UNIVERSITY OF
CAMBRIDGE

MLMI Team 2

Hospital Optimisation Using Emulation

Authors:	*Candidate Number:*
Ryan Crowley	J920R
Alejandro Santorum	J902C
Sotirios Vavaroutas	SV455
Benedict King	J902G

January 24, 2022

Contents

1 Introduction **1**

2 Simulation Setup **1**
2.1 Selecting a Ward to Emulate 1
2.2 Automating the Simulator 1

3 Exploring the Data **2**
3.1 Number of Simulation Steps Chosen 2
3.2 Inputs & Outputs Used 2
3.3 Visualisation of the Inputs & Outputs 2

4 Emulation Methods **3**
4.1 Gaussian Processes vs More Interpretable Regression Models 3
4.2 Ensembling Emulators 5

5 Sensitivity Analysis **6**

6 Optimisation **7**

7 Conclusion **8**

A Appendix **11**
A.1 Neural Networks Hyperparameter Tuning 11
A.2 Poisson Regression 12

1 Introduction

In real-world situations, hospital resources are often scarce [1]. Since patient flow affects multiple measures of healthcare performance, such as occupancy and wait time, patient flow analysis can improve the efficiency of healthcare systems. However, despite the crucial importance of comprehensive analysis of resource management, quality control of hospitals is often still not a data-driven process [2]. Moreover, although the hype around machine learning model applications within healthcare continues to increase [3], effective applications of artificial intelligence in medicine are still relatively rare and prone to biases [4]. In this project, machine learning emulation methods are applied to a healthcare simulator in order to gain a deeper understanding of patient flow dynamics within a given hospital setting.

The employed simulator [5] is accessible here [6], and it models an entire hospital by combining discrete event simulation (DES), a common method for simulating patient flow between wards of a hospital according to treatment needs, with queue theory and graph theory. The hospital is described by a set of wards, each configured with a different number of beds, staff, and resources. Every ward has its own queue policy, resource/staff distribution, and overflow procedures. The simulator is not extremely expensive to run, but it is implemented in JavaScript with inputs parameters that are designed to be specified by hand. Thus, the automation of data acquisition will be key to the realisation of the project.

Given that the simulator is a "black box" in that it's unclear how variation in the input variables directly affects variation in the output variables, we aimed to use emulation methods with sensitivity analysis to investigate hospital flow. These emulation models can help provide surrogate data in situations where computational resources are severely limited. Hence, a variety of different models are used for this emulation task ranging from simple models like linear regression to more complex models like Gaussian Processes, neural networks, and gradient boosted decision trees.

2 Simulation Setup

2.1 Selecting a Ward to Emulate

The original simulator [6] includes multiple connections between hospital wards. However, to keep analysis feasible, we confined our explorations to the emergency department. We chose to focus on the emergency room because its severe time constraints and variable patient population [7] provide the most interesting case study for assessing patient flow. In contrast to other departments like cardiology and surgery, whose planned appointments make the prediction and control of occupancy and queue length relatively straightforward, the emergency department must respond to substantial fluctuations in patient flow. Moreover, patients admitted to the emergency room are often in critical situations and resources can be scarce in surge scenarios, so there lies significant interest in emulating the occupancy of emergency rooms.

2.2 Automating the Simulator

The simulator was initially designed to be used interactively, with user-inputted variables for each individual simulation run through a HTML-based interface. Given that the simulator came with no inherent data exporting functionalities built-in, gathering data naively would have been extremely time-consuming. Hence, we wrote scripts to export the data in a format compatible with GPy [8] and other Python libraries. We extended the simulator's implementation by writing JavaScript code to both run the simulation automatically for a provided range of input values and save the data

after each simulation run. This involved adding a set of loops that sequentially alter the simulator's inputs and invoke its main runtime function with the new input values, followed by exporting the generated data[1]. The final parsed file contains all data useful for our emulation, matching inputs with their respective outputs generated from the simulator[2].

We then carried out a literature review of all the works referencing the paper detailing the simulator [5]. This search returned 13 articles, from which we assessed the relationships between the article and the original simulator paper. We found that, to the best of our knowledge, no deep analysis of this simulation model has been performed to date. Consequently, we added a new button to the simulator's HTML interface invoking the auto-run, providing the possibility of easier data acquisition for future researchers if we were to make our code open source.

3 Exploring the Data

3.1 Number of Simulation Steps Chosen

To generate our dataset, we ran the simulator for 48 steps (corresponding to 48 hours). The model was constructed so that 1 patient is expected to arrive every hour. The hospital simulation begins with a "cold start" each time with an occupancy of zero, so the data for the initial 24 hours are not as representative of real-life scenarios as the data for the final 24 hours. In a real-world scenario, it is the norm for hospitals to carry backlog of patients from a previous day [9], so opting for a 48-hour simulation instead of a 24-hour one provides us with a more realistic representation of a real-life hospital.

