Proceedings of the 2024 Winter Simulation Conference
H. Lam, E. Azar, D. Batur, S. Gao, W. Xie, S. R. Hunter, and M. D. Rossetti, eds.

DEPLOYING REUSABLE HEALTHCARE SIMULATION MODELS IN PYTHON

Alison Harper[1], Thomas Monks[2], and Navonil Mustafee[1]

[1]Centre for Simulation, Analytics, and Modeling, University of Exeter Business School, Exeter, UK
[2] University of Exeter Medical School, Exeter, UK

ABSTRACT

Discrete-event simulation (DES) models for healthcare service planning are time-consuming to develop for both modellers and healthcare stakeholders. Model reuse is seen as a potential solution to reduce duplication of effort and maximise the potential value gained from the model. One approach to model reuse is to deploy a simulation model for the same purpose in a single application area, which can be used for planning by healthcare staff such as managers, clinicians or analysts. A model deployed for reuse by healthcare stakeholders needs to be shared and accessible. However, Python can present accessibility and usability challenges for non-technical users. In this paper we investigate some of the advantages and disadvantages to several methods of deploying interactive DES models to non-technical healthcare users. Combining DES with methods and tools from computer science and software engineering, these hybrid models aim to increase the usability, functionality, and accessibility of simulation models.

1 INTRODUCTION

Simulation models are used extensively for decision-support in healthcare service delivery research. Discrete-event simulation (DES) is the most widely used method as it can effectively and intuitively capture the complexity of health and care processes, though the reported impact of results remains low compared to the number of publications (Roy et al. 2021; Vázquez-Serrano et al. 2021; Kar et al. 2024). Attempts at addressing this include the use of generalizable models as a way of improving model use and reuse, with the aim of reducing the time and cost of developing models for similar processes in similar settings (Hussain et al. 2022; Fletcher and Worthington 2009; Robinson et al. 2004). Re-using generalizable models increases the potential to align model objectives and timescales with those of stakeholders, which are well-known barriers to the uptake of simulation in healthcare (Long and Meadows 2018; Jahangirian et al. 2017). There has been sporadic but rising interest in the use of reusable models for health service delivery over recent years, with the purpose of reducing unnecessary duplication of the clinical and researcher time and expertise required to build models from scratch (Crowe et al. 2023; Boyle and Mackay 2022). Boyle et al. (2022) defined generalizable models as those which can represent multiple units, and can be re-parameterised for site-specific problems. Here, successful re-use may require adapting, composing, integrating, and validating models and components (Fletcher and Worthington 2009; Hussain et al. 2022).

A simpler approach is to re-use a model many times for the same purpose (Robinson et al. 2004). In healthcare services research, where simulation is commonly used, models are generally treated as disposable artifacts. We see few applications of models that can easily be deployed to healthcare stakeholders for reuse. An ongoing challenge is how to deploy a model to a healthcare organisation, to be used by healthcare managers, clinicians, analysts, or for healthcare modeling and simulation (M&S) education. In order to be used, a model needs to be shared, and accessible. The popularity of commericial off-the-shelf software (COTS) is evident, with the majority of DES applications in healthcare using a COTS package such as ARENA, AnyLogic, or Simul8 (Monks and Harper 2023a; Crema and Verbano 2021). These tools offer the advantages of user-friendly interfaces, built-in functionalities for specific industries, and dedicated customer support. However users without simulation knowledge may find the barriers to interacting with simulation

software - such as time, capacity, data issues, and training - to be a significant hurdle (Brailsford et al. 2013). Perhaps a bigger obstacle to access by healthcare users is that COTS packages often come with high license fees, including expensive upgrades to access more powerful features. Free versions may be available for education/self-education, but may not be licensed for research, including reuse. This can particularly present as a barrier for researchers in emerging and developing economies. Nonetheless, while COTS remains the most popular choice for DES applied healthcare research, the majority of recent (2019 to 2022) DES models which have been made openly available have been developed in Free and Open Source Software (FOSS), most commonly Python or R (Monks and Harper 2023a).

FOSS tools such as Python offer many advantages for M&S. For example, they integrate with other tools, such as packages for machine learning or agent-based simulation, or into larger workflows or systems, with extensive resources and communities to support researcher learning and trouble-shooting. FOSS offers transparency in how simulation models are built and run, which can help to ensure reusability, and that results are reproducible. The ability to reproduce published results is central to the scientific method, and increasing emphasis on reproducible research in data science and M&S is encouraging researchers to make their model or code available (Pouwels et al. 2022; Auer et al. 2021; Taylor et al. 2018; Munafò et al. 2017). Nonetheless, FOSS can bring other challenges, such as requirements for code testing; package/software version dependency management; providing a functional user interface; and addressing the needs of different users. In particular for reuse, FOSS tools can present accessibility and usability challenges for non-technical users (Collins et al. 2023).

With the aim of improving uptake of M&S by health and care services, this paper investigates hybrid methods of deployment of DES models for reuse by healthcare staff with limited technical expertise in simulation. In M&S, hybrid modeling refers to the combined use of simulation with methods and tools from other academic disciplines (Mustafee, Harper, and Fakhimi 2022). By combining DES with established methods from computer science and software engineering, the aim is to increase the functionality and accessibility of the model. The focus of this paper is on deploying interactive models for reuse for the same purpose - that is, a model is bespoke to a service (such as discharge planning) or a purpose (such as education), and is intended for repeated-use by the service. In contrasting alternative methods of deployment for non-technical users, our objective is to enable applied DES researchers to improve the usability and reusablity of their FOSS models.

