Transportation Research Part C 166 (2024) 104788

Contents lists available at ScienceDirect

Transportation Research Part C

journal homepage: www.elsevier.com/locate/trc

Explainable active learning metamodeling for simulations: Method and experiments for ATM performance assessment

Christoffer Riis [a,*], Francisco Antunes [b], Tatjana Bolić [c], Gérald Gurtner [c], Andrew Cook [c], Carlos Lima Azevedo [a], Francisco Câmara Pereira [a]

[a] Technical University of Denmark; DTU Management, Machine Learning for Smart Mobility, Denmark
[b] University of Coimbra; Department of Informatics Engineering, Centre for Informatics and Systems, Portugal
[c] University of Westminster; Centre for ATM Research, School of Architecture and Cities, England, United Kingdom

ARTICLE INFO

Keywords:
Air traffic management
Simulation metamodelling
Active learning
Gaussian processes
Machine learning
SHAP values

ABSTRACT

The use of Air traffic management (ATM) simulators for planing and operations can be challenging due to their modelling complexity. This paper presents XALM (eXplainable Active Learning Metamodel), a three-step framework integrating active learning and SHAP (SHapley Additive exPlanations) values into simulation metamodels for supporting ATM decision-making. XALM efficiently uncovers hidden relationships among input and output variables in ATM simulators, which are usually of interest in policy analysis. Our experiments show that XALM's predictive performance is comparable to that of the XGBoost metamodel with fewer simulations. Additionally, XALM exhibits superior explanatory capabilities compared to non-active learning metamodels.

Using the 'Mercury' (flight and passenger) ATM simulator, XALM is applied to a real-world scenario in Paris Charles de Gaulle airport, extending an arrival manager's range and scope by analysing six variables. This case study illustrates the effectiveness of the proposed framework in enhancing simulation interpretability and understanding variable interactions. By addressing computational challenges and improving explainability, it complements traditional simulation-based analyses.

Lastly, we discuss two practical approaches for reducing the computational burden of the metamodelling further: we introduce a stopping criterion for active learning based on the inherent uncertainty of the metamodel, and we show how the simulations used for the metamodel can be reused across key performance indicators, thus decreasing the overall number of simulations needed.

1. Introduction

Accurately modelling modern Air Traffic Management (ATM) systems is challenging due to their complexity. ATM systems involve a wide range of stakeholders, variables, uncertainty, and human behaviour, which interact across both airspace and ground operations levels. Moreover, trying to ensure the safety and efficiency of air traffic flows are encoded in the model introduces further complications, as first set out in Cook and Rivas (2016). However, fast-time simulation[1] approaches provide a way to study and

* Corresponding author.
E-mail address: chrrii@dtu.dk (C. Riis).

[1] Fast-time simulation is a term used in ATM simulations, describing the use of computer models, i.e. simulators. This is to distinguish from real-time simulations that involve humans in the loop when simulating new procedures or tools. The reader is referred to the first book published on European ATM principles (Cook, 2007).

https://doi.org/10.1016/j.trc.2024.104788
Received 31 July 2023; Received in revised form 5 May 2024; Accepted 16 July 2024
Available online 9 August 2024

model ATM systems (Riis et al., 2021; Sánchez-Cauce et al., 2022). These approaches allow researchers and practitioners to explore, test, and propose designs and solutions that would be impractical to implement in the real world. Simulators provide a virtual environment in which these studies can be conducted. Technically, a 'simulator' is the computer implementation of a conceptual 'simulation model', but we use both terms interchangeably in this paper for simplicity.

Despite the benefits of using simulation-based solutions for modelling complex, real-world, stochastic systems and their evolution over time, they often have limited transparency and interpretability regarding the details of the underlying simulation model and the results obtained for *non-expert end-users and policymakers*. This is largely a consequence of the nature of simulation modelling rather than a deliberate design choice made by the simulators' developers. While it is expected that the internal components of such simulators (such as equations, functions, and theoretical foundations) and their logical interactions can be clearly identified and understood, the external, emergent behaviour that may arise may be rather less clear. This can be particularly challenging for end-users, policymakers, and non-experts who are not familiar with the intricate implementation details thereof.

ATM performance assessment in Europe

An important element of SESAR,[2] as the technological pillar of the Single European Sky initiative, is to bring about improvements, as measured through specific key performance indicators (KPIs), and as implemented by a series of so-called SESAR 'Solutions'. SESAR 'Solutions' represent a change in the way air traffic management is performed and are new operational concepts, procedures, and relevant technologies.[3]

Central to performance assessment in SESAR is its Performance Framework. This is partly supported by the European Air Traffic Management Architecture (EATMA), which is the common architecture framework for SESAR, and the means of integrating operational and technical content developments. In various SESAR contexts, the term 'metamodel' is used to describe logical entity relationships, e.g. for performance data and as an architecture mapping and database model, not in a sense presented in this paper, as a model of a model.

Different SESAR Solutions variously deploy different simulations to demonstrate their expected performance contributions across the International Civil Aviation Organisation (ICAO) set of eleven key performance areas (KPAs). They use a number of specific KPIs defined in the Performance Framework. The Solutions are, indeed, compelled to assess performance expectations as part of the SESAR programme. This brings challenges in terms of computational effort, simulation consistency, assessing KPI interdependencies and general integration.

Simulation-based studies in ATM often thus focus on assessing the performance impact of proposed solutions and concepts (SESAR Joint Undertaking, 2018, 2020; Bolić and Ravenhill, 2021), on existing or planned systems, usually focusing on a single solution. Simulation studies often rely on manual exploration of the underlying simulator's behaviour, using domain knowledge, expert-driven scenario design, and 'what-if' approaches to reduce and discretise the input space into a limited set of possible system states to investigate. While these methodologies can be useful for limiting the space of investigation and communicating results to stakeholders, there is a risk of failing to properly assess the behaviour of the simulated system in its entirety, leaving relevant simulation input regions unexplored. As a result, the simulation analysis is restricted to small regions of the uncertainty space, limiting the insights and conclusions that can be drawn from the simulation. To this end, particularly meaningful exploration of new scenarios through complex simulators is often impractical and expensive in the effort required for purposeful analyses of results.

Exploration of simulators The ATM research and development community is thus well-acquainted with the trade-offs between the advantages and disadvantages of fast-time simulation modelling, as they have a long history of using such techniques (Phillips and Marsh, 2000; Cook, 2007; EUROCONTROL, 2010; Gurtner et al., 2017; Delgado et al., 2021). While simulation models are typically simplified representations of real systems (Law, 2015), they can still be complex software programs with many input and output variables and parameters, requiring a large amount of data and computational resources for calibration. The complexity of a simulation model increases with the degree of detail, realism, and purpose of the simulation, as well as the complexity of the system being modelled and the problem being addressed. The large dimensional sizes and value ranges of input spaces can be a significant burden when exploring the behaviour of the simulation as a whole. In addition, the lack of an explicit and manageable closed-form function, as opposed to pure analytical approaches, can make it difficult to understand the true impact of input variables and parameters on the system's output metrics or KPIs and their interrelationships.

To realise the full potential of advanced ATM simulators, the researcher must overcome the computational burden of using the simulator and the subsequent results analyses. One way to reduce the computational burden of using advanced ATM simulators is to combine them with other methods, such as *metamodels*. These methods can interpolate the simulator's outputs as well as the associated uncertainty and variance, thus enabling the researcher to effectively explore the scenario space of the simulator without being limited by the computational cost of running the simulator extensively (Kleijnen, 1997). Therefore, metamodels are an important tool for policymakers and managers to utilise advanced air traffic management simulators efficiently.

Metamodels Metamodelling is a framework for approximating the unknown relationship between the inputs and outputs of a simulation model with an explicit, known functional form, such as a statistical or machine learning model (Kleijnen, 1997; Friedman, 2012). We focus on simulation metamodels, which are specifically designed to reproduce the behaviour of simulation models (Kleijnen and Sargent, 2000; Friedman, 2012; Gramacy, 2020).

A simulation metamodel (for simplicity, 'metamodel') can be seen as an abstraction of a simulator, in the same way that a simulator represents an abstraction of a real-world system or phenomenon, as illustrated in Fig. 1. The main advantage of metamodels

[2] https://www.sesarju.eu

[3] A recently-launched 'digital catalogue' is available at https://www.sesarju.eu/catalogue.

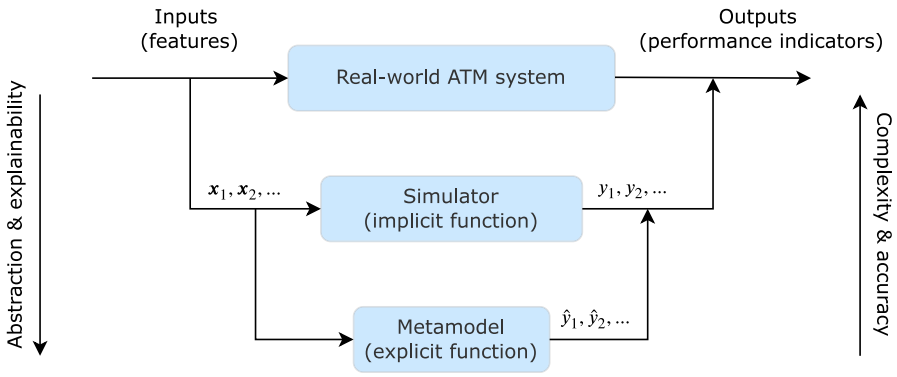

Fig. 1. Relationship between the real-world system under study, the simulator, and the metamodel.

is that they can reduce the computational cost of running time-consuming simulation experiments by exploiting their approximate nature, functional simplicity, and fast computing. Since they approximate the underlying simulation functions, metamodels can achieve a balance between computational speed and controlled accuracy loss, depending on their goals. The metamodels work by mimicking the output behaviour of the simulator as an analytical function of its inputs. While simulators often have complex internal relationships and dynamics, they can be treated as a 'black-box' function with no clear mathematical formula. Nonetheless, the 'emergent behaviour', resulting from the simulators' inner interactions and dynamics that evolve over time, is what the metamodels approximate.

Metamodelling provides a framework to explore the output behaviour of the simulator efficiently, and this has been applied to many problems in transportation, *inter alia*, for calibration of, e.g. origin–destination demand for large networks (Dantsuji et al., 2022), day-to-day dynamics (Cheng et al., 2019), and, more generally, a microscopic traffic simulator (Ciuffo and Azevedo, 2014), and for optimisation of, e.g. bus lane allocation (Li et al., 2022) and traffic signal control (Osorio and Bierlaire, 2013). However, in ATM research, this is quite recent (Riis et al., 2021; Sánchez-Cauce et al., 2022; Cano et al., 2023).

Given a metamodel, it is possible to evaluate thousands of simulation scenarios in a few seconds. Thus, metamodelling enables efficient exploration of the output behaviour of a simulator using computationally fast statistical models. However, by approximating the simulator with a machine learning model, we have, in practice, effectively replaced one black-box model with another, and thus, the internal behaviour and its interactions are still not observable.

Explainable metamodel Recently, Riis et al. (2022) proposed an explainable metamodel by augmenting the metamodel with SHapley Additive exPlanations (SHAP) values (Lundberg and Lee, 2017), enabling the extraction of significantly greater insights from the metamodel. The results show that the proposed methodology can effectively make simulators and their results more explainable, facilitating the interpretation of the obtained associated emergent behaviour and opening new opportunities for novel performance assessment processes in ATM. However, as (Riis et al., 2022) consider, a metamodel based on a fixed data set – a single design of experiment – consisting of 50,000 simulations, they only show the usage of SHAP values given a very large computational budget. In practice, the available computational budget is usually much smaller, and the real potential of metamodels is only really obtainable if the metamodel is constructed using active learning (Burr Settles, 2010), i.e. using active learning metamodels (Antunes et al., 2018; Riis et al., 2022; Sánchez-Cauce et al., 2022).

Our contribution In this paper, we contribute to the field of applied metamodelling for air traffic management performance simulations. The main contribution lies in the introduction of our eXplainable Active Learning Metamodel (XALM), a three-step framework that integrates active learning, metamodelling, and SHAP values. XALM efficiently uncovers hidden relationships among input and output variables in complex simulated ATM systems, supporting ATM decision-making. We show the predictive performance of XALM through our experiments, achieving comparable results to the XGBoost (eXtreme Gradient Boosting) metamodel while using fewer simulations. Furthermore, XALM exhibits superior explanatory capabilities compared to non-active learning metamodels. The practical application of XALM to a real-world scenario in Paris Charles de Gaulle airport, extending the arrival manager's range and scope through the analysis of six variables, highlights its effectiveness in enhancing simulation interpretability and understanding variable interactions. We also discuss two practical approaches to further reduce computational burden: the introduction of a stopping criterion for active learning based on metamodel uncertainty and the reuse of queried simulations for different key performance indicators.

2. Background

In this section, we present the three distinct techniques and components that support the proposed explainable active learning metamodelling framework. We explain (1) the active learning approach, (2) the modelling details of the employed metamodel (a Gaussian process), and (3) the explainable component in the form of the SHAP values.

First, however, we present the notation used in the following sections. In the context of machine learning, a dataset is composed of individual data points, often organised in a matrix form, where the columns represent the dimensions (also known as features) and

the rows represent the observations, additionally being either labelled or unlabelled. A labelled data point contains both an input feature vector, denoted by $\boldsymbol{x}$, and its corresponding output label, denoted by y, whereas an unlabelled data point only contains the features $\boldsymbol{x}$. In the case of data generated from a simulator, acquiring an unlabelled data point $\boldsymbol{x}$ is usually straightforward (specify the input variables to the simulator), whereas obtaining a labelled data point $(\boldsymbol{x}, y)$ requires running the simulator with the input $\boldsymbol{x}$ to obtain the corresponding output y (here the key performance indicator computed by the simulator).

