform (see *Methods*), and batches are distributed to other machines owned by the same user, as sets of parameters. When the batches are processed, the results are returned through the messaging server. The process of generating grids using the browser-based interface using multiple machines is identical, but can reduce the computation time of the grid (e.g., the panels in Figure 3 were distributed across three machines). Thread usage can be managed via the resources tab, by changing the number of utilized threads on each resources (Figure 3, top right corner).

Machines can be connected to `modelRxiv` in two ways. One option is logging into the browser-based interface on `modelRxiv` with the same user account from different machines. After clicking the 'Connect' button in the resources menu, resources from the connected machines will be pooled. This option allows straightforward replication of the browser environment without requiring the installation of software on the connected machines. To pool resources through other environments, it is possible to connect to `modelRxiv` via a back-end utility that is available as part of the platform code. This allows users to compute models using the local environment on the connected machine, so that frameworks installed on this machine can be used. This offers much more flexibility than the browser environment in terms of support for programming languages and frameworks (for details on deployment to multiple environments, see *Methods*).

2.4 Reproducing figures using parameter presets

To maintain continuity between a manuscript and an equivalent model on `modelRxiv`, the platform allows the definition of 'parameter presets' that appear alongside the model as part of the parameter panel (Fig. 4). Presets are an important aspect of making models accessible, as they direct users to specific sets of parameters that are of interest, or that have been referred to in the manuscript. For reviewers, presets allow the reviewer to reproduce dynamics that relate to specific figures or results, and to manipulate other parameters to test the robustness of the results described by the authors.

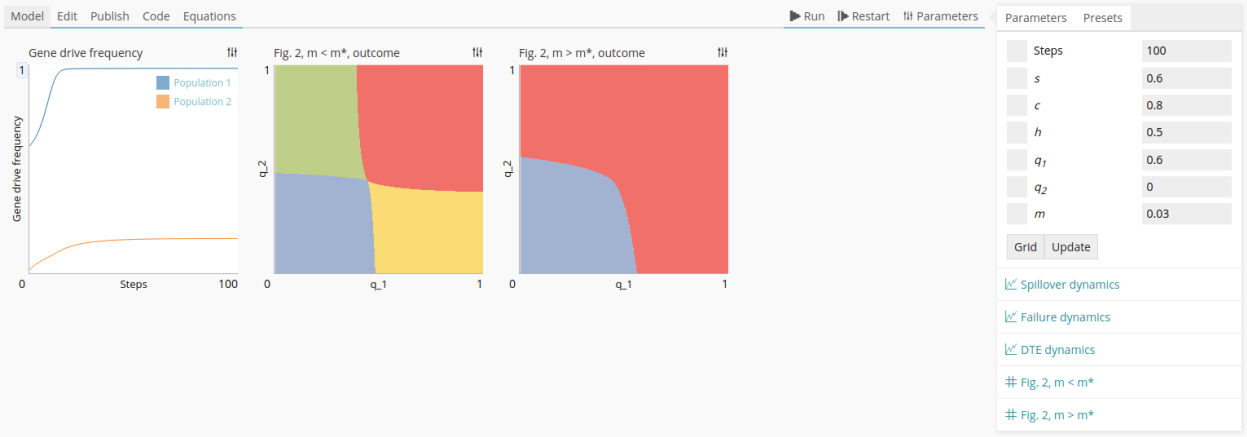

Figure 4: **Using 'parameter presets' to recreate figures from published papers**. We define a 'deployment outcome' as a model output, and assign colors for each possible outcome (colors here are the same as in [8]). The presets appear on the right below the model parameters (e.g., 'spillover dynamics', 'failure dynamics', etc.). On the left we see the dynamics of one outcome of the model, 'differential targeting', and in the second and third panel we reproduce the result of two panels from Figure 2 in the original manuscript.

To demonstrate this feature, we implemented an eco-evolutionary model that tracks gene drive (a genetic element that violates Mendelian inheritance) spread in a two-population system [8]. We chose this model because it combines evolutionary (the spread of the gene drive) and ecological (migration) elements. The model receives as input the gene drive design in the form of three parameters (s, the fitness cost of the gene drive allele; h the dominance of the gene drive allele; c the conversion rate from heterozygotes to gene drive homozygotes), and migration rate between the two populations (m). The model tracks the frequency of the gene drive allele in the two populations (q_1 and q_2). The model returns as an output parameter the eventual 'deployment outcome' of the gene drive (Fig. 4): loss of the gene drive allele in both populations ($q_1 = q_2 = 0$, in blue), fixation in both populations ($q_1 = q_2 = 1$, in red), or 'differential targeting', where the allele has a high frequency in one population and a low frequency in the other ($q_1 > q_2$ in yellow and $q_2 > q_1$ in green). These outcomes allow investigation of how different gene drive designs are expected to generate different deployment outcomes, in relation to the connectivity between the two populations.