2 LITERATURE REVIEW

In previous work, we examined the code and model sharing literature for healthcare DES studies (Monks and Harper 2023a). Although sharing model code enhances transparency (Pouwels and Koffijberg 2024), we found very few shared FOSS DES models that had the potential for reuse, although most shared models used FOSS. The reasons for this varied, and included no license for reuse, code formatting (e.g. code shared as a word document), or a lack of simple instructions for use. These represent basic, essential components for sharing reusable models, as without these, the model is difficult or impossible to reuse. We proposed a framework to support M&S researchers to share their Python code for reusable simulations for healthcare: Sharing Tools and Artifacts for Reusable Simulations (STARS) (Monks, Harper, and Mustafee 2024). Our framework, as with other frameworks designed to provide a conceptual workflow for open science (Van Lissa et al. 2021), aims to enable researchers to meet different standards for open science. Adhering to the 'essential' components that we specified (open licensing; dependency management; minimum documentation for use; research metadata which includes an ORCID and citation information; a maintained code repository; and long-term archiving and preservation of the model) will enable a model to meet the criteria for being discoverable, accessible, and usable, and in principle is platform-independent (The Turing Way Community. 2022; OMF. 2023; Monks, Harper, and Mustafee 2024). Mark W. Isken and Roumani (2023), Anagnostou et al. (2022), Chalk et al. (2021), and Bartz-Beielstein et al. (2021) provide uncommon examples of shared healthcare DES models that are likely to be reusable, based on meeting some or (rarely) all of these criteria. A generalizable model that did not meet these criteria would

be difficult or impossible to reuse, even if the model or code were made available, as other researchers have discovered when attempting model reuse (Boyle and Mackay 2022).

Our framework also considers accessible deployment, and includes options for deploying models to users of differing technical ability. Documentation hosting using open source technical publishing systems offers a way of publishing enhanced documentation alongside a research paper. An online coding environment enables code to be run on a remote kernel for a technical user base. Finally, a model interface allows the model to be accessible to less technical simulation users such as healthcare analysts, clinicians, managers, researchers, or the general public. This can be achieved by providing a simple interface for setting up and executing the model and its experimental scenarios, which can be installed and run locally in a browser.

Focusing on accessibility and model reuse, Tyler et al. (2023) provided an example of an open source model in R aimed at increasing uptake of simulation by health services with a graphical user interface and accompanying documentation. Alternatively, by using a hosting service such as Streamlit Community Cloud or Shiny.io, both of which offer a tier of free hosting, the model can be shared directly with users via a URL, requiring no software to be installed. Example applications have been developed for resource planning directly by healthcare stakeholders (Harper et al. 2023; Hassanzadeh et al. 2023).

Other robust methods have been used for model deployment outside of healthcare. For example, Moreau et al. (2023) promoted the use of containerization for improving reproducibility in computational sciences. Containers are lightweight and share the host's system, while virtual machines are more isolated but use more resources. Containers are being used in a growing number of computational fields, enabling efficient, collaborative, shared research, and ensuring compatibility across operating systems (Ferro et al. 2022; Lin et al. 2021; Van Lissa et al. 2021). Isolating the model and its environment, including software and all required packages, in a container ensures that a DES model can be used across different computing environments by any user, with a consistent runtime environment, supporting reusability standards (OMF. 2023), however examples in healthcare M&S are few. Another approach to deployment is WebAssembly, a technology that allows programming languages to run at near-native speed in web browsers, enhancing the performance and capabilities of web applications. These approaches extend our existing framework, providing advanced options for deployment to healthcare users. Each method offers unique tradeoffs and advantages depending on the specific needs and preferences of users. While containerization excels in ensuring reproducibility and compatibility across different systems, WebAssembly enhances performance and accessibility directly in web browsers. Understanding the opportunities and limitations of different methods helps researchers select the most suitable deployment strategy for their particular application and computational environment.

In this paper, we examine the advantages and trade-offs of four different methods of deploying DES models to non-technical healthcare users. We assume that all 'essential' components have been met (Monks, Harper, and Mustafee 2024), which meets the following UNESCO principles of Open Science (UNESCO. 2021): openly available, openly accessible, reusable, transparent, scrutable, open science infrastructures, open educational resources, sustainability of resources. By focusing on accessible deployment and reuse by healthcare users, the following UNESCO concepts on Open Science are additionally met: increased collaboration, sharing information, communication beyond the scientific community, open scientific knowledge, science communication, open engagement of societal actors, equality of opportunities. These represent good practice, and open, reproducible science is undoubtedly good science (Pouwels and Koffijberg 2024; Munafò et al. 2017). However, while we position our work in the principles of open science, our focus here is on improving decision-making for healthcare service delivery using M&S, and the potential for real-world benefit. Figure 1 illustrates the four methods of deployment of a simulation model developed using *SimPy* (Team SimPy. 2020), with an open license. A use case for WebAssembly using Jupyterlite (Jupyterlite. 2024) is discussed in more detail. A list of papers, tutorials, and code artefacts detailing all methods are archived at Monks and Harper (2024a). For additional detail on the methods in 3.1 and 3.2, we refer readers to Monks et al. (2024); Sections 3.3, 3.4, and 4 extend our previous work.

