Computers and Chemical Engineering 201 (2025) 109243

Contents lists available at ScienceDirect

Computers and Chemical Engineering

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

# Simulator-based surrogate optimisation employing adaptive uncertainty-aware sampling

Yu Liu, Fabricio Oliveira *

*Department of Mathematics and Systems Analysis, School of Science, Aalto University, Espoo, Finland*

ARTICLE INFO

Dataset link: https://github.com/gamma-opt/adaptive-surrogate-opt

*Keywords:*
Mathematical optimisation
Simulation
Mixed-integer programming
Neural networks

ABSTRACT

Optimisation problems involving computationally expensive, black-box functions derived from high-fidelity engineering simulations remain challenging. To efficiently bridge the simulators and optimisation processes, we introduce an adaptive framework for surrogate modelling and optimisation. Our method employs low-discrepancy sequence sampling to select points, followed by training a surrogate model using a piecewise linear neural network (NN) with rectified linear unit (ReLU) activation. Using mixed-integer programming (MIP), we reformulate the ReLU NN as embedded components of an optimisation problem and solve it to find an optimal simulator input. This is achieved by iteratively refining the solution via resampling the simulator, retraining the surrogate model, and rebuilding and resolving the MIP problem. For resampling, an infill strategy that incorporates uncertainty assessment and a solution pool is employed, balancing exploration and exploitation. Moreover, computational efficiency is boosted by bound tightening, lossless model compression, and memory structure reuse. Validation on practical engineering applications confirms significant optimisation efficiency gains from the domain-refined strategy.

## 1. Introduction

The optimisation of complex functions is a frequently encountered challenge in various high-fidelity simulations, where input/output data is obtained through simulators. These simulators are intricately designed to predict the behaviour of physical systems by resolving the mathematical equations that describe underlying physical processes. Employed extensively in diverse sectors, such as automotive engineering to optimise combustion processes for improved fuel efficiency (Aithal and Balaprakash, 2019), chemical engineering to enhance manufacturing processes ensuring consistency and quality (Tsay, 2021; Addis et al., 2023), and aerospace engineering to streamline the aerodynamic design of gas turbine blades (Zhang and Janeway, 2022), simulator-based optimisation methods are instrumental in pushing the boundaries of operational efficiency.

However, despite their precision and utility, the direct use of high-fidelity simulators in optimisation tasks often entails prohibitive computational costs and significant time investments, particularly when exploring large parameter spaces or when multiple iterations are required. Additionally, many simulators are black-box systems and, as such, obtaining the derivatives needed for gradient-based optimisation methods is challenging, limiting the efficiency of these optimisation strategies. These challenges are amplified in industrial applications where decision-making speed is crucial, and in research environments where extensive explorations of theoretical models are necessary.

These shortcomings from direct optimisation using the simulator expose the need for an alternative approach that maintains the integrity of simulation outputs while mitigating the computational burdens. One such approach is the development and use of surrogate models (Bhosekar and Ierapetritou, 2018), which offer a viable solution by approximating the outputs of these complex simulators. Surrogate models are built to capture the essential features and behaviours of the original simulators. These models are conceived to be computationally cheaper to evaluate and smoothly integrate with optimisation algorithms, providing faster convergence rates and facilitating more dynamic exploration of parameter spaces.

Surrogate models come in various forms to leverage simulator-generated data (Alizadeh et al., 2020). Polynomial regression is useful for capturing polynomial relationships, including linear and quadratic forms (Cheng et al., 2019); Kriging is ideal for modelling smooth and continuous functions, providing a robust statistical foundation (Kleijnen, 2014); radial basis functions are used for multidimensional interpolation problems, often applied when data lacks a structured grid (Fasshauer and McCourt, 2012; Wendland, 2017); decision trees provide a hierarchical structure for modelling nonlinear decision boundaries by segmenting the input space (Hehn et al., 2020); neural

* Corresponding author.
*E-mail address:* fabricio.oliveira@aalto.fi (F. Oliveira).

https://doi.org/10.1016/j.compchemeng.2025.109243
Received 17 October 2024; Received in revised form 10 June 2025; Accepted 10 June 2025
Available online 23 June 2025
0098-1354/© 2025 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

networks (NNs) are adept at modelling complex patterns and nonlinear relationships within high-dimensional spaces (Canziani et al., 2017; Advani et al., 2020).

Among these, NNs with rectified linear unit (ReLU) activation functions present a unique advantage in optimisation tasks due to their piecewise linear characteristics. When the network architecture and parameters are fixed after training, a ReLU NN inherently represents a piecewise linear function. This property means that optimisation problems incorporating such networks naturally take the form of piecewise linear problems, simplifying the complexity involved in finding the global optimum (Perakis and Tsiourvas, 2022). In addition to the ReLU function and its variations, other most commonly used piecewise linear activation functions include hard tanh and maxpooling (Tao et al., 2022). Once an NN has been well trained, it can be employed for optimisation tasks through several approaches. Gradient-based methods directly extract gradients from the network and apply gradient-based optimisation techniques (Laurent et al., 2019). Alternative strategies include derivative-free methods, such as swarm optimisation or genetic algorithms, which do not rely on gradients (Bhosekar and Ierapetritou, 2018).

In addition, mathematical programming formulations, particularly mixed-integer programming (MIP), offer a powerful framework for exact optimisation of nonconvex piecewise linear functions (Huchette et al., 2023). ReLU NNs, an MIP-representable class of NNs, have garnered increasing interest for optimising over a trained NN (Fischetti and Jo, 2018; Grimstad and Andersson, 2019; Anderson et al., 2020; Katz et al., 2020; Yang et al., 2022; Maragno et al., 2023; Tong et al., 2024). Additionally, the use of mature off-the-shelf solvers such as Gurobi (Gurobi Optimization, LLC, 2024a) ensures reliable optimisation performance through efficient computation and rigorous guarantees for globally optimal solutions within specified numerical tolerances. The MIP-based approach, combined with mature solvers, provides mathematically guaranteed optimal solutions for applications demanding high solution quality.

While MIP-based optimisation over trained NNs offers optimality guarantees, its effectiveness critically depends on *infill* strategies that iteratively sample points to refine surrogate models (Martins and Ning, 2021). The ability to quantify prediction uncertainty plays a crucial role in guiding efficient sampling (Hüllen et al., 2020).

Kriging is particularly favoured as a surrogate model for its ability to provide both predictions and uncertainty estimates, enabling informed sampling decisions (Lualdi et al., 2024). For Kriging-based methods, several approaches have emerged: ensemble frameworks with adaptive model selection (Lu et al., 2023), hybrid models combining global and local basis functions (Hu et al., 2023), and evolutionary algorithms with multiple infill sampling strategies (Zhu et al., 2024). Other advances include uncertainty-driven approaches through grouping-based criteria (Liu et al., 2021) and dual selection mechanisms based on lower confidence bounds (Li et al., 2023). Nevertheless, while these Kriging-based approaches offer uncertainty estimates, the computational requirements associated with model fitting typically scale cubically with the number of training points due to the associated covariance matrix operations (Kleiber and Nychka, 2015), and, consequently, have limited usefulness in settings with high-dimensional problems, often becoming intractable for large-scale applications (Anahideh et al., 2022). This computational limitation of Kriging methods has motivated research into alternative approaches for uncertainty quantification in high-dimensional settings. NNs present a promising direction, offering both demonstrated favourable scaling properties with sample size and dimension (Guo et al., 2022), as well as various methods for uncertainty estimation such as ensemble techniques (Pearce et al., 2020), Bayesian NNs (Magris and Iosifidis, 2023), and dropout-based approaches (Gal and Ghahramani, 2016). However, despite these advantages, the potential of NN uncertainty estimation in guiding sampling decisions has been relatively unexplored in the context of optimisation. Additionally, existing approaches have not exploited the mathematical rigour and solution space insights available through MIP optimisation over NNs.

To address these limitations, we present several interconnected contributions within a surrogate-based optimisation framework (Fig. 1). Our primary contribution is the development of an infill strategy that leverages uncertainty estimation and solution pools to widen the exploration–exploitation trade-off opportunities, encompassing both active simulator resampling and static dataset analysis. Additionally, we present the novel integration of adaptive sampling, NNs, and MIP into a unified optimisation framework, demonstrating its efficacy in simulation-dependent optimisation problems with respect to computational demands and convergence stability. Specifically, adaptive sampling helps to dynamically adjust the sampling process based on the uncertainty associated with the NN's predictions and incumbent MIP solutions, thereby ensuring that new data points maximise potential improvements in model accuracy.

Our paper is structured as follows, as illustrated in Fig. 1. Section 2 details the MIP modelling method for ReLU NNs, emphasising techniques for efficient solving. Section 3 describes the infill strategy, which utilises estimated uncertainty information and solution pools. Section 4 presents the proposed general framework for surrogate modelling and optimisation. Section 5 presents the numerical experiments involving two different scenarios. Section 6 concludes the paper, highlighting key findings and opening research directions for further work.

## 2. Surrogate embeddings

In this section, we show the use of surrogate embeddings, focusing on how pre-trained ReLU NNs can be embedded as constraints in MIP formulations to optimise complex functions within simulations. Additionally, we discuss techniques like bound tightening and lossless model compression to boost computational efficiency.

### 2.1. Conceptual model

For a given simulation process governed by an underlying black-box function $f$, suppose we have a dataset $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{n_s}$, obtained by running the simulator with $n_s$ different inputs, where the inputs $x^{(i)}$ for each sample $i$ are drawn from the design space, i.e., the domain $\mathcal{X}$, and $y^{(i)}$ are corresponding outcomes of interest. This complex process can be approximated and effectively replaced using a surrogate model $\hat{f}$, constructed based on $n_s$ samples from $\mathcal{D}$. We aim to optimise a predefined function $h$ with the surrogate model $\hat{f}$ embedded while also considering the practical constraint $g$. The optimisation problem is formulated as

$$\begin{aligned} \min_{x\in\mathbb{R}^n, y\in\mathbb{R}^m} \; & h(x, y) \\ \text{s.t.: } & g(x, y) \le 0, \\ & y = \hat{f}_{\mathcal{D}}(x), \\ & x \in \mathcal{X}. \end{aligned} \tag{1}$$

The outcomes of interest $y$, could be constrained by a known function $g$ and/or optimised within a known function $h$, reflecting their dependency on the simulation outcomes according to the specific requirements of the application. To render the optimisation problem (1) manageable from a computational standpoint, we assume that the variables $x$ are bounded.

To be effectively incorporated into an optimisation strategy, simulator-based surrogate optimisation requires two critical elements: high accuracy and reasonable solution times. Misspecifications in the learned surrogate can lead to sub-optimal outcomes; therefore, our focus will be on enhancing the accuracy of the surrogate while ensuring that the solution times remain practical. We will explore these aspects in the subsequent sections.

