Proceedings of the Operational Research Society Simulation Workshop 2023 (SW23)
C. Currie and L. Rhodes-Leader, eds.
https://doi.org/10.36819/SW23.030

A FRAMEWORK TO SHARE HEALTHCARE SIMULATIONS ON THE WEB USING FREE AND OPEN SOURCE TOOLS AND PYTHON

Dr. Alison Harper
University of Exeter
a.l.harper@exeter.ac.uk

Dr. Thomas Monks
University of Exeter
t.m.w.monks@exeter.ac.uk

ABSTRACT

We present a simple framework to support sharing of executable healthcare discrete-event simulation models in python over the web. Our sharing framework is based on combining remote version control repositories with free and open source software - Jupyter Notebooks, Jupyter Books, and streamlit - along with free digital infrastructure provided by Binder, streamlit.io, GitHub pages, and open science repositories such as Zenodo. The framework enables executable models to be shared with users of different technical abilities: from coders to software literate users. We provide an applied example including a full web application of an executable model. Our framework aims to support NHS organisations to preview, validate, and use models. Academic research teams can also benefit from enhanced scrutiny of their work and long term archiving of models.

1 INTRODUCTION

In the UK, health and social care research is often funded using public money. This might be from a funder such as the National Institute for Health and Care Research (NIHR). This money is part of the overall NHS budget. Modelling and simulation within health and social care research often tackles very human problems as well. The systems being modelled affect hundreds, thousands, and potentially hundreds of thousands of people (infants, children, adults, carers, friends, family and potentially yourself) per year. As such research using these funds must maximise transparency, re-usability, adaptability (by the NHS and other researchers), and reduce barriers to uptake by health systems, and scrutiny by research teams. These barriers can include cost (such as the license fees for purchasing a commercial off the shelf simulation package), technical skills (such as coding skills or use of commercial software), security (such as installation on NHS machines) and software dependency management.

One possible route to enable sharing/reuse/adaptation and eliminate licensing costs is to adopt a Free and Open-Source Software (FOSS) for your modelling and simulation. FOSS is best defined using Stallman's four freedoms:

0. The freedom to run the program as you wish, for any purpose.
1. The freedom to study how the program works, and change it so it does your computing as you wish.
2. The freedom to redistribute copies so you can help your neighbour.
3. The freedom to distribute copies of your modified versions to others. By doing this you can give the whole community a chance to benefit from your changes.

Note that FOSS here is more than simply open source code. It grants the rights for users to adapt and distribute copies however they choose. FOSS software within a data science context is often provided with a *permissive* type of license such as the MIT or BSD-2 licenses. These grant the user the four freedoms (within commercial or non-commercial work), waive liability, and require crediting (citation) of that authors and software in subsequent work.

Many FOSS simulation tools exist. For a full review of open source Discrete-Event Simulation (DES) software readers should refer to Dagkakis and Heavey (2016). Here we limit our FOSS focus to tools in Python. We choose Python as it is consistently ranked in the top 4 of Stack OverFlows's most used programming languages (4th in 2022) and top 5 most loved programming languages by developers. Python is also synonymous with *modern data science* and is used extensively in research and development of computational methods and applications.

FOSS and Python have gained significant popularity and use in the UK's NHS in recent years. This transformation has been inspired via communities such as NHS-Python (and NHS-R) as well as NHS-England promoting the use of FOSS tools (NHS England 2022). However, widespread access to Python still remains challenging in the security conscious health care setting. A big challenge is *model installation* on locked down and controlled NHS machines. A possible solution to the installation problem is web based technology and in particular *simulation on the web*.

Here we adopt a straightforward definition of simulation on the web as a simulation model that has been deployed to a remote server, is accessed via a public web URL, and can be executed, in some manner with varying processing power, without local installation of any software or components. The contribution of our paper is to propose a simple python based framework for sharing executable simulation models on the web via Binder, Streamlit and Github pages.

2 AIMS

1. Briefly review related work in FOSS simulation packages implemented in Python, and sharing simulations on the web;
2. Outline a straightforward framework for deploying a simulation developed in Python on the web for users of varying technical skills;
3. Provide an applied simulation example implementing our framework;
4. Provide guidance for modellers to begin sharing models via the web.

3 RELATED WORK

3.1 Python Discrete-Event Simulation Tools

Dagkakis and Heavey (2016) reviewed open source software for discrete-event simulation (DES). Their review identified three FOSS Python packages: Simpy (Team SimPy 2020), Pysimulator (Pfeiffer, Hellerer, Hartweg, Otter, and Reiner 2012) and ScipySim (McInnes and Thorne 2011). PySimulator, and ScipySim were last updated in 2014, and 2010 respectively. Simpy has a process based simulation worldview and has continued to be maintained (last update Apr 2020). It has been used in several Operations Research relevant publications (Bovim, Gullhav, Andersson, Dale, and Karlsen 2021, Allen, Bhanji, Willemsen, Dudfield, Logan, and Monks 2020, Monks, Harper, Anagnostou, and Taylor 2022).