The paper we modeled [8] describes a partial exploration of the four-parameter space (s, h, c, and m), with different figures describing different features of the dynamics. To enable reviewers and readers to navigate the connection between the paper and the model implemented in `modelRxiv`, we defined presents that reproduce results in the paper. For instance, we can define parameters that will display the model dynamics for different deployment outcomes, as a means of illustrating how the dynamics of a particular outcome are characterized. In addition, we reproduce the parameters of two panels from one of the figures (in Figure 4, these appear as "Fig. 2"). In the model analysis page, the first panel shows the dynamics of each population for the differential targeting outcome ('DTE'), while the second and third panel demonstrate the relationship between the initial frequencies of the gene drive and the deployment outcome, for different migration rates. By clicking on the presets, users generate the output summary plots from the paper; they can also modify the parameters and run custom plots derived from the original analyses. To connect dynamics visualization with grid display, users can click on any point in the grid to run the model for that specific parameter set.

2.5 Uploading models

Uploaded models can be written in multiple programming languages. This is determined by the environments available to the user running the model (see *Methods*), and language-specific wrappers that communicate between the platform and the model code. For the initial release of `modelRxiv` we provide three wrappers for *JavaScript*, *Python* and *R*. Otherwise, the upload process is independent of the language used by the model.

To streamline the process of uploading models to `modelRxiv`, we developed an upload interface that includes testing and output validation. The upload form is accessible to authenticated users by clicking the 'gear' icon at the top right corner of the screen and selecting 'Upload model'. The interface is designed as an analog to interfaces for submitting manuscripts for peer review. The upload form is designed to ensure that models operate as expected before deployment. The process is separated into different parts of the upload form: (i) adding metadata associated with the model, such as the title and author list; (ii) adding model code and testing the code for syntax errors or integration issues; (iii) defining the types of inputs and outputs of the model. `modelRxiv` will attempt to identify the inputs and outputs based on the model code; subsequently, users can label these parameters manually. As an alternative to providing model code in a specific programming language, users can generate code from equations that describe the model using a simplified model builder (see the 2.6 section).

Upon uploading (by clicking the 'Submit' button at the end of the upload form), models are visible and accessible only to the account owner, and appear as 'sandbox' models. This gives users the ability to ensure that the model operates correctly before submitting it to the public `modelRxiv` repository. Uploaded models can be edited using the 'Edit' button on the model page. Model code and parameters are versioned, so that previous versions can be browsed and restored if necessary. To submit the model to the public repository, users click the 'Publish' button on the model analysis page. This will transfer the model to a moderator who will review the model manually to ensure it does not contain malicious code, and subsequently the model will be publicly available on `modelRxiv`.

In the model analysis page, plots will be generated automatically based on the type of the output parameters. For example, continuous variables returned by the step function will be translated into a line plot, with the model steps as the x-axis, and the value of the variable as the y-axis, whereas an array of two-dimensional vectors would be represented as a scatter plot that is updated at every step. It is also possible to define units for output parameters; parameters with the same units and domain can be grouped into the same plot (e.g., multiple line plots can be combined into a single plot with different line colors, as in the first panel of Figure 4).

2.6 Simplified model builder for mathematical models

In some cases, models can be explicitly described as a set of mathematical equations. To facilitate adding such "computationally simple" models to `modelRxiv`, we developed a 'simplified model builder' for building models by providing equations in a *Python*-like pseudocode. This utility is geared towards models that can be defined as a step function (e.g., discrete-time models), but it could be extended in the future to other types of mathematical formulations.

The main component of the simplified model builder represents a single iteration of the model dynamics, such as a single time step (Fig. 6). First, the user defines the parameters of the model and their default values. Currently, any variable whose definition does not include other parameters will be interpreted as an input parameter. Next, the user can define a step function as a series of equations, which can include any parameter that has already been defined (e.g., Fig. 6). Finally, the user defines the parameters that are computed at each step of the function. Note that the value of an input parameter that is also returned by the step function will be overridden by its new value; for example, the input parameter 'q1' in Fig. 6 has a default value defined on line 5; this will be the initial value of q_1, and this value will be updated at each step by the equation on line 22. After building the model code, users can define the type and domain of input and output parameters in the form sections below the model builder (Fig. 5).