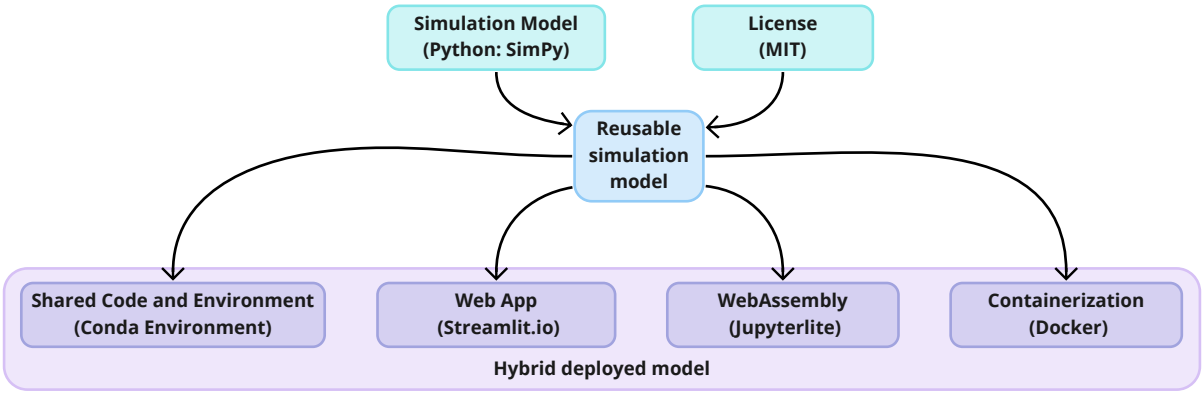

Figure 1: Methods of hybrid model deployment.

3 DEPLOYMENT METHODS

3.1 FOSS and Virtual Environments

Sharing accessible and reusable code requires some basic steps which allows a FOSS model to be available long-term and likely to be at a standard that is reusable in further work (Monks, Harper, and Mustafee 2024). All packages and programs are susceptible to updates, changes, and bugfixes, and FOSS is regularly updated, often by a community of contributors. Changes to code can result in changes to outputs. For deploying a model, a simple approach to tackling software or code deployment challenges is to detail the necessary software packages, versions, and sources that are needed for the model to run without conflicts.

The majority of published healthcare DES studies that share their model do not use formal dependency management (Monks and Harper 2023a). Python dependency management tools like pip virtual environments, conda environments, and poetry offer ways to record and manage these dependencies efficiently, making it easier for others to replicate the setup with minimal effort. Code and virtual environment files are committed to GitHub (or an alternative repository), which can then be downloaded and run locally on a laptop or desktop. The environment manager is installed directly on the users' machine and will attempt to install the correct operating system-specific version of software packages directly. However, conflicts between packages can still arise: different packages may have incompatible dependencies; if the environment is not fully specified then dependencies of key dependencies may change over time, and package managers won't mitigate against differences in outputs across different operating systems. The relative merits are:

- Updates can be pushed to the repository and the latest version can be used.
- Users can view the source code, modify parameters, and understand how the model works in detail.
- An accurate virtual environment ensures that all dependencies are correctly installed and versions are matched, reducing the risks of missing dependency errors that will require solving.
- As the model is used locally, it is possible to run parallel replications, for example using a laptop with multiple virtual cores, reducing run time.
- A user-friendly interface can be provided, to run in a browser locally. These allow users to modify parameters and run simulations, turning complex models into interactive web pages, although the interface will need to be launched.

The trade-offs are:

- Requires users to have some technical knowledge, including understanding of version control systems (like Git) and how to set up and manage virtual environments.
- Users must clone the repository and set up the virtual environment, which can be an obstacle for less technical users.
- Updates to the model may mean that dependencies may change during a project, requiring some re-work of environment files.
- While a user-interface increases accessibility, launching the model interface from a command line may be a barrier for users with limited technical expertise.

3.2 Web Applications

Online deployment of computer models provides significant advantages over the use of executable, shared code for non-technical users. Running code for healthcare DES models requires specialist skills, while web apps offer a user-friendly interface that increases research accessibility to healthcare professionals via a shareable URL. When you run a Streamlit web app, the framework starts a web server and serves a frontend app, dynamically constructed by the Python Streamlit script, using a server-client architecture. Streamlit is server-based - each time there is a user interaction, the server is notified and the whole application is re-run on the server, updating the web page. Monks and Harper (2023b) and Smith and Schneider (2020) provide comprehensive worked examples for structuring code for DES models using Streamlit for Python (Streamlit.io. 2022) and Shiny for R (shinyapps.io. 2024). The model module using any DES package (e.g. SimPy (Team SimPy. 2020), Salabim (van der Ham 2018)) can be structured for interchangeable use with web app software.

Web app software such as Streamlit or Shiny have their own free tier of hosting. For example, Streamlit Community Cloud is a free hosting service for Streamlit apps which provides a robust and easy way to share simulation models to healthcare users. It only needs to be given access to a GitHub repository and it copies the code, builds the app and deploys to a user-customised URL. The relative merits are:

- Provides a graphical interface that is accessible via a web browser, making it easy for non-technical users to interact with the simulation.
- Users can access the simulation immediately without setting up a development environment.
- Can handle multiple users simultaneously, depending on the hosting setup.
- More complex cloud-based solutions are possible to scale deployment, but require ongoing costs.
- Enables early deployment during the model build process to support validation, design, and development decisions by healthcare stakeholders.

The trade-offs are:

- Streamlit apps deployed on Community Cloud can struggle with scaling to handle a high number of concurrent users. If too many users try to run a Streamlit app simultaneously, it may lead to performance issues such as slower response times or timeouts due to the server's limited resources. This can affect the user experience negatively, making the app appear temporarily less responsive.
- If apps are not used for an extended period then the next user that accesses the URL then faces a short wait while the app initializes.
- Paralellization may not be possible, for example Streamlit's community cloud deployment limits model execution to a single CPU, meaning model run time can be slow.
- Users may not be able to modify the simulation code or parameters beyond what the UI allows. This typically means that a web app is bespoke to a single service, and is reusable within that service. However, the source code can be modified and a new bespoke URL can be redeployed.
- Plans for model/app maintenance need to be considered.