An updated list of Python DES packages now includes Salabim (van der Ham 2018, MIT licensed, last updated Apr 2022), Ciw (?? , MIT licensed; last updated Oct 2022), and de-sim (Goldberg and Karr 2020, MIT licensed, last updated Nov, 2020). Salabim is a fork of Simpy2 and includes many simulation tools including automatic results collection and animation. Ciw (the Welsh word for queue) was developed at Cardiff University and allows users to very quickly build complex multi-class queuing networks. An advanced feature of Ciw is network deadlock detection. Palmer and Tian (2021b) provide two reproducible, open source implementations of Ciw for hybrid modelling, archived here (Palmer and Tian 2021a). De-sim is an object orientated (OO) framework for developing complex interacting DES models. The de-sim authors argue that their OO framework is an advancement over Simpy's process-based worldview; although we note that Simpy itself is highly flexible and can easily be used within an OO framework; for example see (Allen, Bhanji, Willemsen, Dudfield, Logan, and Monks 2020).

3.2 Sharing Models on the Web

The idea of simple sharing and executing simulations via the web is not a new one. Fishwick (1996) provides an early example of an executable queuing simulation model accessed via a public URL. Modern takes on simulation on the web include full Open Science Gateways that provide substantive e-infrastructure for controlled model access, parameterisation and execution. Anagnostou, Taylor, Groen, Suleimenova, Anokye, Bruno, and Barbera (2019) provide an impressive example of open science gateways in action using their PALMS model.

Simpler recent examples include a generic hospital ward model developed and made available online (Penn, Monks, Kazmierska, and Alkoheji 2020), which aimed to address the issues raised in this paper. The Simul8 model is made available through Zenodo - an open science repository (Penn and Monks 2018). The authors note that a weakness of their approach is that NHS users must purchase a COTS package license in order to reuse/adapt the model. Tyler, Murch, Vasilakis, and Wood (2022) addressed the need to improve uptake and routine use of simulation by NHS analysts by developing an open source model using *R* which supports user-defined configurations of patient pathways. The *R Shiny* user interface improves usability and functionality, but the model needs to be downloaded and executed locally on the user's machine. Similarly, an open, verifiable model to enable the organisation of dialysis outpatient services during the pandemic was developed by Allen, Bhanji, Willemsen, Dudfield, Logan, and Monks (2020). The code and documentation are available to download and a limited user interface to execute the model is provided via Binder. There is no support for experimentation. Monks, Harper, Anagnostou, and Taylor (2022) proposed a 6-level framework for supporting M&S open working, with a case study demonstrating an approach to achieving openness, including a simple, reproducible pipeline for sharing simulation experimentation and results. The focus of the paper is the practical issues to be considered with different levels of openness.

4 A FRAMEWORK FOR SHARING PYTHON MODELS

We developed a framework to allow an executable simulation model developed in Python to easily be shared to different classes of users with varying knowledge of coding, computer simulation, and software. Our first class considers more technical simulation users that have coding experience; for example NHS analysts or simulation researchers. Our second class of user assumes no experience of coding, but a general familiarity of using software. Figure 1 provides an overview of our framework. In summary, we provide two approaches to sharing executable models:

- FOSS licensed, interactive, and executable scientific notebooks using BinderHub;
- A web app (browser based) model front end built via StreamLit and deployed via Streamlit.io

In addition to the sharing of executable python models our framework also supports enhanced model documentation and open working using Github pages and an (interactive) Jupyter Book.

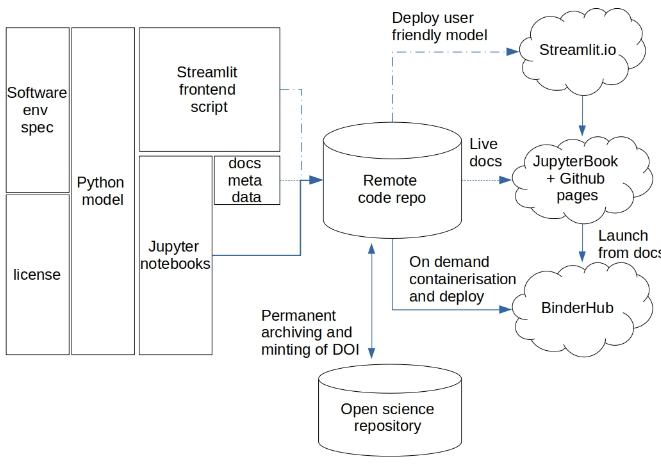

Software env spec
license
Python model
Streamlit frontend script
docs meta data
Jupyter notebooks
Deploy user friendly model
Streamlit.io
Remote code repo
Live docs
JupyterBook + Github pages
Launch from docs
On demand containerisation and deploy
BinderHub
Permanent archiving and minting of DOI
Open science repository

