Journal of Simulation

ISSN: (Print) (Online) Journal homepage: https://www.tandfonline.com/loi/tjsm20

Towards real-time predictions using emulators of agent-based models

Minh Kieu, Hoang Nguyen, Jonathan A. Ward & Nick Malleson

To cite this article: Minh Kieu, Hoang Nguyen, Jonathan A. Ward & Nick Malleson (2022): Towards real-time predictions using emulators of agent-based models, Journal of Simulation, DOI: 10.1080/17477778.2022.2080008

To link to this article: https://doi.org/10.1080/17477778.2022.2080008

Published online: 05 Jun 2022.

JOURNAL OF SIMULATION
https://doi.org/10.1080/17477778.2022.2080008

Towards real-time predictions using emulators of agent-based models

Minh Kieu[a], Hoang Nguyen[b], Jonathan A. Ward[c,d] and Nick Malleson[d,e]

[a]Department of Civil and Environmental Engineering, University of Auckland, Auckland, New Zealand; [b]Trident Pty Ltd, Sydney, NSW 2000, Australia; [c]School of Mathematics, University of Leeds, Leeds, UK; [d]Alan Turing Institute, UK; [e]School of Geography, University of Leeds, Leeds, UK

ABSTRACT
The use of Agent-Based Models (ABMs) to make predictions in real-time is hindered by their high computation cost and the lack of detailed individual data. This paper proposes a new framework to enable the use of emulators, also referred to as surrogate models or meta-models, coupled with ABMs, to allow for real-time predictions of the behaviour of a complex system. The case study is that of pedestrian movements through an environment. We evaluate two different types of emulators: a regression emulator based on a Random Forest and a time-series emulator using a Long Short-Term Memory neural network. Both emulators perform well, but the time-series emulator proves to generalise better to cases where the number of agents in the system is not known *a priori*. The results have implications for the real-time modelling of human crowds, suggesting that emulation is a feasible approach to modelling crowds in real-time, where computational complexity prohibits the use of an ABM directly.

ARTICLE HISTORY
Received 15 March 2021
Accepted 13 May 2022

KEYWORDS
Agent-based modelling; emulators; meta-modelling; machine learning

1. Introduction

Agent-based modelling is a class of computer simulation that excels in its ability to simulate complex systems (Bonabeau, 2002). Instead of deriving aggregated equations of system dynamics, agent-based models (ABMs) encapsulate system-wide characteristics from the behaviours and interactions of individual agents, for instance, humans, animals or vehicles. ABMs have traditionally been used to understand the dynamics of a system in a wide variety of contexts, such as delays in urban traffic (Balmer et al., 2009) and emergency evacuations (Schoenharl & Madey, 2011).

However, the use of ABMs to analyse systems is not usually possible in *real-time*. For example, ABMs are ideally suited to simulating crowds of people (Henein & White, 2005), but such models are currently constrained to "offline" use and cannot inform the management of busy places in real-time. It is, in fact, very challenging to develop real-time ABMs (Swarup & Mortveit, 2020) because there are serious methodological issues associated with updating agent-based models in response to new data that need to be overcome (Clay et al., 2020). In addition, even relatively simple ABMs can become extremely computationally intensive as the number of agents increases, which poses technical challenges as simulation results are required rapidly to be useful for the real-time management of systems. Existing studies report a drastic increase (sometimes exponential) in computation time as the population of agents increases in size (Niemann et al., 2021).

It is also often impractical, from a data perspective, to implement a 1-to-1 simulation of a human system in real-time. Most crowd data are aggregated in the first place – e.g., those created through the use of footfalls sensors or pedestrian counters – and the introduction of regulations such as the General Data Protection Regulation (GDPR) and the California Consumer Privacy Act make it more difficult to capture individual-level data and hence for modellers to model a real person and their intentions in a human system. Aggregated data can be easily handled by statistical and machine learning models for real-time purposes, but these approaches face two challenges. Firstly, statistical models are often purely data-driven and cannot provide additional outputs that are important in understanding the inherent system dynamics, as ABMs can do. For instance, machine learning models can learn to predict future aggregate estimates, such as future footfall counts, but cannot provide additional processed outputs such as delays or pedestrian density, which are provided when using an ABM to simulate the system from the "bottom up". Secondly, it can be very difficult to obtain the substantial volume of data required to train a versatile statistical model. Even if data are collected over a long period, some system states may never be observed, and a statistical model will struggle to make out-of-sample predictions in these cases.

This leads us to the core idea of this paper. Rather than attempting to use ABMs directly to conduct real-time analyses, we use the explanatory power

CONTACT Minh Kieu minh.kieu@auckland.ac.nz Department of Civil and Environmental Engineering, University of Auckland, Auckland 1010, New Zealand

and flexibility of ABMs to empower another model that can be used in real-time for prediction and management purposes. The ABM can be used to simulate the complex system, then in real-time another model, that can make faster predictions, can be used to represent the ABM. Hence this paper aims to develop an *emulators* framework, also referred to as *surrogate models* or *meta-models* that provides a mapping between some aggregated data (that we assume are available in real-time) and outputs from an ABM. In effect, an ABM is used offline to create a large volume of data to train an emulator, and the trained emulator can then use real-world aggregate data to make future predictions in real-time. We also compare two main types of emulators: a regression and a time-series emulator to find out which one is more suitable to make predictions in real-time.

The remainder of this manuscript is structured as follows. Section 2 reviews the literature on emulators for computer codes in general and ABMs specifically. Section 3 outlines the ABM and emulation methods developed in this research. Section 5 describes the results of several numerical case studies that compare the performance of the two emulators considered. Finally, Section 6 concludes and suggests several directions for future research. The source codes for the ABM, as well as the emulators can be found at [TBA link to our Github].

2. Literature review

An emulator is a statistical representation of a simulator, where the simulator itself is considered an unknown function (Bastos & O'Hagan, 2009). The emulator attempts to depict the relationship between the input and output variables of the simulator (Rasouli & Timmermans, 2013), ideally producing output much more efficiently than the simulator. This section reviews the existing approaches to building emulators for complex simulators, including ABMs. The emulator is a fundamental concept in the physical sciences, commonly used in fields such as climate modelling (Krasnopolsky et al., 2005) and biogeochemistry (Conti & O'Hagan, 2010) where simulation models are often too costly for real-time implementation. Recent efforts have shown the benefits of emulators for complex (Krasnopolsky et al., 2005; Rasouli & Timmermans, 2013), dynamic (Conti et al., 2009; Conti & O'Hagan, 2010) and stochastic (Baker et al., 2019; Moutoussamy et al., 2015) simulation models. Despite this enthusiasm, there have been a limited number of attempts at emulating ABMs (Bijak et al., 2013; Heard, 2014; Hilton, 2017; Oyebamiji et al., 2017; Rasouli & Timmermans, 2013), let alone *real-time ABMs*. This section reviews emulators and emulators for ABMs in more detail.

2.1. Analytical emulators

Analytical emulators aim to find a tractable, parametric smoothing function depicting the relationship between input and output variables. Rasouli and Timmermans (2013) aims to emulate the daily distance travelled per person in a microsimulation model and an ABM of traffic flow. A regression model with main effects plus first-order interaction effects was developed. The authors also explored the impact of the number of simulation runs on the performance of the emulator, with the results suggesting that the accuracy of the emulator increases with the number of simulation runs. However, this paper only aims to develop a direct statistical mapping between inputs and outputs, without capturing the dynamic changes in the system. Lafuerza et al. (2016a) developed an analytically tractable emulator of an ABM of social interaction, which allows mathematical analysis to be performed. Emulators such as these are analytically tractable with an elegant parametric formulation that can help to elucidate the relationship between input and output variables. However, they are often limited to the instantaneous dynamics of the system where the analytical formulations are developed, and will need to be revised as the system under study changes over time. Revising the analytical formulations of these emulators is a complex and time-consuming task, that would limit the usefulness of analytical emulators for real-time applications.

2.2. Meta-modelling approaches

Meta-modelling emulators use statistical or machine learning techniques to learn the mapping between the input and output variables of a simulator. These are more flexible than analytical emulators because the same technique can be implemented to "retrain" the emulator if updates are needed, instead of revising the emulator itself. Meta-modelling emulators can learn complex behaviours, making them more widely applicable than analytical approaches.

Gaussian Process (GP) emulators (also referred to as Kriging) are among the most popular emulator techniques (Bastos & O'Hagan, 2009). GP emulators have been developed for univariate (Oakley & O'Hagan, 2002) and multivariate (Higdon et al., 2008) problems, as well as dynamic (Conti et al., 2009; Conti & O'Hagan, 2010) and stochastic (Baker et al., 2019; Moutoussamy et al., 2015) simulators. However, the complexity of parameter inference for a GP is usually $\mathcal{O}(n^3)$, which means that it is actually very expensive to train and adapt a GP emulator of a high-dimension simulator such as an ABM. Other machine learning techniques for emulators, such as Neural and Bayesian Networks (Farah et al., 2014; Shrestha et al., 2009), have also been explored.

Machine learning methods are generally even more flexible than Gaussian Processes, take less time to train and, most importantly, they excel at providing a non-linear mapping between the input and output variables of the simulator. These features make them strong candidates for emulating a real-time simulator.

2.3. Emulators of ABMs

The main difference between emulators of ABMs and emulators of other simulation models is that ABMs are fundamentally driven by the micro-interactions of discrete entities ("agents"). This allows models with even relatively simple behavioural rules to produce complex outcomes. Many emulators of ABMs attempt to re-produce similar dynamics by simplifying the ABM itself, e.g., by reducing the number of agents or simplifying their behaviour (Barnes et al., 2021; Deissenberg et al., 2008; Lafuerza et al., 2016b; Niemann et al., 2021; Rhodes et al., 2016; Tregubov & Blythe, 2020). Rhodes et al. (2016) showed that ABMs can be simplified by several orders of magnitude and still produce a very similar system-level behaviour while reducing runtime and data output. In a similar approach, Lafuerza et al. (2016b) aimed to understand a complex intricate ABM of voting using a series of simplified models and showed that a certain range of modelling capabilities can be maintained in simplified models over a particular range of parameter values (Lafuerza et al., 2016b). In Tregubov and Blythe (2020), several model simplification methods such as sub-sampling agents and simplifying agent behaviour were evaluated, showing improvements in computation time with little reduction in model predictive accuracy. While simplification may help reduce the computation burden of ABMs, such an approach cannot be guaranteed to generalise and, as there are still interactions that need to be computed, even the simplest ABM required to simulate a particular system might be computationally expensive.