3 Discussion

We present and describe the features of the `modelRxiv` platform, an interactive repository of models. The purpose of the platform is to facilitate reviewers and readers in evaluating and investigating models, and to increase the accessibility of published models in ecology and evolution. In addition, the platform can be used as a sandbox for designing and developing models. The platform is designed as a unified web-based interface for working with models, while having the flexibility of running models written using multiple programming languages, and allowing distribution of model computation across multiple environments. In order for the platform to be successful in improving model accessibility and review processes, it would need

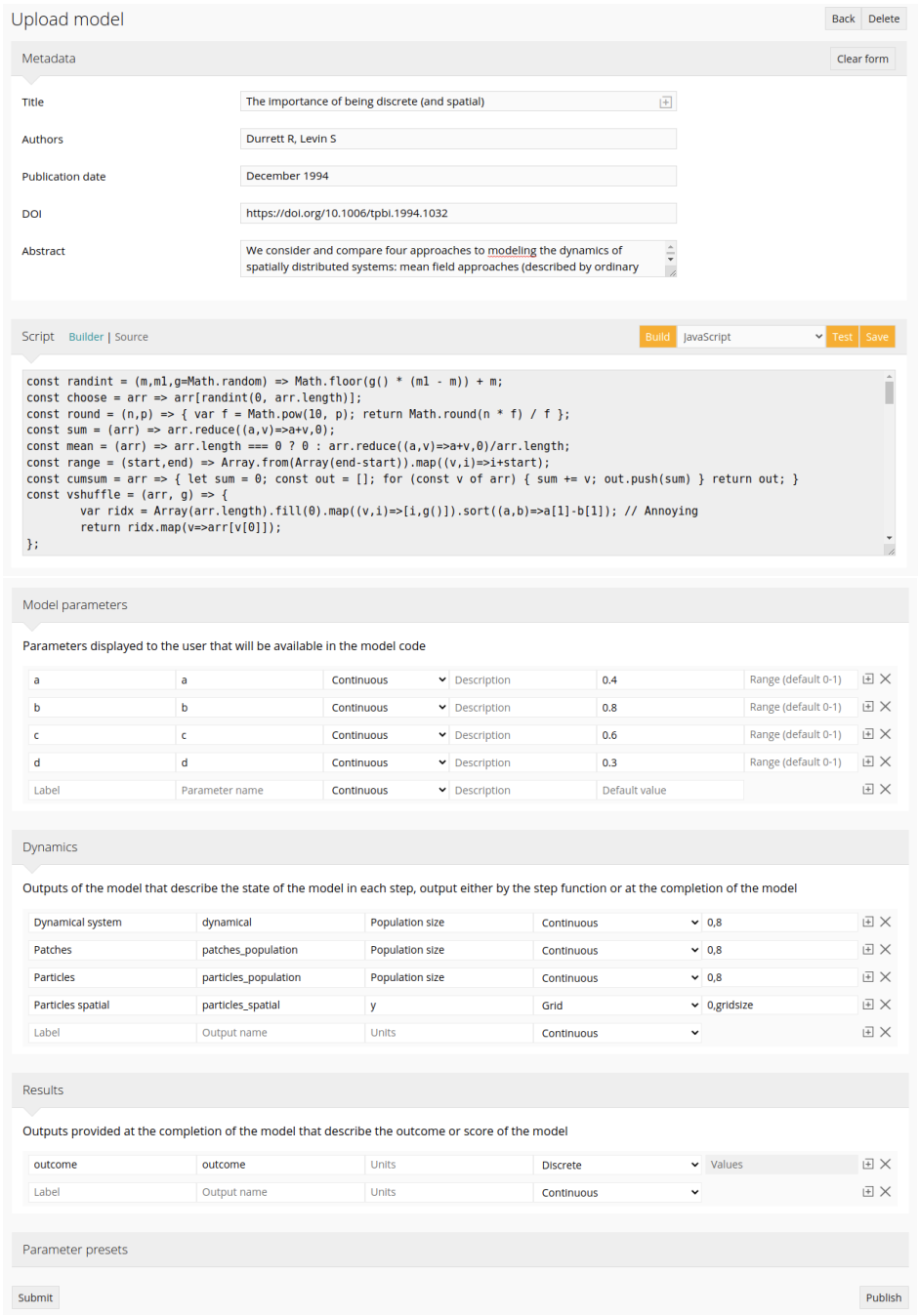

Figure 5: **Interface for uploading models**. The first section contains the model metadata, the 'script' section contains the model code or equations, and the 'model parameters' section defines the types and ranges of input and output parameters of the model. Metadata can be automatically filled out by searching *PubMed* via the search button in the title field, while input and output parameter fields are automatically filled out after successfully testing the model code.