Figure 1: Proposed sharing framework for Python models

4.1 Software Environment Specification

For any type of software deployment problem it is necessary to adopt a language that allows model developers to specify the software libraries, version numbers and sources that are required in order for the model to run. Three options of environment file exist for python: pip requirements, conda environments, and poetry. Our research often makes use of conda virtual environments that documents dependencies in the *Yet Another Mark-up Language (.yml)* format.

4.2 Licensing of Research Artefacts

Before sharing (or making any Python code public), it is essential that authors select an appropriate FOSS license for the code and other research artefacts. This, for example, ensures appropriate use of the model and removes liability of the authors. A common approach in data science is to adopt a permissive license such as the MIT. See Monks, Harper, Anagnostou, and Taylor (2022) for more details on licence types.

4.3 Python Model

Our framework is general to any Python model and Python simulation package. For an application exemplar, we chose to use Simpy as our FOSS simulation package. Simpy provides a simple process based worldview simulation engine. This simplicity makes the package highly flexible and usable in many simulation applications in healthcare. Our research group also has extensive experience of using it in real world simulation applications. One limitation of Simpy relative to commercial off the shelf simulation packages is the lack of a user friendly front end to control simulation execution, scenarios and results analysis. The Python model should be independent of any *front-end* and executable standalone.

4.4 Remote Code Repository

Central to Figure 1 is a remote code repository. The most popular solutions are GitHub, GitLab, and BitBucket. Most developers opt for GitHub, but we recommend modellers consider the unique benefits and cons of each solution. All code artefacts (including licence, meta-data files, and readme) should be committed to the remote repo to provide long lasting version control. Some developers store data here too; although other specialist solutions exist for data version control.

4.5 Jupyter Notebooks

Project Jupyter is a FOSS project that supports scientific coding in Python, *R* and Julia in a classical notebook format. A simple way to conceptualise Jupyter notebooks are as browser based interface and an execution kernel. The interface provides a set of markdown and code sells that a user can control. Markdown is simply a way to present formatted text, mathematics, and code together. As such it provides an excellent way to explain, document and then execute simulation experiments. For example, an analyst may have created a set of Jupyter notebooks for model validation, determining the number of replications, warm-up analysis, defined experiments, or interactive experimentation.

Jupyter's kernel executes all of the code. Typically the kernel is located on the users local machine. But given the server nature of Jupyter, the kernel could be located on a remote (possibly powerful) machine. In these remote cases the user need not have Python, Simpy, or Jupyter installed on their own machine.

4.6 Binderhub

Jupyter's use of the browser and ability to access a remote kernel allows for easy deployment of simulation notebooks to a technical user-base. For example, well trained NHS analysts from the NHS-python / NHS-R community or other researchers familiar with reading and executing code.

BinderHub provides a Jupyter environment in the cloud that others can use to run your simulation model. One simple requirement is that the Python simulation model code, and Jupyter notebooks are available in a public respository hosted in the cloud; for example in GitHub or GitLab.

4.7 Streamlit and streamlit.io

A web app interface to the model is suitable for less technical simulation users such as NHS analysts with no coding experience, NHS managers, researchers not familiar with python, or the general public. A web app will provide simple ways to setup and execute the model. This might include parameter fields, logic diagrams, basic animation, and buttons.

Streamlit is a simple modern Python library that provides a way to script web applications. Streamlit has many built in controls and display functions, for buttons, sliders, text fields, tables, and charts. Using Streamlit, it is possible to create a very simple simulation app with only a few lines of code; for example, basic parameter settings, an execute button, and results displayed as a table in the browser. Streamlit.io is a free hosting service for Streamlit apps. When used together they provide a robust and easy way to share simulation models to healthcare users. *These models can then be accessed from any device*. For example, models can be executed from a phone or a tablet instead of traditional laptop or desktop environment.