Meta-modelling and analytical emulators have the potential to resolve the computation issue by transforming the emulation problem to that of simply finding a mapping function between the inputs and outputs of ABMs. The well-trained emulator might be able to learn all combinations of inputs/outputs such that in real-time the emulator can be used in place of the ABMs for computational efficiency. There are studies such as Niemann et al. (2021) which used ordinary and stochastic differential equations (SDEs) to approximate stochastic ABMs of medium to large agent populations, but to date, many of the emulators of ABMs that have been developed are GPs (Bijak et al., 2013; Dosi et al., 2018; Heard, 2014; Hilton, 2017). GPs provide a confidence interval for each estimate, so fewer model runs are required to explore the parameter space and statistical characteristics of the ABM. Bijak et al. (2013) presented a Semi-Artificial Model of Population, which is a fusion of demographic micro-simulation and an ABM, to address the problem of modelling population dynamics, specifically the impacts of certain parameters on population size and share of married agents. Dosi et al. (2018) analysed policy impacts using ABMs, where GPs were used as part of their sensitivity analysis on key variables and parameters. The benefits of GPs in the model calibration and statistical inference problem of ABMs have also been explored in Heard (2014), where observed data and ABM simulation outputs were used to fit and calibrate GP approximations. In Hilton (2017), a GP emulator was used to quantify the uncertainty in the outputs of ABMs and also to calibrate an ABM against empirical observations. Oyebamiji et al. (2017), Oyebamiji et al. (2019)) developed GP regression models to emulate dynamic and stochastic individual-based models of microbial communities.

The majority of meta-modelling emulators have been designed to support calibration and sensitivity analysis rather than for use in making real-time predictions. When attempting to predict in real-time, the computation complexity is not the only challenge, but also the lack of individual-level data for a 1-to-1 simulation. This paper aims to develop an emulator framework that can work with aggregated data, that are widely available, to represent an ABM and make predictions in real-time.

3. Methodology

We propose a framework for real-time prediction using emulators of ABMs. We have developed this framework to use the emulator to make real-time predictions of the near future, whilst having the flexibility to adapt to incoming data. Broadly, our proposed approach is to develop an ABM to represent the target system, train an emulator using aggregated data outputs from the ABM, and then use the emulator, rather than the ABM itself, to make real-time predictions. The use of the emulator in this setting is advantageous because: (i) the ABM is too computationally expensive to return results in an adequate time; and (ii) the emulator makes predictions using aggregate data, whose availability is more likely than the individual-level data. This section outlines the methodology in detail.

Let $Z_t \in \mathbb{R}^r$ denotes the set of outputs from an ABM at discrete time $t \in \mathbb{Z}$ and since the ABM is stochastic, variables in Z_t are random. The choice of discrete time is motivated by our assumption that aggregated data will be made at discrete points in time and projecting these onto the integers is purely for convenient notation. We assume that the ABM is parameterised by a vector of parameters θ, that might be completely unseen in real-time. Z_t consists of $[X_t, y_t]$. Observation data X_t are assumed available in real-time, and y_t is the additional output from an ABM which is often unobservable in real-time, or completely unobservable. Note that in practice we would have to calibrate the values of parameters θ such that the ABM can adequately represent the underlying system, and replicate X_t to be sufficiently similar to the real aggregated data. In this paper, we assume that our ABM has already been calibrated and the provided X_t are similar to the real data. We are then interested in an emulator function f that can map the available aggregated data X_t with Z_t, i.e., $Z_t = f(X_t)$, and also can make predictions, e.g., $Z_{t+k} = f(X_t)$ with $k > 0$.

We propose a "bi-phase" framework, illustrated in Figure 1. The first phase of the framework, the *Training phase*, aims to train the emulator so that it can predict the behaviour of the real system. We assume that the emulator cannot learn this behaviour from the underlying system *directly* for two reasons. Firstly, there may be important information required by the emulator that can only be extracted from the ABM. For example, in the case study used in this paper we are interested in the delay caused to individual pedestrians in a crowd. This can be calculated by examining the individual agents in the model but not inferred from aggregate (real world) data. Secondly, even if a system is studied over a long period, likely, some more extreme system states will not occur. Therefore the emulator will be trained on "business as usual" system behaviour and will not be able to work well in more extreme scenarios. Using an ABM, not only can a large volume of training data be created, but a wide variety of system states can also be simulated. Therefore we develop an ABM to represent the target system (pedestrians moving through a corridor in this case) and use the ABM to create simulated input and output data for emulator training. This training phase is done before real-time implementation of the emulator as it requires multiple runs of the ABM to generate the training data, which will be a computationally-expensive step. Since the training data do not need to be produced quickly, as would be the case if the ABM were making real-time predictions, the computation time at this stage in the framework is not problematic. Also, the emulator will not need to be retrained unless there is a significant change to the underlying system, in which case the ABM will also need to be recalibrated and re-run to produce new data that reflect the new dynamics of the system. As Section 4.2 will discuss, the emulator is trained and tested using well-established machine learning methods.

In the second phase of the framework, the *Real-time phase*, information about the system is collected in real-time and these data are used as inputs to the trained emulator. In this manner, the emulator can predict the short-term future behaviour of the system without the need for running an ABM. It is important to note that, in this preliminary application, the real-time phase does not actually use real-world data. Instead we follow a "pseudo-truth" or "digital-twin" experiment approach, similar to Wang and Hu (2015) and Kieu et al. (2020) to evaluate this framework. Rather than using real observations, the data will be generated from the ABM. This is advantageous

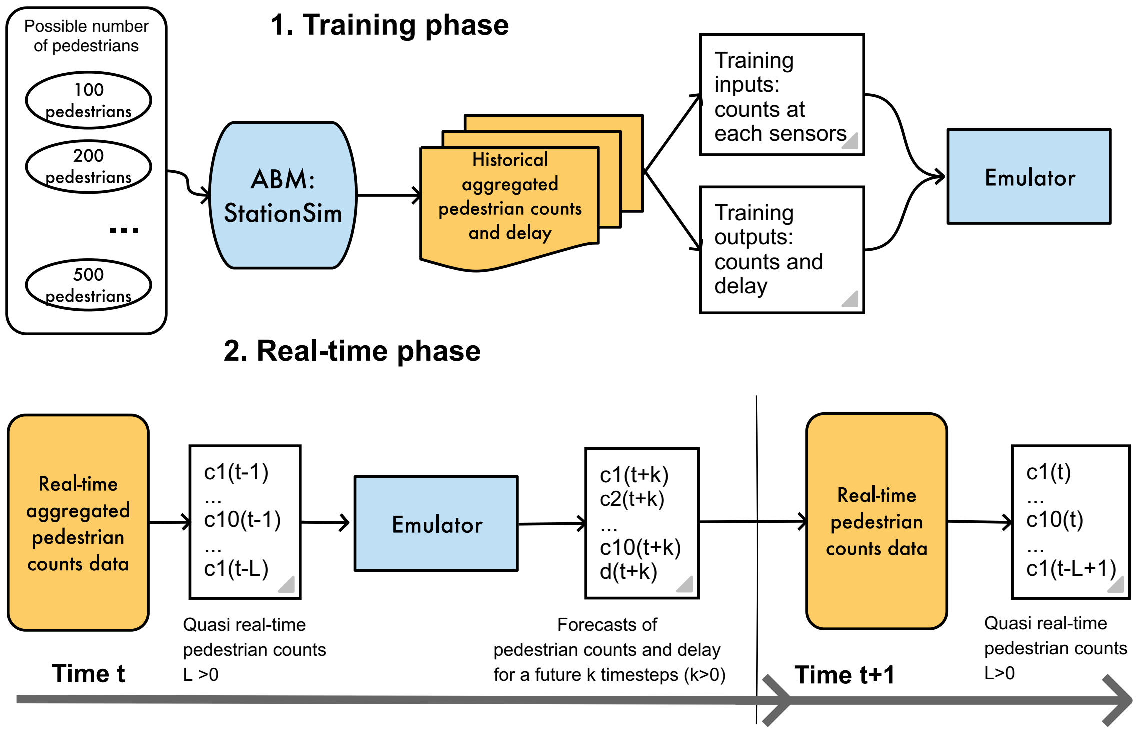
1. Training phase
Possible number of pedestrians
100 pedestrians
200 pedestrians
...
500 pedestrians
ABM: StationSim
Historical aggregated pedestrian counts and delay
Training inputs: counts at each sensors
Training outputs: counts and delay
Emulator
2. Real-time phase
Real-time aggregated pedestrian counts data
c1(t-1) ... c10(t-1) ... c1(t-L)
Emulator
c1(t+k) c2(t+k) ... c10(t+k) d(t+k)
Real-time pedestrian counts data
c1(t) ... c10(t) ... c1(t-L+1)
Quasi real-time pedestrian counts L >0
Forecasts of pedestrian counts and delay for a future k timesteps (k>0)
Quasi real-time pedestrian counts L>0
Time t
Time t+1

Figure 1. Study framework.

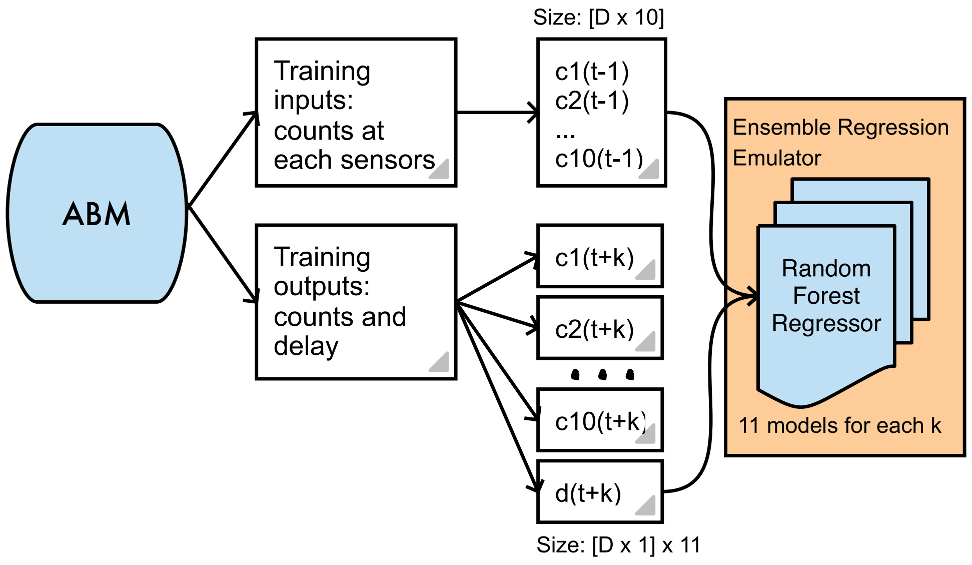

Figure 2. A regression emulator of agent-based models using random forest regression.

because it means that the true system state can be known, so errors can be calculated accurately. In addition, we can evaluate the framework on a wide range of synthetic data to "stress test" it under various scenarios. Future work will move towards using real crowd data, such as those made available by Zhou et al. (2012). As Section 4 will discuss, we assume that ten pedestrian sensors are deployed in the environment, and count the number of pedestrians who pass through them. Hence the task of the emulator is to estimate what the subsequent counts at these sensed points will be. In addition, the emulator is also tasked with estimating the overall *delay* to the pedestrians; in congested scenarios it will take longer for pedestrians to be able to traverse the environment.

