Journal Name

ARTICLE TYPE

Cite this: DOI: 00.0000/xxxxxxxxxx

Supplementary Information

Multi-stage Bayesian optimisation for dynamic decision-making in self-driving labs

Luca Torresi[a,b] and Pascal Friederich[a,b]

1 Synthetic dataset generator

This section details the dataset generator we developed to benchmark sequential optimisation algorithms by creating complex, differentiable, multi-step functions. The problems are constructed using interconnected neural networks, offering tunable complexity, realistic constraints (e.g., additive measurement noise, additive process noise, information masking), and full reproducibility. A key feature is that the true global optimum is accessible via gradient-based methods, which facilitates the comparison and validation of different optimisation algorithms.

1.1 Random function generation

To establish a comprehensive testing framework for optimisation algorithms targeting cascade processes, we developed a system that generates continuous and differentiable random functions using neural networks. These functions are subsequently composed to form interdependent, multi-step cascades. Each constituent function is implicitly defined by training a multi-layer perceptron on a randomly sampled seed dataset, $\mathscr{D}_{\text{seed}} = (\boldsymbol{X}_{\text{seed}}, \boldsymbol{Y}_{\text{seed}})$.

1.1.1 Dataset and boundary constraints.

The input data, $\boldsymbol{X}_{\text{seed}}$, are generated using Sobol sequences to ensure a quasi-random, uniform coverage within the d-dimensional hypercube $[0,1]^d$. The corresponding target outputs, $\boldsymbol{Y}_{\text{seed}}$, are independently drawn from a standard normal distribution, $\mathscr{N}(0,1)$. To prevent the resulting function from exhibiting maxima at the domain boundaries, we employ a mechanism for enforcing an artificial boundary penalty. Points sampled slightly outside the $[0,1]^d$ domain are assigned a low target value (e.g., -1.5). This mechanism effectively pushes the function's response down near the edges, thereby encouraging the presence of internal maxima.

1.1.2 Neural network architecture and training.

The function f is implemented as a multi-layer perceptron with a customizable number of hidden layers and configurable dimensions. In our experiments, we use three hidden layers with dimensions $[64, 128, 32]$ and LeakyReLU as the activation function for all hidden layers. The network is trained for 800 epochs using the Adam optimiser (learning rate $= 0.01$) to minimise the mean squared error between the network's predictions and the random target data, using a batch size of 16. The final output of the network, $\boldsymbol{h}$, is optionally scaled to restrict its range using the sigmoid function. In our multi-step experiments, this scaling is applied to the output of all sub-processes except the final step, thus constraining the input of subsequent steps to the range $[0,1]$.

Figure S1 illustrates examples of the generated functions in a 1D and a 2D parameter space for different sizes of the seed dataset ($\mathscr{D}_{\text{seed}}$). The size of this randomly sampled seed dataset directly correlates with what we define as the *complexity* of the generated function.

1.1.3 Function complexity.

This complexity, in turn, is empirically shown to correlate with the amount of data required by a support vector regressor (implemented using scikit-learn with default settings) to achieve a target modelling accuracy. Figure S2 demonstrates this correlation: for different input dimensionalities ($d = 2, 4, 6,$ and 8, corresponding to panels A, B, C, and D), we plot the coefficient of determination R^2 achieved on a fixed test set while varying the size of the training set, for functions generated with seed datasets of size 2, 15, 50, and 100. As the seed dataset size increases, the functions become more complex, requiring the model to use more training data to reach high accuracy.

1.2 Multi-stage process

To simulate an N-step cascade process (as defined in Section 2.2 and Fig. 1), we generate a sequence of N functions, $f_1, \ldots, f_N$, where the input of each step depends on the output of the preceding step.