2.1. Active learning

While a metamodel is computationally fast compared to the corresponding simulator, the metamodel can only approximate the simulator by actually 'seeing' simulations from the simulator, meaning that to use the metamodel to avoid running the computationally expensive simulations, one first must run a few sets of simulations on which the metamodel will be trained. The framework of *active learning* provides an efficient way to choose which simulations to run in order to make a good approximation in terms of high prediction performance while using the fewest resources spent on simulations.

Active learning aims to select the most informative data points sequentially to improve model prediction performance and control costs. The learning process is guided by a label provider, also called the oracle (in our case, a simulator), which provides labelled data instances that are incorporated into the training data set. The oracle must be permanently accessible to be queried on-demand by the learning algorithm or model and should have the ability to generate labelled instances consistently from the ground truth function defining the process under study. Formally, and following the notation introduced by Li and Sethi (2006), the five fundamental entities of an active learning system are the labelled training set $\mathcal{L}$, the set of unlabelled data points $\mathcal{V}$, the statistical model $\mathcal{M}$ (in our case, the metamodel), the oracle $\mathcal{O}$, and the query function $\mathcal{Q}$, which encodes the strategies and criteria for selecting informative instances from $\mathcal{V}$ to add to $\mathcal{L}$.

For a specific task, both the oracle, the labelled training set, and the unlabelled data points are often defined by the problem under study, whereas the querying strategy $\mathcal{Q}$ and the model $\mathcal{M}$ must be adapted for the task under study. Active learning uses different querying strategies (also known as acquisition functions) $\mathcal{Q}$, including uncertainty sampling (MacKay, 1992), query-by-committee (RayChaudhuri and Hamey, 1995), and variance reduction (Gramacy, 2020). These strategies rely either on the measure of informativeness or the identification of uncertainty regions to select the most informative instances to be added to the labelled training set.

2.2. Gaussian processes

Gaussian processes (GPs) are kernel methods that can be utilised for regression problems in a comparable way to linear regression. They are highly versatile and offer a Bayesian inference framework, being frequently used in both active learning and metamodelling applications. GPs have gained immense popularity for metamodelling tasks across various fields due to their modelling flexibility (Van Beers and Kleijnen, 2004; Kleijnen, 2009; Gramacy, 2020). While initially used for deterministic simulations, the application of GPs was eventually widened also to include stochastic simulation settings (Van Beers and Kleijnen, 2003). Currently, despite vast developments in the machine learning field, traditional GPs, or variations thereof, stand as a common default choice for metamodelling (Erickson et al., 2018; Yue et al., 2020; Knudde et al., 2020; Jiang et al., 2022; Sauer et al., 2023). For a comprehensive introduction to GPs, mostly from a machine learning viewpoint, refer to Rasmussen and Williams (2006).

A GP is a stochastic function fully defined by a mean function $m(\cdot)$ and a covariance function (often called a kernel) $k(\cdot,\cdot)$. Given the data $\mathcal{L} = (X, \boldsymbol{y}) = \{\boldsymbol{x}_i, y_i\}_{i=1}^{N}$, where y_i is the corrupted observation of some latent function $\boldsymbol{f}$ with Gaussian noise ε, i.e. $y_i = f_i + \varepsilon_i,\ \varepsilon_i \in \mathcal{N}(0, \sigma_\varepsilon^2)$, a GP is typically denoted as $\mathcal{GP}(m_{\boldsymbol{f}}(\boldsymbol{x}_i), k_{\boldsymbol{f}}(\boldsymbol{x}_i, \boldsymbol{x}_j))$, with $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ being any two distinct input observations. It is common practice to set the mean function $m_{\boldsymbol{f}}(\boldsymbol{x}_i)$ equal to the zero-value vector and thus, the GP is fully determined by the kernel $k_{\boldsymbol{f}}(\boldsymbol{x}_i, \boldsymbol{x}_j)$. For brevity, we will denote the kernel $K_{\boldsymbol{\theta}}$, which explicitly states that the kernel is parameterised with some hyperparameters $\boldsymbol{\theta}$. Given $\boldsymbol{\theta}$, usually obtained via maximum likelihood estimation methods, the predictive posterior for unknown test inputs $X^\star$ is given by $p(\boldsymbol{f}^\star|\boldsymbol{\theta}, \boldsymbol{y}, X, X^\star) = \mathcal{N}(\boldsymbol{\mu}^\star, \boldsymbol{\Sigma}^\star)$ with

$$\boldsymbol{\mu}^\star = K_{\boldsymbol{\theta}}^\star \left(K_{\boldsymbol{\theta}} + \sigma_\varepsilon^2 \mathbb{I}\right)^{-1} y \quad \text{and} \quad \boldsymbol{\Sigma}^\star = K_{\boldsymbol{\theta}}^{\star\star} - K_{\boldsymbol{\theta}}^\star \left(K_{\boldsymbol{\theta}} + \sigma_\varepsilon^2 \mathbb{I}\right)^{-1} K_{\boldsymbol{\theta}}^{\star\top}, \tag{1}$$

where $K_{\boldsymbol{\theta}}^{\star\star}$ denotes the covariance matrix between the test inputs, and $K_{\boldsymbol{\theta}}^\star$ denotes the covariance matrix between the test inputs and training inputs. For the kernel, we use a popular and general-purpose kernel, namely, the Radial-Basis Function (RBF) with Automatic Relevance Determination (ARD) given by

$$k\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right) = \sigma \exp\left(-\frac{1}{2}(\boldsymbol{x}_i - \boldsymbol{x}_j)^\top \Lambda^{-1} (\boldsymbol{x}_i - \boldsymbol{x}_j)\right), \tag{2}$$

where Λ is a diagonal matrix, where the diagonal is the vector of length scales $\boldsymbol{\ell} = \ell_1, \ldots, \ell_d$, one for each input dimension, and σ is a scalar for the output variance.

2.3. SHAP values

SHapley Additive exPlanations (SHAP) is a unified framework for interpreting model predictions (Lundberg and Lee, 2017). The Shapley values originate from game theory and have lately been used in machine learning to enhance model explainability. To explain a complex predictive model, SHAP constructs a simpler explanation model with M simplified input variables. SHAP builds upon additive variables attribution methods, which have an explanation model that is a linear combination of binary variables, i.e. $g(\boldsymbol{z}) = \phi_0 + \sum_{i=1}^{M} \phi_i z_i$, where $z_i \in \{0,1\}^M$ and $\phi_i \in \mathbb{R}$. The method gives a unique solution and has two properties: (1) additivity of explanations, which means that the sum of all the feature attributions gives the output of the predictive model, allowing for the aggregation of feature contributions while maintaining consistency with the predictive model, even when dealing with categorical variables, and (2) consistency and local accuracy, which means that the output of the explanation model is consistent with the output of the predictive model, and if a feature has a positive impact on the prediction in the explanation model, it will not have a smaller impact in the predictive model.

There are various methods for enhancing model interpretability, including LIME (Ribeiro et al., 2016) and DeepLIFT (Li et al., 2021). However, the SHAP values have emerged as a popular approach due to their unified framework, which aligns well with human intuition and computational efficiency (Lundberg and Lee, 2017). This approach has been successfully used with tree ensemble methods like XGBoost (Chen and Guestrin, 2016) through `TreeExplainers` (Lundberg et al., 2020), which utilise SHAP values to explain the entire model. Compared to previous techniques, `TreeExplainers` provides faster and more consistent results while capturing variable interactions, making it a valuable tool for interpreting tree-based ensemble methods (Lundberg et al., 2020).

For any non-tree-based method, the SHAP values are calculated using the `KernelExplainers`, which can explain any black-box model, including neural networks, support vector machines, and Gaussian processes (Lundberg and Lee, 2017). This method uses a specially-weighted local linear regression to estimate the SHAP values and is slightly more computationally costly than the `TreeExplainers`.

3. Methodology

In this section, we introduce our explainable active learning metamodels and the ATM simulator under investigation. Then we describe the experimental setup, before presenting the details of the metamodel and the simulator.

3.1. Explainable active learning metamodels

Our strategy to overcome the computational obstacles frequently encountered in simulation-based investigations comprise three main components. Firstly, we mitigate the computational complexity by applying a metamodelling framework that provides a fast approximation of the simulation outcomes and allows for predictions in unlabelled regions of the simulation input space. By predicting the output values for unlabelled input data points, a significant amount of simulation runs and time can be saved. Secondly, we adopt an active learning approach that focuses on selecting the most for posterior fitting in a more efficient way. Thirdly, after fitting the metamodel to a data set composed of pre-computed simulation results, we employ the SHAP method to enhance its interpretability and, hence, the transparency of the underlying simulator. We refer to this metamodel as an *'explainable active learning metamodel'* since it gives an efficient model and offers a better understanding of the relationships between the input variables and KPIs, resulting in an increased level of comprehension regarding the problem under consideration. Our approach is illustrated in Fig. 2, and in the following, we describe the framework in more detail.

Active learning metamodels are fitted using the iterative process of active learning. As such, the methodology of active learning metamodels comprises two main stages, each consisting of the sequential steps illustrated in Fig. 2. The first stage involves training the metamodel using the available set of *simulations* $\mathcal{L}$. Once the *active learning metamodel* $\mathcal{M}$ is trained, it is used to predict outputs over the simulation *feature space* $\mathcal{V}$. In the second stage, the query function $\mathcal{Q}$ uses the predictions for the feature space $\{\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}\}$ to select new unlabelled data points, denoted by $\boldsymbol{x}'$, that need to be evaluated by the *simulator* $\mathcal{O}$. The simulator then provides new outputs, denoted by y', corresponding to the selected data points from the previous step, which are then added to the current training set $\mathcal{L}$. The proposed methodology thus enables the efficient and effective construction of a metamodel using active learning. The aforementioned steps are executed repeatedly until a stopping criterion s is met. This criterion can be defined based on the metamodel's performance, such as the reduction in error or improvement in accuracy, or by a predefined number of iterations based on the available resources, budget, and time. Naturally, a set of criteria encompassing the combination of the latter can also be used.

As discussed earlier, active learning aims to improve model training efficiency and predictive performance by identifying the most informative data points while reducing data redundancy and computational resources. To build a fast and efficient approximation of the simulator, it is crucial that the metamodel is capable of effectively modelling complex functions and performing well with small- and medium-sized data sets. Moreover, incorporating some uncertainty measures in the model can facilitate active learning, thereby improving the accuracy of the approximation. GPs are the preferred underlying model for creating metamodels due to their flexibility, ability to handle small data sets, and provision of uncertainty estimates, as well as their ability to model complex functions (Gramacy, 2020). By utilising the GP, active learning can reduce data redundancy and computational resources while improving model training efficiency and predictive performance. For GPs, the most common query is the uncertainty sampling strategy (MacKay, 1992), which is, in such cases, equivalent to minimising the predictive variance or entropy.

Lastly, the trained active learning metamodel and the acquired set of labelled simulations are combined in SHAP to create the eXplainable Active Learning Metamodel (XALM). XALM is then used for performance assessment, as it is able to predict new

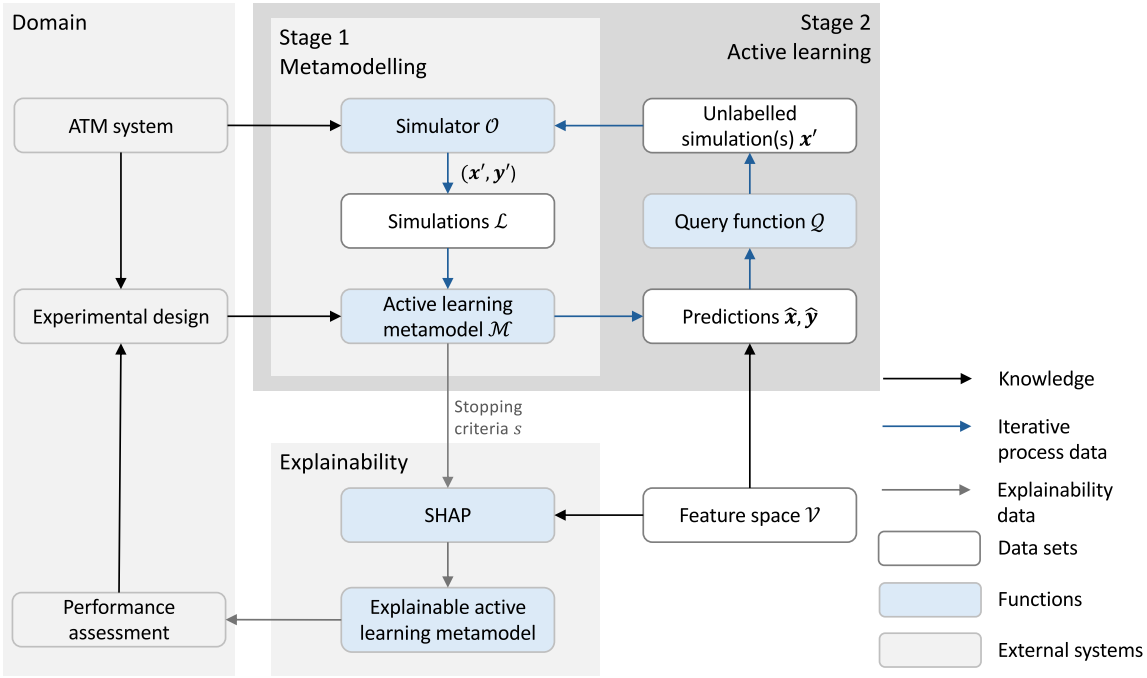

Fig. 2. eXplainable Active Learning Metamodelling (XALM) framework.

scenarios within the unlabelled feature space alongside explanations that enhance the understandability and interpretability of the simulation results. It is worth mentioning that the metamodel might be required to be improved, i.e., retrained. This is a decision to be taken by the ATM researcher/practitioner, allowing for further tuning of XALM with respect to the overall ATM performance assessment goals, expected performance, and domain/expertise knowledge.