The need for fast emulators means that GP emulators are not ideal due to their high computational complexity, so we do not consider them here. Instead, we test two different types of well-known machine learning emulators. We adopt a static emulator, Random Forest (RF) regression, because it is widely used and has been shown to be effective in modelling non-linear relationships. As RF regression does not directly treat the data as a time-series, we also experiment with a Long Short-term Memory (LSTM) neural network, which uses multiple time steps in the near past to forecast the near future. The following sub-sections provide brief descriptions of each these techniques.

3.1. Regression emulator: Random forest

RF regression models are typically trained to predict univariate output, so multiple models have to be trained to predict each of the q required outputs in Y_t (in this case the counts of pedestrians at different sensor locations). In effect, we consider this emulator as an "ensemble regression emulator", as illustrated in Figure 2.

The RF is an ensemble method that makes predictions by combining the decisions from multiple individual models, i.e.,

$$f(x) = \frac{1}{B}\sum_{i=1}^{B} T_i(x), \qquad (1)$$

where each individual model T_i is a simple Decision Tree (Breiman, 2001), built upon a statistical technique called *bagging*. Interested readers may refer to Breiman (2001) for a more detailed and formal description. The effectiveness of RFs comes from two key concepts:

- Each tree is built and learnt from a random sampling of training observations. This is to reduce the overall variance in the entire RF model but not at the cost of increasing bias;
- Splitting nodes in each tree using random subsets. This reduces the potential for over-fitting because each tree only sees a subset of all training features when deciding to split a node.

We train 11 RF models for each value of look-ahead time step k, where the first 10 models predict the counts at each of the 10 pedestrian sensors at time step $t+k$ and the eleventh model predicts the delay to the pedestrians. These models are trained using the same input: the pedestrian counts at the previous time step $t-1$ (as showed in Figure 2). The training input for the regression emulator is 2D array of size $D \times 10$, where D is the sample size of the input data.

3.2. Time-series emulator: Long short-term memory

One major disadvantage of regression techniques in learning time-series patterns, even for powerful techniques such as RFs, is that there is no consideration of longer-term temporal dependency in the data, e.g., the ability to learn from multiple historical observations to predict the next state. In regression models, time can be used as a variable to predict, but there is no dependency between successive time steps. We define the framework for the ABM time-series emulator as in Figure 3.

While it is possible to include historical observation data in the input vector X_t, we might expect that recent observations will have a significant temporal correlation. To account for this, the time-series emulator takes inputs from X_{t-L} to X_{t-1}, where $L > 0$ defines a "look-back window". Defining the value of L is a trade-off problem that has to be solved through experimentation. Choosing L too large leads to unnecessarily long computation time and numerical instability, but too small L will not fully capture the temporal dependency between time steps. After experimenting with different values we take $L = 5$ based on the resulting Mean Absolute Error. Thus, the training input for the time-series emulator is 3D array of size $D \times 10 \times L$.

For our time-series emulator, we adopt the Long Short-term Memory (LSTM) recurrent neural network. Interested readers may refer to the Appendix for a brief description of the LSTM, or to Hochreiter and Schmidhuber (1997) for a detailed explanation of the method. There are several parameters to be determined when constructing an LSTM network. These parameters were selected by both heuristic methods (to limit parameter value ranges) and grid search for a limited set of parameters. The detailed architecture and optimal parameters for our LSTM network is described below:

- LSTM layer with 32 nodes to learn temporal dependency of the time series.
- A dense layer with 32 nodes to enhance the generalisation level of the model.
- A final dense layer with 11 nodes (equal to the number of outputs) to predict next values of 10 sensors and overall delays.
- The model is then compiled with an Adaptive moment estimation optimisation mechanism (ADAM).
- The loss function during training is the mean absolute error (MAE) and the metrics function for model evaluation is the relative mean error.

4. Case study: Pedestrian crowding

4.1. StationSim

We use a simple ABM of pedestrian dynamics (Malleson et al., 2020) named *StationSim* as a case study. The model has been designed to very loosely represent the behaviour of a crowd of people moving from an entrance on one side of a rectangular corridor to an exit on the other side. This is analogous to people disembarking from a train and moving across the concourse of a train station. The model environment is illustrated in Figure 4, with the trajectories of two interacting agents for illustration.

The model does not attempt to match the behavioural realism offered by more developed crowd models, such as those that adopt the Social Force model (Helbing et al., 2000). The reason for this simplicity is so that: (1) the model can execute relatively quickly; (2) the probabilistic elements in the model are limited (we know precisely from where probabilistic behaviour arises); and (3) the model can be described fully using a relatively simple state vector. Importantly, the model can capture the emergence of *crowding*. This occurs because each agent has a different maximum speed that they can travel at, so fast agents will try to overtake slower ones, but also because there is a limited space in which agents can walk and there is a limited rate at which agents can leave the environment.

Given a rectangular 200 m x 100 m corridor (Figure 4), N agents are generated as the model initialises. We assume that within the simulation time interval $[0, T]$, there are W roughly periodic waves of

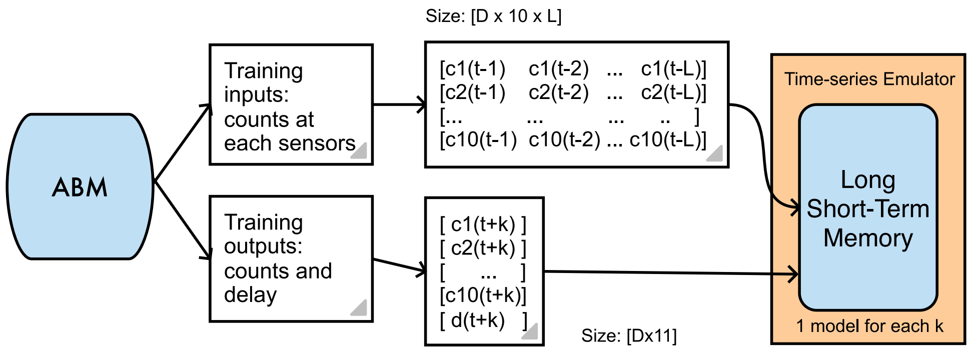
Size: [D x 10 x L]
ABM
Training inputs: counts at each sensors
Training outputs: counts and delay
[c1(t-1) c1(t-2) ... c1(t-L)]
[c2(t-1) c2(t-2) ... c2(t-L)]
[... ... .]
[c10(t-1) c10(t-2) ... c10(t-L)]
[c1(t+k)]
[c2(t+k)]
[...]
[c10(t+k)]
[d(t+k)]
Size: [Dx11]
Time-series Emulator
Long Short-Term Memory
1 model for each k

Figure 3. A time-series emulator of agent-based models using long short-term memory.

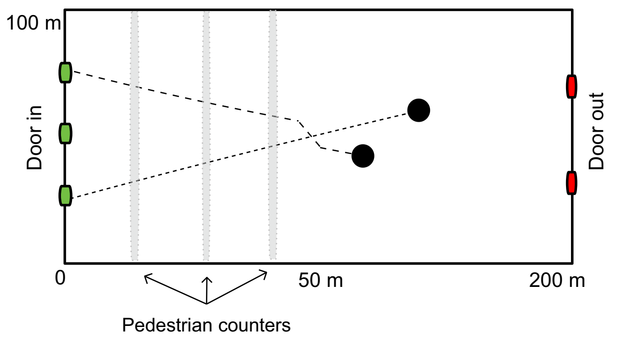

Figure 4. The stationsim model; adapted from. malleson et al. (2020).

arrivals of pedestrians, with some random deviations δ_w. We choose $W = 3$ and these waves start at times $\delta_1, \frac{1}{3}T + \delta_2$ and $\frac{2}{3}T + \delta_3$. This is analogous to multiple trains arriving at near regular times during the simulation time interval. Within each wave, the agents enter the environment (leave their train) at a uniform rate through one of the three entrances. They move across the "concourse" and then leave by one of the two exits. The entrances and exits have a set size, such that only a limited number of agents can pass through them in any given iteration. This configuration represents a typical problem that ABMs excel at because it involves the simulation of individuals with heterogeneous behaviour and with patterns (i.e., congestion) that emerge from the interactions between individuals.

We also assume a set of 10 pedestrian counters are equally spaced along the corridor, which provide the number of pedestrians who walk past the sensor (see, Figure 4). This is a fair assumption in practice because pedestrian counters are widely available in many public places. Figure 5 shows the counts at sensors 1, 5 and 10 at each time step from 0 to 1200s. Because of differences in walking speed between agents and the emergence of crowding, the counts at each sensor are very different. At the first sensor near the entrance (sensor 1), the three waves (i.e., three train arrivals) are easily identified, with the pedestrians uniformly leaving the train. These waves are still visible in sensor 5, in the middle of the environment, but model stochasticity causes them to be much less clearly defined. By the time the agents reach sensor 10, at the end of the corridor, the waves can no longer be easily distinguished.

Despite being simple, this model has three of the most important characteristics of an ABM:

- *individual heterogeneity*: agents have different maximum travel speeds;
- *agent interactions*: agents are not allowed to occupy the same space and try to move around slower agents who are blocking their path;
- *emergence*: crowding is an emergent property of the system that arises as a result of the choice of exit that each agent is heading to and their maximum speed.

4.2. Training the ABM emulators

The Training phase starts with the generation of training data. To expose the emulator to a wide range of possible system behaviours, we randomly generate multiple sets of agents with random maximum speeds, starting entrances and desired exits. Experiments are conducted with populations of $N = 100$ (few collisions and hence largely linear deterministic agent behaviour) up to $N = 500$ agents (many collisions, considerable crowding, large stochasticity). Each set of parameters is fed into the ABM to generate synthetic "historical" aggregated pedestrian counts c and

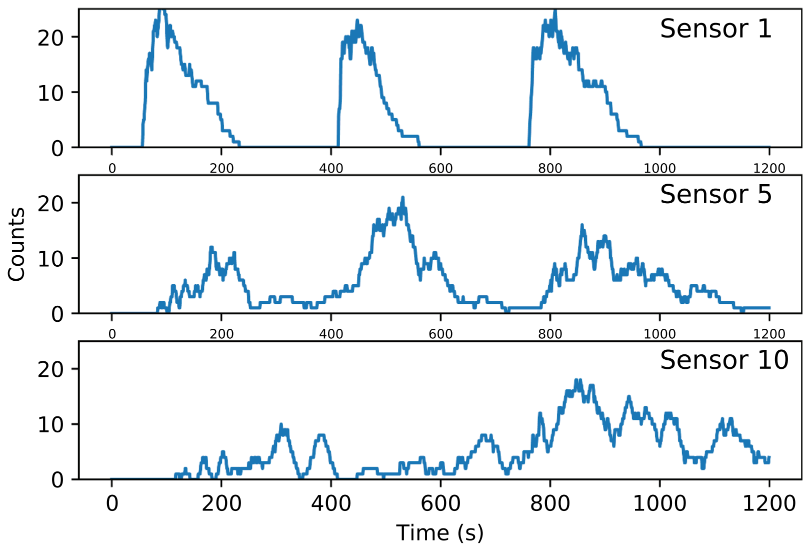

Figure 5. Counts of pedestrians at sensor 1, 5 and 10 at each time step.

mean delay $\bar{d}$. The mean delay $\bar{d}$ is the average of the difference between the time each agent spent to get into its current position versus the time that it would have spent if it could walk with its desired speed, i.e.,

$$\bar{d} = \frac{1}{N}\sum_{i=1}^{N} \tau_i - \frac{x_i}{v_i},$$

where τ_i is the time taken by agent i to cross from where they entered to their current position, x_i is the corresponding straight line distance and v_i is agent i's desired speed.