3.2 Inputs & Outputs Used

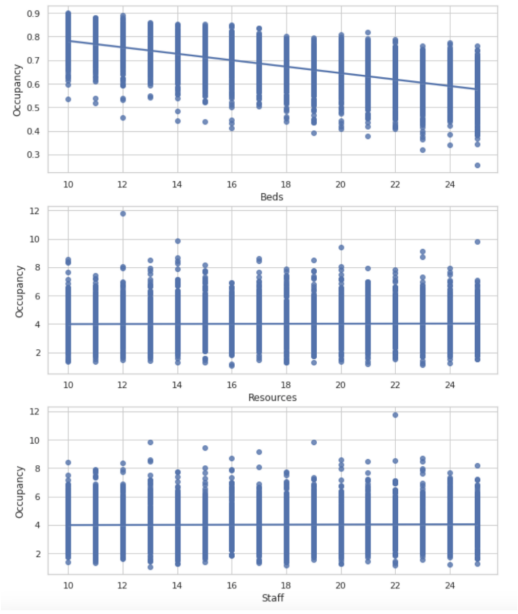
Of the various inputs one can alter when configuring a hospital setup, we chose to focus our analysis on the most salient characteristics of an emergency room setup: the number of beds, the number of staff, and the amount of resources. Moreover, we also assessed various queue policies which included first-come first-served, highest need first, and lowest need first. To generate data for our emulation we ran the simulator under various conditions with inputs ranging from 10 to 25 for the number of beds, resources, and staff in the ward, while also generating data for all three queue policies possible. The simulator was run for all possible permutations of model inputs within these boundaries. At each simulator run, a new set of patients is automatically generated. The stochasticity of the simulator stems from multiple sources, with the most prominent source being that a different subset of patients with varying disease complexity are generated for each new run of the model. We determined the main outputs of interest from each run of the simulator to be the average queue length of patients waiting to be admitted to the hospital and the average ward occupancy.

3.3 Visualisation of the Inputs & Outputs

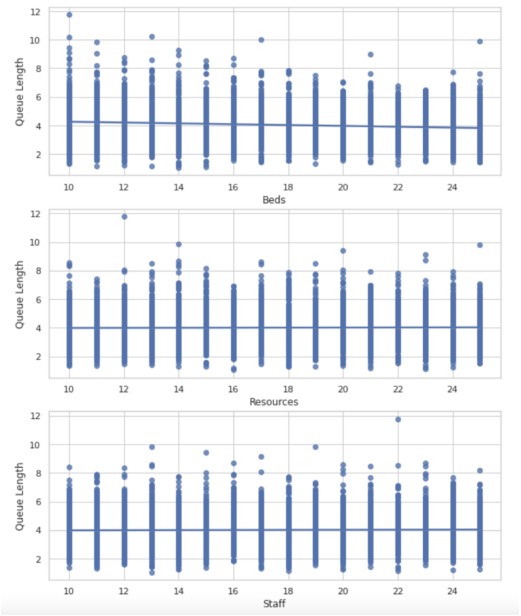
With the points mentioned above in mind, we started exploring the relationships between the inputs and output and the effects that the different queue policies had on the outputs. These relationships are visualised below with lines of best fit.

[1]Each run's data was outputted to a CSV file and, to overcome Chrome's limit on downloading a large number of files, a tailor-made AppleScript was written to press "Allow" repeatedly, automating the data generation process.

[2]All generated files were parsed using a purpose made Python script.



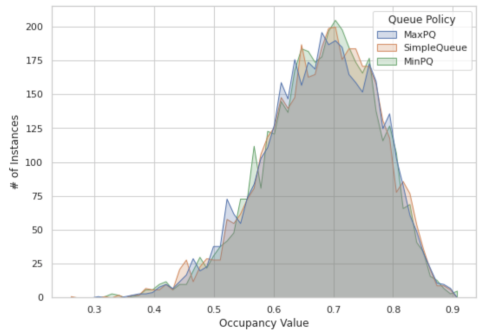
(a) Emergency Occupancy



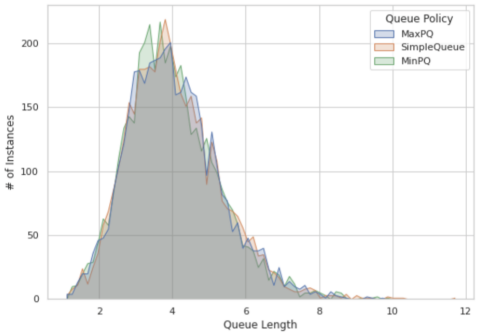
(b) Queue Length

Figure 1: Relationships between inputs and outputs

In order to determine whether the various queue policies (First-Come First-Served, Highest Need First, Lowest Need First) substantially impacted the target outputs of occupancy and queue length, histograms were generated to compare the distributions of occupancy and queue values for each of these three queue policies, and are plotted in Figure 2.