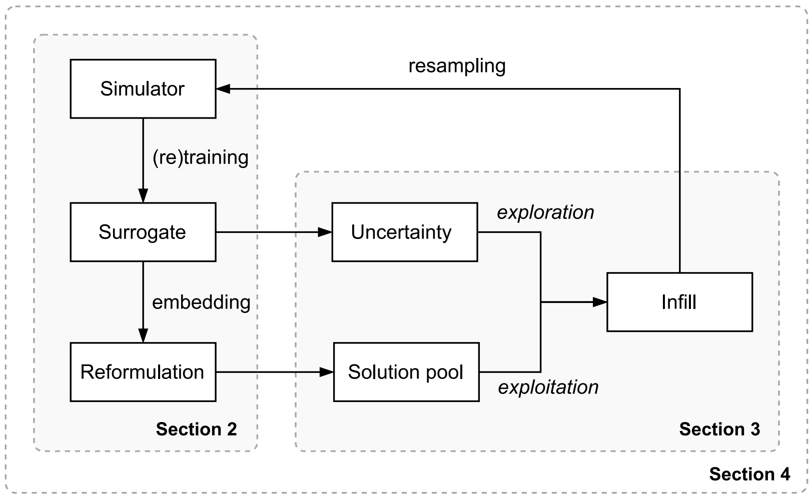

**Fig. 1.** Overview of the proposed simulator-based surrogate optimisation methodology.

### 2.2. MIP formulation

Given the need for efficient optimisation in complex simulation processes, we turn to ReLU NNs as our tool of choice for surrogate modelling as previously discussed in Section 1. We employ the big-M formulation (Fischetti and Jo, 2018) for its implementation simplicity, incorporating techniques to improve its tractability.

Here, we consider a ReLU NN with $L+1$ layer (numbered from 0 to $L$) to build the surrogate model of the simulator $f$. For the input layer, we have $y^0 = x$. For the output layer we have $y^L = \hat{f}_D(x)$. The activation of neurons $i = 1, \dots, N_l$ in the hidden layer $l = 1, \dots, L-1$ are calculated by

$$y_i^l = \mathrm{ReLU}(w_i^{l\top} y_{l-1} + b_i^l) = \max(0, w_i^{l\top} y_{l-1} + b_i^l), \tag{2}$$

where $w_i^l$ and $b_i^l$ denote the weight and bias of the corresponding neuron, respectively. Suppose we are given the lower bounds $L_i^l < 0$ and upper bounds $U_i^l > 0$ such that

$$L_i^l \le w_i^{l\top} y_{l-1} + b_i^l \le U_i^l.$$

Following Fischetti and Jo (2018), the ReLU operator can be encoded into mixed-integer linear constraints by introducing slack variables $s_i^l$ and binary variables $z_i^l$ for $i = 1, \dots, N_l,\ l = 1, \dots, L-1$, which is expressed as

$$\begin{aligned} & y_i^l - s_i^l = w_i^{l\top} y^{l-1} + b_i^l, \\ & 0 \le y_i^l \le U_i^l z_i^l, \\ & 0 \le s_i^l \le -L_i^l \left(1 - z_i^l\right), \\ & z_i^l \in \{0, 1\}. \end{aligned} \tag{3}$$

Combining these constraints (3) with input and output layer bounds

$$L^0 \le y^0 \le U^0, \tag{4}$$

$$L^L \le y^L = w^{L\top} y^{L-1} + b^L \le U^L, \tag{5}$$

we obtain the exact MIP model to embed the ReLU NN surrogate $y = \hat{f}_D(x)$ into the original optimisation problem (1).

The above transformation facilitates seamless integration of the surrogate model into the optimisation framework, providing a direct approach to handling the nonlinearities inherent to the simulator. By leveraging the computational power of a state-of-the-art MIP solver, we can explore the solution space more effectively. This approach ensures that the optimal solution preserves the surrogate model's fidelity in approximating the complex system behaviour.

### 2.3. Techniques for efficiency improvement

The MIP formulation with ReLU NN surrogate embeddings relies on a big-M formulation. The choice of the big-M constants $L$ and $U$ significantly impacts the solution time of the MIP, potentially producing very challenging mixed-integer instances that can test even state-of-the-art solvers (Fischetti and Jo, 2018). Furthermore, this approach introduces as many binary variables as there are ReLU nodes, meaning that the size of the ReLU network directly influences the size of the MIP model. To enhance the computational feasibility of MIP formulations in practical applications, we employed several techniques in our framework.

One such technique involves reducing the big-M values associated with some of the constraints by calculating the minimum and maximum activations of the individual neurons through optimisation. This process, known as *bound tightening*, typically results in smaller big-M values, leading to more efficient MIP formulations. More details on this technique can be found in Grimstad and Andersson (2019).

Utilising these tighter activation bounds, the model can be compressed losslessly by removing units and layers of the NN that do not change the output. This involves removing units with constant outputs regardless of the input, some stable units, and any layers with constant output due to these types of units (Cheng et al., 2020). Consequently, the size and depth of the model are reduced without losing any of its predictive capabilities.

These techniques collectively enhance the computational efficiency of the MIP formulation by addressing key challenges associated with large ReLU networks and big-M constraints. By reducing the big-M values and eliminating redundant units in the NN, the size and complexity of the resulting MIP model are minimised. This not only speeds up the solution process but also makes the framework more scalable and practical for use in real-world applications where computational resources may be limited.

## 3. Infill strategy

This section introduces the key elements of our infill strategy for surrogate refinement, starting with uncertainty estimation to identify areas in the design space that need further exploration. It then covers the resampling procedure, which focuses on selecting new sampling points based on uncertainty and existing solutions.

### 3.1. Uncertainty estimation

When embedding a trained NN into an optimisation problem, there are two primary sources of uncertainty: the functional form of the surrogate model $\hat{f}$ and the parameter estimates that define it (Maragno et al., 2023). While simulator uncertainty from numerical approximations and inherent system randomness constitutes another source of uncertainty, validation and calibration methods can effectively mitigate these effects prior to the NN training (Roy, 2019). Since we are focusing on ReLU NNs as the chosen surrogate model, assuming appropriately validated simulator outputs, we narrow our discussion to *parameter uncertainty*, the uncertainty in the estimates of the model's parameters such as weights and biases.

This uncertainty arises because $\hat{f}$ is trained on a finite dataset $\mathcal{D}$, which may not fully represent the underlying function $f$. As a result, the surrogate model might not perfectly capture the relationship between the input variables $x$ and the output variables $y$. This potential for model misspecification can lead to sub-optimal solutions or even constraint violations in the original optimisation problem (1). Traditionally, optimisation models assume deterministic outcomes, but incorporating uncertainty allows for more robust solutions. Specifically, by factoring in the variability or uncertainty in predictions, the optimisation process can adjust accordingly, ensuring that solutions remain valid even when the surrogate is imperfect.

To quantify the NN parameterisation uncertainty, we employ Monte Carlo Dropout (Gal and Ghahramani, 2016), which introduces controlled randomness into the model during inference (Algorithm 1). Originally introduced as a regularisation method to prevent overfitting during NN training, dropout works by randomly deactivating a subset of neurons in the network based on a given dropout rate $p$ during each training iteration (Hinton et al., 2012). When extended to the testing phase, this approach, known as Monte Carlo Dropout, allows for the generation of a distribution of predictions by performing $M$ multiple forward passes with dropout enabled. The random nature of neuron deactivation in each pass creates variability in the predictions, effectively sampling from different possible network configurations. This method is theoretically justified as an approximate Bayesian inference in deep Gaussian processes, where the prediction variance at each input point provides a local measure of the prediction uncertainty (Gal and Ghahramani, 2016). The local uncertainty measures can be aggregated across multiple input points to construct a comprehensive map of model prediction confidence. We adopt this approach due to its theoretical grounding in approximate Bayesian inference and seamless integration with existing NN architectures, making it particularly suitable for optimisation frameworks requiring repeated uncertainty estimation.

By setting $M$, the number of Monte Carlo samples, to a sufficiently large value, the method can capture a broad range of possible outcomes, thereby providing a more robust estimate of uncertainty. However, there is a trade-off one must consider: increasing $M$ enhances the accuracy of the uncertainty estimation but also increases the computational cost. Therefore, $M$ must be chosen carefully based on the available computational resources and the desired level of confidence in the uncertainty estimates. Notably, this process is highly parallelisable, allowing for a significant reduction in computational overhead when implemented on multi-core processors or distributed systems, making the use of a larger $M$ more feasible.

For each point $x^{(i)}$ in the current sample set $\mathcal{X}_s$, the surrogate model $\hat{f}$ with dropout rate $p$ is evaluated $M$ times under Monte Carlo Dropout, producing a set of predictions $\hat{F}^{(i)} = \{\hat{f}_1(x^{(i)}), \hat{f}_2(x^{(i)}), \ldots, \hat{f}_M(x^{(i)})\}$ (Line 12–19). To estimate uncertainty, the standard deviation $\sigma^{(i)}$ is computed from the set of predictions $\hat{F}^{(i)}$ (Line 20) and then appended to the set $\Sigma$, which stores the uncertainty for each point in $\mathcal{X}_s$ (Line 21). A higher standard deviation $\sigma^{(i)}$ indicates greater uncertainty in the model's predictions at point $x^{(i)}$, signalling regions in the design space that may require additional sampling to improve the model's accuracy and reliability.

This method of uncertainty estimation is particularly useful in surrogate-based optimisation, as it directs attention to areas where the model is less confident, ensuring that subsequent sampling efforts are concentrated where they are most beneficial.

**Algorithm 1** Uncertainty Estimation

```
1:  function MonteCarloDropout($\hat{f}, \mathcal{X}_s, M, p$)
2:      inputs:
3:          $\hat{f}$: ReLU NN surrogate model
4:          $\mathcal{X}_s$: Set of data points $\{x^{(1)}, x^{(2)}, \ldots, x^{(n)}\}$
5:          $M$: Number of Monte Carlo samples
6:          $p$: Dropout rate during inference
7:      outputs:
8:          $\Sigma$: Estimated uncertainty (standard deviations) for each point in $\mathcal{X}_s$
9:      Initialise $\Sigma \leftarrow \{\}$                ▷ Empty set to store uncertainties
10:     for each $x^{(i)}$ in $\mathcal{X}_s$ do
11:         Initialise $\hat{F}^{(i)} \leftarrow \{\}$        ▷ Empty set for predictions of $x^{(i)}$
12:         for $n = 1$ to $M$ do
13:             for each layer in $\hat{f}$ do      ▷ Enable Dropout in $\hat{f}$ during inference
14:                 if the layer is a dropout layer then
15:                     Set the layer to training mode to activate dropout with rate $p$
16:                 end if
17:             end for
18:             Append $\hat{f}(x^{(i)})$ to $\hat{F}^{(i)}$      ▷ Store the prediction with dropout enabled
19:         end for
20:         Compute $\sigma^{(i)} \leftarrow \mathrm{std}(\hat{F}^{(i)})$      ▷ Estimate uncertainty via standard deviation
21:         Append $\sigma^{(i)}$ to $\Sigma$
22:     end for
23:     return $\Sigma$                ▷ Uncertainties for all points in $\mathcal{X}_s$
24: end function
```

## 3.2. Resampling procedure

An effective infill strategy requires a careful balance between exploitation and exploration. Local exploitation focuses on well-performing areas of the search space to enhance current optima, while global exploration broadens the search to poorly represented areas to uncover potential global optima.