to be widely adopted by the members of the scientific ecological and evolutionary modeling community. We have, therefore, designed this version of `modelRxiv` with the features that we believe will be useful for experienced modelers and newcomers alike. Future additions and extensions of features will be guided by feedback from the ecological and evolutionary modeling community. To allow users to provide feedback on bugs and issues using `modelRxiv`, we have opened a *Slack* workspace that can be joined using the invitation

Model Code Equations ▶ Run Parameters

```
## Defaults
s = 0.6
c = 0.8
h = 0.5
q1 = 0.5
q2 = 0
m = 0.01

## Selection coefficients for heterozygotes
sn = 0.5 * (1 - c) * (1 - h * s)
sc = c * (1 - s)

## Migration
qm1 = q1 * (1 - m) + q2 * m
qm2 = q2 * (1 - m) + q1 * m

## Average fitness
w1 = qm1**2 * (1 - s) + 2 * qm1 * (1 - qm1) * (2 * sn + sc) + (1 - qm1)**2
w2 = qm2**2 * (1 - s) + 2 * qm2 * (1 - qm2) * (2 * sn + sc) + (1 - qm2)**2

## Selection
q1 = (qm1**2 * (1 - s) + 2 * qm1 * (1 - qm1) * (sn + sc)) / w1
q2 = (qm2**2 * (1 - s) + 2 * qm2 * (1 - qm2) * (sn + sc)) / w2

return q1, q2
```

Figure 6: **Pseudocode that will be interpreted as a model step function for gene drive spread model**. *Python*-style comments denote different parts of the model: (1) default input parameter values; (2) equations composing the step function; and (3) a return statement that defines the outputs of each step.

link on the 'contribute' page (`https://modelrxiv.org/contribute`).

With development of new platforms such as `modelRxiv`, it is important to evaluate and compare the suit of suggested features to existing platforms to better understand its potential contribution. There are a number of established comparable platforms that provide an interface for manipulating model parameters and visualizing dynamics or results, mainly *Mathematica*, *MATLAB*, *Jupyter* and *NetLogo*. In `modelRxiv`, we have aimed to develop a combination of the features of these platforms that will be most useful for making a wide range of published and unpublished models accessible, summarized in Table 1. On the one hand are 'model-centric' platforms such as *NetLogo* [3], *Numerus* [9], *MATLAB* [10], or *Mathematica* [4]. These require that models be written in a programming language developed specifically for the platform. In principle, such platforms can offer a coherent user experience, where the end-user is able to access the same options as the developer of the model. However, in practice, most models are still developed in different languages and frameworks, mainly because different users are familiar with different coding languages. In addition, only some of these platforms have a web-based implementation. An alternative is 'code-centric' platforms that support execution of multiple programming languages, such as *Jupyter* [5]. *Jupyter* aims to provide a unified wrapper for code that serves both as a development sandbox and as an interactive interface for deploying code. However, because *Jupyter* is programming-oriented, it emphasizes the implementation of the model rather than its logic and introduces technical hurdles that need to be overcome in order to analyze the model. Thus, a platform that is designed specifically for modeling, but also has the flexibility of supporting multiple programming languages and frameworks, has the potential to provide a coherent, accessible solution for many different modeling approaches.

`modelRxiv` also aims to implement features of the repository *EBI BioModels* [11, 12], which provides a browser-based interface for accessing published mathematical models. *EBI BioModels* is designed mostly for biochemical models, and lacks the ability to manipulate models from the repository interface. In designing `modelRxiv`, we implement repository elements similar to those on *EBI BioModels*, and combine these with features specifically suited to analyzing modern eco-evolutionary models, such as parameter manipulation and distribution of computation.

`modelRxiv` offers both the flexibility of deployment in multiple programming languages, and the accessi-

Platform	Approach	Sandbox or repository	Multiple frameworks	No setup required	Distributed computation
modelRxiv	Model-centric	Both	+	+	+
NetLogo [3]	Model-centric	Both	-	NetLogo Web	-
Mathematica [4]	Model-centric	Sandbox	-	-	-
Numerus [9]	Model-centric	Sandbox	-	Export to Web	-
EBI BioModels [11]	Model-centric	Repository	+	+	N/A
Jupyter [5]	Code-centric	Sandbox	+	+	+
CodeOcean [6]	Code-centric	Sandbox	+	+	+

Table 1: **Features of `modelRxiv` compared to other platforms for modeling**. Plus signs indicate that a feature is an integral part of a platform, while minus signs indicate that the feature is not available. Where the feature can be made available with additional software or through special features, the name of the software or feature is indicated. For *EBI BioModels*, which does not serve as a sandbox, distributed computation is irrelevant.

bility of a user interface that requires no technical understanding of the underlying code. By using presets, the model owner can guide users through stages of exploring the parametric space of the model. This can lead to a more intuitive understanding of the model than a static figure, as the user is free to manipulate the model parameters and observe the effect on the model result, or to visualize model dynamics for different regions within a certain figure.