3.2. Simulator

The metamodels presented in this article are trained on an existing open-source[4] simulator called 'Mercury' (Delgado et al., 2023; Gurtner and Delgado, 2023; Delgado et al., 2019), a passenger mobility model able to compute various metrics for flights in European airspace. The model, the indicators selected, the input variables and the experimental setup are described hereafter. In this paper, we focus on a simulation case study that has been studied with Mercury in the past: the problem of Extended Arrival MANager (E-AMAN[5]) for European airports (SESAR 3 JU, 2023). In this case study, we extend the range (over the current maximum range) and scope of the standard arrival manager (AMAN) for Paris Charles de Gaulle Airport, a major hub in Western Europe, and study the influence of this change on the flights.

In the following, we first describe the simulator before moving, in more detail, to the E-AMAN module used for the simulation case study. We then describe the indicators computed by the simulator (the y variables of the explainable active learning metamodels) and the input variables (input variable $\boldsymbol{x}$ of the metamodels).

3.2.1. Core mercury

Mercury is a large-scale, agent-driven air mobility simulator developed over many years to simulate the movements of aircraft and passengers around Europe. The full description of the simulator can be found in Gurtner et al. (2021-12), the simulator itself can be found at Gurtner and Delgado (2023), and its main characteristics are:

- The model features several agents, including flights, Airline Operating Centres (AOC), the Network Manager, etc. Each agent has a private memory and private 'functions'.
- The simulation is based on events, e.g. flight departures, triggering responses from various agents (e.g. checking for missing passengers). Events are dynamically created, rescheduled, and destroyed by agents.
- Aircraft and passengers are tracked by the simulation, checking for turnaround processes, missing connections, etc.

[4] Mercury is publicly available at https://github.com/UoW-ATM/Mercury.

[5] E-AMAN allows for the sequencing of arrival traffic much earlier, up to 200 nautical miles from the airport.

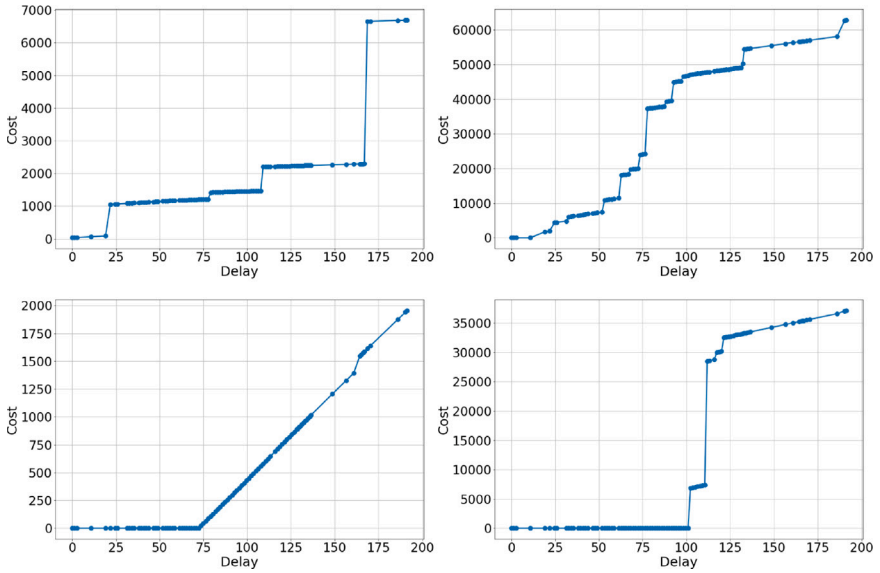

Fig. 3. Examples of cost functions used in Mercury (Cost [€], Delay [min]).

- The simulation includes exogenous delays for flights (related to weather, for instance), and endogenous delays due to missed connections, missed curfews etc.
- The delays, and various other processes, such as aircraft turnaround, are stochastic: values are sampled from distributions (or historical data) chosen by the user.
- The simulator features the entire ECAC[6] space, i.e. it simulates all flights departing or arriving in the ECAC area. Airport processes are simplified for extra-ECAC airports.

Mercury requires several sets of data to run, in particular flight plans, schedules and passenger itineraries. These data sets are described in Gurtner et al. (2021-12). Note that only one day of schedules and passenger data are used.[7] Hence, the different runs of the simulator can be seen as different instances of the same day of operations, randomising delays, turnaround times, regulations, etc.

The agents within the simulator make decisions based on their own information and their inner objectives and logic. In particular, AOCs make various decisions related to flight management, including waiting for passengers or not, reallocating them, cancelling flights in some cases, etc. Their decisions are mostly driven by cost, and hence, an adequate cost model is needed. Mercury includes an implementation of the costs model developed by Cook and Tanner (2011), updated in Cook and Tanner (2015) and Cook et al. (2021-11-15), which allows airlines to derive a cost function for each of their flights, i.e. a relationship between the delay incurred by a flight and the corresponding cost for the airline. Thanks to this model, the cost function takes into account maintenance, crew (in an aggregated way), missed passenger connections (explicitly), turnaround delay (explicitly), and curfew costs (explicitly). As a result, very detailed delay cost functions for flights are obtained, such as those shown in Fig. 3.

For example, the bottom right graph shows two significant and two smaller jumps. The first significant jump is at 100 min of delay (at about 7000 €) and another one at about 110 min (at about 27,000 €). Such significant jumps in costs usually indicate the presence of a group of passengers that would lose a connection in an onward flight and trigger compensation according to Regulation 261 (European Commission, 2004). Another reason for a jump might be due to the need to change the crew when they exceed their permissible hours, and another crew needs to be transferred to operate the flight. Conversely, the bottom left graph shows a flight where even a significant delay (3 h) would cause contained costs (up to 2000 €), which likely does not have connecting passengers.

3.2.2. E-AMAN module

In this article, we selected a particular problem that has been studied in the past with Mercury: extending the range and scope of AMAN.

The AMAN in Mercury is modelled as a stand-alone agent, distinct from the corresponding airport. Its default behaviour is fairly simple, ensuring that the runway capacity is not infringed, otherwise delaying flights en-route or putting them in holding. The detection of the capacity infringement is done tactically every time a flight reaches a 100NM radius circle around the airport (the *'tactical horizon'*, hereafter).

[6] ECAC - European Civil Aviation Conference, encompasses 44 Member States.

[7] The data used are from 2014, as this is the latest, complete data set, containing flights, schedules and passenger data, that is available to us.

Table 1
Input features used in the case studies.

Feature	Short description	Theoretical range	Practical range	Default	Unit
Fuel price	Price of one kg of fuel	[0, ∞)	[0, 5]	1	2014 euros per kg
Planning horizon	Distance horizon where the E-AMAN tries to optimise the arrival.	(100, ∞)	[100, 1000]	300	Nautical miles
Cruise uncertainty scale	Deviation in the aircraft speed during cruise	[0, ∞)	[0, 10]	1	–
Turnaround time scale	Scaler of mean of the distribution of turnaround times	[0, ∞)	[0, 10]	1	–
Minimum connecting time scale	Scaler of mean of the distribution of passenger minimum connecting times	[0, ∞)	[0, 10]	1	–
Claim rate	Proportion of passengers claiming compensation	[0, 1]	[0, 1]	0.14	–

However, this process tends to cause unnecessary fuel burn since flights could, in principle, absorb delay by slowing down much before the 100NM mark. Hence, the module implemented in Mercury features another horizon, called the *planning horizon*, typically much larger than the tactical one, changing the modelled agent from AMAN to Extended-AMAN (E-AMAN). At the planning horizon, flights communicate their intent to the E-AMAN, which builds a 'planning queue' of flights arriving at the airport. E-AMAN then performs optimisation of the queue, minimising total delay, after which it gives a specific command to the flight (either slow down, maintain speed or speed up). We assume in the model that flights always follow this command, to the best of their abilities. We also assume that the flights are cleared regarding potential conflicts in the airspace that they will cross between their current position and arrival. We assume that flights departing within the planned horizon do not get any command, and just fly according to their flight plan, which is not known to the E-AMAN before they depart.

3.2.3. Selection of input variables

Mercury is highly parameterisable, from the exact forms of the probability distribution of delays to the price of fuel. In this case study, we are interested in testing the parameters linked to the E-AMAN itself, as well as macro-parameters that may show that the E-AMAN has different impacts when the system is in one state or another. For instance, we select the typical turnaround time as a parameter. Indeed, if the E-AMAN has a substantial effect, it should be even more substantial when the system is 'tight', i.e. when the aircraft has a small amount of time between two flights.

The final parameters are presented in Table 1 and described below. Note that the parameters serve as individual input variables $\boldsymbol{x}$ of the metamodels. In the table, we included the theoretical range of each parameter, the practical range, pushing some of them to the border of the realistic envelope, and the default value in the simulations when not specified otherwise. Also, we choose very high values for the maximum values of these parameters in order to test the system under massive stress. A factor of 10 on turnaround times, for instance, could happen during heavy disruptions at the airport. The parameters are the following:

- **Fuel price:** the approximated price of one kg of fuel. This has a direct impact on the airlines, since fuel represents a major share of their total costs. The fuel consumption itself varies in the simulations based on the holding and speed performed by the aircraft.
- **Planning horizon:** the radius of the circle where flights enter the planned queue and the subsequent optimisation, as described above.
- **Cruise uncertainty scale:** in the model, wind is modelled as a stochastic process modifying the ground speed of the aircraft (for a constant air speed). The wind has a central component (the mean) and associated variance. This parameter measures the scale of the variance compared to the baseline calibration. For instance, if this parameter is equal to 1.2, the standard deviation of speed is 20% above its default value.
- **Turnaround time scale:** in the model, an aircraft needs a certain time to be ready before its next flight. This time is a random number, drawn from a log-normal distribution. This parameter is the scale of the standard deviation of this distribution with respect to the baseline.
- **Minimum connecting time scale:** passengers need a certain time to transfer from one aircraft to another if they are connecting. This minimum connecting time is a fixed value for each airport, depending only on the type of connection made by the passenger (domestic, international, etc.). This parameter represents the scale of this value with respect to the baseline.
- **Claim rate:** this is the proportion of passengers that claim compensation according to Regulation 261. While this rate is currently quite high, due to increased public awareness and the growth of agencies supporting passenger claims, it was only around 14% in 2014, the date of the dataset for which the model is running for this study.

3.2.4. Selection of key performance indicators (KPIs)

A characteristic of Mercury (and of any complex simulator) is its ability to compute very low-level, disaggregated metrics. This profusion of numbers can be consolidated into a few indicators that are meant to estimate the efficiency of the system from different points of view. As a test case, we selected the following KPIs (they represent the y variables of the explainable active learning metamodels):

- **Flight arrival delay** (in minutes): computed by comparing the actual arrival time with the scheduled arrival time for each flight. The indicator is the average of this difference over all flights in the simulation.
- **Flight departure delay** (in minutes): computed similarly to arrival delay, with departure times.
- **Passenger arrival delay** (in minutes): computed by comparing the scheduled arrival of the last flight of each passenger, compared to the actual arrival. If passengers cannot reach their final destination, a delay of 12 h is considered.
- **Planned absorbed delay** (in minutes): when the E-AMAN gives a 'slow-down' or 'speed-up' command to a flight, the amount of time the flight would absorb en-route with the change of speed is computed. This is the time delay that aircraft could absorb while en-route, instead of being sent to the holding pattern before landing (holding consumes more fuel than cruise). This indicator averages this time overall issued commands.
- **Holding time** (in minutes): the average of the time that the flights spend in holding.
- **Fuel cost** (in 2014 euros): the average cost of the total fuel consumed by flights. Mercury computes this using the BADA3 (Nuic et al., 2010) model for fuel consumption and multiplying by the cost of the fuel.

3.3. Experimental setup

3.3.1. Case-study simulator

Mercury is run on 2014 data from Paris Charles de Gaulle airport, a major hub airport in Europe. All historical flights arriving at and departing from this airport are considered. Passengers on these flights may or may not connect at this airport, depending on the historical passenger itineraries. Exogenous delays are set to a high level in order to stress the system and better study the impact of the E-AMAN mechanism.

This model is then considered as a black box for the metamodel, which can interact with it only by asking for a new simulation on a new set of inputs (only those described in 3.2.3) and getting in return the selected KPIs (only those described in 3.2.4).

3.3.2. Explainable active learning metamodels

For our backbone model in the explainable active learning metamodel, we use a zero-mean GP with the widely known RBF-ARD kernel. In each iteration of the active learning loop, the inputs are rescaled to the unit cube $[0, 1]^6$, and the outputs are standardised to have zero mean and unit variance. The initial data sets consist of ten data points chosen randomly, and in each iteration, one data point is queried. The unlabelled pool $\mathcal{V}$ consists of 50,000 data points from the input space given by practical ranges in Table 1, and the test set consists of another 10,000 simulations randomly sampled from the same space. The query strategy is the uncertainty sampling, which can be computed as $Q(\boldsymbol{x}) = \arg\max_{\boldsymbol{x}} \sigma^2(\boldsymbol{x})$, where $\sigma^2(\boldsymbol{x})$ comes from the diagonal of the covariance matrix of the predictive posterior $\Sigma^\star$. In each active learning iteration, the GP is optimised for 300 gradient steps with Adam (Kingma and Ba, 2015) with a learning rate of 0.1 and early stopping if the performance does not improve for 15 iterations. The GPs are implemented in GPyTorch (Gardner et al., 2018). As there exists no 'ground' truth for the explainability of a model, we instead compute our own set of 'ground' true values using the SHAP values computed with XGBoost trained on 50,000 data points. The data set of the SHAP values consists of 1,000 data points, and in each active learning iteration, we benchmark the explainability by comparing the SHAP values of the current metamodel with these values. For details on XGBoost and the optimisation thereof, see Appendix A.

3.4. Experiments on model performance

In this section, we describe the three experiments in which we evaluate the performance of the explainable active learning metamodels. We make one metamodel per KPI, and all experiments are repeated 30 times with different initial data sets.