This is a useful statistic as it works as a proxy for the level of crowding, hence the emulator can be used to predict the emergence of crowding in the near future.

The simulation is executed 30 times for each population size, N, by generating new sets of random agents and re-running the ABM to create a rich synthetic training dataset. We then split the synthetic data into training input and training outputs. The training inputs are the aggregated counts of pedestrians from each sensor. The training outputs, which the emulator is tasked with estimating, are the aggregated counts and mean delay $\bar{d}$ at future time steps.

In the *Real-time phase* we assume that we only have access to the quasi-real-time aggregated counts of pedestrians at the 10 sensors along the corridor on Figure 4. Hence we need an emulator to use these counts to predict *future* aggregated counts and delay. The data are quasi-real-time because we assume that at the current time step t, only aggregated counts at previous time steps $t - L$ to $t - 1$ are available. The trained emulators process these inputs to predict the aggregated counts and delay at the current time step t and near-future time steps, the forecast period, $t + k$, for $k > 0$. At the next time step $t + 1$, we assume that the data recorded by the sensors at time t are now available and can be used to predict the next time steps in the near future.

5. Numerical experiments

This section describes the experimental setup and results. Following the framework described in Section 3, recall that the emulators are trained using a synthetic "historical" dataset of 30 replications from *StationSim*, with varying agent population sizes. The emulators' predictive ability will now be evaluated using another synthetic "real-time" dataset that is completely unseen to them and created by running StationSim once. This provides "pseudo-truth" aggregated pedestrian counts and delays. We evaluate the two classes of emulator for ABM with two experiments:

(1) The total pedestrian population is known: the emulators are trained and evaluated using datasets with the same population size. For instance, the performance of a regression emulator that has been trained using the population data of $N = 500$ agents will be evaluated against a pseudo-truth "real time" dataset with 500 agents.

(2) The total pedestrian population is unknown: the emulators are trained once with all the training data available (with populations sizes $N \in \{100, 200, 300, 400, 500\}$) and evaluated separately on "real-time" data with different population sizes. For instance, a regression emulator that has been trained using all available population data will be evaluated 5 times, using "real time" data for population sizes $N \in \{100, 200, 300, 400, 500\}$. An emulator that generalises well will be able to estimate the counts at each gate and the mean delay without knowing the size of the population. This experiment is more practical because in reality the total pedestrian is often unknown in real-time.

5.1. Experiment 1: Known pedestrian population size

In the first experiment, we assume that the total number of pedestrians that will walk through the corridor is known. We evaluate the accuracy of each class of emulator when the pedestrian population is $N \in \{100, 200, 300, 400, 500\}$ pedestrians. Figure 6 shows the prediction of the aggregated pedestrian counts at each sensor during the study period using the regression emulator and the time-series emulator for the case where the forecast period is $k = 5$ and $k = 15$ time steps into the future. The red lines show the predicted counts at each sensor and each time step, whereas the black lines show the pseudo-truth real-time counts.

Similarly, Figure 7 shows the prediction results from the time-series emulator.

The two figures illustrate the predicted counts (in red) and the synthetic real-time counts (in black) from each sensor and across time. In Figures 6 and 7, the evaluated pedestrian population is 500.

Overall, both of the emulators perform very well in the prediction of aggregated counts. The overall fluctuations of the aggregated pedestrian counts are captured closely with a few exceptions where the synthetic real-time counts (black line) reach a peak. In both classes of emulators, the further prediction ($k = 15$) has slightly lower accuracy. Figures 6 and 7 clearly show that the emulators have enabled us to work with aggregated data to produce outputs that are similar to those produced by the ABM.

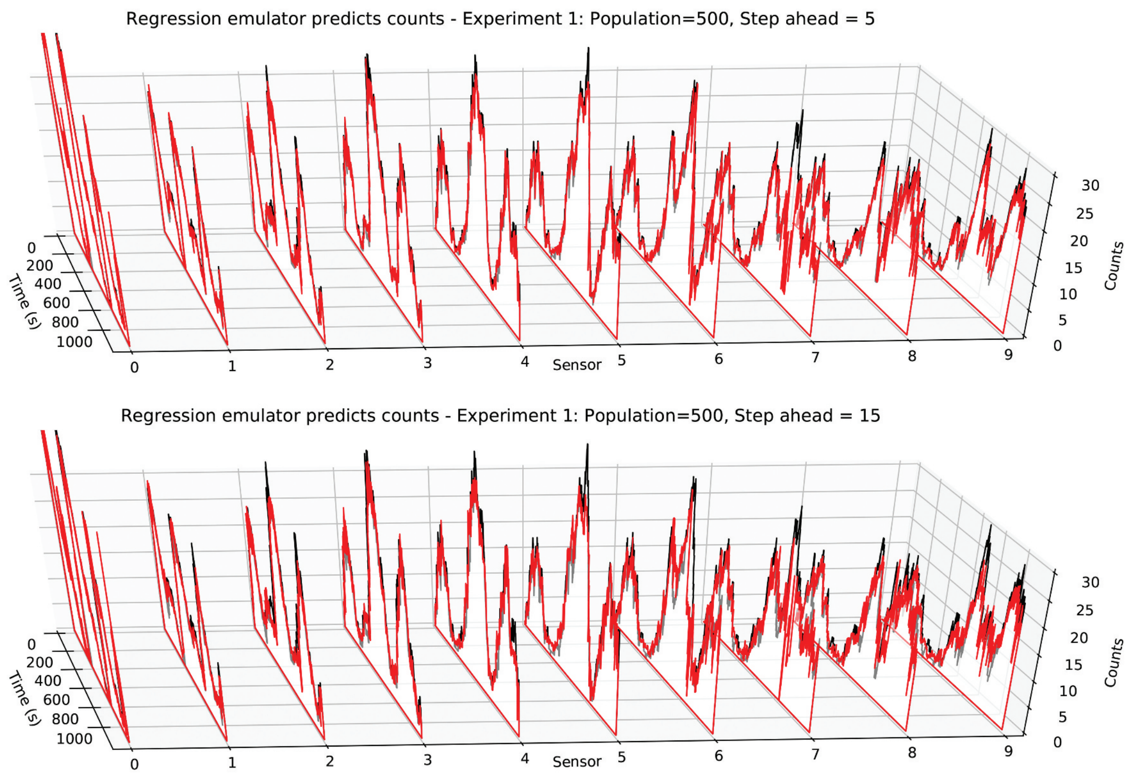

Figure 6. Prediction of pedestrian counts at each sensor using the regression emulator. red line: predicted counts, black line: synthetic real-time counts.

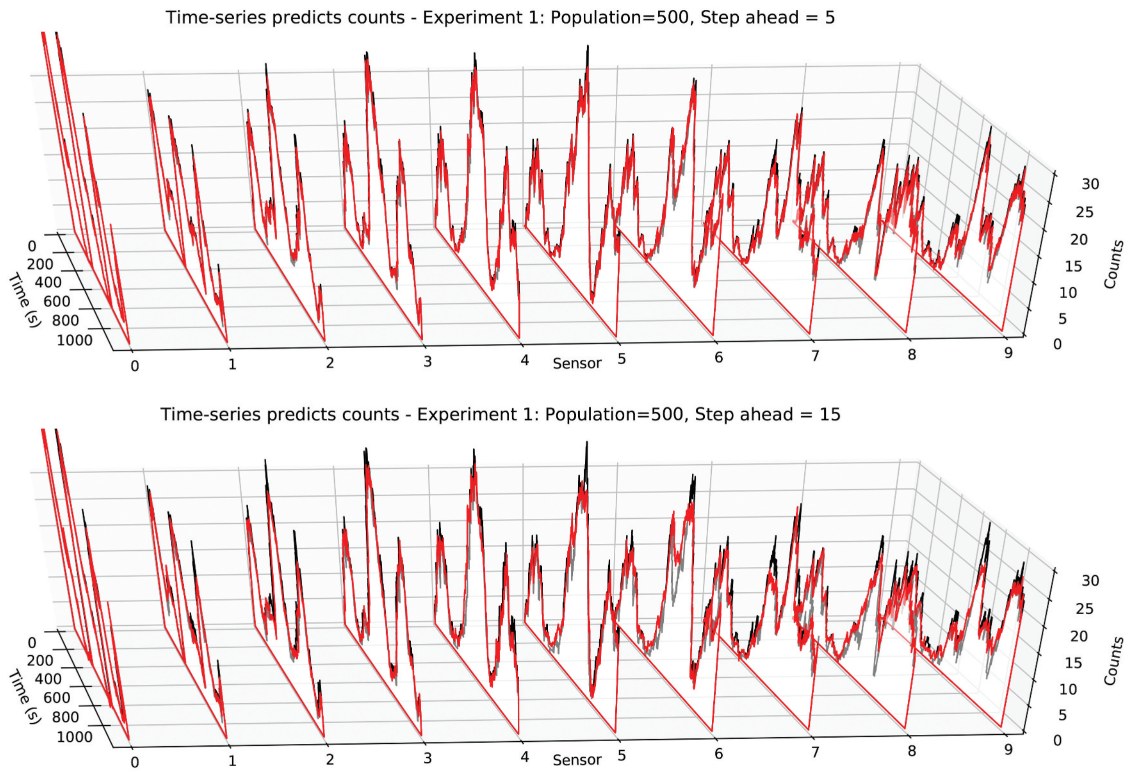

Figure 7. Prediction of pedestrian counts at each sensor using the time-series emulator. red line: predicted counts, black line: synthetic real-time counts.

Figures 8 and 9 show the prediction of mean pedestrian delays (a proxy for crowding) in the study corridor for the regression and time-series emulators respectively. The figures show the predicted delays (in red) versus the synthetic real-time delays (in black) when the agent population is $N \in \{100, 200, 300, 400, 500\}$ and the simulation time is $T = 1200$. The figures clearly show that the delays increase significantly as the agent population increases. The delay is clearly correlated with the three waves of pedestrian arrivals, with a sudden drop in delay after each wave. Predicting these waves of delay is challenging because of this sudden drop and because the location, the peak and the steepness of each curve is stochastic.