(a) Effect on Emergency Occupancy



(b) Effect on Queue Length

Figure 2: Effect of queue policies on occupancy and queue length

The distributions of occupancy and queue length for all three queue policies strongly mirror each other. This indicates that in our formulations of the model structure, queue policy doesn't affect our outputs in a substantial way. Hence, given the negligible impact of queue policy, data from only one of the policies ("First-Come First-Served") was utilised for all further analyses.

4 Emulation Methods

4.1 Gaussian Processes vs More Interpretable Regression Models

Our initial emulation experiments used Gaussian Process regression, with a squared error kernel and tunable exponential coefficient and lengthscale hyperparameters. The benefit of GPs is that they

do not assume a specific curve shape, unlike many of the parametric models discussed later in this report. Therefore, in the absence of prior expectation about the shape of relationship between the inputs and percentage occupancy/queue length, GPs provide a robust way of exploring the data.

The fit of this GP emulator to the acquired data can be seen in Figure 3 for a sample of data points. The entire space has many points and so most are not plotted. While the fits are quite good, they assume that the inputs are in continuous space when they are actually discrete, and they explain a significant proportion of the relationship as noise. Thus, a more case-specific approach might provide better fits. With this in mind, the percentage occupancy appears to follow a linear trend, suggesting that a simple ordinary least squares (OLS) fit may be suitable. In contrast, queue length follows an inverse or exponential decay shape, and so an OLS fit would not work well here, but a Poisson regression could perform well, particularly as it also accounts for integer valued inputs.

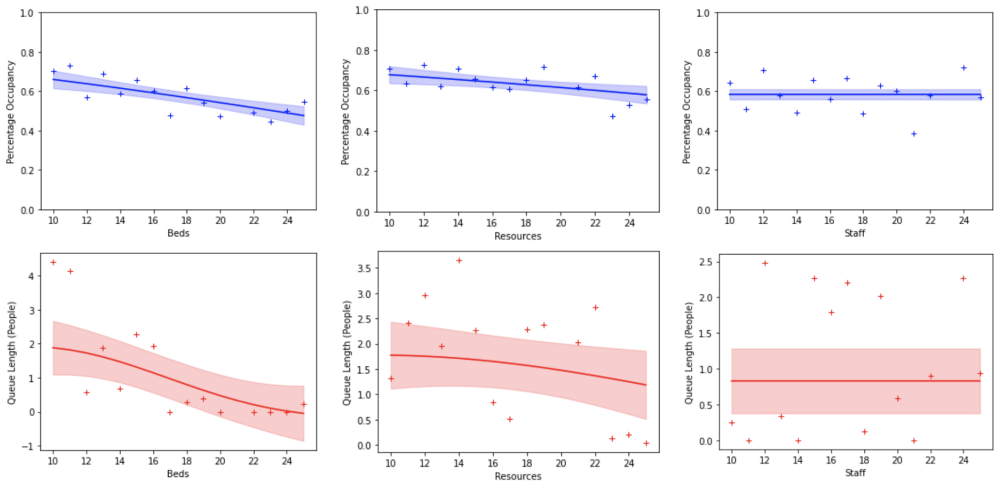


Figure 3: An initial exploration of the data to observe the relationships between the percentage occupancy and queue lengths with the inputs, fitted using Gaussian Process regression. For each row, only a cross-section of the fit for one variable was plotted whilst the others were fixed at 15.

The benefit of OLS and Poisson regression is that they are both able to provide significant insight into the influences of each parameter, in stark contrast to "black box" models. For example, the OLS fit of percentage occupancy gave the coefficients of the beds, resources and staff to be -1.7%, -0.0029% and -0.016% respectively, indicating that increasing the number of beds by one gives approximately a 100 times greater reduction in percentage occupancy than the next most important input, assuming that a linear fit is valid. As discussed above, the linear fit is inappropriate for the queue length (as can be seen from Figure 3), and thus Poisson regression was utilised as an explainable modeling technique[3]. Unlike OLS, Poisson regression coefficients translate to a percentage reduction in the "frequency" of the output per unit change in a specified input[4], and they are well suited to modeling relationships between inputs and outputs that take discrete values. Hence, Poisson regression is valid for this investigation even if we drop our simplification of continuous input space. The Poisson fit therefore predicts a 2.8%, 0.0021%, and 0.023% reduction in queue length per bed, resource and staff member respectively, once again determining beds to be the most influential input by a significant margin.