Experiment 1: The Gaussian process as a metamodel. We test if the performance of the GP is on par with the XGBoost trained on the extensive simulation study using 50,000 simulations (Riis et al., 2022). We evaluate the performance of a Gaussian process trained with 30, 100, and 1,000 simulations against two baselines: the lower baseline *mean predictor*, which on new data points predicts the mean of the observed data points, and the upper baseline XGBoost trained on 50,000 data points, which represent an estimate of the best possible performance. Since the Gaussian process is computationally expensive to train due to the matrix inversion, we limit the largest number of data points to 1,000. Therefore, we compare its performance against XGBoost, fitted on 30, 100, and 1000 data points.

Experiment 2: Active learning vs. passive learning. We test if the active learning approach outperforms the random sampling, also known as passive learning. We report the performance of a Gaussian process trained with active learning after querying a total of 30 and 100 simulations against the Gaussian process trained on a random sample of 30 and 100 simulations, respectively. Furthermore, to not only be dependent on those two specific values, we also visually compare the performance of active and passive learning by investigating the loss curves to show the effect of active learning for all active learning iterations up to a total dataset of 100 simulations.

Experiment 3: Active learning for explainable metamodels. We test if the active learning approach is suitable when the objective is not to increase the performance of the model directly but to obtain SHAP values that correctly explain the predictions of the models. Similarly to experiment 2, we evaluate the performance, in terms of predictive accuracy for the SHAP values, after 30 and 100 simulations, comparing active and passive learning, as well as visual inspection of the loss curves.

Overall, we consider four different metrics to evaluate the performance of the models. We use the three metrics, the root mean square error (RMSE), relative root square error (RRSE), and the Pearson correlation, to evaluate the predictive performance. If we have the data set $\mathcal{L} = \{\boldsymbol{x}_i, y_i\}_{i=1}^N$ and the model f, such that $f(\boldsymbol{x}_i) = \hat{y}_i$ is the prediction of the model, then RMSE and RRSE is

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_i^N (\hat{y}_i - y_i)^2} \quad \text{and} \quad \text{RRSE} = \sqrt{\frac{\sum_i^N (\hat{y}_i - y_i)^2}{\sum_i^N (y_i - \bar{y})^2}} \quad \text{with } \bar{y} = \frac{1}{N}\sum_i^N y_i. \tag{3}$$

The last metric is the *computational time* of the full pipeline, including the simulation and the training time of the models as well as the active learning.

3.5. Analysis of the explainable active learning metamodels

Our explainable active learning metamodels can provide multiple insights into what the model is predicting and, thus, how the outputs of the simulator behave: firstly, due to the explainable component of the SHAP values; and secondly, due to the fast inference of the metamodel, allowing for predicting the output of unseen simulation scenarios, accompanied by the inherent epistemic and aleatoric uncertainty estimates in the Gaussian process. We did an extensive pre-analysis (not included in the paper) to find any particular patterns and trends of interest using the two aforementioned methods. However, in this paper, we only include a condensed analysis, which highlights the main findings. Nonetheless, we recommend the reader to have a look at the documentation of the SHAP values.[8]

In our analysis, we will use the SHAP values to summarise the feature contributions and highlight any particular impactful features, followed by individual plots showing the trends of such features. These plots will be supported by plots showing the predictions of the metamodel with confidence and predictions interval (obtained through the epistemic and aleatoric uncertainty estimates from the GP), computed by fixing all features to their default value (cf. Table 1), and only changing the variable under investigation. Depending on the presence of any interactions between the features, we will include such by using different colours in the individual plots of the features as well as in the predictions of the metamodels. With the analysis, we not only aim to discover the underlying behaviour of the simulator Mercury, but also show how the explainable active learning metamodels can be used in a broader scope. Thus, we will also go into detail about how to interpret the explainable components of the metamodels.

4. Results

We first benchmark the active learning with Gaussian processes by comparing the performance across a different number of simulations and against passive learning.

Experiment 1. In Table 2, the predictive performance is quantified with the RMSE, RRSE, and Pearson correlation for the GPs and the baselines. As expected, all the XGBoost and GP models outperform the mean predictor, and XGBoost trained on 50,000 simulations is, with regards to the predictive performance, the best-performing metamodel across the six KPIs and the three metrics.

If we look at the performance of the three GP models, we see that the more simulations the better across all KPIs, and furthermore, if we compare the performance of the GP trained on 1000 simulations to XGBoost trained on the same number of simulations, we see that the GP surpasses the XGBoost for five out of the six KPIs (XGBoost is slightly better for the planned absorbed delay). Overall, we see that the performance of the GP is on par with XGBoost, and thus, in terms of predictive performance, it is reasonable to use the GP.

The computational time for the different models arises mainly from the simulation time (2.5 min per simulation) and is, thus, approximately constant across the different KPIs. Table 3 shows that for the model with the 50,000 simulations, it takes 2083 h, or 87 days, in CPU time, whereas the model with 30 simulations only takes 1.4 h. As a consequence of the simulation time being the bottleneck, it is definitely beneficial to consider how many simulations are needed to achieve the desired performance. In the next experiment, we benchmark the active learning approach, which exactly deals with this trade-off in performance and computational time. In conclusion, it is possible to get reasonable predictive performance with a much smaller set of simulations, significantly reducing the computational burden of performance assessment using simulators.

[8] Various visualisations of SHAP values: https://shap-lrjball.readthedocs.io/en/latest/examples.html#plots-examples.

Table 2
Performance of the metamodels. The standard deviation is given in parentheses, computed across 30 repetitions.

KPI	Model	#sim	RMSE	RRSE	Correlation
Flight arrival delay	Mean predictor	50,000	106.0	1.00	–
	XGBoost	50,000	3.91 (0.00)	0.04 (0.00)	1.0 (0.00)
	XGBoost	1000	4.27 (0.04)	0.04 (0.00)	1.0 (0.00)
	XGBoost	100	6.41 (0.34)	0.06 (0.00)	1.0 (0.00)
	XGBoost	30	14.67 (5.02)	0.14 (0.05)	.99 (0.01)
	GP	1000	3.97 (0.01)	0.04 (0.00)	1.0 (0.00)
	GP	100	4.41 (0.32)	0.04 (0.00)	1.0 (0.00)
	GP	30	5.77 (0.89)	0.05 (0.01)	1.0 (0.00)
Flight departure delay	Mean predictor	50,000	106.4	1.00	–
	XGBoost	50,000	3.86 (0.00)	0.04 (0.00)	1.0 (0.00)
	XGBoost	1000	4.23 (0.04)	0.04 (0.00)	1.0 (0.00)
	XGBoost	100	6.19 (0.36)	0.06 (0.00)	1.0 (0.00)
	XGBoost	30	14.68 (4.84)	0.14 (0.05)	.99 (0.01)
	GP	1000	3.90 (0.01)	0.04 (0.00)	1.0 (0.00)
	GP	100	4.31 (0.23)	0.04 (0.00)	1.0 (0.00)
	GP	30	5.58 (0.93)	0.05 (0.01)	1.0 (0.00)
Passenger arrival delay	Mean predictor	50,000	202.3	1.00	–
	XGBoost	50,000	43.0 (0.03)	0.21 (0.00)	.98 (0.00)
	XGBoost	1000	46.2 (0.46)	0.23 (0.00)	.97 (0.00)
	XGBoost	100	59.9 (4.10)	0.30 (0.02)	.96 (0.00)
	XGBoost	30	108.0 (23.84)	0.53 (0.12)	.90 (0.04)
	GP	1000	43.7 (0.17)	0.22 (0.00)	.98 (0.00)
	GP	100	46.9 (1.14)	0.23 (0.01)	.97 (0.00)
	GP	30	61.8 (15.72)	0.31 (0.08)	.95 (0.03)
Planned absorbed delay	Mean predictor	50,000	0.242	1.00	–
	XGBoost	50,000	0.093 (0.000)	0.39 (0.00)	.92 (0.00)
	XGBoost	1000	0.098 (0.001)	0.41 (0.00)	.92 (0.00)
	XGBoost	100	0.116 (0.006)	0.48 (0.03)	.57 (0.02)
	XGBoost	30	0.152 (0.025)	0.63 (0.10)	.53 (0.06)
	GP	1000	0.100 (0.001)	0.41 (0.00)	.92 (0.00)
	GP	100	0.105 (0.005)	0.43 (0.02)	.90 (0.01)
	GP	30	0.150 (0.032)	0.62 (0.13)	.80 (0.09)
Holding time	Mean predictor	50,000	0.204	1.00	–
	XGBoost	50,000	0.160 (0.000)	0.78 (0.00)	.62 (0.00)
	XGBoost	1000	0.166 (0.002)	0.81 (0.01)	.59 (0.01)
	XGBoost	100	0.197 (0.010)	0.97 (0.05)	.36 (0.04)
	XGBoost	30	0.212 (0.017)	1.04 (0.08)	.24 (0.12)
	GP	1000	0.163 (0.001)	0.80 (0.00)	.60 (0.00)
	GP	100	0.177 (0.008)	0.87 (0.04)	.52 (0.06)
	GP	30	0.213 (0.023)	1.04 (0.11)	.30 (0.18)
Fuel cost	Mean predictor	50,000	26,401	1.00	–
	XGBoost	50,000	354 (0.4)	0.01 (0.00)	1.0 (0.00)
	XGBoost	1000	474 (10.6)	0.02 (0.00)	1.0 (0.00)
	XGBoost	100	1105 (74.8)	0.04 (0.00)	1.0 (0.00)
	XGBoost	30	3209 (1034)	0.12 (0.04)	.99 (0.01)
	GP	1000	362 (2.6)	0.01 (0.00)	1.0 (0.00)
	GP	100	398 (19.0)	0.02 (0.00)	1.0 (0.00)
	GP	30	531 (97.9)	0.02 (0.00)	1.0 (0.00)

Experiment 2. In Table 4, the predictive performance is evaluated in terms of the RMSE for the GP trained with active learning and compared to the GP trained with passive learning (the results for passive learning are the same as in Table 2). For the GPs trained on 30 simulations, the active learning approach outperforms passive learning across the KPIs (significantly lower RMSE for five of the KPIs and on par for the last KPI). With 100 simulations, the difference in performance between the two approaches is smaller, and passive learning achieves the best performance for the two KPIs, passenger arrival delay and planned absorbed delay. Active learning achieves a significantly lower RMSE for three of the KPIs and is on par for the remaining ones

In Fig. 4, we have two plots for each KPI: one showing the active learning curves based on the RMSE for the predictions of the GPs, and one with the curves based on RMSE for SHAP values (the latter is discussed in experiment 3). The general trend across the KPIs is that the active learning curves are below the passive learning curves, showing how the former is more efficient than the latter. For all the KPIs, except holding time, we see that active learning is more efficient in the first iterations, where only a few simulations are used, compared to the late iterations, where the performance of active learning is only slightly better than passive learning, which is a natural pattern of active learning, well-known in the literature (Riis et al., 2021; Sánchez-Cauce et al., 2022). In summary, the active learning approach outperforms the passive learning approach.

Table 3
Computational time (in hours) for the different models. The time includes the simulation time (2.5 min per simulation) and training time.

Model	#sim	Time [hr]
Mean predictor	50,000	2083.3
XGBoost	50,000	2083.3
XGBoost	1000	41.7
XGBoost	100	4.2
XGBoost	30	1.3
GP	1000	41.8
GP	100	4.9[a]
GP	30	1.4[a]

[a] For fuel cost, the time was 8.51 and 3.05 hr for the GP with 100 and 30 simulations, resp.

Table 4
Performance of the Gaussian processes trained with active and passive learning. The best performance is in bold. Significance is indicated by asterisks, where the p-values are computed using a two-tailed Welch's t-test.

KPI	#sim	RMSE (predictions)		RMSE (SHAP)	
		Active learning	Passive learning	Active learning	Passive learning
Flight arrival delay	100	**4.22 (0.08)****	4.41 (0.32)	**1.43 (0.40)****	2.45 (1.55)
	30	**5.24 (0.29)****	5.77 (0.89)	**1.87 (0.61)*****	3.55 (1.60)
Flight departure delay	100	**4.13 (0.09)*****	4.31 (0.23)	**1.41 (0.37)*****	2.98 (1.63)
	30	**5.17 (0.41)***	5.58 (0.93)	**1.79 (0.86)*****	4.26 (2.82)
Passenger arrival delay	100	47.2 (1.37)	**46.9 (1.14)**	**6.82 (1.18)***	8.03 (2.58)
	30	**54.1 (6.01)***	61.8 (15.72)	**9.64 (2.65)*****	14.50 (5.50)
Planned absorbed delay	100	**0.11 (0.00)**	**0.11 (0.01)**	0.02 (0.00)	**0.01 (0.00)*****
	30	**0.12 (0.01)*****	0.15 (0.03)	**0.03 (0.01)**	**0.03 (0.01)**
Holding time	100	**0.17 (0.00)*****	0.18 (0.01)	**0.02 (0.00)**	**0.02 (0.00)**
	30	**0.20 (0.02)**	0.21 (0.02)	**0.03 (0.01)*****	0.04 (0.01)
Fuel cost	100	**391 (14.3)**	398 (19.4)	678 (166)	**640 (352)**
	30	**468 (48.9)****	531 (97.9)	**748 (379)**	857 (473)

* p-values: < 0.05.
** p-values: < 0.01.
*** p-values: < 0.001 .

Experiment 3. In Table 4, we see that the RMSE for the SHAP values obtained with GP trained on 30 simulations is best for active learning (significantly best for four out of six KPIs). When we use 100 simulations, active learning is better four out of six times (significantly better for three out of six and only significantly worse for one KPI). The active learning curves based on the RMSE for the SHAP values are shown in Fig. 4 and exhibit similar patterns as for the active learning curves based on the RMSE for the predictions. In conclusion, active learning is indeed more efficient in sampling data points that give more precise SHAP values.

4.1. Analysis

In this section, we use the explainable active learning metamodels to describe the behaviour observed while using our case-study simulator (Mercury). First, we look at the overall feature contributions on the different KPIs, and then we investigate selected features' effects on single KPIs.