For the initial step ($i = 1$), the function f_1 accepts external input parameters $\boldsymbol{x}_1 \in [0,1]^{d_1}$, defining the first hidden state as:

$$\boldsymbol{h}_1 = f_1(\boldsymbol{x}_1).$$

For all subsequent steps ($i > 1$), the function f_i accepts a concatenation of the current step's control parameters $\boldsymbol{x}_i \in [0,1]^{d_i}$ and the hidden state output from the previous step, $\boldsymbol{h}_{i-1}$:

$$\boldsymbol{h}_i = f_i([\boldsymbol{x}_i, \boldsymbol{h}_{i-1}]).$$

The complexity of each component function f_i is implicitly controlled by the size of its seed dataset $\mathscr{D}_{\text{seed}}$, as described in the

[a] Institute of Nanotechnology, Karlsruhe Institute of Technology, Kaiserstr. 12, 76131 Karlsruhe, Germany.
[b] Institute of Anthropomatics and Robotics, Karlsruhe Institute of Technology, Kaiserstr. 12, 76131 Karlsruhe, Germany.

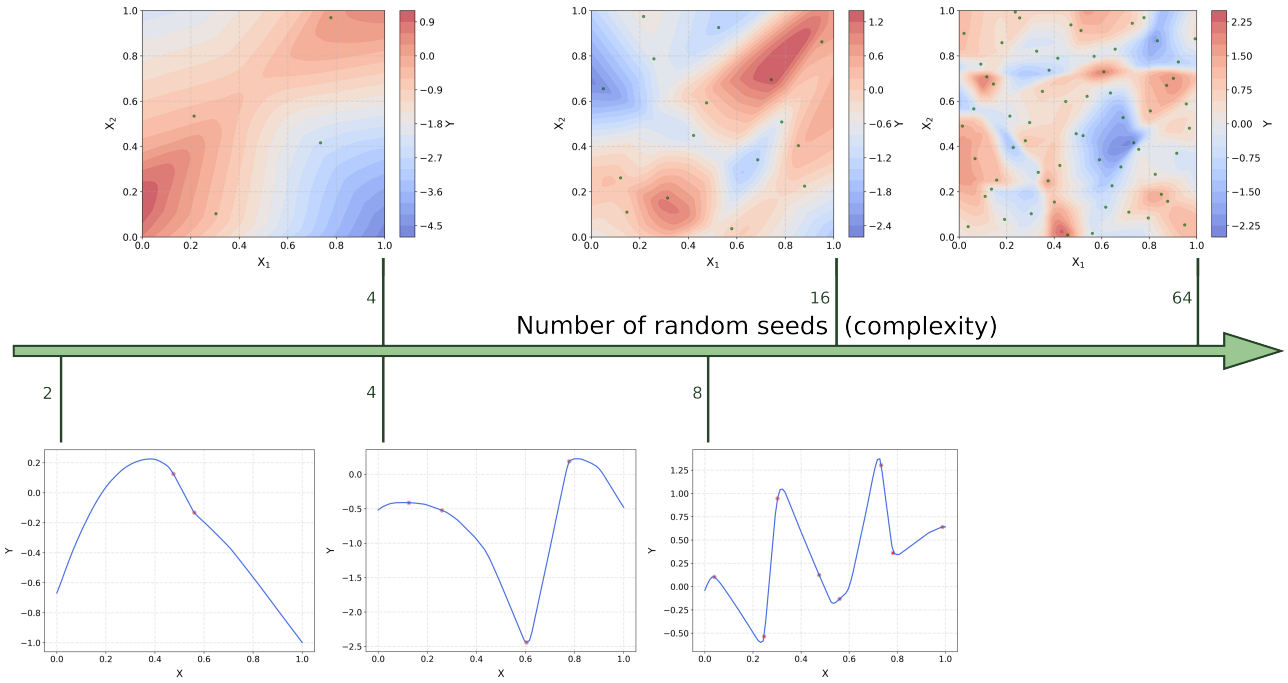

Fig. S1 Examples of synthetic functions generated with varying sizes of the seed dataset $\mathscr{D}_{\text{seed}}$. The bottom row shows $1D$ functions yielding scalar outputs, while the top row shows functions in a $2D$ input space yielding scalar outputs. The size of $\mathscr{D}_{\text{seed}}$ directly controls the complexity of the resulting function landscape.