Nevertheless, one major shortcoming of using fixed shape models like OLS and Poisson regression is that they assume that the outputs follow the same relationship throughout the entire space which can be problematic in certain cases. For example, we can guarantee that there is a set of inputs that lead to 100% occupation, and thus there will be a large domain that gives the same occupancy

[3]See Appendix A.2 for details and justification.

[4]Specifically, for coefficient β_j of input $x^{(j)}$, the multiplicative factor on the output for an increase in the input by 1 is given by e^{β_j}.

percentage output. This directly contradicts the linear fit, and we observed this trend in a subset of inputs sampled from the simulator.

To address this modelling pitfall, we considered other regression models that are more flexible to different relationships than OLS and Poisson regression, and make fewer assumptions about the relationship than GP regression. The models we explored were random forests (one with Poisson splitting, which accounts for the discrete nature of the inputs), the gradient boosted trees variant of random forests, and a shallow neural network (which provided the optimal fit even for low amount of training data, due to its ability to represent more complex relationships than models with either more prior assumptions or fewer parameters[5]). The correlations of each of the tested emulators with the true values (first row) and each other are plotted in Figure 4.

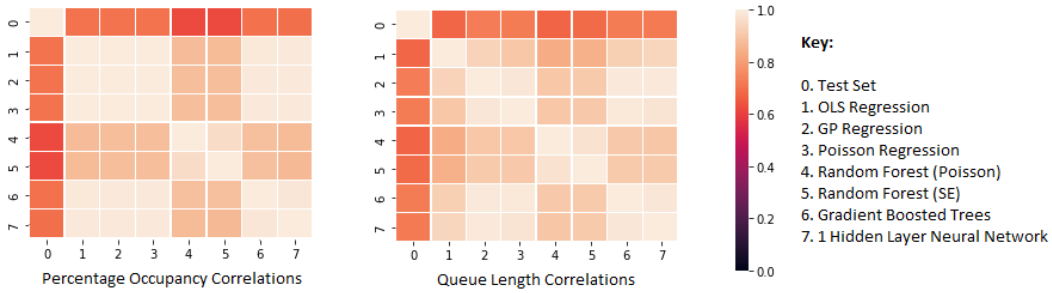


Figure 4: Correlation plots for the different models considered against the true occupancy and queue values of the test-set.

The majority of the models are highly correlated with each other and, consequently, similarly correlated with the true values. This is because the flexible models tend towards the same fits as the interpretable ones (OLS for occupancy and Poisson regression for queue length) when the input samples are in the range giving input/output relationships as linear and exponential decay-like respectively. However, this trend would not hold with very small and very large inputs, where there becomes saturation at 100% occupancy and 0 length of queue. In these cases, the more flexible models would be expected to retain high correlation with the true values, whilst OLS and Poisson regression would perform worse.

4.2 Ensembling Emulators

Ensembling the individual emulators can lead to improved predictions and a more stable final model [10], resulting from reduced variance in the predictions of the model. Given that we trained and evaluated numerous models in our emulation experiments, there lies significant interest in combining them into an ensemble model that is more flexible to different outputs like a GP (i.e. does not assume a relationship shape), while achieving higher accuracy and robustness than the GP regression alone did. Naively, the ensemble method could make predictions by using the mean of all of the individual models' predictions, but adopting such an approach would leave little room for improving the overall performance of our emulations, as we would be giving too much credence to underperforming models. Consequently, we opted for the following more complex approach instead.

Since some models were particularly successful in predicting the emergency ward occupancy while others worked well in predicting queue length, we opted to compute our ensemble model such that it weighs methods according to their correlations with each of the simulator outputs[6]. In this way, the best-performing models play a more significant role in the ensemble's output, ensuring similar performance of the emulator with the optimal model for each case, without needing to

[5]See Appendix A.1 for details.

[6]As long as the correlation was greater than 0.

decide on an optimal model manually or tune additional hyperparameters. This formulation also reduces the variance of the predictions, making the final model more robust and reliable than the individual contributing models. For instance, the correlation resulting from the ensemble emulation of percentage occupancy is 0.701, and although this is not quite as good as the best-performing value of 0.704 from the Neural Network, it is close and has the advantage of the "wisdom of crowds" effect of ensemble methods that could lead to better generalisation.