Fig. 5 shows the SHAP summary plots for the six KPIs. The flight arrival and departure delay are mainly influenced by the turnaround time, where a lower turnaround time gives a lower delay, and vice versa. The two KPIs, planned absorbed delay and fuel cost, are also mainly affected by single variables, namely, planning horizon and fuel price, respectively. The connection between the fuel price and fuel cost (see below) seems to be similar to that of the flight delays, whereas the planning horizon seems to have a particular effect on the planned absorbed delay when the planning horizon is low. Lastly, the passenger arrival delay and the holding time are affected by multiple features — patterns which we will discuss later. Overall, it should be noted that in this specific case study, the metamodel does not capture any effect of the cruise uncertainty or claim rate on any of the six KPIs under study.

In the following, we go through the contribution of specific features on the KPIs, starting with the simpler patterns arising from the fuel price's effect on the fuel cost and the planning horizon's effect on the planned absorbed delay. Afterwards, we investigate the more complicated cases with passenger arrival delay, and lastly, holding time. The plots for the effect of turnaround time on the flight arrival and departure delay are in Appendix B.

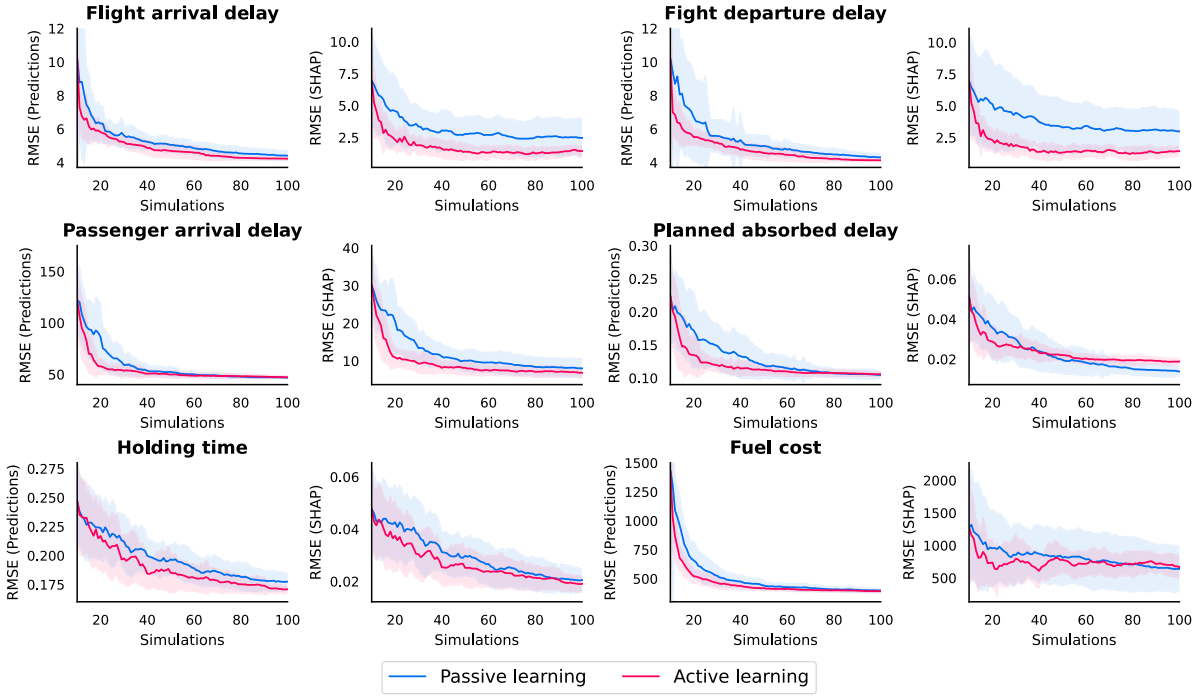

Fig. 4. Active learning curves for the six KPIs. The figure shows the mean results of 30 repetitions, represented by lines, with the shaded area indicating the range of $\mu \pm \sigma$. For each KPI, the left panel shows the root mean square error (RMSE) of the predictions for the two acquisition functions over 90 acquisitions. The right panel displays the RMSE of the SHAP values compared to the SHAP values of the XGBoost using 50,000 simulations.

Understanding the SHAP summary plots

The SHAP summary plots show the contribution of each feature on the corresponding output, i.e. KPI, based on the SHAP values. On the x-axis, we have the SHAP value, which is the impact on the model output, and the colours denote the value of each feature such that a blue and red colour corresponds to a low and high feature value, respectively. Note that the zero on the x-axis corresponds to the mean of the KPIs, and thus the SHAP values are computed with respect to this base value.

Fuel cost. The left plot in Fig. 6 shows the SHAP values fuel price and fuel cost. We observe a linear relationship between the input and the output, and for example that if the fuel price is near 0 €/kg, the fuel cost is the base value (the mean of the fuel cost) subtracted by 40,000 €. On the other hand, a fuel price close to 5 €/kg gives a fuel cost 40,000 € above the base value. The SHAP values give a clear overview of the marginal effect of a feature on the output, but often it is also beneficial to look at the absolute effects for a specific set of features. The right plot in Fig. 6 shows the prediction of the fuel cost when the features are set to the default values, and only the fuel price is changed. This shows clearly the overwhelming effect of the cost of fuel on the costs to the airlines, at least for the tactical costs incurred for these relatively low levels of delay (as refelected in Cook and Tanner (2015), but also mindful that *passenger* costs to the airline dominate at higher levels of delay). Note also that the upper bound of the fuel price modelled here is very high compared to today's.

Both the confidence and prediction interval coincide with the mean prediction, meaning that the metamodel is very certain about its mean predictions and that there is practically no stochasticity in the output.

Planned absorbed delay. Fig. 7 shows how the marginal effect of the planning horizon (PH) has a linear relationship with the planned absorbed delay (PAD), when the PH goes from 100 to 300. For PH values higher than 300, the PAD seems to change periodically with a minor upward trend as the PH increases. However, before suggesting various hypotheses to explain this pattern, it is beneficial to examine the credibility of the metamodels through their predictions. In the right plot in Fig. 7, we see the estimates of the mean, 95% confidence interval, and 95% prediction interval for the simulator's output with default features, only changing the planning horizon. The mean predictions have a similar pattern to that of the SHAP values, however, the 95% confidence interval is rather wide and is, in fact, not excluding that the periodic pattern could be explained by a linear line. We also see that the prediction interval is even wider, meaning that the simulator has high stochasticity for the PAD.

Hence, PAD seems to follow a linear increase with respect to the PH, followed by a saturation (possibly slightly increasing up to 1000NM). This is an interesting pattern, since it shows that the theoretical efficiency of the E-AMAN increases at first (as more

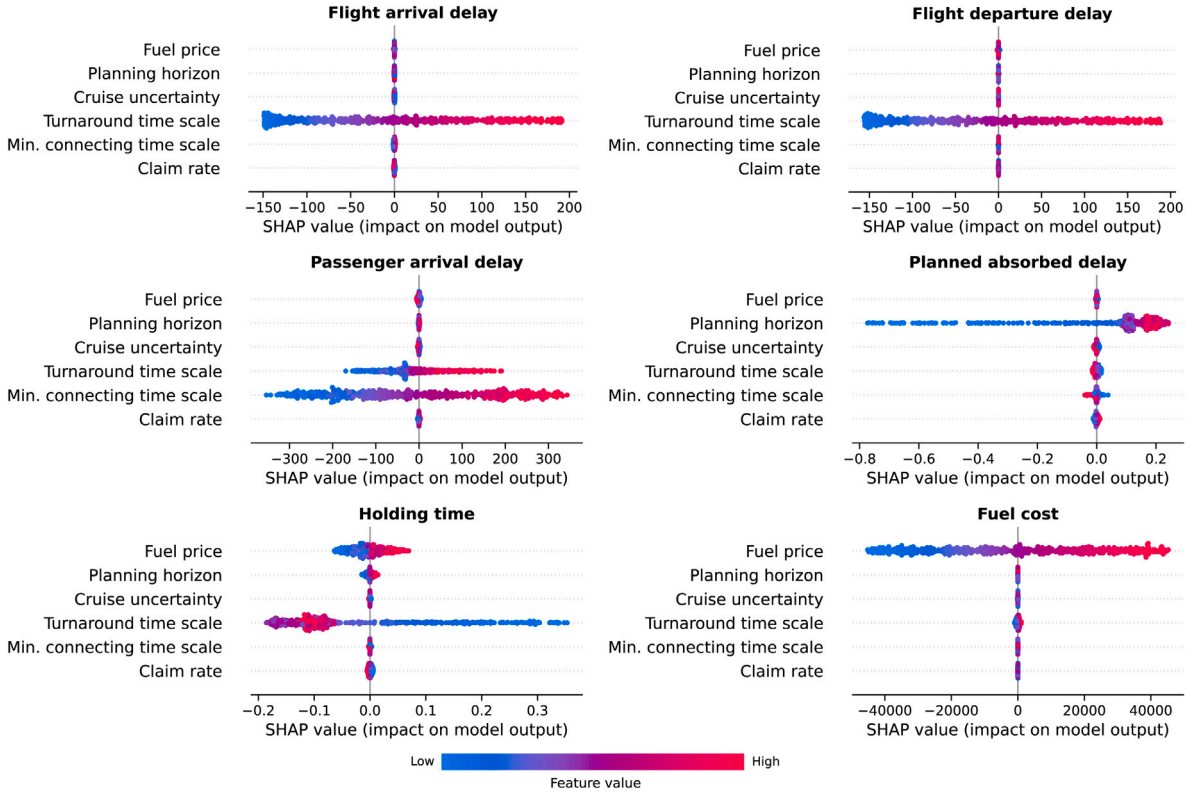

Fig. 5. Feature contribution based on the SHAP values. The x-axis represents the impact on the model outputs, i.e. the KPIs. The colour represents the feature value of the features on the y-axis. All the plots are based on a GP fitted to 100 simulations.

Understanding the predictions of the explainable active learning metamodel

With the GP metamodel, we can estimate the mean, the 95% confidence interval, and the 95% prediction interval. The mean prediction is the metamodel's estimate of the mean of the simulator's output if the simulator is run multiple times with the same features. The confidence interval shows how certain the model is about its mean prediction, and the prediction interval captures the stochasticity in the simulation output, such that if you run the simulator multiple times with the same features, 95% of the simulations will be within the 95% prediction interval.

delay is planned to be absorbed) but reaches a maximum value after around 350NM. In other words, extending the E-AMAN horizon beyond this limit yields very little benefit.

Passenger arrival delay. In Fig. 8, we see the two most important features for predicting the passenger arrival delay, namely, the minimum connecting time scale (MCT) and the turnaround time scale (TT). In the left plot, we have the MCT on the x-axis and the corresponding SHAP values on the y-axis. The colours denote the value of the feature TT in the same way as the colours in the SHAP summary plots, such that a blue point and red point correspond to a low and high TT, respectively. Irrespective of the colours, we see that the higher MCT, the higher the SHAP value, meaning that the higher the MCT, the more it contributes to a higher passenger arrival delay. The colours indicate the value of TT, showing that if TT is low, there is a more considerable change in the passenger arrival delay when MCT is increased compared to when TT is high. In other words, if TT is high, the value of MCT has a smaller impact on the passenger arrival delay. In the right plot, we investigate the same interaction, now having interchanged the MCT and TT. In general, the passenger arrival delay is linearly dependent on the TT. When MCT is low, the linear increase in the passenger arrival delay is higher with respect to TT, whereas, when MCT is high, the value of TT affects the passenger arrival delay less.

The monotonic nature of the passenger delay against both MCT and TT is natural. Since the schedules are fixed in our experiments (and in general, at least during a season), increasing either the time the aircraft takes to be prepared for the next rotation, or the time needed for passengers to connect to the subsequent flight, is going to result in more delays. In the first case, the next rotation is delayed, which may delay next rotations, and/or imply that passengers are going to miss their next connection. Conversely,

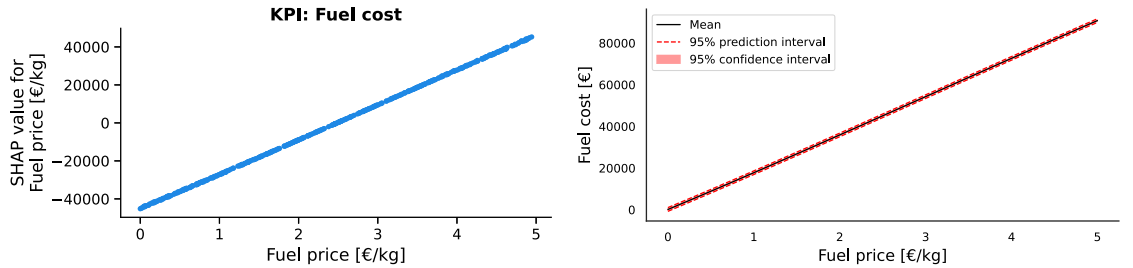

Fig. 6. The impact of the fuel price on the fuel cost. **Left**: SHAP values. **Right**: Model predictions.

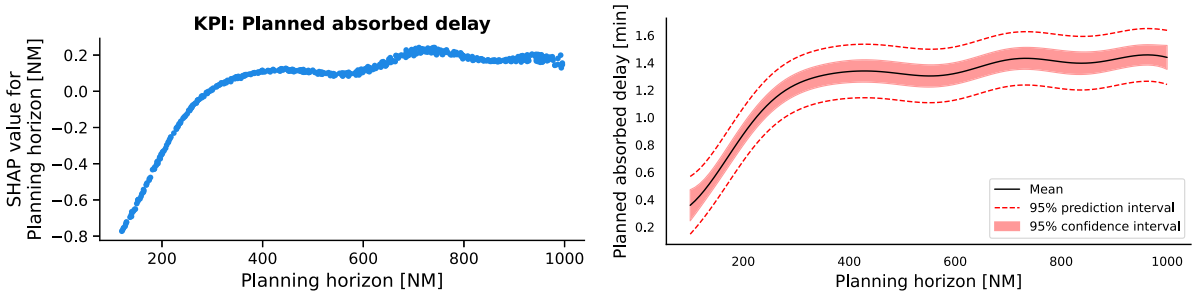

Fig. 7. The impact of the planning horizon on the planned absorbed delay. **Left**: SHAP values. **Right**: Model predictions.