3.1 Facilitating evaluation of models during the review process

One of the most important potential applications of `modelRxiv`, in our opinion, is to facilitate and improve the review processes of modeling studies in ecology and evolution. During the review of modeling studies reviewers should, ideally, evaluate the correctness, robustness, reproducibility, and applicability of models. However, in practice, even when the model code is attached to the submission, there are technical difficulties including setup, installation of dependencies, compatibility, and acquaintance with the specific coding language chosen by the author that makes this time-consuming and unfeasible in many cases. Consequently, models are rarely reproduced by reviewers during the review process, and are even more rarely subjected to manipulation and thorough investigation beyond the specific parameter values chosen by the author.

Improving model evaluation during the review process requires development of user-friendly and accessible tools for reviewers. The understanding that such tools are vital to ensure code validation has led to the adoption of services for deploying data processing code to computational containers by reviewers, such as *CodeOcean* [6, 13]. *CodeOcean* provides an interface through which reviewers can manipulate and deploy code associated with a manuscript during the review process. By removing the technical hurdles of code deployment, code validation can become an integral part of the review process without resulting in a significant additional burden on reviewers.

`modelRxiv` aims to provide a tool for model evaluation, deployment, and manipulation for eco-evolutionary models. By providing a unified user-interface for models written in different programming languages, `modelRxiv` resolves the technical challenge of understanding and deploying model code. With models uploaded to `modelRxiv`, reviewers have access to the full parameter space of the model. Using the presets feature, reviewers could easily generate figures from the manuscript, and assess the robustness of the model in terms of the parameters used to generate these figures.

Making model code validation an integral part of the review process would improve not only confidence in the model results, but also encourage more thorough exploration of the model parameter space by the authors. It could also shorten the review process by allowing reviewers to answer questions regarding model parameters without having to rely on the authors to produce additional figures. As `modelRxiv` is a public repository, using it in the review process has the important benefit of ensuring that the model would be accessible to readers of the manuscript after it has been published.

3.2 Public accessibility to model results

Beyond facilitating the review process, we believe that `modelRxiv` could promote and improve the exploration of published models in the ecological and evolutionary disciplines. First, the ease of accessibility to published models would incentivize researchers to expand existing models and utilize existing modeling frameworks, thereby making model development faster. This would also improve the comparability between published models, ensuring that conclusions pertaining to differences between model results can be coherently attributed to changes in key assumptions, rather than to model design or coding. Second, encouraging the scientific community to participate in manipulation of model parameters, in a deeper examination of model parameter spaces, and in adjustment of model assumptions through simple alteration of underlying code, could generate new insights for existing models. Such inquiries and modifications could lead to the identification of novel model behaviors with biological significance, perhaps undetected due to a different focus of the original study. These investigations could also lead to a deeper understanding of the model behaviors, particularly in terms of the boundaries in which the described behaviors of the models no longer hold, and a discussion on the biological significance of these boundaries. Therefore, adoption of `modelRxiv` by the eco-evolutionary modeling community could encourage collaborations between researchers to extend and elaborate on modeling studies, for example between the publishers of the original modeling study and the researchers identifying interesting behaviors in their models.

3.3 Educational uses

`modelRxiv` also serves as an educational tool. With models that demonstrate basic principles in ecology and evolution, students can visualize pre-prepared dynamics and manipulate model parameters to gain intuition on the phenomenon in question, and they can test hypotheses regarding the relationship between model parameters. At a more advanced level, students can explore the code implementation of the model, to generate similar alternative models and to gain experience in model coding and design. The fact that the user is not exposed to the full complexity of the model from the very beginning is an important aspect when encouraging those not familiar with the model or its implementation to engage in exploration of the model. Therefore, the clear separation of model visualization and manipulation from the underlying code would be helpful in teaching environments, where it is necessary to account for variable technical abilities of students to provide individualized and flatter learning curves for students.

4 Methods

Here we provide the technical details of model integration with the platform, and considerations relating to the current implementation of the features mentioned in the *Platform* section. These details are provided as an explanatory layer for the platform code, and to make integration with and extension of the platform accessible to new users and developers.