5 Sensitivity Analysis

In order to determine how uncertainty in the output of our simulation and emulation models can be allocated to the different inputs, we employed a variance-based sensitivity analysis. We utilised the SALib library [11] to incorporate analysis using Sobol indices with Monte Carlo sampling. SALib was chosen as it utilises a method that minimises computational costs when calculating Sobol indices [12]. Sobol indices were calculated for the simulator as well as all emulation methods, and these can be observed in Figure 5 and Figure 6, which display the first-order (S1) and total-effect (ST) Sobol indices for emergency occupancy and queue length respectively.

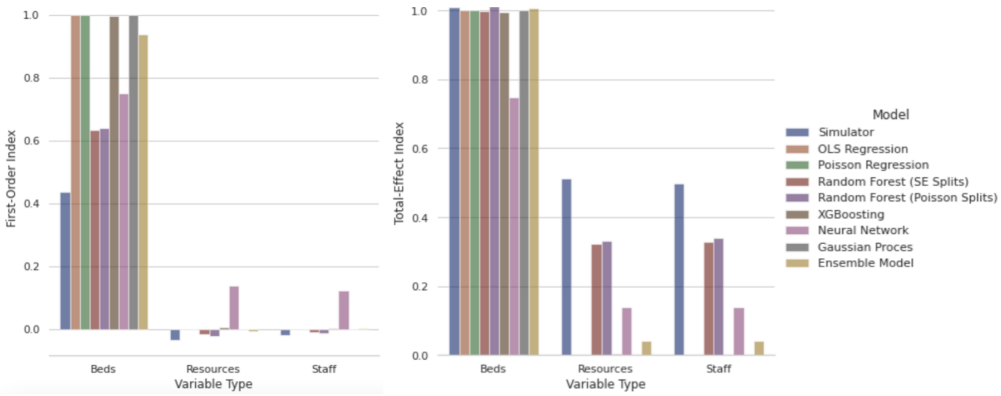


Figure 5: Sobol indices for the occupancy (S1 pictured left; ST pictured right)

Assessing the results from the Sobol indices for occupancy, a variety of interesting findings emerge. First, Sobol indices theoretically should only vary from 0 to 1 but some values within our data are negative while others are greater than one. This occurs as when using Monte Carlo estimation values outside of the strict bounds can arise due to numerical estimation problems. Secondly, beds appears to be the most important variable driving occupancy, shown by its comparatively higher first-order index and total-effect index. This finding aligns with prior findings from the interpretation of regression coefficients. In response to the demonstrated importance of beds within the data, many methods, such as ordinary least squares regression and the Gaussian Processes take on a linear fit that prioritises beds substantially over staff and resources. Although the number of beds is the strongest signal in the data, it only accounts for 40% of the variance in occupancy. Hence, these methods that attribute all of the difference in occupancy to beds show a limited ability to effectively emulate the simulator. Surprisingly, in contrast to the correlation results, random forests now appear to more closely align with the simulated data when considering variance decomposition. This could perhaps be a reflection that the use of correlation to measure closeness of fit is not perfect, and therefore whilst the correlation heatmaps provide information about the fits, they have to be considered alongside the sensitivity analysis to get a better picture.

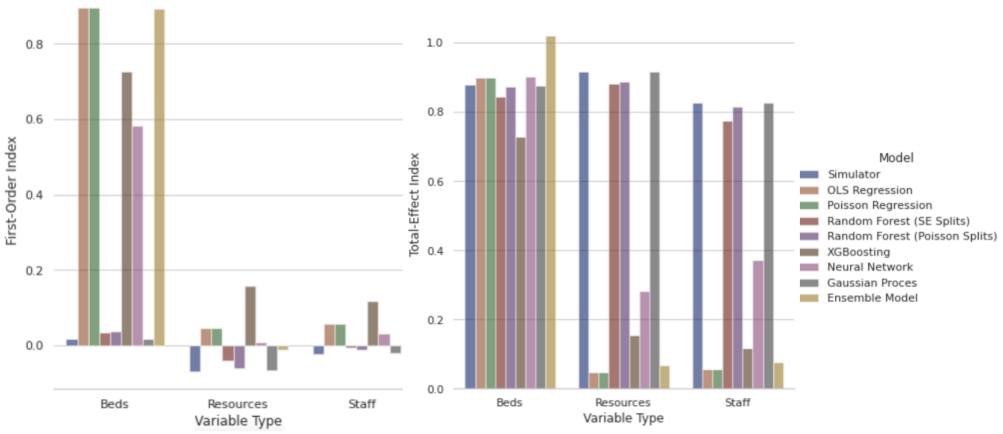


Figure 6: Sobol indices for the queue (S1 pictured left; ST pictured right)

In contrast, the Sobol indices for queue time provide different insights. The simulator has near-zero first-order Sobol indices for beds, resources, and staff. This demonstrates that the complex, non-linear relationship between queue length and our input variables is difficult to disentangle. Hence, there is no substantial signal for these models to attempt to match for first-order indices. Despite this lack of signal, simple regression methods like ordinary least square and Poisson regression account for nearly all variation in the output based upon variation in the number of beds as shown by their high first-order indices. Moreover, assessing the total-effect indices we can see that the random forest models and the Gaussian Process model again most effectively match the ST indices, indicating their effectiveness in emulation of queue length, despite their poorer performances with correlation to the test set.