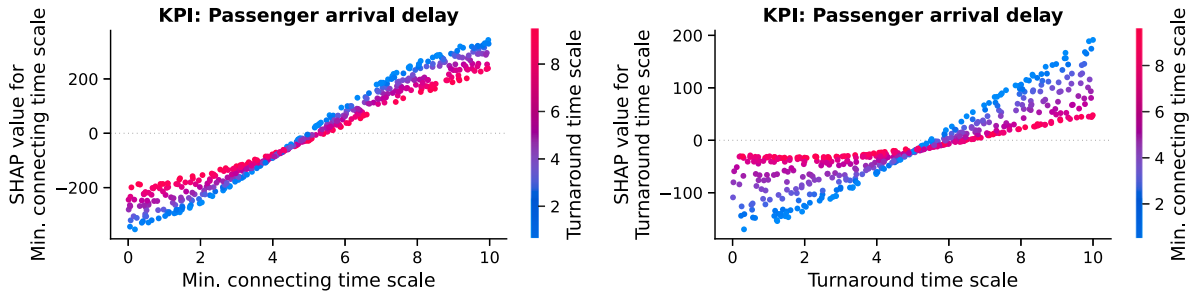

Fig. 8. The two features with the highest impact on the passenger arrival delay.

increasing the TT will imply that more passengers are going to miss their flights when connecting. Both of these will increase the delay at passengers' final destination. It was also expected that higher TT values decrease the impact of MCT on the delay, and vice versa. If, for example, aircraft take more time to be prepared for their next rotation, then passengers will have more time to reach them, *ceteris paribus*.

In the following, we examine the two features' role in predicting the passenger arrival delay as well as their interactions by looking at the predictions from the metamodel. We incorporate the interaction into the plots by choosing three different values for the secondary feature such that we see, for example, how MCT affects the passenger arrival delay for three different values of the TT.

The left plot in Fig. 9 shows that the passenger arrival delay increases when MCT increases. When TT is low (blue line), the passenger arrival delay increases from 200 to 1000 as MCT is increased from 0 to 10. If TT is high (red line), the passenger arrival delay is already high for low values of MCT, and for MCT higher than six, the passenger arrival delay is only slightly affected by TT. The right plot in Fig. 9 shows that for a high MCT, the passenger arrival delay is around 1000 and is slightly decreasing as TT increases. Conversely, when MCT is low, TT has a high effect on the passenger arrival delay. In both plots, the confidence and prediction intervals are stable across the different values of MCT and TT, and we see that there is some stochasticity in the output of the passenger arrival delay. Note how the decrease in passenger arrival delay might seem contrary to the fact that the SHAP value's impact on the passenger arrival delay for TT is increasing as TT increases in Fig. 8. However, the previously mentioned interaction with the minimum connecting time scale is causing this decrease. Consider the SHAP values in Fig. 8: in the right plot, the red points represent the SHAP values for TT, when MCT is high, and ranges from −20 to 40, approximately linearly increasing with TT. Now, if we consider the left plot and examine the SHAP values for high values of MCT, e.g. MCT equal to 8, the SHAP values are ranging from approximately 180 to 350, as TT decreases. Thus, combining the SHAP values from TT and MCT yields a decrease in the passenger arrival delay as TT increases, as seen in the predictions in Fig. 9

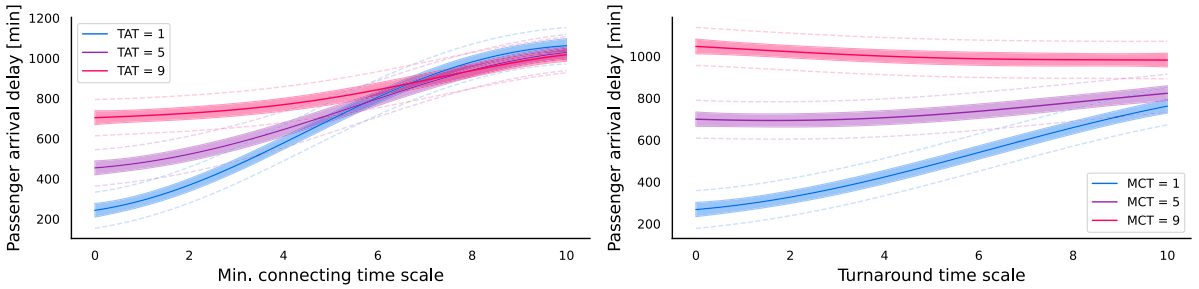

Fig. 9. The two features with the highest impact on the passenger arrival delay.

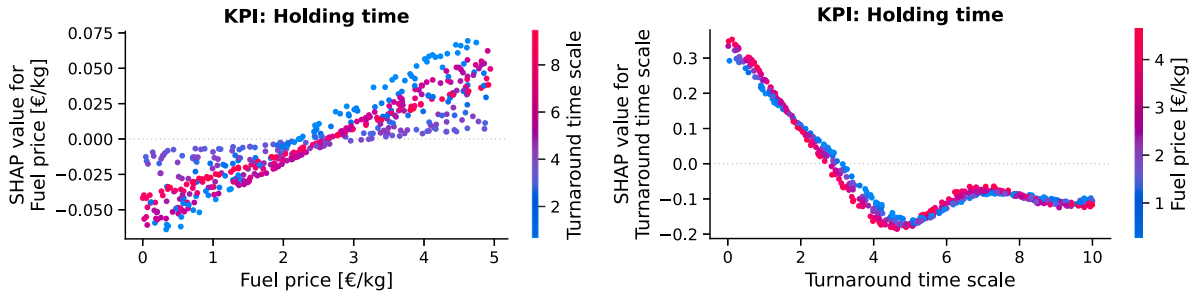

Fig. 10. The two features with the highest impact on the holding time.

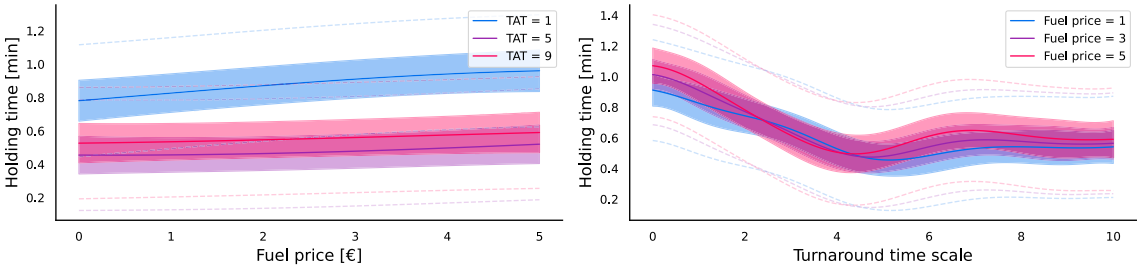

Fig. 11. The two features with the highest impact on the passenger arrival delay.

Holding time. In Fig. 10, we see the two most important features for predicting the holding time, namely, the fuel price (FP) and the turnaround time scale (TT). For the SHAP values for the FP, we see that the higher the FP, the higher the holding time, although when the TT is around 5, we see only a small increase in the holding time. The fuel price has the highest impact on the holding time when TT is low. In the right plot, we see that the marginal effect of TT on the holding time has a negative linear relationship when TT is below 4. For TT higher than 4, we note an increase in the holding time before a minor decrease. Again, it is helpful to consider the validity of the SHAP values by looking at the credibility of the metamodel through its predictions of confidence intervals.

In the left plot in Fig. 11, we see that the predictions also reflect the linear relationship between the fuel price and the holding time. However, we also note that both the confidence and prediction intervals are rather wide, where the former means that the metamodel is uncertain regarding the intercept and coefficient of the linear relationships, and the latter that there is high stochasticity in the holding time. In the right plot, we see a similar pattern in the confidence and prediction intervals, showing high uncertainty about the mean predictions and in particular the pattern in holding time for TT higher than 4.

Hence, the holding time is an increasing function of the fuel price, although less strongly related when the TT is higher. It is somewhat of a puzzling result since it should be expected to have less holding time when the fuel is more expensive. This is probably because the E-AMAN does not work as intended. Indeed, when the price of fuel is higher, the E-AMAN will try to give 'slowing down' commands more often, to save fuel during cruise. However, it may be that the randomness of the trajectory means that the flight, when arriving at the tactical horizon, has lost the slot it was supposed to take when the command was issued. Hence, the flight then has to take extra holding time.

This effect is slightly mitigated by the turnaround time, up to a certain level, because higher TTs mean less slack for the next flight. Hence, slowing down commands are less likely to be issued, decreasing the side effect of having to hold longer afterwards.

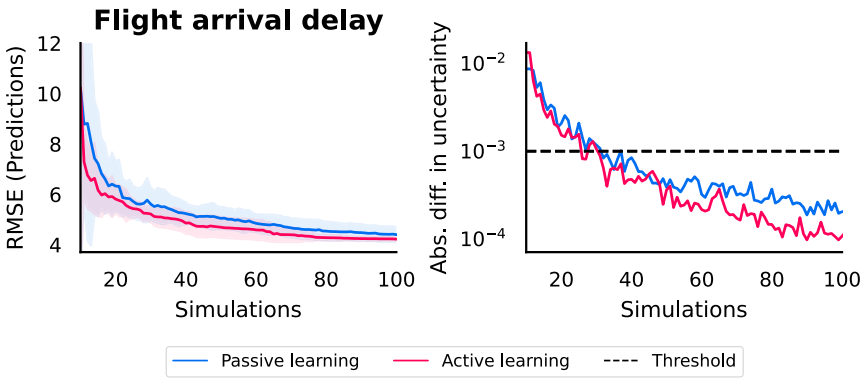

Fig. 12. Active learning curves for the flight arrival delay. The figure shows the mean results of 30 repetitions, represented by lines, with the shaded area indicating the range of $\mu \pm \sigma$. **Left**: The root mean square error (RMSE) of the predictions for the two acquisition functions over 90 acquisitions. **Right**: The absolute difference in the epistemic uncertainty between the active learning iterations, where the epistemic uncertainty for the model is predicted for the unlabelled data set.

Table 5
The mean (of 30 runs) correlation coefficients between the normalised noise-free predictive uncertainty and the RMSE of the predictions and RMSE of the SHAP values, computed across the 100 active learning iterations.

KPI	$\rho_{\text{RMSE (pred)}}$	$\rho_{\text{RMSE (SHAP)}}$
Flight arrival delay	0.81 (0.10)	0.68 (0.15)
Flight departure delay	0.82 (0.09)	0.67 (0.17)
Passenger arrival delay	0.75 (0.16)	0.74 (0.16)
Planned absorbed delay	0.69 (0.31)	0.57 (0.40)
Holding time	0.71 (0.20)	0.69 (0.22)
Fuel cost	0.93 (0.04)	0.25 (0.35)

5. Further discussion

In this section, we discuss two important technical points that contribute to the efficiency of the proposed method. First, we look at stopping criteria for the active learning process because, contrary to conventional machine learning techniques, the 'ground truth' is being computed during training, making stopping criteria based on standard errors moot. Second, we discuss the feasibility and merit of single versus multi KPI active learning and how one can help the other.

5.1. Practical considerations on stopping criterion for the active learning

While there is sometimes a hard constraint on the budget for simulation runs, which limits the feasible number of simulations, in some cases, it is more natural to have a stopping criterion, i.e. a threshold in some pre-defined value after which the model is deemed good enough to be used without the simulator. However, contrary to what is typically done in standard machine learning, with active learning, we cannot test the model on a hold-out test dataset. Indeed, building this dataset would require a considerable amount of simulation runs, which precisely defeats the purpose of active learning.

A common tool to utilise in such situations is cross-validation, which could be used to estimate the generalisation performance of the model, although it implies a computational overhead. However, since the GPs have an intrinsic measure of the epistemic uncertainty in the form of the noise-free predictive uncertainty, a computational-free measure is to evaluate it and define the stopping criterion based on the epistemic uncertainty, or changes therein, to be below a certain threshold (Antunes et al., 2018; Hino, 2020). To do this, one needs to check the uncertainty, which is a good approximation for the performance of the model itself.

In Fig. 12, on the right, we show the absolute difference in the epistemic uncertainty between the active learning iterations, where the epistemic uncertainty for the model is predicted for the unlabelled data set. Comparing it to the left plot in Fig. 12, we see that curves for the absolute difference in the uncertainty approximately follow that of the RMSE of the SHAP values. Other examples of these curves for the other KPIs are in Appendix C.

For a quantitative evaluation of how closely the epistemic uncertainty follows the RMSE of the predictions, we show the correlation coefficients ρ between the two in Table 5. The coefficients ρ range from 0.69 to 0.93, and we thus see that the epistemic uncertainty from the GP is sufficiently correlated with the RMSE to be used as a proxy for the predictive performance.

We now use the epistemic uncertainty estimate as a stopping criterion in practice. More specifically, we compute the absolute difference between the normalised epistemic uncertainty, such that it is independent of the scale of the output. We define the stopping criterion such that we either stop if this metric is below a certain value (0.001) three iterations in a row or after we have queried 100 simulations. In Table 6, we see that the stopping criterion requires a different number of simulations for the different

Table 6
The performance of the XALM using the stopping criterion averaged across 30 experiments. The first column contains the average number of iterations needed for converge.

KPI	Simulations	RMSE (pred)	RMSE (SHAP)	Time [hr]
Flight arrival delay	16.3 (2.7)	6.32 (1.10)	1.88 (0.76)	0.78 (0.13)
Flight departure delay	15.5 (1.5)	6.27 (1.41)	1.90 (0.91)	0.73 (0.07)
Passenger arrival delay	26.5 (14.8)	53.98 (5.89)	9.26 (2.49)	1.28 (0.75)
Planned absorbed delay	41.5 (32.2)	0.12 (0.01)	0.03 (0.01)	2.05 (1.62)
Holding time	96.4 (13.8)	0.17 (0.01)	0.03 (0.01)	4.49 (0.66)
Fuel cost	13.1 (1.2)	785 (193)	795 (385)	0.67 (0.13)

Table 7
The performance of the metamodels trained on the simulations queried by the XALM used for the 'holding time'.