4.1 Elements of model code

`modelRxiv` acts as a wrapper for existing models. Integration with models relies on the definition of common inputs and outputs of a set of functions that constitute the model (Table 2). These inputs and outputs include: (i) *parameters*, an associative array of model parameters; and (ii) *result*, an associative array of model result variables.

The following functions must be implemented by the model code: (i) *defaults*, which returns the default *parameters* of the model; (ii) *run*, which returns a model result for a set of *parameters*. For step-wise models that can be run in the browser, the model code should also implement the following functions: (iii) *step*, which returns step $t + 1$ given step t and a set of *parameters*, and (iv) *result*, which returns a model result for a series of steps and a set of *parameters*.

This structure reflects different parts of the model logic. Because `modelRxiv` provides standardized visualization based on the definition of model result variables, the model code should include only the model logic, without any plotting functions. In addition, as `modelRxiv` can also define parameter grids for model analysis, such analyses need not be coded in the *run* function.

4.2 Distribution of computation

`modelRxiv` includes a number of features that are designed to reduce the computation time of output summary plots. This includes distribution of model computation across threads of the local machine, and also the ability to pool resources from multiple machines. This distribution system is also designed to connect to institution and cloud service resources. With models requiring increasing computational power, it is important to include distribution features as part of modeling platforms.

Modern browsers implement a background task system (*Web Workers*) [14] to prevent interface interaction being degraded by computational requirements of web applications, with each 'worker' deployed to a separate thread. `modelRxiv` uses Web Workers to parallelize model computation, combining model runs into batches and distributing these across multiple threads. In addition, multiple machines can pool resources by connecting to a messaging server using the *WebSocket* protocol, which allows bidirectional communication between clients and a central server [15]. Importantly, *WebSocket* has native browser and back-end implementations, allowing it to serve as a cross-platform means of transmitting data.

By deploying a *WebSocket* server, we are able to link multiple machines that connect to `modelRxiv`. This allows a user to distribute the generation of output summary plots requiring extensive computational power on the local machine, and also on any other machines connected to the same user, by sending batches through the *WebSocket* server. When the individual batches have been processed, the results are sent back through the *WebSocket* server and combined to produce the plot.

4.3 Deployment to different environments

At present, browsers have native support only for *JavaScript*. In addition, the majority of modern browsers support *WebAssembly* [16], which allows programs written in *C/C++*, *C#* and *Rust* to be compiled and run natively. This has paved the way for projects such as *Pyodide* [17], which allows *Python* and many core scientific *Python* packages (e.g. *SciPy* [18]) to be run in the browser. Nonetheless, the browser environment is limited for many reasons (including security considerations that are irrelevant to model computation), differs between browsers, and is optimized for *JavaScript*.

To enable computation of models written for a wider range of programming languages and frameworks, `modelRxiv` includes support for distribution of computation to additional environments (e.g., dedicated servers, institute computation clusters, cloud environments). Any environment connected via the *WebSocket* server using the same user account will have its resources pooled and available to use for model computation (model computation will be distributed according to the relevant frameworks available on each environment). Below we describe some examples of environments to which `modelRxiv` can be deployed.

4.3.1 Back-end utility

To allow users to utilize frameworks that are inaccessible from the browser environment, or to connect machines without the browser-based interface, we provide, as part of the platform code, a back-end utility that communicates with the WebSocket server but lacks an interface. Connecting through the back-end utility will allow batches to be received and processed by the connected machine. As the utility lacks an

Function	Input	Output
defaults		*parameters*
run	*parameters*	*result*
step	*parameters*, *step*	*step*
result	*parameters*, [*step*]	*result*

Table 2: **Functions that constitute model code**. Model code is organized so that its components are composable: *defaults* return a parameter set that can be passed to *run* or *step*, *step* output can be passed to *step* (recursion) and multiple steps can be passed to *result*, which has the same output as *run*. Model code must include *defaults* and *run*; step-wise models must also include *step* and *result* in order to allow dynamics to be generated "on-the-fly" in the browser.

interface for interacting with models, it does not allow the initiation of analyses. The back-end process utility comes with wrappers written for specific programming languages; for the sake of demonstrating the capabilities of this utility, we provide three initial wrappers for *JavaScript*, *Python* and *R*. Similarly to the browser-based interface, the utility is written in *JavaScript* but is run using *Node.js*, and spawns a thread using the wrapper of the respective programming language. Packages installed for *Python* or *R* on the machine will also be available to the utility (this will be the same for any programming language or framework supported by the utility). When running the back-end utility, users will be prompted for their credentials for `modelRxiv` as they would on the browser-based interface; after authenticating, the resources on the connected machine will become available for use via the browser-based interface (as in the resource menu in Figure 3). For instructions on installing and using the utility, see the GitHub repository (`https://github.com/carrowkeel/modelrxiv/README.md`).