3.3 WebAssembly

Several of the disadvantages of running simulations on the web can be overcome by running a Python model entirely in the client's web browser using WebAssembly: a compact, binary instruction language that runs with near-native performance. A key distribution is Pyodide: the CPython interpreter compiled to WebAssembly using emscripten. A number of key packages from the Python scientific stack have also been compiled to WebAssembly and are included in the distribution e.g. NumPy, Pandas, and Matplotlib. Pure Python packages, (e.g. SimPy or Salabim) can also be installed at runtime and run against Pyodide.

Jupyterlite (Jupyterlite. 2024) is a WebAssembly build of the popular Jupyter IDE and runs in a user's browser. It serves static content and uses either the Pyodide or xeus-Python kernels to execute Python code. This means it is possible to build interactive webpages on sites such as GitHub pages that contain simulation models in Jupyter notebooks (JupyterLab. 2024). As JupyterLite serves static content and model computation happens in the client's web browser, users do not need to rely on limited-capacity or costly hosting sites. In comparison, server-reliant cloud-based services like mybinder.org can be slow to build the environment, especially with infrequent access, and have limited resource allocations.

In Jupyterlite, notebooks are downloaded to the local machine from a URL, and are stored in the browser's cache. The user can make changes to the model or create new files, and these persist until the browser cache is cleared. The relative merits are:

- As all the components are on the client side, once all resources are loaded from web servers the model can run offline.
- Model parameters/data are not sent to any remote servers and are only processed locally within users' machines. Without the need to manage servers, the security risks associated with server vulnerabilities are mitigated. Similarly, files can be downloaded and output data can be safely exported locally.
- Without the need for server side computation, there are no costs associated with server maintenance and scaling.
- Unlike environments that require building and launching a server, Jupyterlite has no server startup time and launches instantly.
- As it runs in the browser, the simulation is accessible on any device without needing specific software installations. This provides an interactive web application with the computational capabilities of Python but no need to install Python or any libraries.
- WebAssembly can be potentially suitable for computationally intensive simulations. However, it is limited by the client's machine resources (CPU, memory) and the browser's capabilities.
- For M&S reuse, a link to a JupyterLite model with all notebooks, datasets and libraries configured can make it very easy for users to get started without setup barriers.
- Every user accesses the same software environment directly in their browser, ensuring consistency and reliability.

The trade-offs are:

- JupyterLite kernels (e.g. Pyodide) include a limited set of Python packages compiled to WebAssembly. This means not all Python libraries are available, and not all (but most) of the features of JupyterLab and Classic Notebooks are supported.
- The simulation's performance and capability might be limited by the user's browser and hardware.
- JupyterLite, being browser-based, does not support traditional multithreading due to its architecture.
- Although JupyterLite is currently being developed by core Jupyter developers, the project is still unofficial as of the time of writing.
- The deployed model in Jupyter notebooks will display a mixture of exposed code, markdown, and outputs such as results, tables, and charts. Changing parameters will require interacting with code

directly. The notebooks therefore need to be designed for clarity, interactivity, and aesthetic appeal. Stlite (Stlite. 2024) offers a user-friendly alternative, with a web app user interface using Streamlit.

3.4 Containerization

Deploying models via containerization robustly overcomes issues of dependency management. Containers refer to lightweight, portable, self-contained software units that package together an application, such as a simulation model, and all its software dependencies, including libraries, frameworks, data, and system tools. Containers provide a consistent and isolated runtime environment for running applications, ensuring that they behave the same way across different computing environments. Each container operates as a separate entity, with its own file system, networking, and resources, while sharing the host operating system's kernel. Containers enable efficient deployment, scaling, and management of applications, as they can be easily moved between different computing environments without compatibility issues. Containers support reproducibility, as the exact same environment is used across all users' machines. Containerization provides an excellent solution for deploying interactive applications such as Streamlit and Shiny. Wrapping everything in a container such as Docker (Docker. 2024) and posting it to a public (or private) container registry such as DockerHub, reduces the need to recreate the environment and makes the simulation model more accessible for users (Harper, Monks, and Manzi 2023). The relative merits are:

- Although more complex to set up than Jupyterlite/Pyodide, containerization is more versatile.
- Containers package the application and its environment, ensuring it runs consistently across different computing environments.
- The software environment is specified at build time eliminating dependency management issues for a user.
- Containers can be easily scaled and deployed across multiple instances for high availability and load balancing. This makes them ideal for large-scale deployments where control over the environment and dependencies is critical. In comparison, WebAssembly is more suitable for educational purposes, or environments such as healthcare where installation privileges are restricted.

The trade-offs are:

- Requires some technical knowledge to set up and manage containers, and is therefore a less accessible approach than web apps or WebAssembly.
- Requires users to have a container manager such as Docker installed, which might be a barrier for those unfamiliar with containerization technologies.
- Docker requires commercial licensing for healthcare organisations.

4 APPLIED EXAMPLE USING WEBASSEMBLY

WebAssembly has potentially wide application in healthcare for model reuse, but we know of no examples in practice. We detail a demonstration DES model that can be run in a user's browser using WebAssembly. The model can be accessed and run via this URL: https://pythonhealthdatascience.github.io/stars-simpy-jupterlite. The model is tested in Firefox, Chrome, and Chromium-based browsers. Open the notebook file 01_urgent_care_model.ipynb. The Jupyter notebook can be run cell-by-cell, or all at once. Parameters can be modified in the code. We provide this example as a simplified demonstration model to illustrate approaches to structuring, sharing, and deploying FOSS models.