KPI	RMSE (pred)	RMSE (SHAP)
Flight arrival delay	4.39 (0.38)	1.72 (0.82)
Flight departure delay	4.26 (0.35)	1.69 (0.81)
Passenger arrival delay	47.16 (2.33)	7.70 (1.82)
Planned absorbed delay	0.11 (0.01)	0.03 (0.01)
Holding time	0.17 (0.01)	0.03 (0.01)
Fuel cost	399 (26)	815 (236)

KPIs, where the flight arrival and departure delay together with fuel cost requires fewer than 17 simulations to achieve reasonable performance, whereas the holding time almost needs the full computational budget, on average, with 97 simulations.

5.2. Reusing simulations across KPIs

Simulators usually output multiple KPIs at once, and thus, for each simulation input variable vector $\boldsymbol{x}$, there are multiple outputs $\boldsymbol{y}$. This means that we can consider multiple KPIs simultaneously and create a single explainable active learning metamodel to predict all the KPIs of interest, and thus save a significant amount of the computational burden of the simulator. Riis et al. (2021) apply active learning such that new data points are acquired based on all the KPIs, resulting in new data points that add as much information about the KPIs as possible in each iteration. However, if we know or expect *a priori* that an output is more complex to learn and requires more simulations than the others, e.g. such as 'holding time', in this specific case, we can instead perform active learning only with respect to this KPI and then reuse the simulations to fit the metamodels for the other KPIs.

Table 7 shows the performance of the metamodels trained on the data points queried by the XALM used for a single KPI, 'holding time'. If we compare the results in Table 7 with those in Table 4, we see that the RMSEs for the predictions are slightly worse than for XALMs optimised for the specific KPIs, though still on a par with the passive sampling. If we compare the RMSEs for the SHAP values, our XALM optimised for the holding time is better than random sampling in three out of six KPIs and is on par with the other three. Reusing the simulations to train the metamodels for the other KPIs reduces the overall computational time from 10 to 4.5 h in total.

In sum, XALM is perfectly compatible with the reuse of simulation results originating from previous experiments conducted on the same simulator as long as the same set of input variables and KPIs are used. This characteristic is aligned with the computationally economical approach meant for the proposed framework since its conception. Moreover, advanced strategies could be used to build metamodels that combine different active learning schemes optimised for the distinct complexities exhibited by the different KPIs while reusing, whenever necessary, old simulation results during the process.

6. Conclusions

In this paper, we proposed and demonstrated XALM (eXplainable Active Learning Metamodel), a three-step framework that integrates active learning and SHAP values with simulation metamodels, thus addressing computational and interpretational limitations while improving interpretability. XALM offers a unified modelling approach to efficiently discover the intricate hidden relationships between the input and output variables of a simulator. Through two experiments based on an ATM scenario (E-AMAN), we demonstrated the predictive performance of XALM, showing that it achieves comparable results to an XGBoost metamodel with significantly fewer simulations.

In a third experiment on explainability, we highlighted the superior explanatory capabilities of XALM compared to non-active learning metamodels. We also discussed two practical approaches to further reduce computational burden by introducing a stopping criterion based on metamodel uncertainty and by reusing previously queried simulations for different key performance indicators.

Applying XALM to the ATM case study using the Mercury simulator, we also explored the effects of extending the range and scope of the arrival manager by analysing six variables. The ATM case study highlighted the effectiveness of XALM in enhancing the interpretability of simulations, enabling a deeper understanding of variable interactions. Our approach complements traditional simulation-based analyses, providing a practical solution to computational challenges and improving explainability.

We close with some concluding thoughts as to when, in particular, one would want to apply a model of a model in the realm of ATM simulations. Of course, not all analyses for which simulations are used would benefit from metamodelling. For example, if the simulation's goal was to assess whether a certain new procedure respects given safety requirements, the precision, accuracy and eventual outlier events are of utmost importance, which would not be expected to lend itself to the application of metamodelling. However, when modelling the ATM system with the goal of performance assessment (for example), metamodelling offers the following benefits:

- It facilitates more comprehensive analyses due to the efficient utilisation of a few simulation runs to generalise across scenarios, increasing the potential number of scenarios considered in a given amount of (modelling) time and strengthening the scalability of Solution/s assessment.
- It makes simulation results more explainable, facilitating interpretation, which can be of help to convey the results to policy-makers (and other decision-makers), especially those who are not simulation experts, and to improve predictability relating to potential operational and policy changes.[9]
- It can especially enhance scenario-based and 'what-if' analyses, as it can be used to explore the scenario space of the simulator with higher efficiency, often including an improved understanding of broad KPI interdependencies.
- As a broader *enabler*, it facilitates and promotes a potential move towards a more fully consolidated performance assessment framework, ultimately accommodating a suite of simulators, each tailored to one or more KPAs or modelling capabilities (such as environmental impact, airspace structure, passenger itineraries); this, in turn, would support a future truly common database of parameters (such as traffic assumptions, estimated aircraft performance, assumed fuel and carbon prices) and a common API syntax (one simulator being able to call elements of another), preferably in an open source environment — thus driving a more service-oriented architecture.

CRediT authorship contribution statement

Christoffer Riis: Conceptualization, Data curation, Formal analysis, Methodology, Software, Writing – original draft, Writing – review & editing. **Francisco Antunes:** Formal analysis, Methodology, Writing – original draft, Writing – review & editing. **Tatjana Bolić:** Formal analysis, Writing – original draft, Writing – review & editing. **Gérald Gurtner:** Data curation, Formal analysis, Writing – review & editing. **Andrew Cook:** Formal analysis, Writing – review & editing. **Carlos Lima Azevedo:** Formal analysis, Writing – review & editing. **Francisco Câmara Pereira:** Formal analysis, Supervision, Writing – review & editing.

Acknowledgements

This work was supported by the NOSTROMO (Next-generation Open-Source Tools for ATM PeRfOrmance Modelling and Optimisation) project, framed in the scope of the SESAR 2020 Exploratory Research topic SESAR-ER4-26-2019, 'ATM Validation for a Digitalised ATM,' with focus on the 'Macro-modelling applied to Air Traffic Management' area and funded by SESAR Joint Undertaking through the European Union's Horizon 2020 research and innovation programme under grant agreement No 892517.

Appendix A. XGBoost

A Gradient Boosting Machine (GBM) is an ensemble of weak predictors used for regression and classifications in machine learning with great success (Nielsen, 2016). The model is built in a stage-wise fashion similar to other ensemble methods, such as AdaBoost or Random Forest. We will consider the weak predictors to be a small regression tree $h_m(\boldsymbol{x};\boldsymbol{a}_m)$ with hyperparameters $\boldsymbol{a}_m$ for the input data point $\boldsymbol{x}$. Then the GBM is given as

$$F\left(\boldsymbol{x};\left\{\beta_m,\boldsymbol{a}_m\right\}_{m=1}^{M}\right)=\sum_{m=1}^{M}\beta_m h_m\left(\boldsymbol{x};\boldsymbol{a}_m\right), \tag{4}$$

where β_m is the learning rate and M is the number of boosting steps. The optimal solution is then given by the function minimising the expected loss of all the data $D=(X,\boldsymbol{y})$ as

$$F^*=\arg\min_F \mathbb{E}_{X,\boldsymbol{y}}[L(\boldsymbol{y},F(X;\left\{\beta_m,\boldsymbol{a}_m\right\}_{m=1}^{M}))], \tag{5}$$

where $L(\boldsymbol{y},F(X;\left\{\beta_m,\boldsymbol{a}_m\right\}_{m=1}^{M}))$ is the loss function, e.g. root mean square error (RMSE).

To achieve an efficient implementation of a gradient boosting machine (GBM) with decision trees, the XGBoost framework (Chen and Guestrin, 2016) is utilised. XGBoost employs a depth-wise tree growth approach and constructs trees up to a certain depth. This implementation benefits from the high speed of GBM by enabling the computation of the splits in parallel. Additionally, XGBoost incorporates several techniques to prevent overfitting, such as L2-regularisation and the learning rate as a shrinkage parameter.

[9] For example, the Implementing Regulation 2021/116 defines the establishment of the Common Project One supporting the implementation of the European Air Traffic Management Master Plan, where certain Solutions/functionalities are chosen for synchronised, subsidised deployment. The choices are based on the performance assessment results.

Table 8
Hyperparameters optimised with a grid search.

Hyperparameter	Space
Max depth	[3, 4, 5, 6, 7, 8, 9]
L2-regularisation	[0, 0.1, 0.2, 0.3, 0.4, 0.5]
Learning rate	[0.01, 0.05, 0.1, 0.2, 0.3, 0.4]

Table 9
Hyperparameters for XGBoost. The hyperparameters have been found using the grid search technique.

KPI	#sim	Max depth	Learning rate	L2-reg.
Flight arrival delay	50,000	3	0.01	0.5
	1000	3	0.05	0.2
	100	4	0.30	0.1
	30	3	0.30	0.4
Flight departure delay	50,000	3	0.01	0.4
	1000	3	0.10	0.0
	100	6	0.20	0.1
	30	3	0.30	0.4
Passenger arrival delay	50,000	5	0.01	0.5
	1000	4	0.01	0.5
	100	3	0.10	0.1
	30	5	0.40	0.5
Planned absorbed delay	50,000	4	0.05	0.2
	1000	4	0.30	0.1
	100	5	0.30	0.5
	30	5	0.10	0.2
Holding time	50,000	3	0.20	0.5
	1000	3	0.20	0.0
	100	6	0.40	0.5
	30	3	0.40	0.1
Fuel cost	50,000	4	0.01	0.4
	1000	5	0.01	0.0
	100	9	0.20	0.1
	30	4	0.40	0.3

A.1. Optimising XGBoost

In accordance with the standard practice in machine learning, we optimise the hyperparameters of each XGBoost model by employing the train-validation-test split technique to ensure strong generalisation performance across all relevant key performance indicators. We follow the setup of Riis et al. (2022) and construct a training dataset comprising 50,000 simulations, as well as a test dataset of 10,000 simulations, both generated through Latin hypercube sampling. Subsequently, we randomly divide the training dataset into a smaller training dataset of 40,000 simulations and a validation dataset of 10,000 simulations, which allows us to fine-tune the hyperparameters of XGBoost using a grid search. Through 10-fold cross-validation splits, we optimise the maximum depth of each decision tree, L2 regularisation, and learning rate by specifying a search space for each hyperparameter, as in Table 8.

Additionally, we employ the exact greedy algorithm to select the decision tree splits. We imposed a maximum limit of 1000 boosting iterations for all models, and early stopping was employed if the validation accuracy did not improve over a span of five boosting steps. The optimal hyperparameters are listed in Table 9.

Appendix B. The effect of turnaround time on flight arrival and departure delay

The SHAP values and model predictions for the flight arrival and departure delay are very similar, cf. Figs. 13 and 14. In the following, we describe the arrival delay. In the left plot in Fig. 13, we see an almost linear relationship between the input and the output, and, for example, if the turnaround time is near 0, the delay is the base value subtracted by 150 min. On the other hand, a turnaround time close to 10 gives a delay at 200 min above the mean. The right plot in Fig. 13 shows the prediction of the delays, when the features are set to the default values, and only the turnaround time scale is changed. The predictions confirm the trend of the SHAP values, and we see that there is only little uncertainty about the mean predictions and almost no stochasticity in the delays.

Appendix C. Active learning curves with stopping criterion

In Fig. 15, we show two plots for each key performance indicator. To the left, we show the active learning loss curves, and to the right, we show the absolute difference in the epistemic uncertainty between the active learning iterations, where the epistemic

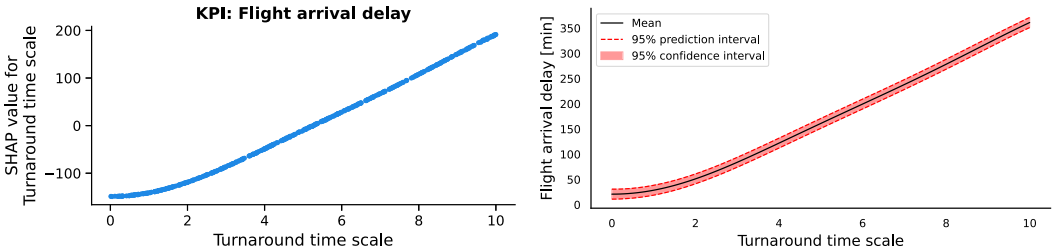

Fig. 13. The impact of the turnaround time scale on flight arrival delay. **Left**: SHAP values. **Right**: Model predictions.

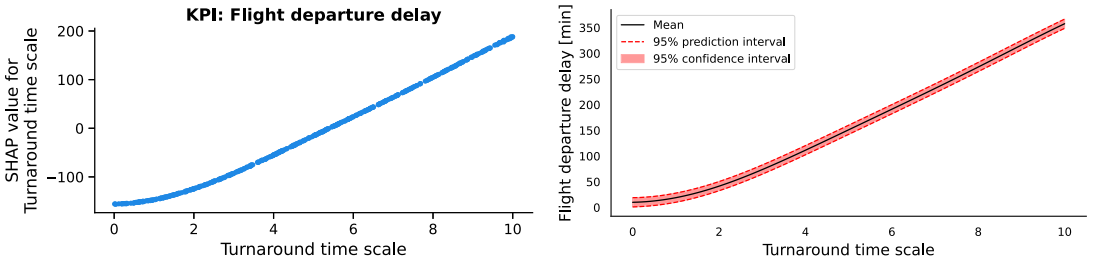

Fig. 14. The impact of the turnaround time scale on flight departure delay. **Left**: SHAP values. **Right**: Model predictions.