4.3.2 Cluster or queue system

By default, the back-end utility assumes that allocated resources (threads) can be utilized immediately, and runs computation by spawning subprocesses of the utility. In most academic institutions, computation clusters rely on queue systems to manage thread usage. To facilitate integration of `modelRxiv` with such computational resources, we developed an additional "mode of operation" of the back-end utility that is compatible with *Slurm Workload Manager*. Integration for additional software used by clusters can be added in future versions by adding additional modes.

This mode can be activated by setting `--mode=slurm` when running the back-end utility (for detailed instructions see `https://github.com/carrowkeel/modelrxiv/README.md`). In this mode, the back-end utility launched by the user serves only as a 'relay instance' for submitting jobs to *Slurm*, and does not directly initiate model computation. After launching the utility in this mode and authentication, received batches are submitted as jobs to *Slurm* using the `sbatch` command. When the job is processed by *Slurm*, it launches a new instance of the back-end utility, which distributes computation of the batch among threads allocated to the job. When the batch is processed, the results are returned to the *WebSocket* server through the new instance of the back-end utility, which is then terminated, while the 'relay instance' continues to run.

4.4 Integration with cloud services

It is also possible to pool resources from cloud services using similar schemes to those presented above. For the sake of demonstrating the potential integration with cloud services, we present two approaches: (i) launching 'on-demand' compute containers, or (ii) using managed compute containers. It is important to note that, as opposed to using resources that are free or belong to the same research institute, using cloud services without close monitoring of usage, and without an intermediate service that manages usage, can lead to rapidly exceeding budgets set out for a project. Thus, we offer examples of integration with cloud services only for the sake of users who are already using these services and are aware of the risks involved. In addition, to fully integrate this option as a feature in the `modelRxiv` platform, it would be necessary to consider the relative cost of the different resources available for computation when distributing batches, and to design fail-safes that prevent unintended use of paid services. These features are not available in the initial version of the platform, and we recommend care in developing any integration with cloud services to avoid unintentional costly deployments.

4.4.1 Compute containers

A straightforward approach to integrating cloud services with `modelRxiv` is to launch compute containers, which are similar to temporary back-end environments. The compute containers can be launched using a container image that includes the dependencies required by the model code, and are given a fixed timeout to ensure that containers are terminated when no jobs are received (implementation of the termination of the container depends on the cloud provider). In this scheme, a user wanting to increase computational power for a specific analysis launches multiple containers with the same image. Containers can connect using the back-end utility when launched, and will appear in the resource management tab. In this implementation, the containers would terminate after all jobs have been processed.

Language/Framework	Browser deployment	Back-end deployment
JavaScript	+	*Node.js/Deno*
Python	*Pyodide* (*WebAssembly*)	+
C/C++/C#/Rust	*WebAssembly*	+
Java/R	-	+

Table 3: **Programming languages that can currently be used with `modelRxiv` in browser and back-end environments.** Plus signs indicate that a language is fully implemented while minus signs indicate that a language is not available. Where a language is available through third-party software, the name of the software package is indicated.

4.4.2 Managed compute

A more responsive approach to scaling computational power is to use managed compute containers. These containers are launched only for the duration of batch processing, and can send the results of model computation to the *WebSocket* server when complete. The benefit of managed compute is the elimination of deployment time, decreased "cloud waste" owing to idle containers, and more redundancy in case of a specific thread or container failing. These benefits come at a higher cost for equivalent computational power and require the development of additional features to prevent unintended use. To distribute model computation using managed compute services, it would be possible to replicate the current option for deploying jobs to *Slurm* to instead launch an instance of the container via a command-line utility on the local machine (most cloud services offer a command-line utility to interact with their APIs). The container would have a local copy of the back-end utility, which would receive the batch when launched, process the batch, and return the results to the user via *WebSocket*. This implementation would require only minor modification of the current utility in order to successfully process batches, but would need to be combined with additional features that can restrict use of such cloud services.

4.5 Platform architecture

`modelRxiv` is browser-based, and was designed as a serverless application. This greatly reduces operational costs as there is no need to operate a back-end server, and costs scale with use. To further reduce running costs, dynamic content is served as static files: model code is uploaded as script files, while metadata is available as *JavaScript Object Notation* (JSON) files. These are indexed as larger JSON files, reducing the need for database requests for common actions such as browsing the index of models or manipulating models. User authentication uses *JSON Web Tokens* (JWT), so that user information is retrieved only when logging in; subsequent requests are sent along with a token that contains the user information (specifically, the user ID). These considerations are important to make the project sustainable in the long-term as a free, open-source repository.