4.7.1 Documentation via Jupyter Book and Github Pages

While not strictly required for sharing the simulation model our framework advocates that any simulation used for health decision making should provide open and high quality documentation on how the model is implemented and works. In our example, we provide a link to another web based technology: a Jupyter Book deployed in GitHub pages https://tommonks.github.io/treatment-centre-sim. Jupyter Book is a high level tool to create a structured website based on meta-data (a table of contents and config file), markdown, and Jupyter Notebooks. GitHub Pages is a free web hosting service provided by GitHub (owned by Microsoft). As such it can publish cleanly annotated Python code describing the working of each component within the model. In our example, we have supplemented this with discrete sections for the Strengthening the Reporting of Empirical Simulation Studies (STRESS) for DES models (Monks, Currie, Onggo, Robinson, Kunc, and Taylor 2019).

One benefit of a Jupyter Book deployed on the web in this manner is that it automatically provides links to BinderHub. This means that different parts of the model and model process (such as model testing) can be viewed on a static website and then opened live to run via a link in the book if desired. A second benefit of online documentation managed with Jupyter book is that we have provided a simple mechanism for users/readers to provided feedback (such as queries, clarification requests, and bug reports). Within our Jupyter Book we have provided *contribution* guidelines and implemented the technology using GitHub Issues. The latter allows a conversation between authors and contributors to facilitate improvement in the documentation and model web app.

4.8 Open Science Repository

Lastly, for long term archiving (and preservation) of simulation models our framework links code in the remote repository to an open science repository. Examples, include Figshare, Zenodo, and the Open Science Framework. Each of these has guarantees on storage and mints DOIs to enable citation and improve discoverability.

5 APPLIED EXAMPLE

This section describes a text-book simulator implemented in Python and shared via our framework. In addition to the code we provide, we include some basic instructions for use of Binder and required modifications to sharing desktop based simulations.

5.1 Case Study Model

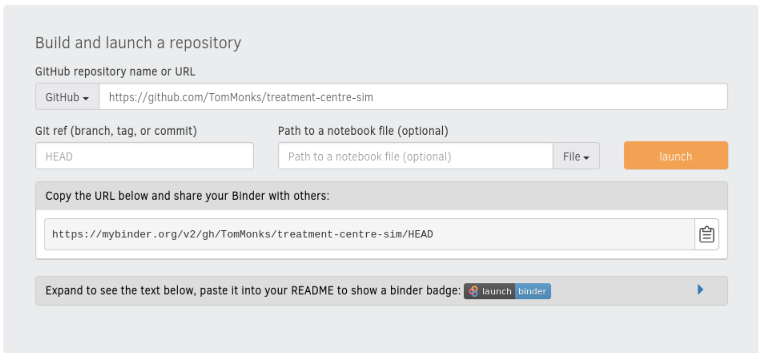
We adapt a textbook example from Nelson (2013, p.170): a discrete-event simulation model of a U.S based treatment centre. In the model, patients arrive to the health centre between 6am and 12am following a non-stationary Poisson process. On arrival, all patients sign-in and are triaged into two classes: trauma and non-trauma. Trauma patients include impact injuries, broken bones, strains or cuts etc. Non-trauma include acute sickness, pain, and general feelings of being unwell etc. Trauma patients must first be stabilised in a trauma room. These patients then undergo treatment in a cubicle before being discharged. Non-trauma patients go through registration and examination activities. A proportion of non-trauma patients require treatment in a cubicle before being discharged. The model predicts waiting time and resource utilisation statistics for the treatment centre. The model allows managers to ask questions about the physical design and layout of the treatment centre, the order in which patients are seen, the diagnostic equipment needed by patients, and the speed of treatments. For example: “what if we converted a doctors examination room into a room where nurses assess the urgency of the patients needs.”; or “what if the number of patients we treat in the afternoon doubled”.

5.2 Code and Software

We have shared our code via GitHub. Binder examples can be found at https://github.com/TomMonks/treatment-centre-sim while the Streamlit example can be found at https://github.com/TomMonks/treat_sim_streamlit. We used Conda to manage our dependencies. In summary, we used Python 3.8. Data manipulation, simulation, and general mathematical modeling were done using Simpy v4.0.1 (Team SimPy 2020), NumPy v1.19.2 (van der Walt, Colbert, and Varoquaux 2011) and Pandas v1.2.3 (McKinney 2011). Charts were produced with MatPlotLib v3.3.4 (Hunter 2007). All analyses were conducted in Jupyter-Lab v2.4.3 (JupyterLab 2022). The web app was produced using StreamLit v1.13.0 (streamlit.io 2022). Documentation was created using Jupyter Book v0.13.1 (Executable Books Community 2020).