During our optimisation process, we identify an optimal solution $\hat{x}^*$, along with additional (sub-)optimal solutions kept in a solution pool, represented as the set $\mathcal{S}_{\text{pool}}$. The solution pool is a feature that some MIP solvers provide, including Gurobi (Gurobi Optimization, LLC, 2024b). As the solver navigates the MIP search space, it identifies not only the proven optimal solution but also alternative next-best feasible solutions, which can be systematically retrieved and utilised. Specifically, the resampling is focused around multiple solutions from $\mathcal{S}_{\text{pool}}$ and points with high uncertainty in the surrogate model's predictions, represented as the set $\mathcal{U}$. Together, these points form our *centre points* $C$ that guide the resampling.

Within this framework, resampling around high-uncertainty points $\mathcal{U}$ is aligned with exploration, as it focuses on regions with greater uncertainty to identify potential new optima. Conversely, resampling around $\hat{x}^*$ and other solutions from the solution pool $\mathcal{S}_{\text{pool}}$ corresponds to exploitation, as it seeks to improve the quality of the current best solution by gathering more data in promising regions.

The resampling procedure is outlined in Algorithm 2. To implement the resampling, the sampling radius and the number of samples are adjusted based on the available computational resources, controlled by the parameters $\alpha$ and $\beta$, ensuring that the search space is effectively covered.

A sampling radius $r$, defined around each centre point in the set $C$ (Line 12), is determined using the scaling factor $\alpha$ and the difference between the lower ($\underline{x}$) and upper ($\bar{x}$) bounds of current sampling parameterisation $\xi$:

$$r = \frac{(\bar{x} - \underline{x}) \cdot \alpha}{2}. \tag{6}$$

The algorithm then adjusts the bounds for each centre point $x^{(i)}$ in $C$ (Line 15), ensuring that new samples are generated within an appropriate range:

$$\underline{x}^{(i)} = \max(\underline{x}, x^{(i)} - r^{(i)}), \quad \bar{x}^{(i)} = \min(x^{(i)} + r^{(i)}, \bar{x}). \tag{7}$$

**Algorithm 2** Resampling Procedure

```
 1: function Resample($\mathcal{X}_s, C, \xi, \alpha, \beta$)
 2:     inputs:
 3:         $\mathcal{X}_s$: Current sample set
 4:         $C$: Centre points
 5:         $\xi = (n_s, \underline{x}, \bar{x})$: Current sampling parameterisation
 6:         $\alpha, \beta$: Scaling factors for radius and number of samples
 7:     outputs:
 8:         $\mathcal{X}_s$: Enriched sample set
 9:         $\mathcal{F}_s$: Corresponding simulator values
10:         $\xi = (n_s, \underline{x}, \bar{x})$: Updated sampling parameterisation
11:     Initialise $\mathcal{X}_{\text{new}} \leftarrow \{\}$                ▷ Store new samples
12:     $r = \frac{(\underline{x} - \bar{x}) \cdot \alpha}{2}$                ▷ Calculate the radius
13:     $n_s = \frac{n_s \cdot \beta}{|C|}$                ▷ Adjust sampling number
14:     for each $x^{(i)}$ in $C$ do
15:         $\underline{x}^{(i)} = \max(\underline{x}, x^{(i)} - r)$, $\bar{x}^{(i)} = \min(x^{(i)} + r, \bar{x})$  ▷ Set new bounds around $x^{(i)}$
16:         $\xi^{(i)} \leftarrow (n_s, \underline{x}^{(i)}, \bar{x}^{(i)})$      ▷ Resampling parameterisation for $x^{(i)}$
17:         $\mathcal{X}^{(i)} \leftarrow$ Sample($\xi^{(i)}$)                ▷ Generate new samples
18:         Append $\mathcal{X}^{(i)}$ to $\mathcal{X}_{\text{new}}$
19:         $\underline{x} \leftarrow \min(\underline{x}, \underline{x}^{(i)})$, $\bar{x} \leftarrow \max(\bar{x}, \bar{x}^{(i)})$      ▷ Update global bounds
20:     end for
21:     $\xi \leftarrow (|\mathcal{X}_{\text{new}}|, \underline{x}, \bar{x})$          ▷ Updated sampling parameterisation
22:     $(\mathcal{X}_s, \mathcal{F}_s) \leftarrow \{(x, \text{Evaluate}(x)) \mid x \in \mathcal{X}_s \cup \mathcal{X}_{\text{new}}, \underline{x} \leq x \leq \bar{x}\}$ ▷ Enrich sample set
23:     return $(\mathcal{X}_s, \mathcal{F}_s, \xi)$
24: end function
```

Using these updated bounds, the resampling parameterisation $\xi^{(i)} = (n_s, \underline{x}^{(i)}, \bar{x}^{(i)})$ for each centre point $x^{(i)}$ is defined (Line 16), where $n_s$ represents the number of samples allocated to each region, scaled by the factor $\beta$ and divided by the cardinality of the set $C$:

$$n_s \leftarrow \frac{n_s \cdot \beta}{|C|}. \tag{8}$$

New samples $\mathcal{X}^{(i)}$ are then generated around each centre point $x^{(i)}$ using the updated parameterisation through Sample($\xi^{(i)}$) (Line 17) and then appended to the entire set $\mathcal{X}_{\text{new}}$, which stores the resampled data around all centre points. When relying on a pre-acquired simulation data points (e.g., when querying the simulator in real-time is not feasile), up to $n_s$ points are randomly selected within the defined sample area around each centre point. If fewer points exist within that area than $n_s$, all of them are selected to enrich the sample set.

After generating the new samples $\mathcal{X}_{\text{new}}$, the overall sampling parameterisation $\xi$ is updated to reflect the total number of samples and the new combined bounds (Line 21). Instead of merging the entire existing dataset $\mathcal{X}_s$ with the newly sampled data $\mathcal{X}_{\text{new}}$, only the union of the new samples and the previously sampled data within the new bounds $[\underline{x}, \bar{x}]$ is merged (Line 22). The corresponding simulator values $\mathcal{F}_s$ are obtained using Evaluate($\cdot$). This ensures that only the relevant data, which lies within the updated bounds, is used to refine the surrogate model.

By prioritising regions around the centre points during the resampling procedure, the optimisation process can systematically reduce uncertainty across the design space, leading to more reliable and accurate outcomes.

## 4. Iterative optimisation framework

Building upon the infill strategy presented in Section 3.2, we describe our complete framework for simulator-based optimisation. The framework integrates efficient sampling techniques, ReLU NNs, and MIP to handle the challenges of high-fidelity simulations. We detail the iterative processes along with strategies to streamline the computational workload.

### *4.1. Framework overview*

For conciseness, the outcome of the surrogate model for the simulator, denoted as $y$, is directly taken as the objective in the following description of the proposed framework, i.e., setting $h(x, y) = y$ in problem (1). All other settings and assumptions of the original optimisation problem (1) remain unchanged. In practice, a custom objective function involving the outcome of interest from the simulator can be defined without affecting the implementation of the method.

The general framework for optimising complex simulations is illustrated in Fig. 2. The framework initiates with an efficient sampling process using low-discrepancy sequences (quasirandom sequences), which are deterministically constructed to achieve superior uniformity in multi-dimensional spaces compared to pseudorandom samples (Kucherenko and Sytsko, 2005). We consider two scenarios: (i) having access to the simulator for active resampling; or (ii) being limited to pre-acquired datasets for data selection. If evaluating the simulator is too expensive, low-discrepancy sample points can be pregenerated to create a static dataset for future data selection. While several low-discrepancy sequences exist, we employ Sobol sequences due to their demonstrated superiority in numerous practical applications (L'Ecuyer and Lemieux, 2002). These sequences generate successive points by considering binary fractions and systematically constructing primitive polynomials, which reduces the number of required evaluations and enhances the representativeness of the sample points (Renardy et al., 2021).

The process of obtaining data points can be parallelised to enhance efficiency, particularly in large-scale simulations. Multiple sample points can be evaluated simultaneously by distributing simulation runs across various computational nodes or processors, enabling the concurrent execution of simulations (Ndih and Cherkaoui, 2015). Techniques such as multithreading or distributed computing can be employed to implement parallel processing, significantly accelerating the sampling phase.

Once sampling is completed, the next phase involves building a surrogate model using the obtained dataset. In this framework, a ReLU NN is trained to develop the surrogate model. The choice of NN architecture is crucial, as it balances computational efficiency with the ability to approximate nonlinear relationships within the data. The trained surrogate is then embedded into an MIP problem, as discussed in Section 2. By solving the MIP formulation, we can determine an optimal parameterisation of the simulator's inputs based on the initial surrogate, leveraging the exact optimisation capabilities of MIP methods. It is important to achieve a reasonably accurate surrogate from the start, as a low-accuracy surrogate model embedded in the MIP may lead to sub-optimal solutions in the initial stage, potentially steering the subsequent optimisation towards a misleading direction.

The framework's core strength lies in its iterative refinement process driven by a resampling infill strategy as discussed in Section 3. This iterative process forms a loop where resampling, ReLU NN surrogate model retraining, MIP reformulation, and MIP resolving are repeated until the termination conditions are met. Each iteration enriches the model's dataset, making the surrogate progressively more reliable, while the use of MIP ensures that the optimisation leverages the most accurate model available at each step.

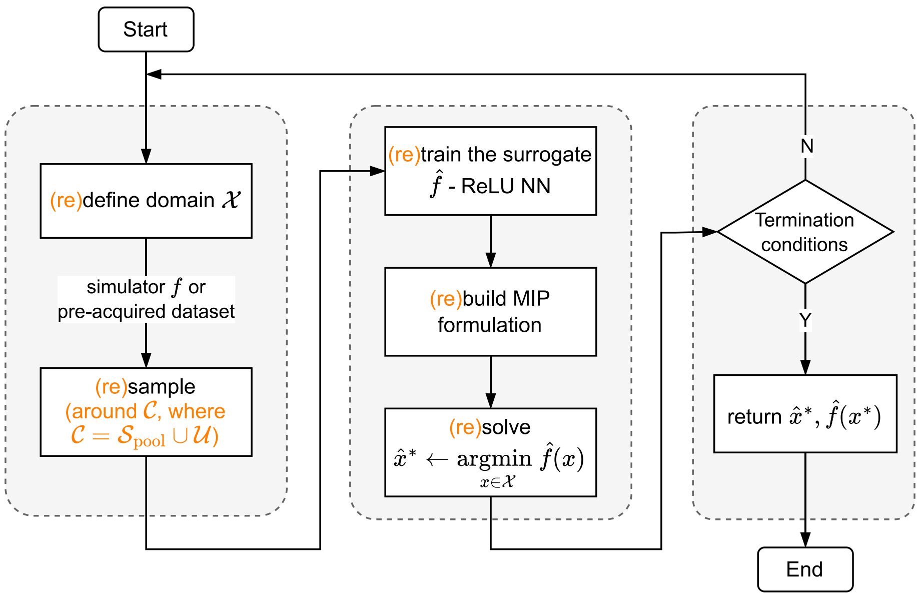
Start
(re)define domain $\mathcal{X}$
simulator $f$ or pre-acquired dataset
(re)sample (around $\mathcal{C}$, where $\mathcal{C} = \mathcal{S}_{\text{pool}} \cup \mathcal{U}$)
(re)train the surrogate $\hat{f}$ - ReLU NN
(re)build MIP formulation
(re)solve $\hat{x}^* \leftarrow \underset{x\in\mathcal{X}}{\text{argmin}}\ \hat{f}(x)$
N
Termination conditions
Y
return $\hat{x}^*$, $\hat{f}(x^*)$
End

**Fig. 2.** Flowchart of the optimisation process, outlining key steps: beginning with sampling, followed by training the ReLU NN surrogate model, formulating and solving the MIP (middle), and iteratively (right) optimising the simulator inputs.