4.1 Use Case

1. As part of a research study, a simulation model of complexity typically seen in healthcare DES has been built using a Python-based simulation package (here, using *SimPy*).

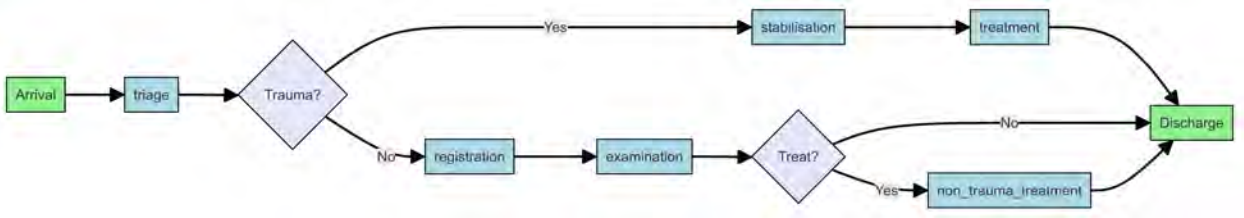

Figure 2: Conceptual flow of the DES model, a US-based urgent care centre, from Nelson (2013).

2. A researcher wishes to share a runnable version of the simulation model with their publication. The code allows others to replicate the simulation results, tables and charts in a paper and allows others to reuse the model.
3. The researcher wants the model to be immediately usable. Users should not need to install Python, *SimPy* or any dependencies.
4. The researcher either wants to reduce load on online open science compute infrastructure (e.g. mybinder.org) or does not want to rely on it.
5. Users may want to use a version of their own data but due to governance, ethics or other reasons *cannot* upload the data to a remote instance of the model.

4.2 Case Study Model

Here we adapt an MIT licensed SimPy (Team SimPy. 2020) terminating DES model of an urgent care department, adapted from Monks and Harper (2023). The model (Figure 2) is an implementation of a stylized U.S.-based urgent care department described in Nelson (2013). Patients arrive to the health clinic between 6am and 12am following a non-stationary Poisson process. A thinning algorithm is employed to generate inter-arrival times. After 12am arriving patients are diverted elsewhere and remaining work-in-process is completed. On arrival, all patients sign-in and are triaged. The health clinic expects two types of patient arrivals: trauma (serious conditions), and non-trauma (minor conditions).

The simulation model provides a framework for decision makers to evaluate the impact of alternative working processes on queues and other efficiency criteria in the urgent treatment health centre. At the end of a model run the average waiting times and resource utilisation are calculated for each of the following activities: (i) Patient triage; (ii) Registration; (iii) Examination (iv) Non-trauma patient treatment (v) Trauma patient treatment. The model also calculates total time in system separately for trauma and non-trauma patients. Confidence intervals are generated using multiple replications. Model logic, baseline model parameters, and details of experimentation and implementation are available here: https://pythonhealthdatascience.github.io/stars-simpy-example-docs/content/front_page.html. Running the default model deployed using Jupyterlite to the URL above will run simple scenario analysis with the following experiments:

- Increasing triage capacity by +1
- Increasing examination capacity by +1
- Increasing non-trauma capacity by +1
- Combining additional triage and additional examination capacity

4.3 Computational Materials

Simulation logic was executed using SimPy v4.1.1 (Team SimPy. 2020) and installed from conda-forge. Data manipulation and visualisation was conducted using NumPy v1.25.2 (van der Walt et al. 2011), Pandas v2.0.3 (McKinney 2011), MatPlotLib v3.8.4 (Hunter 2007) compiled to WebAssembly, and installed from emscripten-forge (Empscripten-forge. 2024). The model is runnable through Jupyterlite

with a xeus-Python kernel running Python 3.11. All code is archived in Zenodo (Monks and Harper 2024b), and a template for automating the creation of new GitHub repositories is available online at https://github.com/pythonhealthdatascience/stars-simpy-jupterlite.

4.4 Method

The model will run directly in the user's browser without the need to install any components on the local machine. This functionality is achieved using WebAssembly technology, specifically *JupterLite* (Jupyterlite. 2024) and the *xeus-Python* kernel (Xeus-python. 2024). We chose the xeus-Python kernel over Pyodide for this implementation as it is simple to install dependencies at build time from conda-forge, providing robust and straightforward package management.

The first time the URL is clicked, a Jupyter notebook containing the model is downloaded to a user's local machine along with all dependencies (i.e. SimPy) that were pre-installed from conda-forge. The model notebook is stored in the browser's cache. JuypterLite is loaded as an IDE and a user can make changes to the model or create new notebooks. Any changes made are saved and persisted within the browser cache. These stored models and notebooks remain accessible until the browser cache is cleared by the user, ensuring that all work is retained across sessions.

This approach ensures an efficient user experience, combining local computation with the convenience of a web-based interface. WebAssembly eliminates the need for users to manage local installations and dependencies, making advanced healthcare modeling more accessible and user-friendly. Additionally, ensuring that all computations are performed locally enhances data security by keeping sensitive healthcare information within the user's own browser environment.

4.5 Results: A Reproducible Pipeline

The notebook included in our method can be used as a reproducible pipeline. For example, if all cells in the notebook are run in sequence the final cell generates the latex used in Table 1 to report the mean waiting time results (in minutes) of six urgent care activities across five experiments.

Table 1: Simulation results that can be verified by our example reproducible pipeline.