5.3 Binder

For this example, we have created a simulation model and committed the code to a public GitHub repository https://github.com/TomMonks/treatment-centre-sim. Binder expects that a conda virtual `environment.yml` file is placed into a sub-directory `binder/`. Once this has been committed it is a simple case of navigating to https://mybinder.org/ and copy and pasting the URL into the build and launch text box. Binder will then create a remote instance that contains both your code and dependencies specified in the `binder/environment.yml` file. The first time the instance is built it will likely take several minutes. This process may be need to be repeated if the instance is not used regularly. The setup form is illustrated by Figure 2



Build and launch a repository

GitHub repository name or URL

GitHub ▾ https://github.com/TomMonks/treatment-centre-sim

Git ref (branch, tag, or commit): HEAD

Path to a notebook file (optional): Path to a notebook file (optional) File ▾

launch

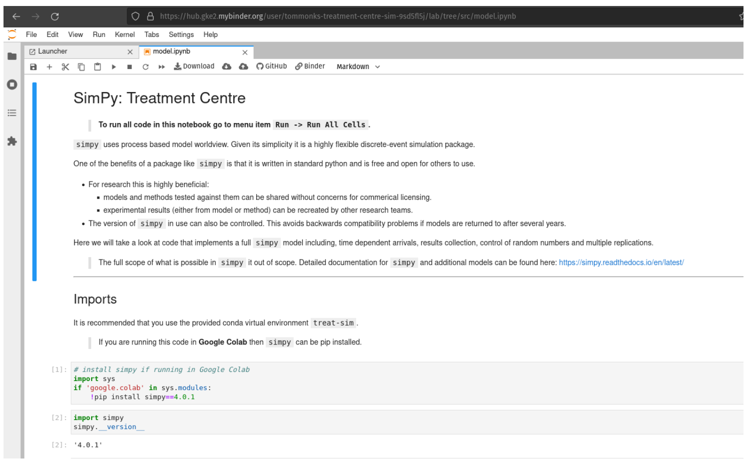
Copy the URL below and share your Binder with others:

`https://mybinder.org/v2/gh/TomMonks/treatment-centre-sim/HEAD`

Expand to see the text below, paste it into your README to show a binder badge: launch binder

Figure 2: Binderhub setup

We invite interested readers to launch our treatment sim instance on Binder via the following link: https://bit.ly/treat_sim_binder. The model notebook can be found `src/full_model.ipynb`. You should be presented with a notebook in Figure 3.


https://hub.gke2.mybinder.org/user/tommonks-treatment-centre-sim-9sd5fl5j/lab/tree/src/model.ipynb
File Edit View Run Kernel Tabs Settings Help
Launcher
model.ipynb
Download
GitHub
Binder
Markdown

SimPy: Treatment Centre

> **To run all code in this notebook go to menu item `Run -> Run All Cells`.**

`simpy` uses process based model worldview. Given its simplicity it is a highly flexible discrete-event simulation package.

One of the benefits of a package like `simpy` is that it is written in standard python and is free and open for others to use.

- For research this is highly beneficial:
  - models and methods tested against them can be shared without concerns for commerical licensing.
  - experimental results (either from model or method) can be recreated by other research teams.
- The version of `simpy` in use can also be controlled. This avoids backwards compatibility problems if models are returned to after several years.

Here we will take a look at code that implements a full `simpy` model including, time dependent arrivals, results collection, control of random numbers and multiple replications.

> The full scope of what is possible in `simpy` it out of scope. Detailed documentation for `simpy` and additional models can be found here: https://simpy.readthedocs.io/en/latest/

Imports

It is recommended that you use the provided conda virtual environment `treat-sim`.

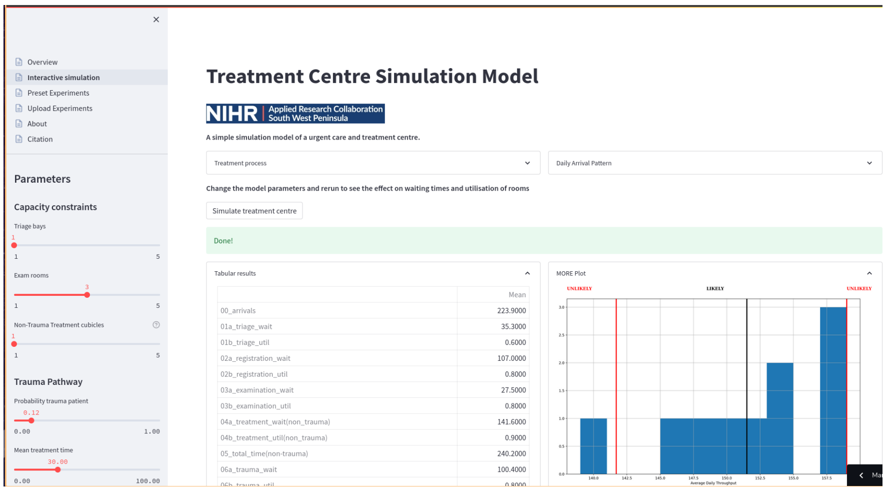
> If you are running this code in **Google Colab** then `simpy` can be pip installed.

```
[1]: # install simpy if running in Google Colab
     import sys
     if 'google.colab' in sys.modules:
         !pip install simpy==4.0.1

[2]: import simpy
     simpy.__version__

[2]: '4.0.1'
```