### *4.2. Algorithm implementation*

The implementation of the simulator-based surrogate optimisation framework is outlined in Algorithm 3, focusing on the specific procedural steps and iterative mechanisms. Unlike the broader conceptual discussion in Section 3.1, this section emphasises the algorithm's technical elements, including initialisation, iterative resampling, model retraining, and convergence criteria.

The algorithm starts with defining the necessary input parameters. These include the initial sampling configuration $\xi$ which sets the number of samples $n_s$ and the bounds $[\underline{x}, \bar{x}]$ within which the algorithm will explore. The ReLU NN parameters $\theta$ specify the architecture and training specifics, ensuring the surrogate model is appropriately configured to approximate the simulator's outputs. Other critical inputs involve the Monte Carlo Dropout rate $p$ for uncertainty estimation, the number of Monte Carlo samples $M$, and parameters for adaptive resampling, including the number of high-uncertainty points to target ($N$) and the scaling factors $(\alpha, \beta)$ used to dynamically adjust the sampling density.

The algorithm proceeds with an initialisation phase where initial samples $\mathcal{X}_s$ are drawn within the defined bounds using $\textsc{Sample}(\xi)$ (Line 14) and evaluated using either the high-fidelity simulator or a pre-existing dataset with $\textsc{Evaluate}(\mathcal{X}_s)$ (Line 15). The dataset $(\mathcal{X}_s, \mathcal{F}_s)$ is then used to train the initial surrogate model $\hat{f}$, a ReLU NN, which approximates the simulator's behaviour using $\textsc{TrainNN}(\mathcal{X}_s, \mathcal{F}_s, \theta)$ (Line 16). This surrogate model is then embedded into the MIP problem $\mathcal{M}_{\text{MIP}}$, allowing the algorithm to build a solution pool $\mathcal{S}_{\text{pool}}$ by solving it using $\textsc{Solve}(\mathcal{M}_{\text{MIP}})$ (Line 18) and identify an initial optimal solution $\hat{x}^*$.

After building the initial solution pool, the algorithm estimates the uncertainty in the surrogate model's predictions using Monte Carlo Dropout, implemented as $\textsc{MonteCarloDropout}(\hat{f}, \mathcal{X}_s, M, p)$ (Line 23). The algorithm then selects points from $\mathcal{U}$ and $\mathcal{S}_{\text{pool}}$, forming our centre points $C$ for resampling (Line 25). This focused resampling enriches the dataset by adding new, informative points using $\textsc{Resample}(\mathcal{X}_s, C, \xi, \alpha, \beta)$ (Line 26). The enriched dataset is used to retrain the NN. The updated surrogate model is then reformulated as a new MIP problem, and the optimisation process is repeated to identify a potentially improved solution using $\textsc{Solve}(\mathcal{M}_{\text{MIP}})$ (Line 28–29).

The looped process continues until the algorithm meets the convergence criteria, which are based on either the relative improvement in the objective function or the maximum number of iterations (Line 22). Convergence is assessed by comparing the predicted optimum $\hat{f}^*$ with a reference value $f_{\text{ref}}$. For cases with access to the simulator, $f_{\text{ref}}$ is the actual simulated value obtained through $\textsc{Assess}(\hat{x}^*)$ at the current best point $\hat{x}^*$ (Line 30). For pre-acquired datasets, $f_{\text{ref}}$ is the predicted optimum from the previous iteration, obtained similarly through $\textsc{Assess}(\hat{x}^*)$. Convergence is reached when the relative difference between $\hat{f}^*$ and $f_{\text{ref}}$ falls below a predefined tolerance, $\tau$, indicating that further iterations are unlikely to significantly enhance the solution. This condition suggests that the optimisation has plateaued, and the algorithm is terminated.

### *4.3. Memory structure reuse*

The computational efficiency of the iterative process, described in Algorithm 3, is enhanced through the reuse of memory structures. A crucial aspect of this strategy is freezing the initial layers of the NN during the retraining phase. By maintaining the weights of the first few layers, the algorithm retains the features and representations learned from earlier iterations, while adapting to the new data. These layers typically capture fundamental patterns and structures in the data, which remain relevant as new data is introduced through resampling (Yosinski et al., 2014). This reuse of memory not only reduces the computational cost associated with retraining the entire network but also stabilises the training process while incorporating the latest data for improved accuracy.

In addition to NN layer freezing, the reuse of the existing MIP model plays a necessary role in streamlining the optimisation process. Once the MIP formulation has been established for the initial surrogate model, it can be efficiently adapted for subsequent iterations. This reuse minimises the overhead associated with rebuilding the MIP model from scratch in each iteration. Instead, the existing structure is modified to incorporate the updated surrogate model, allowing the algorithm to focus computational efforts on solving the optimisation problem rather than repeatedly reformulating it.

These memory reuse techniques, freezing NN layers and reusing the MIP model, are integral to enhancing the computational feasibility of the optimisation, especially in complex scenarios where high-dimensional data and large-scale simulations are involved. In the broader context of the surrogate-based optimisation framework,

**Algorithm 3** Simulator-based Surrogate Optimisation

```
1: inputs:
2:     $\xi = (n_s, \underline{x}, \bar{x})$: Initial sampling parameterisation (number of samples, lower and upper bounds)
3:     $\theta$: ReLU NN parameterisation (layers' architecture, loss function, optimiser, dropout rate, batch size and number of epochs)
4:     $p$: Dropout rate for Monte Carlo Dropout
5:     $M$: Number of Monte Carlo Dropout samples
6:     $N$: Number of points with the highest uncertainty to identify
7:     $\alpha, \beta$: scaling factors for number of samples and radius
8:     $k_{\max}$: Maximum number of iterations
9:     $\tau$: Convergence tolerance
10: outputs:
11:     $\hat{x}^*$: Best point identified
12:     $\hat{f}^*$: Corresponding surrogate outputs
13: initialise:
14:     $\mathcal{X}_s \leftarrow$ SAMPLE($\xi$)    ▷ Sample initial points within bounds
15:     $\mathcal{F}_s \leftarrow$ EVALUATE($\mathcal{X}_s$)    ▷ Evaluate function using simulator or referring to pre-acquired dataset
16:     $\hat{f} \leftarrow$ TRAINNN($\mathcal{X}_s, \mathcal{F}_s, \theta$)    ▷ Train a ReLU NN and store the trained model
17:     Formulate $\mathcal{M}_{\text{MIP}}(\hat{f}, \bar{x}, \underline{x})$    ▷ Covert ReLU NN to MIP
18:     $\mathcal{S}_{\text{pool}} \leftarrow$ SOLVE($\mathcal{M}_{\text{MIP}}$)    ▷ Solve MIP and build solution pool
19:     $\hat{x}^* \leftarrow$ best point in $\mathcal{S}_{\text{pool}}$, $\hat{f}^* = \hat{f}(\hat{x}^*)$
20:     $f_{\text{ref}} \leftarrow$ ASSESS($\hat{x}^*$)    ▷ Reference value for convergence check
21:     $k \leftarrow 0$
22: while $k < k_{\max}$ and $\frac{|\hat{f}^* - f_{\text{ref}}|}{|f_{\text{ref}}|} > \tau$ do    ▷ Convergence criteria
23:     $\Sigma$ = MONTECARLODROPOUT($\hat{f}, \mathcal{X}_s, M, p$)    ▷ Estimate uncertainty of each point
24:     $\mathcal{U} \leftarrow$ top $N$ points with highest uncertainty from $\mathcal{X}_s$ based on $\Sigma$
25:     $\mathcal{C} \leftarrow \mathcal{U} \cup \mathcal{S}_{\text{pool}}$    ▷ Identify centre points
26:     $(\mathcal{X}_s, \mathcal{F}_s, \xi)$ = RESAMPLE($\mathcal{X}_s, \mathcal{C}, \xi, \alpha, \beta$)    ▷ Enrich sample set
27:     Formulate $\mathcal{M}_{\text{MIP}}(\hat{f}, \bar{x}, \underline{x})$    ▷ Reformulate MIP with updated model
28:     $\mathcal{S}_{\text{pool}} \leftarrow$ SOLVE($\mathcal{M}_{\text{MIP}}$)    ▷ Solve MIP and update solution pool
29:     $\hat{x}^* \leftarrow$ best point in $\mathcal{S}_{\text{pool}}$, $\hat{f}^* = \hat{f}(\hat{x}^*)$
30:     $f_{\text{ref}} \leftarrow$ ASSESS($\hat{x}^*$)    ▷ Update reference value for next iteration
31:     $k = k + 1$    ▷ Increment iteration counter
32: end while
33: return $(\hat{x}^*, \hat{f}^*)$
```

these strategies ensure that each iteration builds upon the progress made in previous steps, accelerating convergence while maintaining the robustness of the optimisation process.

## 5. Numerical experiments

This section presents numerical experiments to assess the effectiveness of the proposed framework through two real-world simulator-based engineering applications: a jet engine turbine blade design problem (The MathWorks, Inc., 2024b) with an accessible simulator and an auto-thermal reformer process optimisation (Miller et al., 2018) using pre-acquired simulator data. In addition, we provide further numerical validation concerning our method's global optimisation capabilities through the benchmark Rastrigin function (Surjanovic and Bingham, 2013), presented in the Supplementary Materials, Section 1.

### *5.1. Setup and tools*

All experiments were conducted on a computing platform equipped with an Intel Core i5-1145G7 processor running at 2.60 GHz, paired with 32 GB of RAM. The optimisation and modelling tasks were implemented in Julia 1.10.3 (Bezanson et al., 2017), utilising Gurobi 11.0.2 solver (Gurobi Optimization, LLC, 2024a) for solving MIP problems. For supervised learning tasks, Flux 0.14.15 library (Innes, 2018) was employed to train the ReLU NN surrogate.