Both emulators capture the general trend in the mean delay reasonably well. The performances are lower than the prediction for aggregated pedestrian counts because the emulators cannot capture some of the peaks and troughs in the values of mean delay. This can be explained by the lack of the historical values of delay in the input, as mentioned in Section 4. The

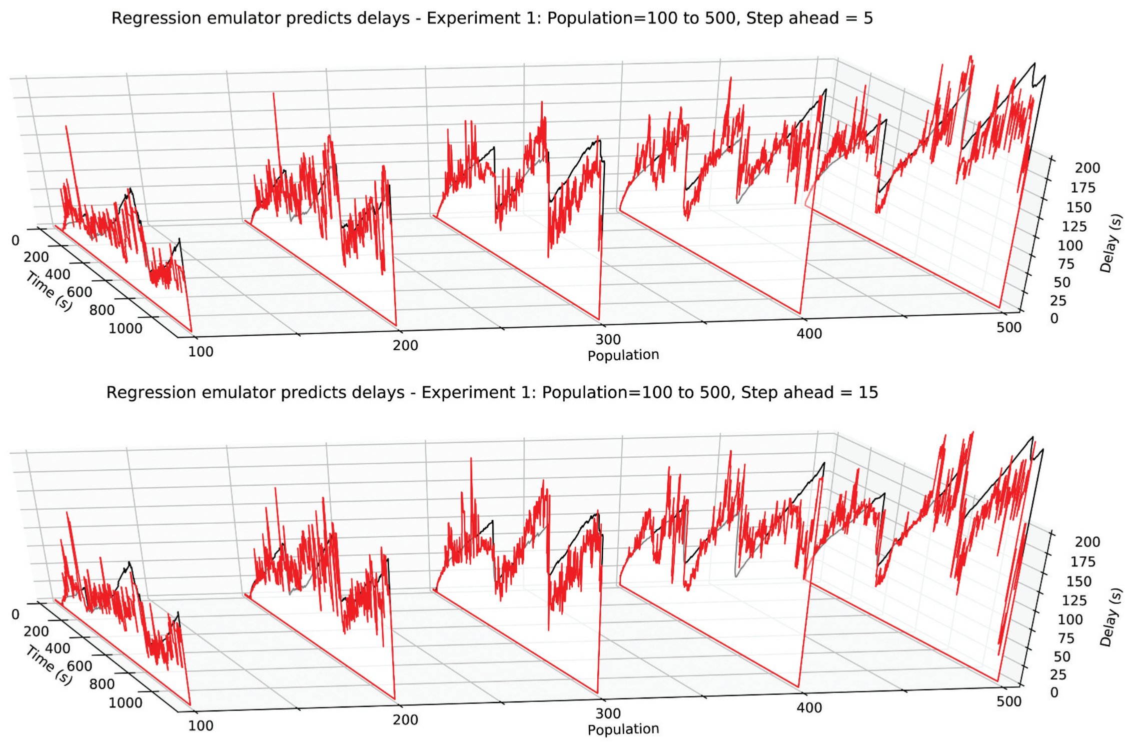

Figure 8. Prediction of delays using the regression emulator. red line: predicted delay, black line: synthetic real-time delay.

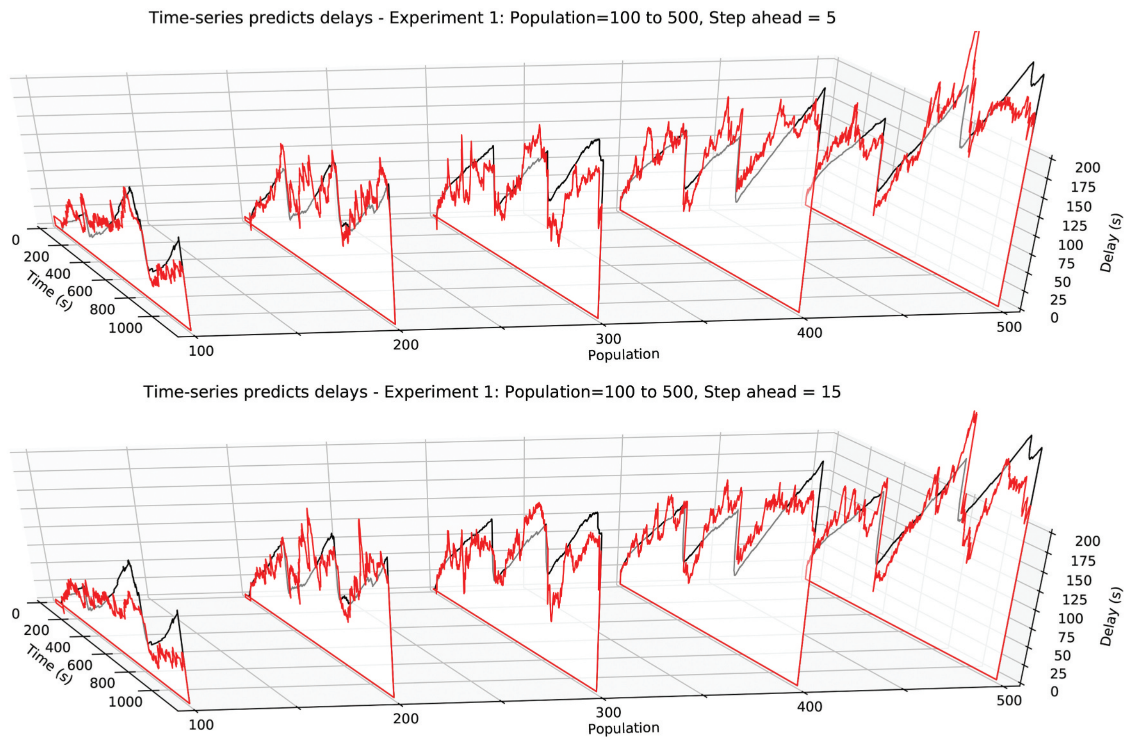

figure 9. Prediction of delays using the time-series emulator. red line: predicted delay, black line: synthetic real-time delay.

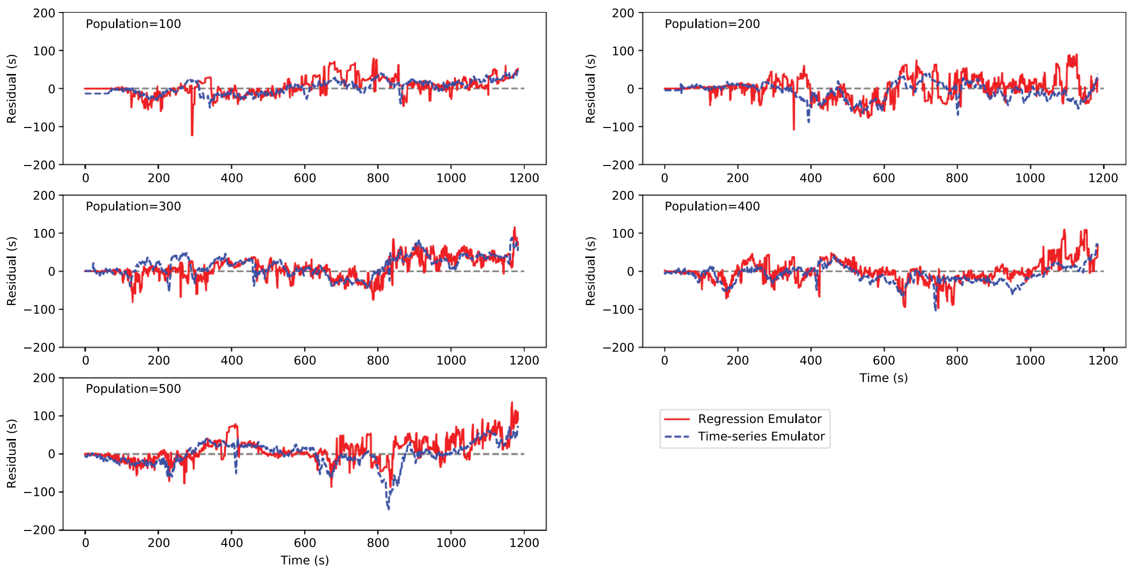

Figure 10. Comparison between the prediction errors from regression versus time-series emulators.

Table 1. Comparison of prediction accuracy (MSE) for Experiment 1. The bold number shows the lower error when comparing the regression to the time-series emulator.

	Regression emulator			Time-series emulator		
Population	5s	10s	15s	5s	10s	15s
100	18.21	20.82	**21.02**	**15.12**	**17.37**	22.25
200	22.09	20.72	21.53	**17.14**	**15.38**	**18.72**
300	**23.05**	**23.59**	23.83	23.26	23.78	**21.93**
400	21.37	23.03	25.36	**18.75**	**16.78**	**20.11**
500	24.06	26.32	29.51	**23.16**	**20.36**	**24.86**

regression emulator (Figure 8) shows noticeably more fluctuations in its prediction than the time-series emulator (Figure 9), with several cases of large prediction errors. It shows that the regression emulator is less reliable in its prediction compared to the time-series prediction. We further illustrate this point in Figure 10, which shows the difference between the predicted mean delay with the synthetic real-time mean delay for the case where the forecast period is $k = 5$, and the agent population is 100 and 200. The closer the lines are to zero, the better the prediction. The residuals from the regression emulator are more volatile, indicating that the regression emulator is a less stable prediction compared to the time-series emulator.

Table 1 shows the comparison of prediction accuracy between the regression emulator (RF) and the time-series emulator (LSTM). The comparison values are of the Mean Squared Error (MSE),

$$\text{MSE} = \frac{1}{T}\sum_{t=1}^{T}(\bar{d}_t - \hat{d}_t)^2, \tag{2}$$

where $\bar{d}_t$ is the synthetic real-time mean delay at time t, and $\hat{d}_t$ is the predicted mean delay at time t. The predictions in Table 1 are made for forecast periods $k \in \{5, 10, 15\}$. The numbers on the time-series emulator are showed in bold if the MSE is lower than its regression counterpart, and vice-versa. The time-series emulator has a better accuracy compared to the regression emulator, by 13.16% on average.

5.2. *Experiment 2: Unknown pedestrian population*

The next experiment assumes that in real-time the total number of pedestrians who will enter the system is unknown, so only the aggregated counts at each sensor are available to the emulators. This is much closer to the situation that would arise in reality. The framework is exactly the same as the previous experiment, apart from the fact that now the training input data of *all* agent population sizes $N \in \{100, 200, 300, 400, 500\}$ are fed into the emulators for training. The trained emulators are then evaluated against five different sets of "real time" data, one created for each of the five different population sizes, to see how well they can predict different crowd sizes.