Figure 3: Simulation model running in binderhub

5.4 StreamLit.io App

Our Streamlit implementation of the case study model can be accessed via https://treat-sim.streamlitapp.com. To create the Streamlit app we logged into streamlit.io, created a new app, and specified the GitHub repo plus app launch file (Overview.py). Streamlit.io requires information on app dependencies. We used a pip `requirements.txt`. The initial build is automatic and takes 5-10 minutes. Figure 4 illustrates the interactive simulation page of our app.



Treatment Centre Simulation Model

A simple simulation model of a urgent care and treatment centre.

Change the model parameters and rerun to see the effect on waiting times and utilisation of rooms

Done!

Parameters

Capacity constraints

Triage bays: 1 (range 1–5)

Exam rooms: 3 (range 1–5)

Non-Trauma Treatment cubicles: 1 (range 1–5)

Trauma Pathway

Probability trauma patient: 0.12 (range 0.00–1.00)

Mean treatment time: 30.00 (range 0.00–100.00)

Tabular results

	Mean
00_arrivals	223.9000
01a_triage_wait	35.3000
01b_triage_util	0.6000
02a_registration_wait	107.0000
02b_registration_util	0.8000
03a_examination_wait	27.5000
03b_examination_util	0.8000
04a_treatment_wait(non_trauma)	141.6000
04b_treatment_util(non_trauma)	0.9000
05_total_time(non-trauma)	240.2000
06a_trauma_wait	100.4000
06b_trauma_util	0.8000

Figure 4: Treatment centre user interactive simulation

6 GUIDELINES FOR SHARING MODELS VIA STREAMLIT

When deployed, a Streamlit application must be thought of as a remote instance of a simulation model in the cloud. The model is accessed via a URL. Each user that accesses the URL is provided with a unique instance of the web app (i.e. it is not shared between users). To be successful, a **slight change** in a developers way of thinking is needed as you are working with an instance of a web application as opposed to a model on a local machine.

6.1 Parameterisation

A simulation model that will be executed by clients and people other than those that developed it will typically provide a way to parameterise a model for one or more experiments. Two such ways are:

- Default parameters, interactive controls, and fields built into the simulation interface;
- An external file that contains parameter configurations for a single or multiple experiments.

The first of these options is perhaps best used for setting up a singular experimental run of the model. For example, one or multiple replications of the model to test if changing a parameter has an impact on results. For interactive simulation of this type a web app requires no modification from classical desktop simulation. For example, a set of sliders representing model parameters, each with a default value, could be provided for users to manipulate and rerun the model.

The second of these options in its simplest form might use a format such as a Comma Separated Values (.CSV), containing a simple row and column format for experiment and parameters. The user would populate (or more likely modify) these parameters in some way before a simulation model loads the parameters on each run. If a simulation model is deployed via a web-app the model may be setup with an option for a user to *upload* the CSV file to the instance (e.g. by click and dragging a file into a web form). It is recommended that an example file is provided. The format might vary depending on the application, for example, a format might be rows of named experiments and columns of parameter values (or vice versa). One option is for the web-app to auto-generate an exemplar file that a user can download, modify, and then upload. A secondary option for parameterisation is to specify a URL where parameter data are stored. This would still require a user to ensure the data are in the correct location and it may be simpler to upload directly. Alternatively an application might allow a user to specify a Digital Object Identifier (DOI) where data are stored. This latter option might be useful if the web app is part of a publication and a permanent record of data used in experiments is required.

6.2 Results Visualisation

One significant advantage of developing a front end to the simulation model in Python is the opportunity to produce publication quality charts and outputs directly from each run of the model and then allow a user to export them. For example, Python's Matplotlib library could be used to create a Measure of Risk and Error (MORE) plot after a user specified experiment. These packages offer superior plotting options and output resolution than utilities found in a typical spreadsheet package, or specialist simulation tool. Once a model run is complete, a high resolution image, meeting printing specifications such as DPI > 300, can then be downloaded by the user. Of course, like desktop simulation packages the data underlying plots might be of more interest to users; particularly if they wish to create a custom set of plots. In these cases web-apps need to provide the functionality to *download* the data in a suitable file format (for example, a CSV file). Another option is to allow a user to copy data from the web-app to the local machine's clipboard. The data can then be pasted into whatever software the user desires; for example, a spreadsheet or statistical analysis package. In these instances, the developer needs to remember that the data reside on a remote instance of the model not the local machine. For external plotting web-apps may also contain links to (or generate) Python code templates that users can use to quickly produce plots and customise.