The MIP models were formulated using Gogeta.jl package (Reijonen et al., 2024), a Julia package designed to represent machine learning models within a mathematical programming framework. In our experiments, we utilised the package's key features of bound tightening and lossless model compression to improve the efficiency of the optimisation process, as introduced in Section 2.3.

Specifically, we employed feasibility-based bound tightening (Belotti et al., 2012), leveraging interval arithmetic, which is computationally efficient. This technique computes bounds on constraint activations over the variable domains (forward propagation) (Gleixner et al., 2017), considering the variables and constraints defined up to the previous layer of the neuron undergoing bound tightening. Additionally, lossless model compression was used to simplify the NN surrogate, reducing the complexity of the corresponding MIP formulation without losing crucial information. This was achieved by pruning stably active or inactive neurons, adjusting the weights and biases accordingly, and reconstructing the network with the remaining neurons while preserving local equivalence (Serra et al., 2020).

For the visual analysis of unstructured data, the Makie.jl (Danisch and Krumbiegel, 2021) data visualisation ecosystem was utilised. Its tricontour function, which employs Delaunay triangulation (Ito, 2015), enables continuous surface representations through linear interpolation of scattered data points, facilitating interpretable visualisation of the optimisation outcomes.

### *5.2. Case 1: Having access to the simulator*

In this experiment, we apply the proposed optimisation framework to the design of a jet engine turbine blade. In a jet engine, the turbine converts energy from high-temperature exhaust gases into mechanical work (Gowreesh et al., 2012). However, the turbine blades must endure extremely harsh thermal conditions. Controlling the blade's maximum temperature is critical to ensuring the reliability and longevity of the engine during the turbine blade design process.

To tackle this challenge, a thermal analysis based on Finite Element Analysis (FEA) using the Partial Differential Equation Toolbox (The MathWorks, Inc., 2024a) of MATLAB is conducted. FEA models the turbine blade geometry by discretising it into smaller elements, allowing for the simulation and analysis of temperature distributions under different thermal loads. Fig. 3 presents both the structural model of the turbine blade and the corresponding temperature distribution predicted by the FEA. The structural model illustrates the physical geometry of the blade, while the FEA predicts how the blade responds to thermal loads under different operating conditions.

The simulation is treated as a black box, focusing solely on the relationship between the six input variables and the output. The inputs include the cooling air temperature $T_{\text{air}}$, gas temperature $T_{\text{gas}}$, and heat transfer coefficients for different regions of the blade. Specifically, the heat transfer coefficients correspond to various faces of the structural model shown on the left-hand side of Fig. 3. These include coefficients for interior cooling (covering faces 15, 12, and 14), as well as for the pressure side (face 11), the suction side (face 10), and the tip (face 13) of the blade.

The objective of the optimisation is to minimise the maximum temperature that the blade will experience under various conditions. The optimisation framework explores the design space within defined bounds for these six input variables. The initial bounds are set at [120, 900, 20, 40, 30, 10] for the lower bounds and [180, 1200, 40, 60, 50, 30] for the upper bounds (Guo, 2021), ensuring a comprehensive range of operating conditions is covered.

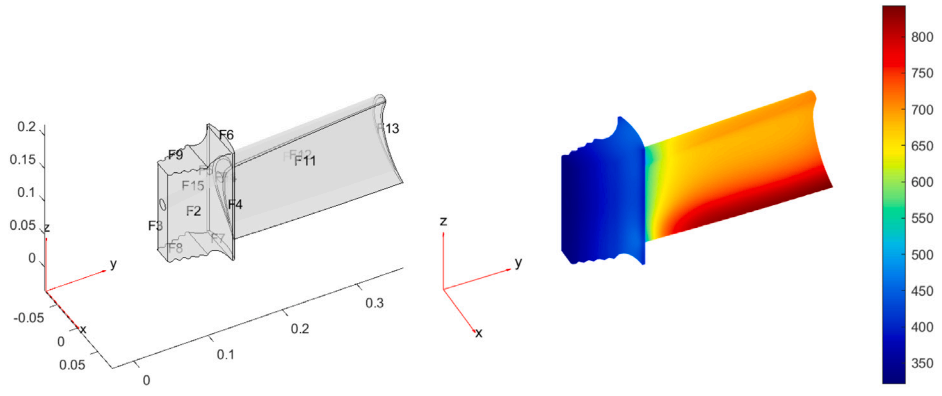

**Fig. 3.** The structural model used in this experiment (left) and a temperature (°C) distribution example after Finite Element Analysis (FEA) (right).

**Table 1**
Iterative optimisation results: Number of resampled points, total dataset size, and Mean Absolute Percentage Error (MAPE) for training and test sets, along with the relative optimisation gap.

| Iteration step | Resampled points | Training MAPE (%) | Test MAPE (%) | Relative gap (%) |
|---|---|---|---|---|
| Initial | – | 0.445 | 0.427 | 1.842 |
| 1 | 300 | 0.357 | 0.368 | 0.606 |
| 2 | 96 | 0.288 | 0.299 | 0.387 |
| 3 | 33 | 0.196 | 0.197 | 0.025 |

The surrogate model was initially trained using 1000 data points, where the input values were generated using a Sobol sequence within the defined bounds and the corresponding outputs were obtained by evaluating these inputs with the simulator. The ReLU NN used for the surrogate model had an architecture of $6 \rightarrow 50 \rightarrow 50 \rightarrow 1$, and it was trained using the *Adam* optimiser (Kingma and Ba, 2017) with a learning rate schedule of exponential decay. The Mean Squared Error (MSE) was used as the loss function. During retraining, the first hidden layer was intentionally frozen.

To estimate uncertainty, we utilised Monte Carlo Dropout with the following parameters: $M = 100$ for the number of Monte Carlo samples, and a dropout rate of $p = 0.1$. During each iteration, the framework identified $N = 10$ points with the highest standard deviation among the samples. The resampling was controlled by scaling factors set as $\alpha = 0.10$ for the radius and $\beta = 0.3$ for the number of samples, determining the scope and density of the additional data points sampled during each iteration.

After three iterations of resampling, surrogate retraining, and MIP reformulating and resolving, the algorithm converged, meeting the termination condition of $\tau = 0.1\%$. Table 1 summaries the iterative optimisation process, detailing the number of resampled points, the total dataset size used for the ReLU NN training, and the Mean Absolute Percentage Error (MAPE) for both the training and test sets at each step. The table also includes the relative gap, which is the difference between the predicted optimum $\hat{f}^*$ and the actual simulated value $f^*$ at the current best point, serving as the termination condition for the optimisation process.

The iterative optimisation process shows a significant reduction in both the training and test MAPE as the optimisation progresses, indicating that the surrogate model becomes increasingly accurate with each iteration. The number of resampled points decreases with each step, reflecting the framework's ability to focus on the most informative regions of the design space. The relative gap also narrows, indicating the model's convergence towards an optimal solution.

For the sake of illustration, we selected the first two of the six input variables, $x_1$ and $x_2$, to further illustrate the optimisation process. Fig. 4 highlights the initial step's optimum and the 10 highest-uncertainty points within the sample set. Fig. 5 shows the resampled points during the 3rd iteration, emphasising how the framework hones in on centre points to improve model accuracy. Fig. 6 presents the final optimisation results after the 3rd iteration, where the optimisation process converged, meeting the termination condition.

The framework effectively handled this complex problem through iterative surrogate model refinement and optimisation. Starting with a surrogate model trained on 1000 Sobol-sampled data points from the black-box simulator, the framework iteratively refined the model. The convergence was visually confirmed in the final iteration (Fig. 6), where the contours of the surrogate model closely matched those of the simulator, and the optimal solution was accurately pinpointed.

### 5.3. Case 2: Being limited to pre-acquired datasets

In this case, the optimisation framework operates under the constraint of relying solely on pre-acquired datasets, without the ability to interact with a simulator in real-time. This scenario is common in industrial and engineering contexts where generating new simulator data is costly or impractical. The framework's performance is evaluated using an optimisation problem involving an auto-thermal reformer process, a complex example from process engineering.

The reformer process generates synthesis gas (syngas) from air, steam, and natural gas (NG) inputs, as depicted in Fig. 7, which is subsequently used in a solid-oxide fuel cell. The process involves 12 outputs of interest, including the steam flow rate, the reformer duty, and the properties of the outlet stream. The two operating (input) variables are the fraction of NG that bypasses the reformer, $x_1$, ranging from $[0.1, 0.8]$, and the steam to NG flow ratio, $x_2$, ranging from $[0.8, 1.2]$.

The optimisation setup follows the OMLT (Optimisation and Machine Learning Toolkit) (Ceccon et al., 2022) implementation, which is a toolkit designed to integrate machine learning models with optimisation frameworks. In this case, the constraints ensure that the nitrogen ($N_2$) concentration in the product stream remains below 34 mol%. The objective function to be maximised is hydrogen ($H_2$) production.

In the implementation of OMLT, the surrogate model is trained using 2800 static simulated data points generated via a grid sampling method. The ReLU NN, consisting of layers sized $2 \rightarrow 10 \rightarrow 10 \rightarrow 10 \rightarrow 10 \rightarrow 12$ neurons, is optimised using the *Adam* optimiser with MSE as the loss function. OMLT supports the integration of the NN within the optimisation problem by offering various formulations for machine learning models, including the MIP approach. The ReLU NN

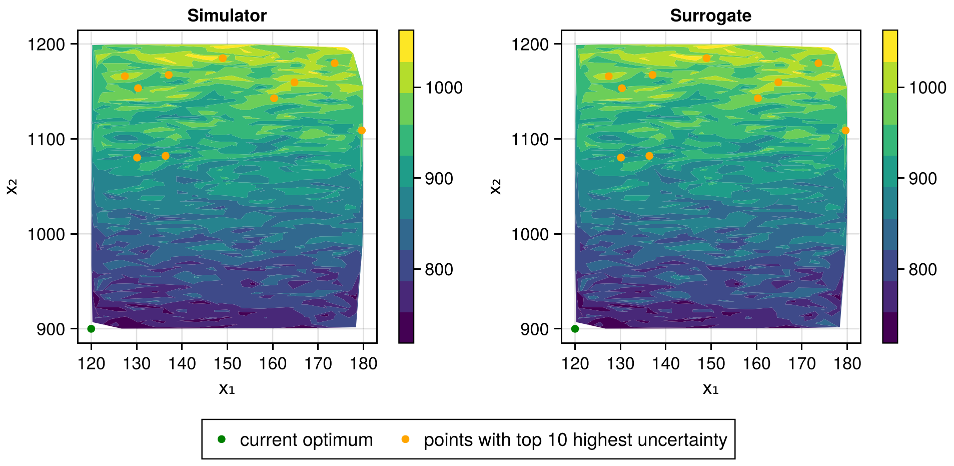

**Fig. 4.** Contours of the simulator and surrogate model, highlighting the current optimal solution point and 10 highest-uncertainty points.