Mean waiting time (mins)	**base**	**triage+1**	**exam+1**	**treat+1**	**triage+exam**
Triage	32.56	1.26	32.56	32.56	1.26
Registation	104.69	131.82	104.69	104.69	131.82
Examination	23.36	24.44	0.14	23.36	0.14
Non-trauma treatment	130.73	133.09	144.50	2.15	147.81
Trauma stabilisation	166.98	189.67	166.98	166.98	189.67
Trauma treatment	14.39	14.77	14.39	14.39	14.77

5 CONCLUSION

In this paper, we have examined current technologies to enhance the deployment of healthcare DES models developed in Python, aiming to enhance functionality and usability. Different hybrid methods present opportunities and limitations for reuse by healthcare services. For models intended for light use, developing a front-end user interface in Streamlit, deployed to Streamlit Community Cloud (or other tools such as Shiny), provides high usability. This method allows models to be easily accessible via a URL, and is particular beneficial for small teams without extensive technical support. More costly cloud-based servers can also be considered.

For applications where consistency, long-term usage, or large user bases are anticipated, containerization could be considered. This method ensures reproducibility across different computing environments, and simplifies updates and maintenance. It may suit larger teams that require stable, scalable deployment.

Where data security is an issue, such as handling sensitive patient data, local model deployment may be required. Using conda environments allows for a controlled setup that maintains dependencies without complex configuration, though users require some technical ability to install and set up. WebAssembly implementation also facilitates local processing, mitigating the risk of data breaches by avoiding server-based computations, and eliminates the need for complex software installation processes. JupyterLite notebooks load and become operational nearly instantly, as they don't require server-side interaction. This also removes the need for ongoing server costs, which can be advantageous for organizations with limited budgets. We have demonstrated how Jupyter notebooks can provide a reproducible pipeline; this consistency is important to ensure model credibility with users, and reliable, valid decision-support for healthcare service planning.

JupyterLite additionally supports the use of interactive widgets, allowing users to easily manipulate simulation parameters. For streamlined, interactive user interfaces, *Stlite* (Stlite. 2024) is a stripped-down version of Streamlit which is an alternative to Jupyterlite. This can be particularly suitable for rapid prototyping or where ease-of-use is prioritised over extensive functionality. Additionally, healthcare professionals can access JupyterLite or Stlite from any device that has a web browser via a URL.

By adhering to the 'essential' components specified in Monks et al. (2024), M&S researchers are employing principles of open science such as transparency and scrutiny, increasing the rigor of their work, longevity of their models, and trust in their results. By considering model reuse for a wide audience base, such as those directly impacted by the results of the research, the potential to increase the societal impact of the work increases (Munafò et al. 2017; UNESCO. 2021). The choice of deployment method should be tailored to fit the specific needs of stakeholders and the user-base within a project. For healthcare M&S researchers, a long-standing challenge is translating model results into real-world practice. While many factors contribute to successful model uptake and reuse, each of the hybrid methods described overcomes some of the barriers to model reuse in healthcare by enabling decision-makers to access and use the model. While we have focused on the specific needs of healthcare users, these methods are widely applicable to other domains where knowledge transfer, model reuse and usability are priorities (Kogler and Rauch 2020).

Our applied example provides educational resources to support capacity building in M&S, by providing open, documented code of our demonstration model, and a template to automate the creation of new GitHub repositories using WebAssembly. Future work will expand the approach to other methods and tools, and will investigate the barriers and challenges to the use of WebAssembly in practice for healthcare end users.

ACKNOWLEDGMENTS

TM is supported by the NIHR Applied Research Collaboration South West Peninsula. The views expressed in this publication are those of the author(s) and not necessarily those of the NIHR or the Department of Health and Social Care. This work was supported by the Medical Research Council [MR/Z503915/1].

REFERENCES

Anagnostou, A., D. Groen, S. J. Taylor, D. Suleimenova, N. Abubakar, A. Saha *et al*. 2022. "FACS-CHARM: A Hybrid Agent-Based and Discrete-Event Simulation Approach for Covid-19 Management at Regional Level". In *Proceedings of the 2022 Winter Simulation Conference (WSC)*, 1223–1234 https://doi.org/10.1109/WSC57314.2022.10015462.

Auer, S., N. A. Haeltermann, T. L. Weissgerber, J. C. Erlich, D. Susilaradeya, M. Julkowska *et al*. 2021. "Science Forum: A Community-led Initiative for Training in Reproducible Research". *eLife* 10:e64719.

Bartz-Beielstein, T., M. Dröscher, A. Gür, A. Hinterleitner, T. Lawton, O. Mersmann *et al*. 2021. "Optimization and Adaptation of a Resource Planning Tool for Hospitals Under Special Consideration of the COVID-19 Pandemic". In *2021 IEEE Congress on Evolutionary Computation (CEC)*, 728–735.

Boyle, L. and M. Mackay. 2022. "A Reusable Discrete Event Simulation Model for Improving Orthopedic Waiting Lists". In *2022 Winter Simulation Conference (WSC)*, 973–984 https://doi.org/10.1109/WSC57314.2022.10015512.

Boyle, L. M., A. H. Marshall, and M. Mackay. 2022. "A Framework for Developing Generalisable Discrete Event Simulation Models of Hospital Emergency Departments". *European Journal of Operational Research* 302(1):337–347.

Brailsford, S. C., T. B. Bolt, G. Bucci, T. M. Chaussalet, N. A. Connell, P. R. Harper *et al*. 2013. "Overcoming the Barriers: a Qualitative Study of Simulation Adoption in the NHS". *Journal of the Operational Research Society* 64(2):157–168.