4.6 Authentication

When registering to `modelRxiv`, users provide only an arbitrary username ('Nickname') and password. Both the username and the password are stored encrypted so that the `modelRxiv` platform and other users have no access to this information. A user ID is generated when registering, and when logging in, this user ID is encoded in a JWT using a private key (RS256). When performing actions with the API (such as uploading or editing models), the token is decoded using the public key, confirming that the JWT was encoded by the correct private key, and providing the user ID. To provide authenticated access to private static resources (such as sandboxed models), upon log-in in the user is provided with an additional token (a *AWS CloudFront* signed cookie) that provides access to a specific URL pattern including the user ID. Both tokens have the same expiry time; when expired, users must log-in again to receive new tokens. Using JWTs and *AWS CloudFront* signed cookies reduces the need for authentication and database access. A similar architecture could be deployed to other cloud providers.

5 Accessibility of data

The `modelRxiv` platform, including all models analyzed here, is freely accessible at `https://modelrxiv.org`. The platform code is available at `github.com/carrowkeel/modelrxiv`. All platform code is licensed under AGPLv3. Models are licensed under CC-BY 4.0 unless otherwise stated.

6 Author contributions

KDH designed and developed the platform, GH consulted on cloud integration and GG supervised the development of modeling features. KDH and GG wrote the manuscript. All authors read and approved the final version of the manuscript.

7 Funding

We would like thank David Gokhman, Oren Kolodny and Royi Zur for helpful comments and discussions. This project was supported by Israel Science Foundation (ISF) Grant 2049/21, and by German-Israel Foundation (GIF) Grant I-1526-500.15/2021.

References

1. Otto, S. P. & Day, T. *A biologist's guide to mathematical modeling in ecology and evolution.* Princeton University Press (2007).
2. Grimm, V. & Berger, U. Structural realism, emergence, and predictions in next-generation ecological modelling: Synthesis from a special issue. *Ecological Modelling* **326,** 177–187 (2016).
3. Tisue, S. & Wilensky, U. Netlogo: A simple environment for modeling complexity. *International Conference on Complex Systems* **21,** 16–21 (2004).
4. Wolfram, S. *Mathematica: a system for doing mathematics by computer.* Addison Wesley Longman Publishing Co., Inc. (1991).
5. Kluyver, T. *et al. Jupyter Notebooks-a publishing format for reproducible computational workflows.* IOS Press (2016).
6. Staubitz, T., Klement, H., Teusner, R., Renz, J. & Meinel, C. CodeOcean-A versatile platform for practical programming excercises in online environments. *2016 IEEE Global Engineering Education Conference,* 314–323 (2016).
7. Durrett, R. & Levin, S. The importance of being discrete (and spatial). *Theoretical Population Biology* **46,** 363–394 (1994).
8. Greenbaum, G., Feldman, M. W., Rosenberg, N. A. & Kim, J. Designing gene drives to limit spillover to non-target populations. *PLoS Genetics* **17,** e1009278 (2021).
9. Getz, W. M., Salter, R., Muellerklein, O., Yoon, H. S. & Tallam, K. Modeling epidemics: A primer and Numerus Model Builder implementation. *Epidemics* **25,** 9–19 (2018).
10. MATLAB. MATLAB 9.11 (R2021b). The MathWorks Inc. (2021).
11. Le Novere, N. *et al.* BioModels Database: a free, centralized database of curated, published, quantitative kinetic models of biochemical and cellular systems. *Nucleic Acids Research* **34,** D689–D691 (2006).
12. Malik-Sheriff, R. S. *et al.* BioModels—15 years of sharing computational models in life science. *Nucleic Acids Research* **48,** D407–D415 (2020).
13. Cheifet, B. Promoting reproducibility with Code Ocean. *Genome Biology* **22,** 65 (2021).
14. Moon, S. *et al.* HTML 5.3. `w3.org/TR/2021/NOTE-html53-20210128` (2021).
15. Fette, I. & Melnikov, A. The WebSocket Protocol. RFC 6455 (2011).

16. Haas, A. *et al.* Bringing the web up to speed with WebAssembly. *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation,* 185–200 (2017).

17. The Pyodide development team. pyodide/pyodide. *Zenodo.* `doi.org/10.5281/zenodo.5156931` (2021).

18. Virtanen, P. *et al.* SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nature Methods* **17,** 261–272 (2020).