6 Optimisation

The original motivation behind this project was to investigate how emulation of a hospital simulator could help predict critical hospital functionality through indicators such as the percentage of the emergency ward being occupied and the queue length, with the ultimate goal of optimising hospital efficiency while minimising operating costs. As such, a loss function for this optimisation must satisfy two properties: 1) A longer queue time and a larger percentage occupancy are penalised. 2) Increases in the number of beds, resources and staff are penalised, since these are directly related to the operating costs. With these conditions met, the choice of function is quite flexible as long as its convex, so the following quadratic cost function with input regularisation was chosen:

$$\mathcal{L}(q(\vec{x}), o(\vec{x}), \vec{x}; \alpha, \vec{\beta}) = \frac{1}{N} \sum_{i=1}^{N} \left(\alpha_1 q(\vec{x}_i)^2 + \alpha_2 o(\vec{x}_i)^2 + \begin{bmatrix} \beta_1 & \beta_2 & \beta_2 \end{bmatrix}^T \vec{x}_i \odot \vec{x}_i \right) \tag{1}$$

where $q(\vec{x}_i)$ indicates the output of the ensemble that has been trained on queue data from the training set, and $o(\vec{x}_i)$ similarly representing the output of the percentage occupancy ensemble emulator. In the equation above, the associated costs in the simulator are \$400 per bed, \$150 per resource, and \$150 per member of staff, so we have a greater penalisation term for beds than for the other two inputs ($\beta_1 = (8/3)\beta_2$). Also, as the queue length values are approximately an order of magnitude smaller than the inputs and the occupancy magnitudes are approximately two orders of magnitude smaller than the inputs, the penalisation terms are weighted accordingly. With this in mind we chose values of $\alpha_1 = 0.1$, $\alpha_2 = 1$, $\beta_1 = 0.08/3$, $\beta_2 = 0.01$.

Using these parameters and the Bounded BFGS algorithm for optimisation [13] (in which the inputs are bounded to be strictly positive), the optimal inputs were 14 beds, 1 resources, and 1 staff, giving an average daily queue length of 0.874 and average daily percentage occupancy of 52%. The cost

associated with this configuration is $\$5,900$ per day. This is as would be expected when looking at the plots in Figure 3, as the effect of staff on both variables is predicted to be negligible, and the effect of resources is also far less than beds. Therefore the resources and staff can be reduced far more than beds to keep costs down without adversely effecting ward functions. However, this also highlights a major oversimplification of the simulator since, in reality, reducing the number of staff members means that the fewer staff members are overstretched; thus, their average performance (i.e. number of patients that they can process in a set time-frame) will decrease as a function of the ratio of beds to staff, whereas the simulator appears to treat staff and resources as independent to beds[7].

7 Conclusion

Machine Learning has already made a substantial impact in healthcare transformation [14], and this exploration has explored additional ways that machine learning can impact medicine by analysing how to improve medical efficiency and performance through emulating patient flows. Our work contributes to the research areas of fairly allocating scarce medical resources [15] and improving the interpretability and reproducibility of machine learning for healthcare settings [16].

Although some models in our emulation experiments showed success in predicting hospital occupancy and queue length values, there still appears to be some irreducible error. The majority of this irreducible error likely stems from the randomness within the dataset due to the simulation generating a unique set of patients at each run. This stochasticity was found to be especially significant for queue length, where there weren't strong signals in the data tying the input parameters to the output parameters, as can be seen by the relevant S1 indices being close to zero.

This task also demonstrates the fallibility of a variety of different model types when tasked with emulation. For instance, simpler models, like linear regression, oversimplified the relationships within the data and mistakenly identify all of the variance in occupancy as being due to fluctuations in the number of beds and not due to number of staff or resources. While this can be addressed by turning to more complex models, one must consider the trade-off between interpretability and model complexity. Given that hospitals are making life or death decisions regarding the patients that they are treating, it's crucial that these models are both effective and interpretable.