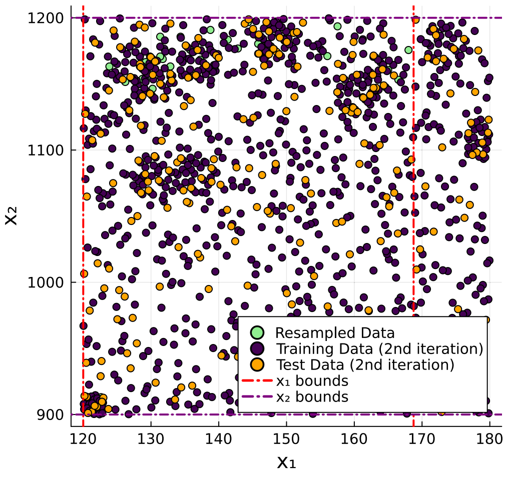

**Fig. 5.** Data points used in the 2nd iteration and newly resampled data points from the 3rd iteration, focusing on regions around $C$.

surrogate is incorporated into the optimisation problem using the big-M method, and the problem is subsequently solved using an MIP solver to determine the optimal operating conditions that satisfy the specified constraints.

A key distinction between our method and the OMLT implementation is that their optimisation is performed on the initially trained surrogate model, using the full pre-acquired dataset without any further updates or resampling. In contrast, our proposed framework focuses on refining the sampling strategy to make more efficient use of the dataset, minimising the need for additional data points. We use the OMLT implementation as a baseline for evaluating our algorithm, applying the same simulated dataset and NN parameters as in the OMLT example. While our algorithm is implemented in Julia and OMLT in Python, we emphasise that our comparative analysis focuses on the relative magnitude of performance differences rather than exact numerical equivalencies. This comparison helps demonstrate how our framework handles optimisation with finite and fixed data, underscoring the effectiveness of our approach in surrogate modelling and data selection under constrained conditions.

In the initial sampling phase, 1000 data points were randomly selected from the entire 2800-point dataset. To estimate uncertainty, Monte Carlo Dropout was applied with a dropout rate of $p = 0.2$, generating $M = 100$ predictions for each data point. As shown in Fig. 8, the highest uncertainty points were primarily located in the upper right corner of the plot, highlighting regions where the model exhibited lower confidence. Conversely, the model showed greater confidence in predictions at the initial solution point. To balance the need for precision improvement in the most uncertain regions while avoiding unnecessary computational overhead, the process concentrated on the top 10 points with the highest uncertainty rather than considering a larger candidate set with, i.e., 50 points.

Points are then randomly selected from the remaining pre-acquired dataset within the defined sample area surrounding the points from $\mathcal{S}_{\text{pool}}$ and $\mathcal{U}$ identified earlier. The bounds for the variables were updated accordingly to focus the search in these critical regions. Fig. 9 shows the original and resampled data points during the first iteration, with new samples concentrated around $C$. As before, the first layer of the NN was frozen during retraining to preserve the features learned in the previous training phase. An analysis of the model's performance with and without layer freezing is presented in the Supplementary Materials, Section 2.

The iterative process of resampling, retraining the NN, reformulating the MIP, and resolving the optimisation problem continued until convergence (tolerance $\tau = 0.1\%$) was achieved after four iterations. During each iteration, the data was normalised to enhance training performance by ensuring that the input features are on a comparable scale, which helps in faster convergence. Fig. 10 presents the contours of both the simulator and the surrogate model, along with the solution points from $\mathcal{S}_{\text{pool}}$ following the initial and final (4th) iterations. These figures visually confirm the model's improved accuracy and the progressive refinement of the optimal solution region as the iterations advanced.

The iterative optimisation results are summarised in Table 2, highlighting the number of resampled points, training and test MSEs, and the relative optimisation gap. As the iterations progressed, both the training and test MSEs decreased significantly, indicating a marked improvement in the model's accuracy. Additionally, the relative gap narrowed progressively, demonstrating the framework's effectiveness in efficiently converging towards an optimal solution.

The performance comparison between our proposed algorithm and the OMLT implementation is presented in Table 3. We provide two variants of our method: one with ("Our Alg.") and one without ("Our Alg. reduced") considering uncertainty quantification. By presenting this reduced variant, our objective is to measure the importance of uncertainty quantification and infill sampling in the performance of our approach. Although our full algorithm utilised fewer than half the data points compared to OMLT, leading to a significant reduction in training time, it achieved comparable final loss values, underscoring the effectiveness of our method in maintaining high accuracy while being resource-efficient. In contrast, the reduced variant, despite using a similar number of data points to our full algorithm, required three times more iterations (12 versus 4) to achieve convergence and resulted in a substantially higher loss value. This performance degradation

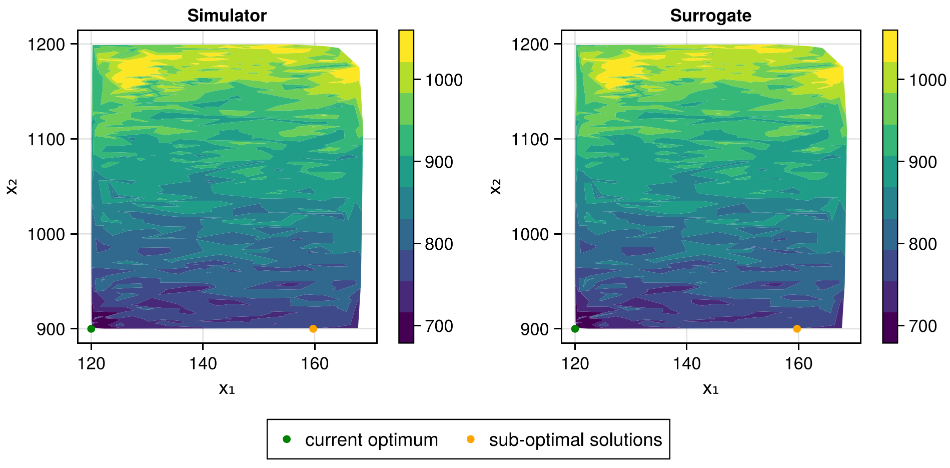

**Fig. 6.** Final optimisation results after the 3rd iteration, showing simulator and surrogate model contours with the optimal solution point and another solution stored in $S_{pool}$ (which is less optimal). The optimisation process converged after meeting the termination condition $\tau = 0.1\%$.

**Table 2**
Summary of iterative optimisation results, showing the number of resampled points, training and test MSE, and the relative optimisation gap across iterations.

| Iteration step | Resampled points | Training MSE | Test MSE | Relative gap (%) |
|---|---|---|---|---|
| Initial | – | 0.503186 | 0.444885 | – |
| 1 | 73 | 0.000930 | 0.001071 | 2.03 |
| 2 | 68 | 0.000463 | 0.000375 | 0.79 |
| 3 | 10 | 0.000438 | 0.000325 | 0.41 |
| 4 | 7 | 0.000184 | 0.000351 | 0.04 |

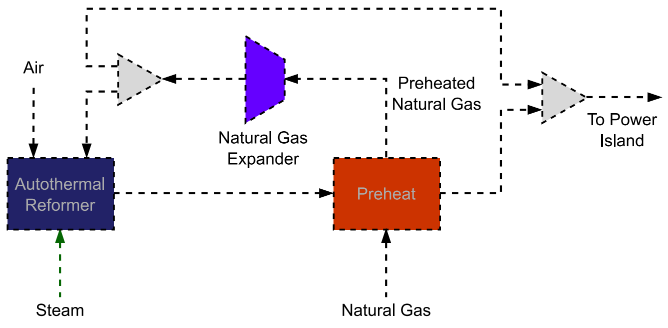

**Fig. 7.** The auto-thermal reformer flowsheet as modelled in Turcani and Sadler (2024).

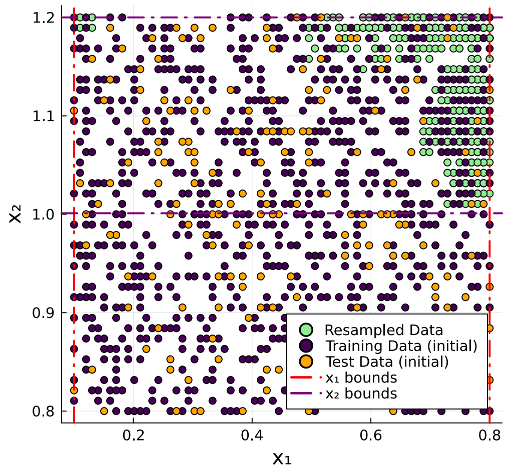

**Fig. 9.** Original and resampled data points during the 1st iteration, highlighting the new samples added around $C$.

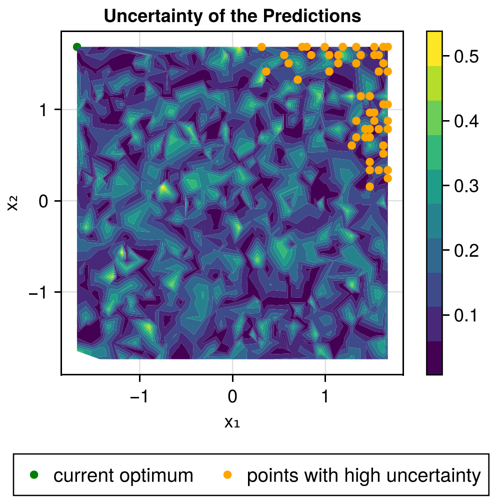

**Fig. 8.** Initial uncertainty estimation in predictions, with current optimum and 50 highest-uncertainty points highlighted.

not only indicates the crucial role of uncertainty quantification in solution quality but also suggests that relying solely on solution pool information yields less enriched resampling strategies. Notably, our approach reduced the total execution time by an order of magnitude compared to OMLT, primarily in the training phase. Though the reduced variant achieved faster resampling, its poor solution quality makes it impractical for reliable applications.

The final optimisation solutions obtained from both our algorithm and the OMLT implementation are compared in Table 4. Both methods produced nearly identical outcomes for the operating variables and the output variables of interest. Importantly, both approaches successfully maintained the $N_2$ concentration below 34 mol% while maximising $H_2$ production, confirming that our method is effective in achieving the key optimisation objectives.