Chalk, D., S. Robbins, R. Kandasamy, K. Rush, A. Aggarwal, R. Sullivan *et al*. 2021. "Modelling Palliative and End-of-life Resource Requirements During COVID-19: Implications for Quality Care". *BMJ open* 11(5):e043795.

Collins, A. J., F. Sabz Ali Pour, and C. A. Jordan. 2023. "Past Challenges and the Future of Discrete Event Simulation". *The Journal of Defense Modeling and Simulation* 20(3):351–369.

Crema, M. and C. Verbano. 2021. "Simulation Modelling and Lean Management in Healthcare: First Evidences and Research Agenda". *Total Quality Management & Business Excellence* 32(3-4):448–466.

Crowe, S., L. Grieco, T. Monks, B. Keogh, M. Penn, M. Clancy *et al*. 2023. "Here's Something we Prepared Earlier: Development, Use and Reuse of a Configurable, Inter-disciplinary Approach for Tackling Overcrowding in NHS Hospitals". *Journal of the Operational Research Society*:1–16.

Docker. 2024. "Docker Builds: Now Lightning Fast Announcing Docker Build Cloud general availability". https://www.docker.com/, accessed 18th April 2024.

Empscripten-forge. 2024. "Empscripten-forge". https://repo.mamba.pm/emscripten-forge, accessed 18th April 2024.

Ferro, M. V., A. Lee, C. Pritchard, C. M. Barton and M. A. Janssen. 2022. "Containerization for Creating Reusable Model Code". *Socio-Environmental Systems Modelling* 3:18074–18074.

Fletcher, A. and D. Worthington. 2009. "What is a 'Generic' Hospital Model?—A Comparison of 'Generic' and 'Specific' Hospital Models of Emergency Patient Flows". *Health Care Management Science* 12:374–391.

Harper, A., T. Monks, and S. Manzi. 2023. "Deploying Healthcare Simulation Models Using Containerization and Continuous Integration". *OSF Preprints*.

Harper, A., T. Monks, R. Wilson, M. T. Redaniel, E. Eyles, T. Jones *et al*. 2023. "Development and Application of Simulation Modelling for Orthopaedic Elective Resource Planning in England". *BMJ open* 13(12).

Hassanzadeh, H., J. Boyle, S. Khanna, B. Biki, F. Syed, L. Sweeney *et al*. 2023. "A Discrete Event Simulation for Improving Operating Theatre Efficiency". *The International Journal of Health Planning and Management* 38(2):360–379.

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

Hussain, M., N. Masoudi, G. Mocko, and C. Paredis. 2022. "Approaches for Simulation Model Reuse in Systems Design—A review". *SAE International Journal of Advances and Current Practices in Mobility* 4(2022-01-0355):1457–1471.

Jahangirian, M., S. J. Taylor, T. Young, and S. Robinson. 2017. "Key Performance Indicators for Successful Simulation Projects". *Journal of the Operational Research Society* 68(7):747–765.

JupyterLab. 2024. "JupyterLab". https://github.com/jupyterlab/jupyterlab, accessed 18th April 2024.

Jupyterlite. 2024. "Wasm Powered Jupyter". https://jupyterlite.readthedocs.io/en/stable/, accessed 18th April 2024.

Kar, E., M. Fakhimi, C. Turner, and T. Eldabi. 2024. "Hybrid Simulation in Healthcare: A Systematic Exploration of Models, Applications, and Emerging Trends". *Journal of Simulation*:1–19.

Kogler, C. and P. Rauch. 2020. "Game-based Workshops for the Wood Supply Chain to Facilitate Knowledge Transfer". *International Journal of Simulation Modelling* 19(3):446–457.

Lin, T. Y., G. Shi, C. Yang, Y. Zhang, J. Wang, Z. Jia *et al*. 2021. "Efficient Container Virtualization-based Digital Twin Simulation of Smart Industrial Systems". *Journal of cleaner production* 281:124443.

Long, K. M. and G. N. Meadows. 2018. "Simulation Modelling in Mental Health: A Systematic Review". *Journal of Simulation* 12(1):76–85.

Mark W. Isken, O. T. A. and Y. F. Roumani. 2023. "Queueing Inspired Feature Engineering to Improve and Simplify Patient Flow Simulation Metamodels". *Journal of Simulation* 0(0):1–18.

McKinney, W. 2011. "Pandas: A Foundational Python Library for Data Analysis and Statistics". *Python for High Performance and Scientific Computing* 14(9):1–9.

Monks, T. and A. Harper. 2023a. "Computer Model and Code Sharing Practices in Healthcare Discrete-event Simulation: a Systematic Scoping Review". *Journal of Simulation* 0(0):1–16.

Monks, T. and A. Harper. 2023b. "Improving the Usability of Open Health Service Delivery Simulation Models Using Python and Web Apps". *NIHR open research* 3:48.

Monks, T. and Harper, A. 2023. "Zenodo: Towards Sharing Tools and Artifacts for Reusable Simulation: Example Enhanced Documentation for a 'Simpy' Model". https://doi.org/10.5281/zenodo.10054063, accessed 13th March 2024.

Monks, T. and Harper, A. 2024a. "Zenodo: List of STARS Publications and Artefacts." https://doi.org/10.5281/zenodo.11092744. https://doi.org/10.5281/zenodo.11092744, accessed 13th March 2024.

Monks, T. and Harper, A. 2024b. "Zenodo: Simpy JupyterLite Template". https://doi.org/10.5281/zenodo.10987817, accessed 13th March 2024.

Monks, T., A. Harper, and N. Mustafee. 2024. "Towards Sharing Tools and Artefacts for Reusable Simulations in Healthcare.". *Journal of Simulation* 0(0):1–16.