Using the regression and time-series emulators, we found that predictions of pedestrian counts are very similar to previous experiments (e.g., when compared to the results in Figures 6 and 7). To better differentiate the results, Figure 11 shows a heat map of the Mean Absolute Error of the predictions of aggregated pedestrian counts using the regression emulator (on the top row) and the time-series emulator (on the bottom row) when the emulators know the pedestrian population (on the left) and with an unknown population (all the data are used, on the right). All the tests are done for the

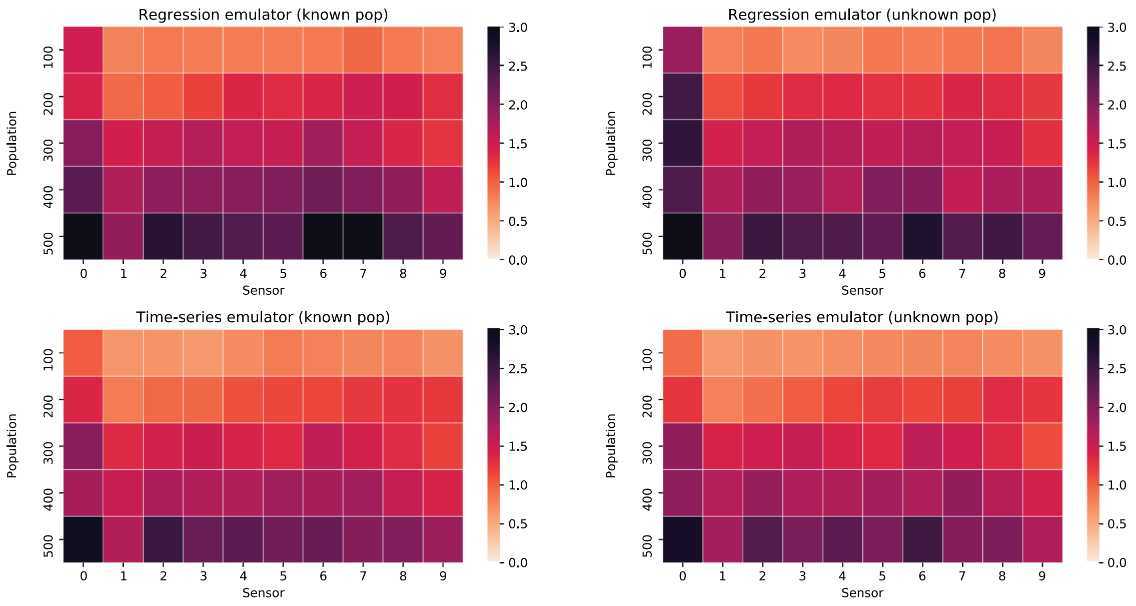

Figure 11. Comparison of the prediction results for aggregated pedestrian counts at each sensor.

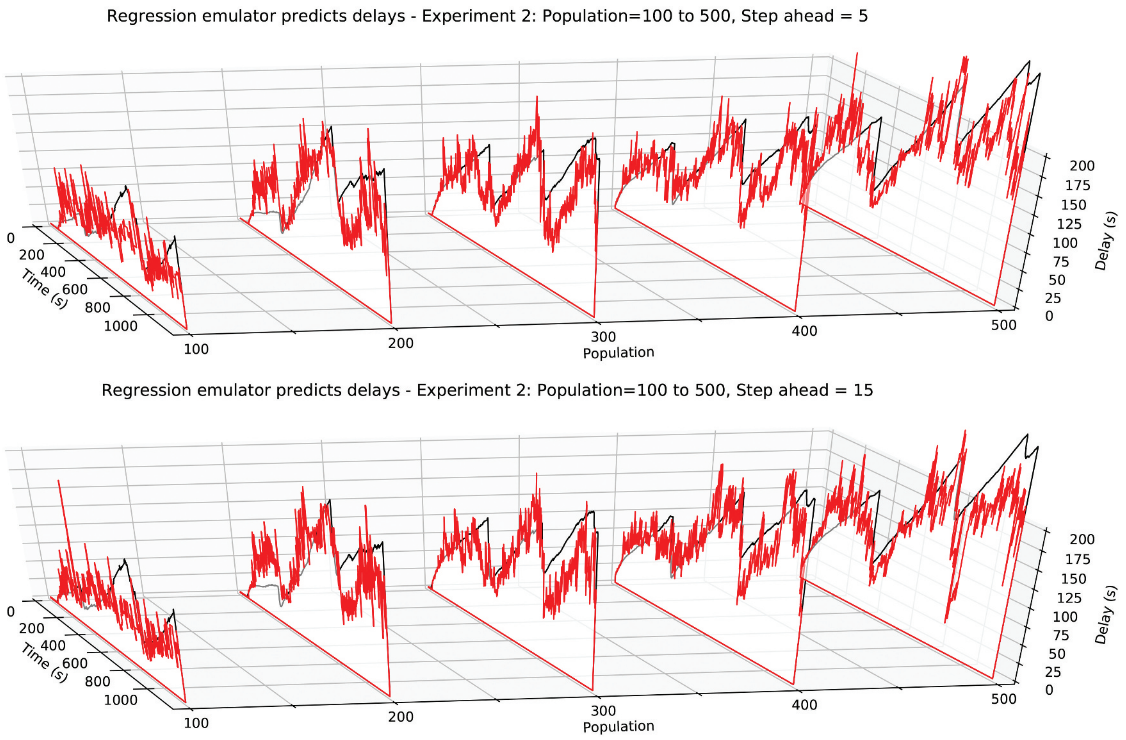

Figure 12. Experiment 2: prediction of delays using the regression emulator. red line: predicted delay, black line: synthetic real-time delay.

forecast period $k = 15$. The lighter colour illustrates better prediction accuracy. Three distinct patterns are visible:

- First, the prediction accuracy is better with smaller agent populations. This is expected because with fewer agents there are fewer collisions, so the behaviour of the agents is nearly deterministic.
- Second, the time-series emulator shows better prediction accuracy (lighter colours on Figure 11) especially at higher agent populations (400 and 500 agents). The regression emulator performs worse at the first sensor (Sensor 0), because it does not take into consideration the historical counts. When the train has not yet arrived and there are no pedestrians in the corridor, the

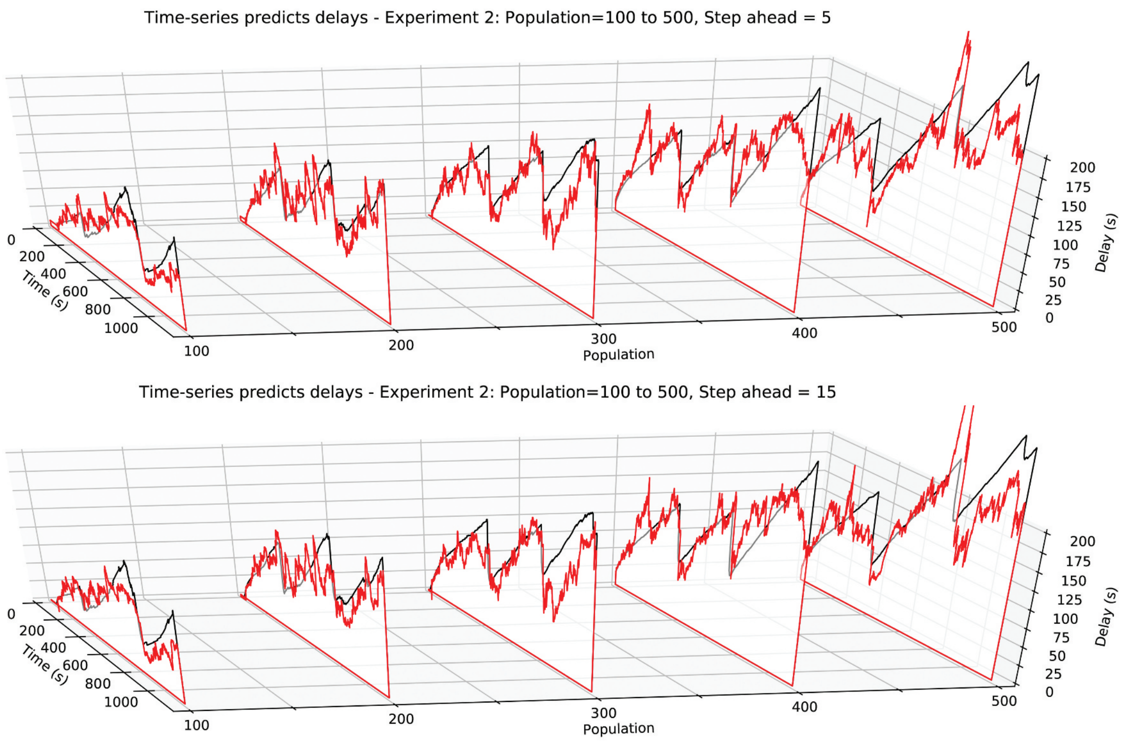

figure 13. Experiment 2: prediction of delays using the time-series emulator. red line: predicted delay, black line: synthetic real-time delay.

Table 2. Comparison of prediction accuracy: experiment 2.

	Regression emulator			Time-series emulator		
Population	5s	10s	15s	5s	10s	15s
100	23.58	23.50	25.14	**16.09**	**16.44**	**16.97**
200	29.13	29.59	30.43	**16.43**	**17.14**	**18.95**
300	26.68	26.00	26.70	**21.12**	**21.65**	**22.98**
400	21.25	23.61	24.95	**21.14**	**18.51**	**20.14**
500	32.30	35.38	33.85	**30.77**	**30.34**	**33.17**

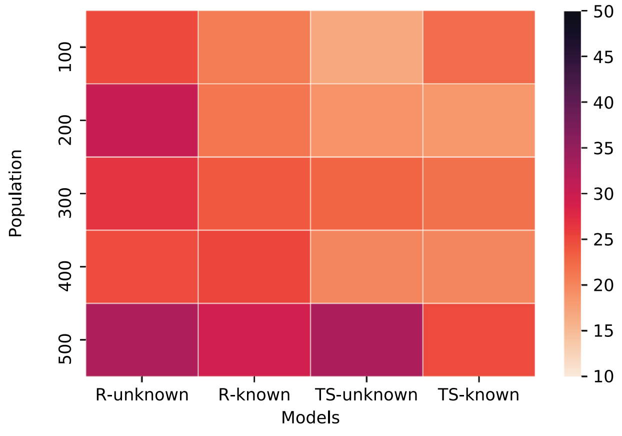

Figure 14. Comparison of the prediction results for delay. r-unknown: regression emulator predicts unknown population, r-known: regression emulator predict known population, ts-unknown: time-series emulator predicts unknown population, ts-known: time-series emulator predicts known population.

emulators still make a prediction based on historical patterns that some pedestrians should be in the corridor. The time-series emulator partly solves this problem by including more data from the near past time steps, as illustrated in Figure 11.

- Third, there are no significant differences in terms of prediction accuracy between the known and the unknown pedestrian population scenario (the colours are very similar from the left to the right of Figure 11) apart from a few occasions when the emulators perform slightly better when the pedestrian population is known. This pattern can be seen near Sensor 0, where differences in pedestrian demand are really important.

We will now look at the prediction of mean pedestrian delay in Figures 12 and 13. Both emulators capture the trends in mean delay, with comparable results to those of the Experiment 1 (see, Figures 8 and 9).

Table 2 compares the emulators in more detail. Again, we see that the prediction accuracy from the time-series emulator is significantly better than the regression emulator in all tests, by 22.1% on average. The time-series emulator is noticeably better in Experiment 2 – when the emulator has to generalise to make prediction – as the number of pedestrian is not observed. The LSTM takes advantage of the historical pedestrian counts from the last 5 time steps to predict much more accurately than the regression emulator.

Figure 14 shows a heat map of the Mean Absolute Error of the predictions of delay using the regression emulator (left two columns) and the time-series emulator (right two columns) when the emulators know the pedestrian population and with unknown population.