Many interactive visualisation technologies have been developed in Python for web applications. Two such powerful and popular visualisation libraries are bokeh and plotly. Interactive options might include 'hover and see more detail', such as data point values, or zooming in on different parts of a plot, for example in the case of a time series plot.

6.3 Content and Organisation

The digital content that is included with a simulation model deployed online will vary depending on the use case. However, we recommend that developers consider a number of simple options to enhance user experience for the model. Our basic guidelines include:

- A non-expert summary of model and its use
- Interactive experimentation options
- Advanced experimentation or experimental design (aka scenario analysis) facilities
- A page describing the project and funding used to develop the model
- Model citation instructions
- Links to any external code repositories and model technical documentation

With a Streamlit implementation of a Python simulation model we propose that these can be organised as separate *pages* within the web app. The user can then navigate to the appropriate page to access the functionality. A developer will need to decide on a *landing page* for the app. In our example, we have opted for the non-expert summary. Writing a non-expert summary is a non-trivial task and out of scope of this article. It is worth noting that a health care simulation deployed via a web-app may be private to an organisation, organisational sub-group such as a department or team, or as in the case of streamlit.io public on the internet. Particularly in the latter case we recommend a high quality non-expert summary that provides a general overview of the model along with its use case.

7 DISCUSSION

We present a simple, practical framework for sharing free and open simulation of healthcare systems developed in Python.

7.1 Strengths and Contributions

Our FOSS approach benefits health services, such as the NHS in the UK, students of simulation and researchers other than the authors of the original simulation. Our framework has the following benefits for the simulation and health services communities:

Previewing models for potential users

The FOSS tools within our framework enable potential users of a model to test it out without going through any installation, source, or dependency management. This can enable fast feedback and revision of a model to fix mistakes or better meet user requirements. More technical users have direct access to executable code in this process.

Sharing runnable code with an organisation that cannot run python

NHS organisations have slowly been adopting Python, but when working with new organisations IT departments may be initially reluctant to allow model installation. Python and other free and open software are sometimes viewed with suspicion by IT departments and cannot be installed on a collaborators machine.

Making the results of a publication repeatable

In science, particularly within computational science, there's a concept called the *reproducibility crisis*. This is a problem with understanding how results of a particular study were generated, and an inability to (fully) reproduce (repeat) the results with the same code and data. In lay terms it means that most academic authors write a nicely crafted paper that doesn't manage to fully reveal how they got the figures they reported in their simulation study. Modern simulation projects are complicated and writing up a brief summary of methods may fail to enable reuse, verification, or peer learning. Our framework supports authors to provide an executable version of their model with their publications.

7.2 Limitations

Our framework aims to share Python models, although several of technologies used are language agnostic and overall it is easily adapted for *R* and to some extent Julia. Modifications for *R* would include switching from StreamLit to *R Shiny* (and Shinyapps.io). *R* users are less likely to be familiar with rigorous dependency management than Python and Julia developers; however, dependency management can be achieved via the *renv* library (or to a lesser extent via conda). Julia currently lacks a native StreamLit, equivalent, but our framework would allow sharing with more technical users. A limitation of all free remote hosting services is the amount of compute available. For expensive computer simulation, two potential modifications could be considered. The first is to pay to host a web app (or Binder server) that provides sufficient compute (e.g via Heroku). The second is to *containerise* the app and model via a technology such as *docker* and *dockerhub*. Model users will require docker installed on their local machine in order to run it. One drawback is that for commercial organisations this will involve a licence fee.

ACKNOWLEDGEMENTS

This article is independent research supported by the National Institute for Health and Care Research Applied Research Collaboration South West Peninsula. The views expressed in this publication are those of the author(s) and not necessarily those of the National Institute for Health and Care Research or the Department of Health and Social Care.

REFERENCES

Allen, M., A. Bhanji, J. Willemsen, S. Dudfield, S. Logan, and T. Monks. 2020, Aug. "A simulation modelling toolkit for organising outpatient dialysis services during the COVID-19 pandemic". *PLOS ONE* 15 (8): e0237628. Publisher: Public Library of Science.