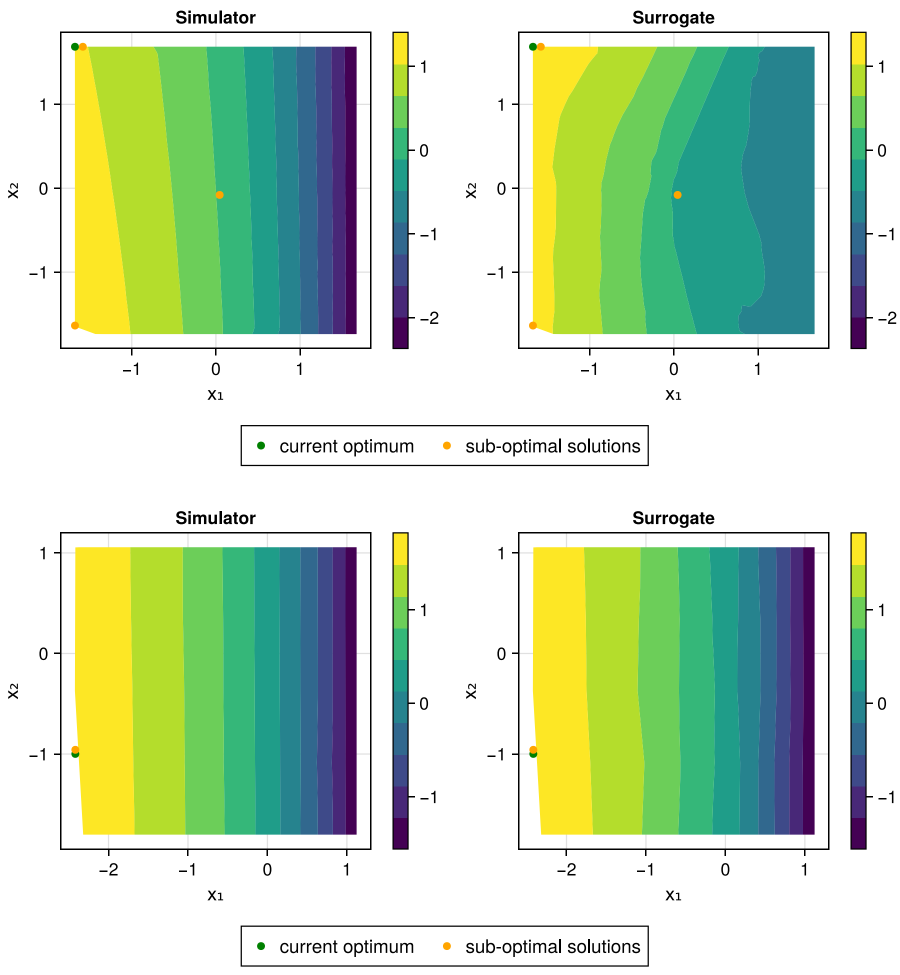

**Fig. 10.** Normalised simulator and surrogate model contours with solution points from $\mathcal{S}_{\text{pool}}$ after the initial iteration (top) and the final 4th iteration (bottom). These figures illustrate the convergence of the optimisation process and the progressive refinement of the surrogate model.

**Table 3**
Comparative performance between our algorithm and the OMLT implementation: data usage, final loss, and computational time.

| Method | Data used | Final loss | Time (s)[a] | | | | |
|---|---|---|---|---|---|---|---|
| | | | Uncertainty estimation | Resampling | Training | MIP solving | Total |
| Our Alg. | 1158 | 0.000184 | 1.32 | 0.52 | 1.08 | 0.73 | 3.65 |
| Our Alg. reduced | 1106 | 0.591645 | – | 0.04 | 3.78 | 1.26 | 5.07 |
| OMLT[b] | 2800 | 0.000169 | – | – | 18.3 | 0.77 | 19.07 |

[a] All reported times represent the sum of execution times across iterations; Our Alg. converged in 4 iterations while the reduced variant required 12 iterations.

[b] Coded in Python 3.10.13 (Pyomo 6.7.1, TensorFlow 2.16.1, Keras 3.3.3); solved with Gurobi 11.0.2.

**Table 4**
Comparison of optimisation solutions between our algorithm and the OMLT implementation, showing the key variables.

| Method | Operating variables | | Output variables of interest | |
|---|---|---|---|---|
| | $x_1$: Bypass fraction | $x_2$: NG steam ratio | $N_2$ Concentration[a] | $H_2$ Concentration[b] |
| Our Alg. | 0.1 | 1.174901 | 0.34 | 0.333138 |
| OMLT | 0.1 | 1.132608 | 0.34 | 0.332007 |

[a] The $N_2$ concentration is constrained below 34 mol%.

[b] The objective is to maximise $H_2$ production.

This experiment demonstrates the effectiveness of our proposed optimisation framework when operating with pre-acquired datasets. The comparison with the OMLT implementation highlights the framework's capability to deliver comparable performance metrics with less computational effort and fewer resources.

## 6. Conclusions

This paper introduces a framework that effectively connects simulators with optimisation processes, representing an advancement in surrogate-based optimisation across various engineering domains.

Specifically, the framework is best suited to optimisation problems involving deterministic simulation models where gradients are computationally expensive or unavailable.

The iterative nature of the framework, which combines adaptive resampling, uncertainty-informed retraining, and MIP reformulation, not only optimises data usage but also significantly reduces the computational burden associated with complex simulations. Furthermore, by incorporating techniques like bound tightening, lossless model compression, and memory structure reuse, the framework is equipped to handle large-scale, high-dimensional problems with increased efficiency.

Future work should focus on addressing challenges associated with undesirably distributed pre-acquired datasets and extending the application across various complex systems. There is also potential for exploring alternative NN architectures, different mathematical programming formulations, such as strong MIP formulations (Anderson et al., 2020) and P-split approaches (Kronqvist et al., 2022), and advanced sampling strategies, along with parallelisation techniques where applicable. Our work establishes a foundation for more intricate algorithmic design, which lies beyond the scope of this study but offers a promising direction for future research.

## CRediT authorship contribution statement

**Yu Liu:** Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Formal analysis, Conceptualization. **Fabricio Oliveira:** Writing – review & editing, Writing – original draft, Validation, Resources, Project administration, Methodology, Investigation, Funding acquisition, Formal analysis, Conceptualization.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Acknowledgements

Fabricio Oliveira gratefully acknowledges the support of the Research Council of Finland, decision number 348094. Yu Liu gratefully acknowledges the support from the China Scholarship Council, China, grant number 202206260034.

## Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.compchemeng.2025.109243.

## Data availability

The data and code are available at a GitHub repository https://github.com/gamma-opt/adaptive-surrogate-opt.

## References

Addis, B., Castel, C., Macali, A., Misener, R., Piccialli, V., 2023. Data augmentation driven by optimization for membrane separation process synthesis. Comput. Chem. Eng. 177, 108342. http://dx.doi.org/10.1016/j.compchemeng.2023.108342.

Advani, M.S., Saxe, A.M., Sompolinsky, H., 2020. High-dimensional dynamics of generalization error in neural networks. Neural Netw. 132, 428–446. http://dx.doi.org/10.1016/j.neunet.2020.08.022.

Aithal, S.M., Balaprakash, P., 2019. MaLTESE: Large-scale simulation-driven machine learning for transient driving cycles. In: High Performance Computing. Springer International Publishing, Cham, pp. 186–205. http://dx.doi.org/10.1007/978-3-030-20656-7_10.

Alizadeh, R., Allen, J.K., Mistree, F., 2020. Managing computational complexity using surrogate models: a critical review. Res. Eng. Des. 31 (3), 275–298. http://dx.doi.org/10.1007/s00163-020-00336-7.

Anahideh, H., Rosenberger, J., Chen, V., 2022. High-dimensional black-box optimization under uncertainty. Comput. Oper. Res. 137, 105444. http://dx.doi.org/10.1016/j.cor.2021.105444.

Anderson, R., Huchette, J., Ma, W., Tjandraatmadja, C., Vielma, J.P., 2020. Strong mixed-integer programming formulations for trained neural networks. Math. Program. 183 (1), 3–39. http://dx.doi.org/10.1007/s10107-020-01474-5.

Belotti, P., Cafieri, S., Lee, J., Liberti, L., 2012. On feasibility based bounds tightening. URL: https://optimization-online.org/?p=11891.

Bezanson, J., Edelman, A., Karpinski, S., Shah, V.B., 2017. Julia: A fresh approach to numerical computing. SIAM Rev. 59 (1), 65–98. http://dx.doi.org/10.1137/141000671.

Bhosekar, A., Ierapetritou, M., 2018. Advances in surrogate based modeling, feasibility analysis, and optimization: A review. Comput. Chem. Eng. 108, 250–267. http://dx.doi.org/10.1016/j.compchemeng.2017.09.017.

Canziani, A., Paszke, A., Culurciello, E., 2017. An analysis of deep neural network models for practical applications. http://dx.doi.org/10.48550/arXiv.1605.07678.

Ceccon, F., Jalving, J., Haddad, J., Thebelt, A., Tsay, C., Laird, C.D., Misener, R., 2022. OMLT: Optimization & machine learning toolkit. J. Mach. Learn. Res. 23 (349), 1–8, URL: http://jmlr.org/papers/v23/22-0277.html.

Cheng, X., Khomtchouk, B., Matloff, N., Mohanty, P., 2019. Polynomial regression as an alternative to neural nets. http://dx.doi.org/10.48550/arXiv.1806.06850.

Cheng, Y., Wang, D., Zhou, P., Zhang, T., 2020. A survey of model compression and acceleration for deep neural networks. http://dx.doi.org/10.48550/arXiv.1710.09282.

Danisch, S., Krumbiegel, J., 2021. Makie.jl: Flexible high-performance data visualization for Julia. J. Open Source Softw. 6 (65), 3349. http://dx.doi.org/10.21105/joss.03349.

Fasshauer, G.E., McCourt, M.J., 2012. Stable evaluation of gaussian radial basis function interpolants. SIAM J. Sci. Comput. 34 (2), A737–A762. http://dx.doi.org/10.1137/110824784.

Fischetti, M., Jo, J., 2018. Deep neural networks and mixed integer linear optimization. Constraints 23 (3), 296–309. http://dx.doi.org/10.1007/s10601-018-9285-6.

Gal, Y., Ghahramani, Z., 2016. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In: Proceedings of the 33rd International Conference on Machine Learning. vol. 48, PMLR, New York, USA, pp. 1050–1059, URL: https://proceedings.mlr.press/v48/gal16.html.

Gleixner, A.M., Berthold, T., Müller, B., Weltge, S., 2017. Three enhancements for optimization-based bound tightening. J. Global Optim. 67, 731–757. http://dx.doi.org/10.1007/s10898-016-0450-4.

Gowreesh, S., Pravin, V.K., Rajagopal, K., Veena, P.H., 2012. Thermal stresses investigation of a gas turbine blade. AIP Conf. Proc. 1440 (1), 374–383. http://dx.doi.org/10.1063/1.4704239.

Grimstad, B., Andersson, H., 2019. ReLU networks as surrogate models in mixed-integer linear programs. Comput. Chem. Eng. 131, 106580. http://dx.doi.org/10.1016/j.compchemeng.2019.106580.

Guo, S., 2021. An introduction to Surrogate modeling, Part II: Case study. URL: https://towardsdatascience.com/an-introduction-to-surrogate-modeling-part-ii-case-study-426d8035179e.

Guo, M., Manzoni, A., Amendt, M., Conti, P., Hesthaven, J.S., 2022. Multi-fidelity regression using artificial neural networks: Efficient approximation of parameter-dependent output quantities. Comput. Methods Appl. Mech. Engrg. 389, 114378. http://dx.doi.org/10.1016/j.cma.2021.114378.

Gurobi Optimization, LLC, 2024a. Gurobi optimizer reference manual. URL: https://www.gurobi.com.