For future analyses, it would be worth examining additional models for our emulation experiments to identify better-performing models that better correlate with test data and fit Sobol indices more accurately. Nevertheless, this exploration has shed light on the usefulness of hospital simulations, and it has demonstrated how choosing the appropriate setup of beds, resources and staff can have a true impact on hospital performance and patient outcomes. Moreover, if this topic area were to be explored further, we would like to look at how the ensemble emulator can be used in stress-testing situations, where the mean and maximum arrivals per step are expanded greatly to increase demands on the emergency wards. This would be an extremely valuable analysis for hospital management, as crisis scenarios are less common in hospitals; therefore, simulations are of utmost importance in these situations as there exists less historical data to analyse.

[7]Note that staff in the model is used to calculate the rate at which a patient is processed and discharged, and this depends on the number of staff only, without factoring in staff exhaustion.

References

[1] Ezekiel J. Emanuel, Govind Persad, Ross Upshur, Beatriz Thome, Michael Parker, Aaron Glickman, Cathy Zhang, Connor Boyle, Maxwell Smith, and James P. Phillips. Fair Allocation of Scarce Medical Resources in the Time of COVID-19. *New England Journal of Medicine*, 382 (21):2049–2055, 2020. https://doi.org/10.1056/NEJMsb2005114.

[2] Quality Control in Hospitals. *New England Journal of Medicine*, 277(22):1205–1206, 1967. https://doi.org/10.1056/NEJM196711302772212.

[3] Adam Bohr1 and Kaveh Memarzadeh. The Rise of Artificial Intelligence in Healthcare Applications. *Artificial Intelligence in Healthcare*, pages 25–60, 2020. https://doi.org/10.1016/B978-0-12-818438-7.00002-2.

[4] McDonald L, Ramagopalan SV, Cox AP, and Oguz M. Unintended Consequences of Machine Learning in Medicine? *F1000Research*, 6(1707), 2017. https://doi.org/10.12688/f1000research.12693.1.

[5] Daniel M Bean, Paul Taylor, and Richard J B Dobson. A Patient Flow Simulator for Healthcare Management Education. *BMJ Simulation and Technology Enhanced Learning*, 5(1):46–48, 2018. https://doi.org/10.1136/bmjstel-2017-000251.

[6] Daniel M Bean, Paul Taylor, and Richard J B Dobson. Patient Flow Simulator. https://khp-informatics.github.io/patient-flow-simulator/.

[7] Ò Mir, M Sánchez, G Espinosa, B Coll-Vinent, E Bragulat, and J Millá. Analysis of Patient Flow in the Emergency Department and the Effect of an Extensive Reorganisation. *Emergency Medicine Journal*, 20(2):143–148, 2003. https://doi.org/10.1136/emj.20.2.143.

[8] The GPyOpt authors. GPyOpt: A Bayesian Optimization framework in Python, 2016. http://github.com/SheffieldML/GPyOpt.

[9] Mel Dyson. Pressure Points in the NHS, Jan 2022. https://www.bma.org.uk/advice-and-support/nhs-delivery-and-workforce/pressures/pressure-points-in-the-nhs.

[10] Thomas G. Dietterich. Ensemble Methods in Machine Learning. In *Multiple Classifier Systems*, pages 1–15, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg. https://link.springer.com/chapter/10.1007/3-540-45014-9_1.

[11] Jon Herman and Will Usher. SALib: An Open-Source Python library for Sensitivity Analysis. *The Journal of Open Source Software*, 2(9), jan 2017. https://doi.org/10.21105/joss.00097.

[12] Andrea Saltelli. Making Best Use of Model Evaluations to Compute Sensitivity Indices. *Computer Physics Communications*, 145(2):280–297, 2002. https://doi.org/10.1016/S0010-4655(02)00280-1.

[13] R. Fletcher. *Practical Methods of Optimization*. Wiley, 1986. ISBN 978-0-471-91547-8.

[14] Christine Cutillo, Karlie Sharma, Luca Foschini, Shinjini Kundu, Maxine Mackintosh, Kenneth Mandl, Tyler Beck, Elaine Collier, Christine Colvis, Kenneth Gersing, Valery Gordon, Roxanne Jensen, Behrouz Shabestari, and Noel Southall. Machine Intelligence in Healthcare - Perspectives on Trustworthiness, Explainability, Usability, and Transparency. *NPJ Digital Medicine*, 3:47, 03 2020. https://doi.org/10.1038/s41746-020-0254-2.