Anagnostou, A., S. J. E. Taylor, D. Groen, D. Suleimenova, N. Anokye, R. Bruno, and R. Barbera. 2019, December. "Building Global Research Capacity in Public Health: The Case of a Science Gateway for Physical Activity Lifelong Modelling and Simulation". In *2019 Winter Simulation Conference (WSC)*, 1067–1078. ISSN: 1558-4305.

Bovim, T. R., A. N. Gullhav, H. Andersson, J. Dale, and K. Karlsen. 2021, December. "Simulating emergency patient flow during the COVID-19 pandemic". *Journal of Simulation* 0 (0): 1–15. Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/17477778.2021.2015259.

Dagkakis, G., and C. Heavey. 2016. "A review of open source discrete event simulation software for operations research". *Journal of Simulation* 10 (3): 193–206.

Executable Books Community 2020, February. "Jupyter Book".

Fishwick, P. A. 1996, November. "Web-based simulation: some personal observations". In *Proceedings of the 28th conference on Winter simulation*, WSC '96, 772–779. USA: IEEE Computer Society.

Goldberg, A. P., and J. R. Karr. 2020. "DE-Sim: an object-oriented, discrete-event simulation tool for data-intensive modeling of complex systems in Python". *Journal of Open Source Software* 5 (55): 2685.

Hunter, J. D. 2007. "Matplotlib: A 2D graphics environment". *Computing in Science & Engineering* 9 (3): 90–95.

JupyterLab 2022, October. "JupyterLab". original-date: 2016-06-03T20:09:17Z.

McInnes, A. I., and B. R. Thorne. 2011. "ScipySim: Towards Distributed Heterogeneous System Simulation for the SciPy Platform".

McKinney, W. 2011. "pandas: a foundational Python library for data analysis and statistics". *Python for High Performance and Scientific Computing* 14.

Monks, T., C. S. Currie, B. S. Onggo, S. Robinson, M. Kunc, and S. J. Taylor. 2019. "Strengthening the reporting of empirical simulation studies: Introducing the STRESS guidelines". *Journal of Simulation* 13 (1): 55–67.

Monks, T., A. Harper, A. Anagnostou, and S. J. Taylor. 2022, Jul. "Open Science for Computer Simulation".

Nelson, B. 2013. *Foundations and methods of stochastic simulation: a first course*. London: Spinger.

NHS England 2022. "NHS England Open Source Programme". https://www.england.nhs.uk/digitaltechnology/open-source/.

Palmer, Geraint and Tian, Yawen 2021a, March. "Source code for Ciw hybrid simulations.".

Palmer, G. I., and Y. Tian. 2021b. "Implementing hybrid simulations that integrate DES+ SD in Python". *Journal of Simulation*:1–17.

Penn, M., T. Monks, A. Kazmierska, and M. Alkoheji. 2020, April. "Towards generic modelling of hospital wards: Reuse and redevelopment of simple models". *Journal of Simulation* 14 (2): 107–118. Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/17477778.2019.1664264.

Penn, M. L. and Monks, T. 2018, October. "Generic Ward Configuration Simulation Model".

Pfeiffer, A., M. Hellerer, S. Hartweg, M. Otter, and M. Reiner. 2012. "PySimulator–A simulation and analysis environment in Python with plugin infrastructure".

streamlit.io 2022. "Streamlit. The fastest way to build and share data apps". https://streamlit.io/.

Team SimPy 2020. "SimPy 3.0.11". https://simpy.readthedocs.io/en/latest/index.html.

Tyler, J. M., B. J. Murch, C. Vasilakis, and R. M. Wood. 2022, June. "Improving uptake of simulation in healthcare: User-driven development of an open-source tool for modelling patient flow". *Journal of Simulation* 0 (0): 1–18. Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/17477778.2022.2081521.

van der Ham, R. 2018. "Salabim: open source discrete event simulation and animation in python". In *Proceedings of the 2018 Winter Simulation Conference*, 4186–4187.

van der Walt, S., S. C. Colbert, and G. Varoquaux. 2011. "The NumPy Array: A Structure for Efficient Numerical Computation". *Computing in Science Engineering* 13 (2): 22–30.

AUTHOR BIOGRAPHIES

ALISON HARPER is an ESRC post-doctoral Research Fellow at the University of Exeter Medical School, in the Peninsula Collaboration for Health Operational Research and Development. Her email

address is a.l.harper@exeter.ac.uk.

THOMAS MONKS is Associate Professor of Health Data Science at University of Exeter Medical School and Turing Fellow at the Alan Turing Institute. His email address is t.m.w.monks@exeter.ac.uk. His website is https://arc-swp.nihr.ac.uk/about-penarc/people/tom-monks/.