All the tests are done when the forecast period is $k = 15$. The lighter colour illustrates better prediction accuracy. Two patterns are visible:

- Similar to the prediction of counts, the prediction accuracy is better with smaller agent populations. The time-series emulators outperform the regression emulator counterparts.
- There is now a distinct difference between the prediction accuracy of delay when the population is known, versus when the population is known. Models with known populations perform much better.

6. Discussion and conclusion

6.1. Summary

This paper proposes a new framework that enables ABM researchers and practitioners to benefit from emulators to make predictions based on ABMs in real-time. The framework, as outlined in Figure 1, allows emulators to learn from the outputs of an ABM to achieve two major points. First, the emulators enable a real-time prediction using a fraction of the computation time required when compared to using the ABM to make predictions. We test two classes of emulators: a random forest regression and a time-series model. While they both need to be trained using the data generated from the ABM, which is an expensive process, they can provide predictions of pedestrian counts and mean delay. Second, the emulators enable us to predict directly from aggregated data (i.e., counts from each sensor), instead of relying on individual-level data. This is especially important if individual data are not available in real-time due to technical or privacy challenges, which is extremely common in practice.

Both emulators perform well at predicting aggregated pedestrian counts and mean delay for both Experiment 1 and 2. Between the two emulators, the time-series model performs slightly better in Experiment 1 (when the number of pedestrians is observed) and significantly better in Experiment 2 (when the number of pedestrians is unknown). Experiment 2 is more practical, showing the greater generalisability of the time-series emulator, as in practice the number of pedestrians entering the environment (i.e., travellers on a train) may be unknown. The time-series emulator generally takes more time to train, but takes less time to make a prediction. This is because the regression emulator needs 11 individual Random Forests to predict the $q = 11$ values of the output variable at each time step, while we need only one time-series model to make predictions for the same output. Therefore, the time-series emulator is more feasible to use in practice.

6.2. Caveats

This paper has focused on developing the framework and evaluating it using synthetic data. Although we have shown that the framework enables us to use emulators to predict the output of ABMs in real-time, a limitation of this paper is the lack of *real* data in the evaluation. The identical twin experimental framework has advantages in that it allows us to accurately calculate error because the "truth" is known, but it is limited in that the pseudo-truth data might not be as rich as data derived from a real system. In addition, we assume that the ABM that is used to train the emulator is able to adequately capture the dynamics of the target system. If the ABM is not able to model the target adequately – e.g., due to a lack of accurate observational data or unknown behavioural parameters – then the emulator will not be able to reliably make predictions about the real system. Future research directions include the use of real data for a pedestrian system, which would allow the same framework to be used with the only difference being that the ABM should be first calibrated to the real-world observations.

One of the advantages of the proposed framework is that the most computationally-intensive part – running the ABM a large number of times to generate training data for the emulators – can be conducted offline in advance. However, it is important to note that for larger, more complex agent-based models, the time taken for them to execute, and the amount of data they generate, might become problematic. One solution to this problem would be to adapt the framework so that the creation of synthetic data and the training of the emulators could take place simultaneously, rather than creating all of the synthetic data in one step, and then training the emulators in the next. Whilst this does not solve the problems associated with computationally-expensive ABMs, it significantly reduces the amount of data storage required – once a model result has been incorporated into the emulator it can be discarded – and reduces the extra time that would be required to train the emulator after the ABM has finished running.

7. Conclusion

This paper proposes a new framework to enable real-time agent-based modelling through the use of emulators. Experiments with a regression emulator and a time-series emulator suggest that the time-series emulator proves to generalise better to cases where the number of agents in the system is not known *a priori*. The results have implications for the real-time modelling of human crowds, suggesting that emulation is a feasible approach to modelling crowds in real-time, where computational complexity prohibits the use of an ABM directly.

Disclosure statement

No potential conflict of interest was reported by the author(s).

Funding

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme [grant agreement No. 757455].

ORCID

Minh Kieu http://orcid.org/0000-0001-7798-6195
Hoang Nguyen http://orcid.org/0000-0002-1489-0142
Jonathan A. Ward http://orcid.org/0000-0002-2469-7768
Nick Malleson http://orcid.org/0000-0002-6977-0615

References

Baker, E., Challenor, P., & Eames, M. (2019). Diagnostics for stochastic Gaussian process emulators. *arXiv:1902.01289 [stat.ME]*. https://arxiv.org/abs/1902.01289

Balmer, M., Rieser, M., Meister, K., Charypar, D., Lefebvre, N., & Nagel, K. (2009). Matsim-t: Architecture and simulation times. In A. Bazzan & F. Klügl(Eds.), *Multi-agent systems for traffic and transportation engineering* (pp. 57–78). IGI Global.

Barnes, C. M., Ghouri, A., & Lewis, P. R. (2021). Explaining evolutionary agent-based models via principled simplification. *Artificial Life*, *27*(3–4), 143–163. https://doi.org/10.1162/artl_a_00339

Bastos, L. S., & O'Hagan, A. (2009). Diagnostics for Gaussian process emulators. *Technometrics*, *51*(4), 425–438. https://doi.org/10.1198/TECH.2009.08019

Bijak, J., Hilton, J., Silverman, E., & Cao, V. D. (2013). From agent-based models to statistical emulators. *Joint Eurostat/ UNECE Work Session On Demographic Projections* (pp. 12).

Bonabeau, E. (2002). Agent based modeling: Methods and techniques for simulating human systems. *Proceedings of the National Academy of Sciences*, *99*(90003), 7280–7287. https://doi.org/10.1073/pnas.082080899

Breiman, L. (2001). Random forests. *Machine Learning*, *45* (1), 5–32. https://doi.org/10.1023/A:1010933404324

Clay, R., Kieu, L.-M., Ward, J. A., Heppenstall, A., & Malleson, N. (2020). Advances in practical applications of agents, multi-agent systems, and trustworthiness.The PAAMS collection. In Y. Demazeau, T. Holvoet, J. M. Corchado, & S. Costantini (Eds.), *Towards realtime crowd simulation under uncertainty using an agent-based model and an unscented Kalman filter* (Vol. 12092, pp. 68–79). Springer.

Conti, S., Gosling, J. P., Oakley, J. E., & O'Hagan, A. (2009). Gaussian process emulation of dynamic computer codes. *Biometrika*, *96*(3), 663–676. https://doi.org/10.1093/biomet/asp028

Conti, S., & O'Hagan, A. (2010). Bayesian emulation of complex multi-output and dynamic computer models. *Journal of Statistical Planning and Inference*, *140*(3), 640–651. https://doi.org/10.1016/j.jspi.2009.08.006

Deissenberg, C., Van Der Hoog, S., & Dawid, H. (2008). Eurace: A massively parallel agent-based model of the European economy. *Applied Mathematics and Computation*, *204*(2), 541–552. https://doi.org/10.1016/j.amc.2008.05.116

Dosi, G., Pereira, M. C., Roventini, A., & Virgillito, M. E. (2018). The effects of labour market reforms upon unemployment and income inequalities: An agent-based model. *Socio-Economic Review*, *16*(4), 687–720. https://doi.org/10.1093/ser/mwx054

Farah, M., Birrell, P., Conti, S., & Angelis, D. D. (2014). Bayesian emulation and calibration of a dynamic epidemic model for A/H1n1 influenza. *Journal of the American Statistical Association*, *109*(508), 1398–1411. https://doi.org/10.1080/01621459.2014.934453

Heard, D. (2014). *Statistical inference utilizing agent based models* [PhD thesis]. Duke University.

Helbing, D., Farkas, I., & Vicsek, T. (2000). Simulating dynamical features of escape panic. *Nature*, *407*(6803), 487. https://doi.org/10.1038/35035023

Henein, C. M., & White, T. (2005). Agent-based modelling of forces in crowds. In P. Davidsson, B. Logan, & K. Takadama (Eds.), *Multi-agent and multi-agent-based simulation* (pp. 173–184). Springer.

Higdon, D., Gattiker, J., Williams, B., & Rightley, M. (2008). Computer model calibration using high-dimensional output. *Journal of the American Statistical Association*, *103*(482), 570–583. https://doi.org/10.1198/016214507000000888

Hilton, J. (2017). *Managing uncertainty in agent-based demographic models* [PhD thesis]. University of Southam- ton

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*(8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

Kieu, L.-M., Malleson, N., & Heppenstall, A. (2020). Dealing with uncertainty in agent-based models for short- term predictions. *Royal Soc Ety Open Science*, *7*(1), 191074. https://doi.org/10.1098/rsos.191074

Krasnopolsky, V. M., Fox-Rabinovitz, M. S., & Chalikov, D. V. (2005). New approach to calculation of atmospheric model physics: Accurate and fast neural network emulation of longwave radiation in a climate model. *Monthly Weather Review*, *133*(5), 1370–1383. https://doi.org/10.1175/MWR2923.1

Lafuerza, L. F., Dyson, L., Edmonds, B., & McKane, A. J. (2016a). Simplification and analysis of a model of social interaction in voting. *The European Physical Journal. B*, *89*(7), 159. https://doi.org/10.1140/epjb/e2016-70062-2

Lafuerza, L. F., Dyson, L., Edmonds, B., & McKane, A. J. (2016b). Staged models for interdisciplinary research. *PloS one*, *11*(6), e0157261. https://doi.org/10.1371/journal.pone.0157261

Malleson, N., Minors, K., Kieu, L.-M., Ward, J. A., West, A., & Heppenstall, A. (2020). Simulating crowds in real time with agent-based modelling and a particle filter. *Journal of Artificial Societies and Social Simulation*, *23*(3), 3. https://doi.org/10.18564/jasss.4266

Moutoussamy, V., Nanty, S., & Pauwels, B. (2015). Emulators for stochastic simulation codes. *ESAIM: Proceedings and Surveys*, *48* , 116–155. https://doi.org/10.1051/proc/201448005

Niemann, J.-H., Winkelmann, S., Wolf, S., & Schütte, C. (2021). Agent-based modeling: Population limits and large timescales. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, *31*(3), 033140. https://doi.org/10.1063/5.0031373

Oakley, J., & O'Hagan, A. (2002). Bayesian inference for the uncertainty distribution of computer model outputs. *Biometrika*, *89*(4), 769–784. https://doi.org/10.1093/biomet/89.4.769

Oyebamiji, O. K., Wilkinson, D. J., Jayathilake, P. G., Curtis, T. P., Rushton, S. P., Li, B., & Gupta, P. (2017). Gaussian process emulation of an individual-based model simulation of microbial communities. *Journal of Com Putational Science*, *22*, 69–84. https://doi.org/10.1016/j.jocs.2017.08.006

Oyebamiji, O. K., Wilkinson, D. J., Li, B., Jayathilake, P. G., Zuliani, P., & Curtis, T. P. (2019). Bayesian emulation and calibration of an individual-based model of microbial communities. *Journal of Computational Science*, *30*, 194–208. https://doi.org/10.1016/j.jocs.2018.12.007

Rasouli, S., & Timmermans, H. (2013). Using emulators to approximate predicted performance indicators of complex microsimulation and multiagent models of travel demand. *Transportation Letters*, *5*(2), 96–103. https://doi.org/10.1179/1942786713Z.0000000008

Rhodes, D. M., Holcombe, M., & Qwarnstrom, E. E. (2016). Reducing complexity in an agent based reaction model—benefits and limitations of simplifications in relation to run time and system level output. *Biosystems*, *147*, 21–27. https://doi.org/10.1016/j.biosystems.2016.06.002

Schoenharl, T., & Madey, G. (2011). Design and implementation of an agent-based simulation for emergency response and crisis management. *Journal of Algorithms & Computational Technology*, *5*(4), 601–622. https://doi.org/10.1260/1748-3018.5.4.601

Shrestha, D. L., Kayastha, N., & Solomatine, D. P. (2009). A novel approach to parameter uncertainty analysis of hydrological models using neural networks. *Hydrology and Earth System Sciences*, *13*(7), 1235–1248. https://doi.org/10.5194/hess-13-1235-2009

Swarup, S., & Mortveit, H. S. (2020). Live simulations. *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent. Systems, AAMAS '20*, Auckland, New Zealand. (pp. 1721–1725). International Foundation for Autonomous Agents and Multiagent Systems.