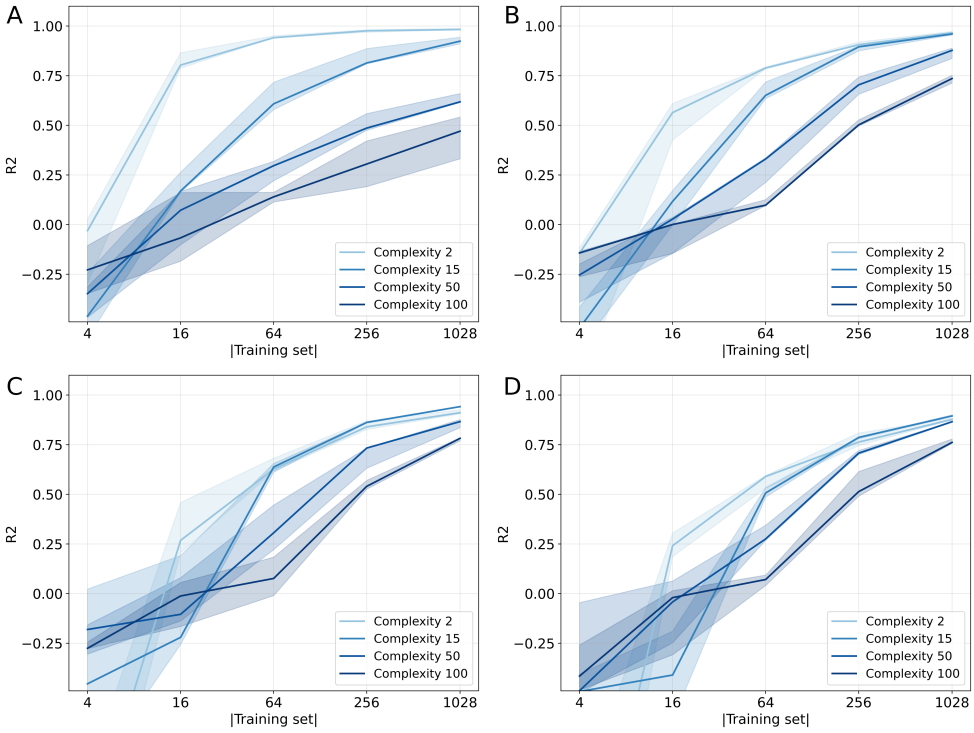

Fig. S2 Correlation of $\mathscr{D}_{\text{seed}}$ size with function complexity. The plots show the coefficient of determination (R^2) on a test set for a support vector regressor as a function of training set size. Panels (A), (B), (C), and (D) correspond to input dimensions $d = 2, 4, 6$, and 8, respectively. Different lines represent synthetic functions generated with varying $\mathscr{D}_{\text{seed}}$ size.

previous section. A random generator initialised with a fixed seed is used for all data sampling and network initialisations, ensuring the complete process is fully reproducible.

To benchmark algorithms under realistic laboratory conditions, the framework supports the injection of stochasticity and information masking at various stages of the cascade. We distinguish between intrinsic process noise and extrinsic measurement uncertainty.

1.2.1 Process noise.

Additive Gaussian noise can be injected directly into the latent process outputs $\boldsymbol{h}$ before they propagate to the next stage. This simulates inherent variability in the underlying physical or chemical transformation. The noisy process output is given by:

$$\boldsymbol{h}_i = f_i([\boldsymbol{x}_i, \boldsymbol{h}_{i-1}]) + \boldsymbol{\varepsilon}^p, \tag{1}$$

where $\boldsymbol{\varepsilon}^p \sim \mathcal{N}(0, \sigma_p^2 \boldsymbol{I})$ This noise propagates through the system, as $\boldsymbol{h}_i$ becomes the input for the subsequent step f_{i+1}, altering the trajectory of that specific sample in the cascade.

1.2.2 Measurements.

To simulate partial observability, we apply a masking matrix $\boldsymbol{M}_i$ that selects a subset of the state $\boldsymbol{h}_i$ for observation, forcing the optimiser to infer the latent state of the system from partial data. Furthermore, independent Gaussian measurement noise $\boldsymbol{\varepsilon}^m \sim \mathcal{N}(0, \sigma_m^2 \boldsymbol{I})$ can be added to these observations:

$$\boldsymbol{m}_i = \boldsymbol{M}_i \boldsymbol{h}_i + \boldsymbol{\varepsilon}_i^m \tag{2}$$

Unlike process noise, measurement noise affects only the data observed by the optimiser; it does not propagate to subsequent physical steps in the cascade.