[15] Muhannad H. Yousef, Yazan N. Alhalaseh, Razan Mansour, Hala Sultan, Naseem Alnadi, Ahmad Maswadeh, Yasmeen M. Al-Sheble, Raghda Sinokrot, Khawlah Ammar, Asem Mansour, and Maysa Al-Hussaini. The Fair Allocation of Scarce Medical Resources: A Comparative Study From Jordan. *Frontiers in Medicine*, 7, 2021. https://doi.org/10.3389/fmed.2020.603406.

[16] Matthew B. A. McDermott, Shirly Wang, Nikki Marinsek, Rajesh Ranganath, Marzyeh Ghassemi, and Luca Foschini. Reproducibility in Machine Learning for Health. *CoRR*, abs/1907.01463, 2019. http://arxiv.org/abs/1907.01463.

[17] Scikit-Learn MLPRegressor. https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html.

A Appendix

A.1 Neural Networks Hyperparameter Tuning

Neural Networks have been tuned with the goal of finding the best hyperparameters that fit the data, and the results are shown in Table 1. For each tuning configuration, the entire dataset has been shuffled and divided into training set and test set before fitting and testing the model, and this process has been repeated 10 times in order to obtain a reliable estimation of the generalisation error.

Neural Networks up to 4 hidden layers were investigated, but employing two or more hidden layers resulted in considerable overfitting, so the obtained performances are not included. Additionally, several regularisation rates were studied, but the majority of them ended up underfitting or overfitting the data.

The hyperparameters were generated from a grid of values. This tuning method is known as "Grid Search", and it is commonly used for hyperparameter tuning. However, other techniques like "Random Search" are usually performed too, since they can find different well-performing optimums.

Early Stopping regularization technique was executed in order avoid overfitting as much as possible. This method is already implemented by the imported library, and it essentially shut downs the training when the loss does not significantly change for several epochs.

All the NN models were evaluated using R^2 coefficient (also known as Pearson coefficient or coefficient of determination) because it is the score used by scikit-learn MLPRegressor [17]. The best possible score is 1, and scores can be negative because the model can be arbitrarily worse. A constant model that always predicts the expected value of the targets, disregarding the input features, would get a score of 0. However, the coefficient of correlation was calculated to compare NN models with the rest of the systems in this project.

The best performing Neural Network model consisted of just 1 hidden layer with 8 neurons, employing as learning rate and regularization rate 0.001.

Hidden layers config.	**Learning rate**	**Reg. rate**	**R^2 coeff.**
1 HL of 4 neurons	0.001	0.001	0.2843
1 HL of 4 neurons	0.01	0.001	0.3336
1 HL of 8 neurons	**0.001**	**0.001**	**0.4326**
1 HL of 8 neurons	0.01	0.001	0.3619
1 HL of 12 neurons	0.001	0.001	0.3843
1 HL of 12 neurons	0.01	0.001	0.3276
1 HL of 16 neurons	0.001	0.001	0.2200
1 HL of 16 neurons	0.01	0.001	0.3587
2 HL of 4 neurons	0.001	0.001	0.4319
2 HL of 4 neurons	0.01	0.001	0.3072
2 HL of 8 neurons	0.001	0.001	0.4008
2 HL of 8 neurons	0.01	0.001	0.2995
2 HL of 12 neurons	0.001	0.001	0.3297
2 HL of 12 neurons	0.01	0.001	0.3385
2 HL of 16 neurons	0.001	0.001	0.3241
2 HL of 16 neurons	0.01	0.001	0.3172

Table 1: Neural Network hyperparameter tuning

A.2 Poisson Regression

Poisson regression is used in scenarios where the output or input variables are counts or frequencies, and it is thus suited to discrete variables. The general equation for Poisson regression between an input $\vec{x}_i$ and an output Y is:

$$\ln \lambda := \mathbb{E}\left[\ln(Y|\vec{x}_i)\right] = \sum_j \beta_j x_i^{(j)} + \beta_0 \tag{2}$$

which therefore means that Poisson regression is well-suited to relationships that appear exponentially increasing or decreasing (linear in log space). As the log-linear equation is defined to be the mean of the likelihood function, when we substitute the expression into the Poisson distribution, we obtain:

$$P(Y|\vec{x}_i) = \frac{\lambda^Y}{Y!}e^{-\lambda} \tag{3}$$

which gives the likelihood that can be used to fit the coefficients via maximum likelihood estimation. Specifically, the function to minimise (e.g. via gradient descent) is:

$$\begin{aligned}\mathcal{L}(\{\beta_i\}) &= \prod_i \ln\left(\frac{\lambda^Y}{Y!}e^{-\lambda}\right) \\ &= \sum_i \left\{Y_i \vec{\beta}^T \vec{x}_i - \exp(\vec{\beta}^T \vec{x}_i)\right\} - N\ln(Y!)\end{aligned} \tag{4}$$