Tregubov, A., & Blythe, J. (2020). Optimization of large-scale agent-based simulations through automated abstraction and simplification. *International Workshop on Multi-Agent Systems and Agent-Based Simulation*, Auckland, New Zealand. (pp. 81–93). Springer.

Wang, M., & Hu, X. (2015). Data assimilation in agent based simulation of smart environments using particle filters. *Simulation Modelling Practice and Theory*, *56*(56), 36–54. https://doi.org/10.1016/j.simpat.2015.05.001

Zhou, B., Wang, X., & Tang, X. (2012). Understanding collective crowd behaviors: Learning a mixture model of dynamic pedestrian-agents In *2012 IEEE Conference on Computer Vision and Pattern Recognition* (pp. 2871--2878), Providence, RI: IEEE.

Appendix. Long Short-term Memory

This appendix briefly describes the Long Short-term Memory (LSTM), a recurrent neural network (RNN). The RNN was first introduced where a loop connection is added to each unit along a sequence (e.g., organised by timestamps) mikolov2010recurrent. A hidden vector **h** is maintained in the RNN, which is updated at time stamp t via

$$\mathbf{h_t} = \tanh(\mathbf{W} * \mathbf{h_{t-1}} + \mathbf{I} * x_t) \tag{A1}$$

where **W** is the recurrent weight matrix and I is a projection matrix. Then a prediction will be made based on the hidden state **h** via

$$\mathbf{y}_t = softmax(\mathbf{W} * \mathbf{h}_{t-1}), \tag{A2}$$

where the function *softmax* provides a normalised probability distribution over the possible classes. By using **h** as the input to another RNN, we can stack RNNs, creating deeper architectures pascanu2013construct, i.e.,

$$\mathbf{h}_t^l = \sigma(\mathbf{W} * \mathbf{h}_{t-1}^l + \mathbf{I} * \mathbf{h}_t^{l-1}). \tag{A3}$$

As a consequence, an RNN can be considered as multiple repetitions of the same structure, where each is connected to the successor. The internal design of an RNN layer with compatible unrolled network is presented in Figure A1.

A chunk of neural network **M** takes some input X_t and outputs a value h_t. The RNN architecture can be seen as multiple copies of the same network M, each transmitting a message to a successor, as also illustrated in Figure A1.

The RNN architecture can record the information for a few previous timestamps. However, while RNN outperforms ANN in modelling the temporal dependency between sequences, the RNN has the problem of long-term dependency. Instead of having a "memory" to store the information, RNN simply carries the information forward and updates it in every time step sak2014long. In other words, RNN only uses recent information to perform the present task, which is not ideal for dealing with time-series prediction.

This shortcoming is addressed by the Long Short-Term Memory (LSTM) network hochreiter 1997 long, which is an extended RNN architecture capable of learning long-term dependencies. LSTM carries the information forward through multiple time steps as a long-term memory. It has a similar chain architecture as in the RNN but the repeating modules have a special internal design. An RNN module only has single neural network layer (e.g., tanh layer) while there are four different interacting layers inside a LSTM module. Figure A2 illustrates a standard LSTM with four interacting layers.

At each time stamp, a hidden vector **h** and a memory vector **m** is maintained by the LSTM, which are responsible for controlling state updates and outputs.

The information to be "forgotten" when passing through a gate is controlled by the forget function f_t,

$$\mathbf{f}_t = \sigma(\mathbf{W}^f \times [h_{t-1}, x_t] + b_f). \tag{A4}$$

There are two steps which control the information to be carried in the cell state. Firstly, a sigmoid layer called the "input gate layer",

$$\mathbf{i}_t = \sigma(\mathbf{W}^i \times [h_{t-1}, x_t] + b_i), \tag{A5}$$

decides which values are to be updated. Next, a vector of new candidate values is generated by a *tanh* layer,

$$\tilde{\mathbf{C}}_\mathrm{t} - \tanh(\mathbf{W}^c \times [h_{t-1}, x_t] + b_c), \tag{A6}$$

that could be added to the state.

The current cell state $\tilde{C}_t$ will be updated by multiplying the forget function f_t with the old state, then combining with $i_t * \tilde{C}_t$, i.e.,

$$\mathbf{C}_t = \mathbf{f}_t \times \mathbf{C}_{t-1} + \mathbf{i}_t \times \mathbf{C_t}. \tag{A7}$$

Finally, the output of a cell state is decided by

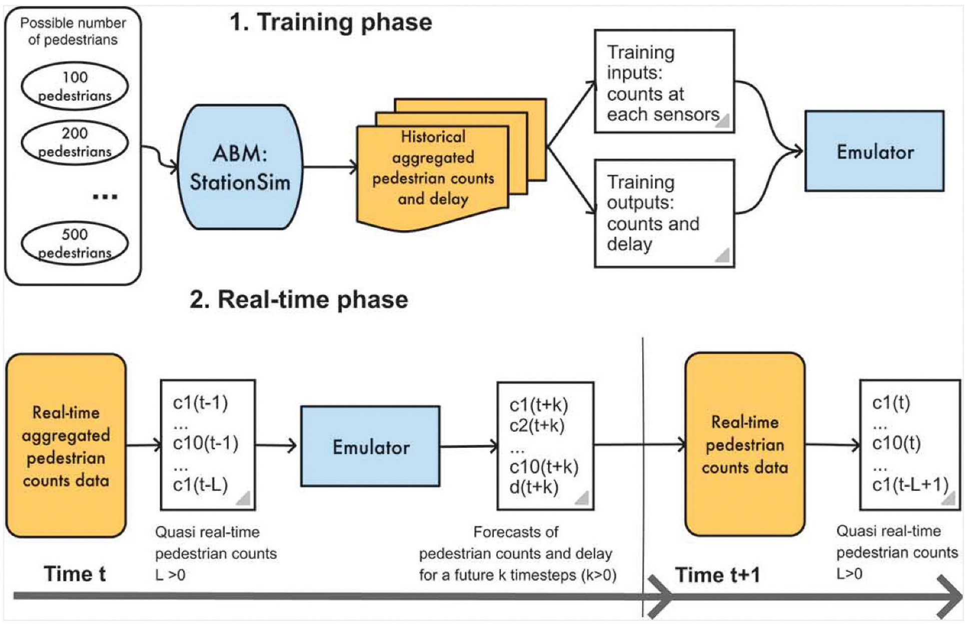
1. Training phase
Possible number of pedestrians
100 pedestrians
200 pedestrians
...
500 pedestrians
ABM: StationSim
Historical aggregated pedestrian counts and delay
Training inputs: counts at each sensors
Training outputs: counts and delay
Emulator
2. Real-time phase
Real-time aggregated pedestrian counts data
c1(t-1) ... c10(t-1) ... c1(t-L)
Emulator
c1(t+k) c2(t+k) ... c10(t+k) d(t+k)
Real-time pedestrian counts data
c1(t) ... c10(t) ... c1(t-L+1)
Quasi real-time pedestrian counts L >0
Forecasts of pedestrian counts and delay for a future k timesteps (k>0)
Quasi real-time pedestrian counts L>0
Time t
Time t+1

Figure A1. RNN and unrolled RNN.

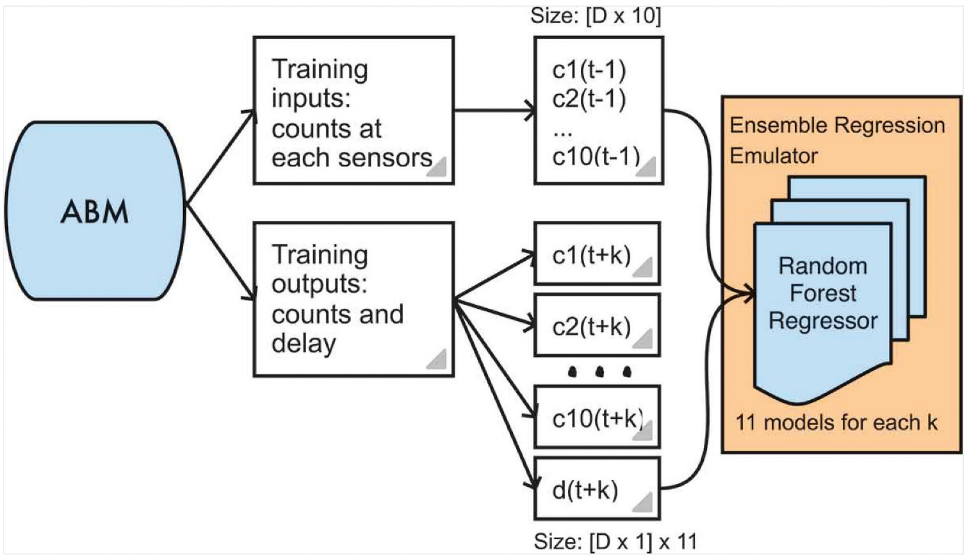

Figure A2. A standard LSTM module with four interacting layers.fv

$$\mathbf{o}_t = \sigma(\mathbf{W}^o \times [h_{t-1}, x_t] + b_o) \tag{A8}$$

and

$$\mathbf{h}_t = \mathbf{o}_t \times \tanh(\mathbf{C_t}), \tag{A9}$$

where σ is the sigmoid function, $\mathbf{W}^f, \mathbf{W}^i, \mathbf{W}^C, \mathbf{W}^o$ are recurrent weight matrices and b_f, b_i, b_C, b_o are projection matrices.

Besides different module structure, the key factor in the LSTM is the connection line (on the top of the diagram) running through the entire architecture with only simple linear combinations. This connection allows long-term information to flow smoothly along the sequences. Each cell state in a LSTM is able to control the amount of information allowed to pass through the cell using the regulated elements called gates. A sigmoid layer under a point-wise multiplication operation will make up a gate and a LSTM has three of them. Because a sigmoid layer outputs numbers between zero and one, the amount of information passing through a gate can range from “nothing” to “everything”.