2 MSBO decision rule

Algorithm 1 MSBO stage selection and inventory update

Require: Budget parameters (E_{total}, E_{current}), tolerance $\tau = 5 \cdot 10^{-3}$, minimum cost $c_{\min} = 10^{-6}$, stage costs $\{c_i\}$, cascade model $\mathscr{M}$, inventory $\mathscr{I}$, frequency constraints $\mathscr{F}$, Monte Carlo samples $S = 256$

$\beta \leftarrow 100 \cdot \left(\frac{E_{\text{total}} - E_{\text{current}}}{E_{\text{total}}}\right) + 1$ ▷ Compute linear decay schedule for exploration
$i_{\text{forced}} \leftarrow \text{CheckFrequencyConstraints}(\mathscr{I}, \mathscr{F})$
for each $\alpha \in \{\text{EI}, \text{UCB}\}$ **do**
  Initialise candidate set $\mathscr{C} \leftarrow \emptyset$
  for each stage i causally feasible given $\mathscr{I}$ **do**
    if $i_{\text{forced}} \neq \text{Null}$ **and** $i \neq i_{\text{forced}}$ **then**
      continue
    end if
    if $c_i < c_{\min}$ **then**
      raise ValueError("Costs must be strictly positive; assign a constant $\geq 10^{-6}$")
    end if
    if $i = 0$ **then**
      $x_i^*, v_i \leftarrow \text{OptimiseAcquisition}(\alpha, \mathscr{M}, i, \text{Null}, \beta, S)$
      $\mathscr{C} \leftarrow \mathscr{C} \cup \{(i, \text{Null}, x_i^*, v_i/c_i)\}$
    else
      for each available sample j at stage i in $\mathscr{I}$ **do**
        $m_{i-1}^j \leftarrow \text{GetHistory}(\mathscr{I}, j)$
        $x_i^*, v_i \leftarrow \text{OptimiseAcquisition}(\alpha, \mathscr{M}, i, m_{i-1}^j, \beta, S)$
        $\mathscr{C} \leftarrow \mathscr{C} \cup \{(i, j, x_i^*, v_i/c_i)\}$
      end for
    end if
  end for
  $(i^*, j^*, x^*, v^*) \leftarrow \arg\max_{(i,j,x,v) \in \mathscr{C}} v$
  if $\alpha = \text{EI}$ **and** $v^* < \tau$ **then**
    continue ▷ Fallback to UCB if EI is numerically negligible
  else
    break ▷ Valid optimum acquired
  end if
end for
$m^* \leftarrow \text{ExecuteExperiment}(i^*, x^*)$
if $i^* = 0$ **then**
  $\mathscr{I}.\text{AddSample}(i^*, x^*, m^*)$ ▷ Initialise new sample
else
  $\mathscr{I}.\text{UpdateSample}(j^*, i^*, x^*, m^*)$ ▷ Update existing sample
end if
$\text{UpdateFrequencyCounters}(i^*, \mathscr{F})$