Gurobi Optimization, LLC, 2024b. Solution pool - gurobi optimization. URL: https://www.gurobi.com/documentation/current/refman/solution_pool.html.

Hehn, T.M., Kooij, J.F.P., Hamprecht, F.A., 2020. End-to-end learning of decision trees and forests. Int. J. Comput. Vis. 128 (4), 997–1011. http://dx.doi.org/10.1007/s11263-019-01237-6.

Hinton, G.E., Srivastava, N., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.R., 2012. Improving neural networks by preventing co-adaptation of feature detectors. http://dx.doi.org/10.48550/arXiv.1207.0580.

Hu, C., Zeng, S., Li, C., 2023. A framework of global exploration and local exploitation using surrogates for expensive optimization. Knowl.-Based Syst. 280, 111018. http://dx.doi.org/10.1016/j.knosys.2023.111018.

Huchette, J., Muñoz, G., Serra, T., Tsay, C., 2023. When deep learning meets polyhedral theory: A survey. http://dx.doi.org/10.48550/arXiv.2305.00241.

Hüllen, G., Zhai, J., Kim, S.H., Sinha, A., Realff, M.J., Boukouvala, F., 2020. Managing uncertainty in data-driven simulation-based optimization. Comput. Chem. Eng. 136, 106519. http://dx.doi.org/10.1016/j.compchemeng.2019.106519.

Innes, M., 2018. Flux: Elegant machine learning with Julia. J. Open Source Softw. 3 (25), 602. http://dx.doi.org/10.21105/joss.00602.

Ito, Y., 2015. Delaunay triangulation. In: Engquist, B. (Ed.), Encyclopedia of Applied and Computational Mathematics. Springer, Berlin, Heidelberg, pp. 332–334. http://dx.doi.org/10.1007/978-3-540-70529-1_314.

Katz, J., Pappas, I., Avraamidou, S., Pistikopoulos, E.N., 2020. Integrating deep learning models and multiparametric programming. Comput. Chem. Eng. 136, 106801. http://dx.doi.org/10.1016/j.compchemeng.2020.106801.

Kingma, D.P., Ba, J., 2017. Adam: A method for stochastic optimization. URL: https://arxiv.org/abs/1412.6980.

Kleiber, W., Nychka, D.W., 2015. Equivalent kriging. Spat. Stat. 12, 31–49. http://dx.doi.org/10.1016/j.spasta.2015.01.004.

Kleijnen, J.P.C., 2014. Simulation-optimization via Kriging and bootstrapping: A survey. J. Simul. 8 (4), 241–250. http://dx.doi.org/10.1057/jos.2014.4.

Kronqvist, J., Misener, R., Tsay, C., 2022. P-split formulations: A class of intermediate formulations between big-M and convex hull for disjunctive constraints. http://dx.doi.org/10.48550/arXiv.2202.05198.

Kucherenko, S., Sytsko, Y., 2005. Application of deterministic low-discrepancy sequences in global optimization. Comput. Optim. Appl. 30 (3), 297–318. http://dx.doi.org/10.1007/s10589-005-4615-1.

Laurent, L., Le Riche, R., Soulier, B., Boucard, P.-A., 2019. An overview of gradient-enhanced metamodels with applications. Arch. Comput. Methods Eng. 26 (1), 61–106. http://dx.doi.org/10.1007/s11831-017-9226-3.

L'Ecuyer, P., Lemieux, C., 2002. Recent advances in randomized Quasi-Monte Carlo methods. In: Dror, M., L'Ecuyer, P., Szidarovszky, F. (Eds.), Modeling Uncertainty: an Examination of Stochastic Theory, Methods, and Applications. Springer US, New York, NY, pp. 419–474. http://dx.doi.org/10.1007/0-306-48102-2_20.

Li, F., Yang, Y., Shang, Z., Li, S., Ouyang, H., 2023. Kriging-assisted indicator-based evolutionary algorithm for expensive multi-objective optimization. Appl. Soft Comput. 147, 110736. http://dx.doi.org/10.1016/j.asoc.2023.110736.

Liu, Q., Wu, X., Lin, Q., Ji, J., Wong, K.-C., 2021. A novel surrogate-assisted evolutionary algorithm with an uncertainty grouping based infill criterion. Swarm Evol. Comput. 60, 100787. http://dx.doi.org/10.1016/j.swevo.2020.100787.

Lu, Q., Polyzos, K.D., Li, B., Giannakis, G.B., 2023. Surrogate modeling for Bayesian optimization beyond a single Gaussian process. IEEE Trans. Pattern Anal. Mach. Intell. 45 (9), 11283–11296. http://dx.doi.org/10.1109/TPAMI.2023.3264741.

Lualdi, P., Sturm, R., Camero, A., Siefkes, T., 2024. An uncertainty-based objective function for hyperparameter optimization in Gaussian processes applied to expensive black-box problems. Appl. Soft Comput. 154, 111325. http://dx.doi.org/10.1016/j.asoc.2024.111325.

Magris, M., Iosifidis, A., 2023. Bayesian learning for neural networks: an algorithmic survey. Artif. Intell. Rev. 56 (10), 11773–11823. http://dx.doi.org/10.1007/s10462-023-10443-1.

Maragno, D., Wiberg, H., Bertsimas, D., Birbil, Ş.İ., den Hertog, D., Fajemisin, A.O., 2023. Mixed-integer optimization with constraint learning. Oper. Res. http://dx.doi.org/10.1287/opre.2021.0707.

Martins, J.R.R.A., Ning, A., 2021. Surrogate-based optimization. In: Engineering Design Optimization, first ed. Cambridge University Press, pp. 373–419. http://dx.doi.org/10.1017/9781108980647.

Miller, D.C., Siirola, J.D., Agarwal, D., Burgard, A.P., Lee, A., Eslick, J.C., Nicholson, B., Laird, C., Biegler, L.T., Bhattacharyya, D., Sahinidis, N.V., Grossmann, I.E., Gounaris, C.E., Gunter, D., 2018. Next generation multi-scale process systems engineering framework. Comput. Aided Chem. Eng. 44, 2209–2214. http://dx.doi.org/10.1016/B978-0-444-64241-7.50363-3.

Ndih, E.D.N., Cherkaoui, S., 2015. Simulation methods, techniques and tools of computer systems and networks. In: Modeling and Simulation of Computer Networks and Systems. Morgan Kaufmann, Boston, pp. 485–504. http://dx.doi.org/10.1016/B978-0-12-800887-4.00017-1.

Pearce, T., Leibfried, F., Brintrup, A., 2020. Uncertainty in neural networks: Approximately bayesian ensembling. In: Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics. URL: https://proceedings.mlr.press/v108/pearce20a.html.

Perakis, G., Tsiourvas, A., 2022. Optimizing objective functions from trained relu neural networks via sampling. http://dx.doi.org/10.48550/arXiv.2205.14189.

Reijonen, E., Begantsova, M., Toivonen, V., Belyak, N., Linkola, J., Oliveira, F., 2024. Gogeta.jl. URL: https://github.com/gamma-opt/Gogeta.jl.

Renardy, M., Joslyn, L.R., Millar, J.A., Kirschner, D.E., 2021. To Sobol or not to Sobol? The effects of sampling schemes in systems biology applications. Math. Biosci. 337, 108593. http://dx.doi.org/10.1016/j.mbs.2021.108593.

Roy, C.J., 2019. Errors and uncertainties: Their sources and treatment. In: Beisbart, C., Saam, N.J. (Eds.), Computer Simulation Validation: Fundamental Concepts, Methodological Frameworks, and Philosophical Perspectives. Springer International Publishing, Cham, pp. 119–141. http://dx.doi.org/10.1007/978-3-319-70766-2_5.

Serra, T., Kumar, A., Ramalingam, S., 2020. Lossless compression of deep neural networks. In: Hebrard, E., Musliu, N. (Eds.), Integration of Constraint Programming, Artificial Intelligence, and Operations Research. Springer International Publishing, Cham, pp. 417–430. http://dx.doi.org/10.1007/978-3-030-58942-4_27.

Surjanovic, S., Bingham, D., 2013. Optimization test problems: Rastrigin function. URL: https://www.sfu.ca/~ssurjano/rastr.html.

Tao, Q., Li, L., Huang, X., Xi, X., Wang, S., Suykens, J.A.K., 2022. Piecewise linear neural networks and deep learning. Nat. Rev. Methods Prim. 2 (1), 1–17. http://dx.doi.org/10.1038/s43586-022-00125-7.

The MathWorks, Inc., 2024a. MATLAB partial differential equation toolbox, R2024a. URL: https://se.mathworks.com/products/pde.html.

The MathWorks, Inc., 2024b. Thermal stress analysis of jet engine turbine blade - MATLAB & simulink - mathworks nordic. URL: https://se.mathworks.com/help/pde/ug/thermal-stress-analysis-of-jet-engine-turbine-blade.html.

Tong, J., Cai, J., Serra, T., 2024. Optimization over trained neural networks: Taking a relaxing walk. In: Dilkina, B. (Ed.), Integration of Constraint Programming, Artificial Intelligence, and Operations Research. Springer Nature Switzerland, Cham, pp. 221–233. http://dx.doi.org/10.1007/978-3-031-60599-4_14.

Tsay, C., 2021. Sobolev trained neural network surrogate models for optimization. Comput. Chem. Eng. 153, 107419. http://dx.doi.org/10.1016/j.compchemeng.2021.107419.

Turcani, L., Sadler, J., 2024. ML surrogates for chemical processes with OMLT. URL: https://github.com/cog-imperial/OMLT/blob/main/docs/notebooks/neuralnet/auto-thermal-reformer-relu.ipynb.

Wendland, H., 2017. Multiscale radial basis functions. In: Pesenson, I., Le Gia, Q.T., Mayeli, A., Mhaskar, H., Zhou, D.-X. (Eds.), Frames and Other Bases in Abstract and Function Spaces: Novel Methods in Harmonic Analysis, Volume 1. Springer International Publishing, Cham, pp. 265–299. http://dx.doi.org/10.1007/978-3-319-55550-8_12.

Yang, D., Balaprakash, P., Leyffer, S., 2022. Modeling design and control problems involving neural network surrogates. Comput. Optim. Appl. 83 (3), 759–800. http://dx.doi.org/10.1007/s10589-022-00404-9.

Yosinski, J., Clune, J., Bengio, Y., Lipson, H., 2014. How transferable are features in deep neural networks?. http://dx.doi.org/10.48550/arXiv.1411.1792.

Zhang, C., Janeway, M., 2022. Optimization of turbine blade aerodynamic designs using CFD and neural network models. Int. J. Turbomach. Propuls. Power 7 (3), 20. http://dx.doi.org/10.3390/ijtpp7030020.

Zhu, Q., Kang, G., Wu, X., Lin, Q., Tang, H., Chen, J., 2024. A Kriging-assisted evolutionary algorithm with multiple infill sampling for expensive many-objective optimization. Eng. Appl. Artif. Intell. 135, 108505. http://dx.doi.org/10.1016/j.engappai.2024.108505.