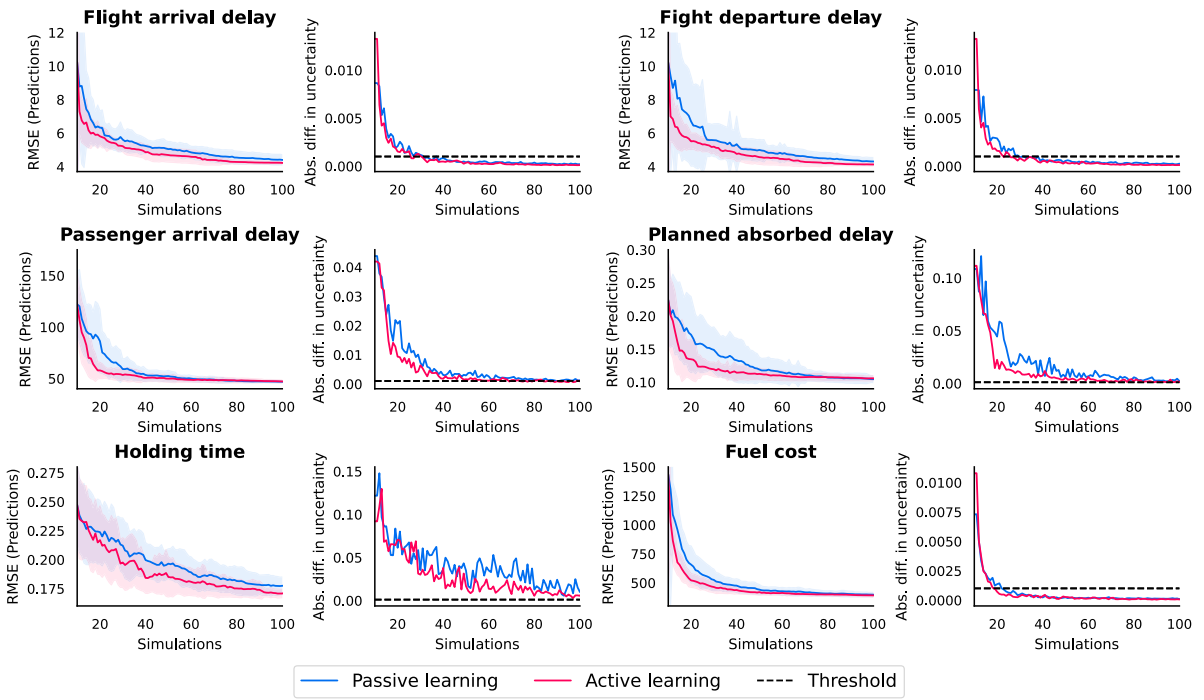

Fig. 15. Active learning curves for the six KPIs. The figure shows the mean results of 30 repetitions, represented by lines, with the shaded area indicating the range of $\mu \pm \sigma$. For each KPI, the left panel shows the root mean square error (RMSE) of the predictions for the two acquisition functions over 90 acquisitions, and the right panel displays the absolute difference in the epistemic uncertainty between the active learning iterations, where the epistemic uncertainty for the model is predicted for the unlabelled data set.

uncertainty for the model is predicted for the unlabelled data set. Comparing the right plots to the left plots in Fig. 12, we see that curves for the absolute difference in the uncertainty approximately follow that of the RMSE of the SHAP values.

References

Antunes, F., Ribeiro, B., Pereira, F.C., Gomes, R., 2018. Efficient Transport Simulation With Restricted Batch-Mode Active Learning. IEEE Trans. Intell. Transp. Syst. 19 (11), 3642–3651. http://dx.doi.org/10.1109/TITS.2018.2842695, URL https://ieeexplore.ieee.org/document/8419064/.

Bolić, T., Ravenhill, P., 2021. SESAR: The past, present, and future of European air traffic management research. Engineering 7 (4), 448–451.

Burr Settles, 2010. Active Learning Literature Survey. Technical Report 1648, University of Wisconsin–Madison.

Cano, M., Perillo, A., López, J.A., Tello, F., Poveda, J., Câmara, F., Antunes, F., Riis, C., Crook, I., Tibichte, A., et al., 2023. NOSTROMO: Lessons learned, conclusions and way forward. arXiv preprint arXiv:2303.18060.

Chen, T., Guestrin, C., 2016. Xgboost: A scalable tree boosting system. In: Proceedings of the 22nd Acm Sigkdd International Conference on Knowledge Discovery and Data Mining. pp. 785–794.

Cheng, Q., Wang, S., Liu, Z., Yuan, Y., 2019. Surrogate-based simulation optimization approach for day-to-day dynamics model calibration with real data. Transp. Res. C 105, 422–438.

Ciuffo, B., Azevedo, C.L., 2014. A sensitivity-analysis-based approach for the calibration of traffic simulation models. IEEE Trans. Intell. Transp. Syst. 15 (3), 1298–1309.

Cook, A., 2007. European Air Traffic Management: Principles, Practice, and Research. Ashgate Publishing, Ltd..

Cook, A., Rivas, D., 2016. Complexity Science in Air Traffic Management. Routledge London.

Cook, A.J., Tanner, G., 2011. European Airline Delay Cost Reference Values. Technical Report, EUROCONTROL Performance Review Unit.

Cook, A.J., Tanner, G., 2015. European Airline Delay Cost Reference Values - Updated and Extended Values (Version 4.1). Technical Report, University of Westminster.

Cook, A.J., Tanner, G., Bolic, T., 2021-11-15. D3.2 Industry Briefing on Updates to the European Cost of Delay. Technical Report, BEACON Consortium.

Dantsuji, T., Hoang, N.H., Zheng, N., Vu, H.L., 2022. A novel metamodel-based framework for large-scale dynamic origin–destination demand calibration. Transp. Res. C 136, 103545.

Delgado, L., Gurtner, G., Mazzarisi, P., Zaoli, S., Valput, D., Cook, A., Lillo, F., 2021. Network-wide assessment of ATM mechanisms using an agent-based model. J. Air Transp. Manag. 95, 102108.

Delgado, L., Gurtner, G., Weiszer, M., Bolić, T., Cook, A., 2023. Mercury: an open source platform for the evaluation of air transport mobility. In: SESAR Innovation Days 2023. SESAR 3 Joint Undertaking, pp. 1–9. http://dx.doi.org/10.61009/SID.2023.1.36.

Delgado, L., Gurtner, G., Zaoli, S., Mazzarisi, P., Valput, D., Cook, A., Lillo, F., 2019. Final tool and model description, and case studies results. In: Domino Project, Deliverable 5.3. SESAR.

Erickson, C.B., Ankenman, B.E., Sanchez, S.M., 2018. Comparison of Gaussian process modeling software. European J. Oper. Res. 266 (1), 179–192.

EUROCONTROL, 2010. European Operational Concept Validation Methodology, E-OCVM v3. European Organisation for the Safety of Air Navigation (EUROCONTROL) Brussels.

European Commission, 2004. Regulation (EC) No 261/2004 of the European Parliament and of the Council of 11 February 2004 Establishing Common Rules on Compensation and Assistance to Passengers in the Event of Denied Boarding and of Cancellation or Long Delay of Flights, and Repealing Regulation (EEC) No 295/91, 17 February, 1-7. Technical Report, European Union, URL http://data.europa.eu/eli/reg/2004/261/oj.

Friedman, L.W., 2012. The Simulation Metamodel. Springer Science & Business Media.

Gardner, J.R., Pleiss, G., Bindel, D., Weinberger, K.Q., Wilson, A.G., 2018. Gpytorch: Blackbox matrix-matrix Gaussian process inference with GPU acceleration. Adv. Neural Inf. Process. Syst. 31, 7576–7586, arXiv:1809.11165.

Gramacy, R.B., 2020. Surrogates: Gaussian Process Modeling, Design, and Optimization for the Applied Sciences. CRC Press.

Gurtner, G., Bongiorno, C., Ducci, M., Miccichè, S., 2017. An empirically grounded agent based simulator for the air traffic management in the SESAR scenario. J. Air Transp. Manag. 59, 26–43.

Gurtner, G., Delgado, L., 2023. URL https://github.com/UoW-ATM/Mercury.

Gurtner, G., Delgado, L., Valput, D., 2021-12. An agent-based model for air transportation to capture network effects in assessing delay management mechanisms. Transp. Res. C 133, 103358. http://dx.doi.org/10.1016/j.trc.2021.103358.

Hino, H., 2020. Active learning: Problem settings and recent developments. arXiv preprint arXiv:2012.04225.

Jiang, M., Pedrielli, G., Ng, S.H., 2022. Gaussian processes for high-dimensional, large data sets: A review. In: 2022 Winter Simulation Conference. WSC, IEEE, pp. 49–60.

Kingma, D.P., Ba, J.L., 2015. Adam: A method for stochastic optimization. In: 3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings. pp. 1–15, arXiv:1412.6980.

Kleijnen, J.P., 1997. Experimental design for sensitivity analysis, optimization and validation of simulation models.

Kleijnen, J.P., 2009. Kriging metamodeling in simulation: A review. European J. Oper. Res. 192 (3), 707–716.

Kleijnen, J.P., Sargent, R.G., 2000. A methodology for fitting and validating metamodels in simulation. European J. Oper. Res. 120 (1), 14–29.

Knudde, N., Dutordoir, V., Herten, J.V.D., Couckuyt, I., Dhaene, T., 2020. Hierarchical gaussian process models for improved metamodeling. ACM Trans. Model. Comput. Simul. 30 (4), 1–17.

Law, A.M., 2015. Simulation Modeling and Analysis, fifth ed. McGraw-Hill Higher Education.

Li, M., Sethi, I.K., 2006. Confidence-based active learning. IEEE Trans. Pattern Anal. Mach. Intell. 28 (8), 1251–1261.

Li, Z., Tian, Y., Sun, J., Lu, X., Kan, Y., 2022. Simulation-based optimization of large-scale dedicated bus lanes allocation: Using efficient machine learning models as surrogates. Transp. Res. C 143, 103827.

Li, J., Zhang, C., Zhou, J.T., Fu, H., Xia, S., Hu, Q., 2021. Deep-LIFT: Deep label-specific feature learning for image annotation. IEEE Trans. Cybern. 52 (8), 7732–7741.

Lundberg, S.M., Erion, G., Chen, H., DeGrave, A., Prutkin, J.M., Nair, B., Katz, R., Himmelfarb, J., Bansal, N., Lee, S.-I., 2020. From local explanations to global understanding with explainable AI for trees. Nat. Mach. Intell. 2 (1), 56–67.

Lundberg, S.M., Lee, S.-I., 2017. A unified approach to interpreting model predictions. Adv. Neural Inf. Process. Syst. 30.

MacKay, D.J.C., 1992. Information-based objective functions for active data selection. Neural Comput. 4 (4), 590–604. http://dx.doi.org/10.1162/neco.1992.4.4.590.

Nielsen, D., 2016. Tree Boosting with Xgboost - Why Does Xgboost Win "Every" Machine Learning Competition?. NTNU.

Nuic, A., Poles, D., Mouillet, V., 2010. BADA: An advanced aircraft performance model for present and future ATM systems. Int. J. Adapt. Control Signal Process. 24 (10), 850–866.

Osorio, C., Bierlaire, M., 2013. A simulation-based optimization framework for urban transportation problems. Oper. Res. 61 (6), 1333–1345.

Phillips, M., Marsh, D.T., 2000. The validation of fast-time air traffic simulations in practice. J. Oper. Res. Soc. 51 (4), 457–464.

Rasmussen, C.E., Williams, C., 2006. Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning). The MIT Press.

RayChaudhuri, T., Hamey, L.G., 1995. Minimization of data collection by active learning. In: IEEE International Conference on Neural Networks - Conference Proceedings, Vol. 3. pp. 1338–1341. http://dx.doi.org/10.1109/icnn.1995.487351.

Ribeiro, M.T., Singh, S., Guestrin, C., 2016. "Why should i trust you?" Explaining the predictions of any classifier. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 1135–1144.

Riis, C., Antunes, F., Bolic, T., Gurtner, G., Pereira, F.C., Azevedo, C.M.L., 2022. Explainable metamodels for ATM performance assessment. In: Proceedings of the 12th SESAR Innovation Days, Vol. 2022.

Riis, C., Antunes, F., Gurtner, G., Pereira, F.C., Delgado, L., Azevedo, C.M.L., 2021. Active learning metamodels for ATM simulation modeling. In: Proceedings of the 11th SESAR Innovation Days, Vol.2021.

Sánchez-Cauce, R., Riis, C., Antunes, F., Mocholí, D., G. Cantú Ros, O., Pereira, F.C., Herranz, R., Azevedo, C.M.L., 2022. Active learning metamodelling for R-NEST. In: Proceedings of the 12th SESAR Innovation Days, Vol. 2022.

Sauer, A., Cooper, A., Gramacy, R.B., 2023. Non-stationary Gaussian process surrogates. arXiv preprint arXiv:2305.19242.

SESAR 3 JU, 2023. Extended Arrival Management (AMAN) horizon. URL https://www.sesarju.eu/sesar-solutions/extended-arrival-management-aman-horizon.

SESAR Joint Undertaking, 2018. Vision of the Future Performance Research in SESAR. Technical Report, Project PJ19 CI.

SESAR Joint Undertaking, 2020. European ATM master plan: digitalising europe's aviation infrastructure, executive view.

Van Beers, W.C., Kleijnen, J.P.C., 2003. Kriging for interpolation in random simulation. J. Oper. Res. Soc. 54 (3), 255–262.

Van Beers, W.C., Kleijnen, J.P., 2004. Kriging interpolation in simulation: a survey. In: Simulation Conference, 2004. Proceedings of the 2004 Winter, Vol. 1. IEEE.

Yue, X., Wen, Y., Hunt, J.H., Shi, J., 2020. Active learning for Gaussian process considering uncertainties with application to shape control of composite fuselage. IEEE Trans. Autom. Sci. Eng. 18 (1), 36–46.