Moreau, D., K. Wiebels, and C. Boettiger. 2023. "Containers for Computational Reproducibility". *Nature Reviews Methods Primers* 3(1):50.

Munafò, M. R., B. A. Nosek, D. V. Bishop, K. S. Button, C. D. Chambers, N. Percie du Sert *et al*. 2017. "A Manifesto for Reproducible Science". *Nature human behaviour* 1(1):1–9.

Mustafee, N., A. Harper, and M. Fakhimi. 2022. "From Conceptualization of Hybrid Modelling & Simulation to Empirical Studies in Hybrid Modelling". In *2022 Winter Simulation Conference (WSC)*, 1199–1210 https://doi.org/10.1109/WSC57314.2022.10015400.

Nelson, B. 2013. *Foundations and Methods of Stochastic Simulation: A First Course*. London: Spinger.

OMF. 2023. "The Modeling Foundation". https://www.openmodelingfoundation.org/standards/reusability/, accessed 8th May 2024.

Pouwels, X. G. and H. Koffijberg. 2024. "Introducing Open Science in Teaching Health Economic Modelling". *PharmacoEconomics-Open*:1–11.

Pouwels, X. G., C. J. Sampson, R. J. Arnold, M. D. Janodia, R. Henderson, M. Lamotte *et al*. 2022. "Opportunities and Barriers to the Development and Use of Open Source Health Economic Models: A Survey". *Value in health* 25(4):473–479.

Robinson, S., R. E. Nance, R. J. Paul, M. Pidd and S. J. Taylor. 2004. "Simulation Model Reuse: Definitions, Benefits and Obstacles". *Simulation modelling practice and theory* 12(7-8):479–494.

Roy, S., S. Prasanna Venkatesan, and M. Goh. 2021. "Healthcare Services: A Systematic Review of Patient-centric Logistics Issues using Simulation". *Journal of the Operational Research Society* 72(10):2342–2364.

shinyapps.io. 2024. "Easy Web Apps for Data Science Without the Compromises". https://shiny.posit.co/, accessed 18th April 2024.

Smith, R. and P. Schneider. 2020. "Making Health Economic Models Shiny: A Tutorial". *Wellcome Open Research* 5(69).

Stlite. 2024. "Serverless Streamlit App Platform". https://edit.share.stlite.net/?sampleAppId=intro, accessed 30th April 2024.

Streamlit.io. 2022. "Streamlit. The Fastest Way to Build and Share Data Apps". https://streamlit.io/, accessed 18th October 2023.

Taylor, S. J., T. Eldabi, T. Monks, M. Rabe and A. M. Uhrmacher. 2018. "Crisis, What Crisis–Does Reproducibility in Modeling & Simulation Really Matter?". In *Proceedings of 2018 winter simulation conference (WSC)*, 749–762.

Team SimPy. 2020. "SimPy 3.0.11". https://simpy.readthedocs.io/en/latest/index.html, accessed 18th October 2023.

The Turing Way Community. 2022. "The Turing Way: A Handbook for Reproducible, Ethical and Collaborative Research." https://doi.org/10.5281/zenodo.7470333. https://doi.org/10.5281/zenodo.7470333, accessed 13th March 2024.

Tyler, J. M., B. J. Murch, C. Vasilakis, and R. M. Wood. 2023. "Improving Uptake of Simulation in Healthcare: User-driven Development of an Open-Source Tool for Modelling Patient Flow". *Journal of Simulation* 17(6):765–782.

UNESCO. 2021. "Recommendation on Open Science". https://unesdoc.unesco.org/ark:/48223/pf0000379949, accessed 2nd February 2024.

van der Ham, R. 2018. "Salabim: discrete event simulation and animation in Python". *Journal of Open Source Software* 3(27):767.

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.

Van Lissa, C. J., A. M. Brandmaier, L. Brinkman, A.-L. Lamprecht, A. Peikert, M. E. Struiksma *et al*. 2021. "WORCS: A Workflow for Open Reproducible Code in Science". *Data Science* 4(1):29–49.

Vázquez-Serrano, J. I., R. E. Peimbert-García, and L. E. Cárdenas-Barrón. 2021. "Discrete-event Simulation Modeling in Healthcare: A Comprehensive Review". *International journal of environmental research and public health* 18(22):12262.

Xeus-python. 2024. "A Jupyter Dernel for Python Based on The Native Implementation of the Jupyter Protocol Zeus". https://github.com/jupyter-xeus/xeus-python, accessed 30th April 2024.

AUTHOR BIOGRAPHIES

ALISON HARPER is a lecturer in operations and analytics at the Centre for Simulation, Analytics and modeling, University of Exeter. Her research interests include applied health and social care modeling and simulation, real-time simulation, and reusable modeling in healthcare. Her email address is a.l.harper@exeter.ac.uk.

THOMAS MONKS is an Associate Professor of Health Data Science at University of Exeter Medical School. His research interests include open science for computer simulation, urgent and emergency care, and real-time discrete-event simulation. His email address is t.m.w.monks@exeter.ac.uk and his website is https://arc-swp.nihr.ac.uk/about-penarc/people/tom-monks/.

NAVONIL MUSTAFEE is Professor of Analytics and Operations Management at the University of Exeter Business School, UK. His research focuses on modeling and simulation methodologies, including hybrid modeling and real-time simulation, and their application in healthcare, supply chain management, circular economy and climate change adaptation and resilience. He is a Joint Editor-in-Chief of the Journal of Simulation (UK OR Society journal). His email address is n.mustafee@exeter.ac.uk.