3 Additional results

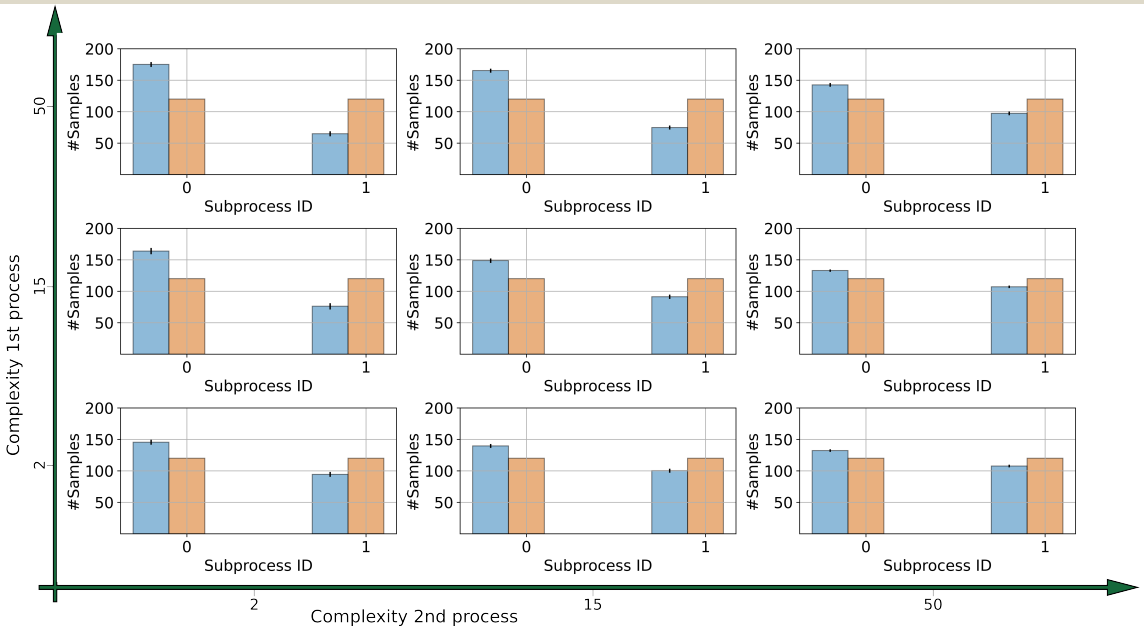

Fig. S3 Sample counts across sub-processes. The bar charts display the number of samples collected for the first stage and the second stage, for MSBO (blue) and the baselines (orange). The grid arrangement corresponds to the complexity settings defined in Fig. S2, with rows representing increasing stage-one complexity and columns representing increasing stage-two complexity. The sampling distribution shifts dynamically according to the relative difficulty of each stage.

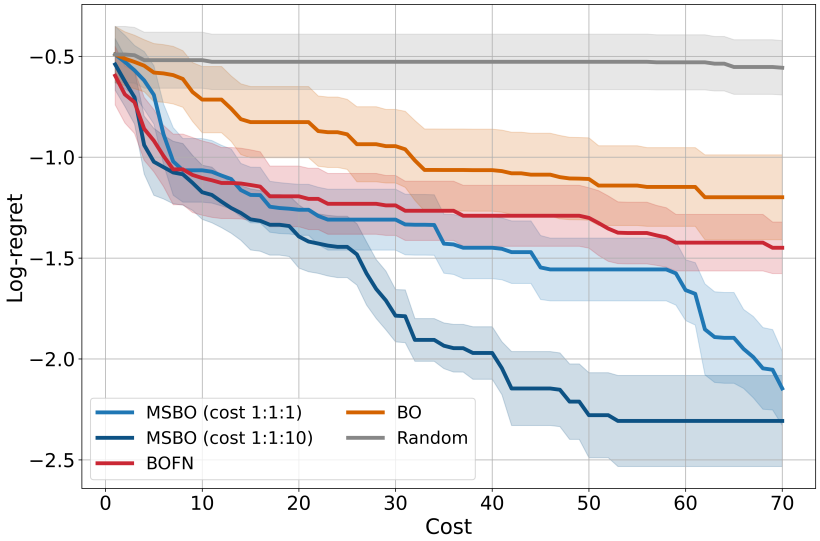

Fig. S4 Performance on a three-stage cascade process. Aggregated log-regret plotted against cost for a synthetic three-stage process $f_1(\boldsymbol{x}_1) \to \boldsymbol{h}_1, f_2(\boldsymbol{h}_1, \boldsymbol{x}_2) \to \boldsymbol{h}_2, f_3(\boldsymbol{h}_2, \boldsymbol{x}_3) \to y$, with $\boldsymbol{x}_1 \in \mathbb{R}^4$, and $\boldsymbol{x}_2, \boldsymbol{x}_3, \boldsymbol{h}_1, \boldsymbol{h}_2 \in \mathbb{R}^2$, and $\mathscr{D}_{seed}$ size 15, 15, and 5, respectively. The curves display MSBO performance under uniform (1:1:1) and heterogeneous (1:1:10) cost ratios, compared to BO and BOFN baselines. The trajectories represent the aggregated results over 10 optimisation runs. Shaded regions indicate a 1 standard deviation interval. Cost and regret are considered unitless here.

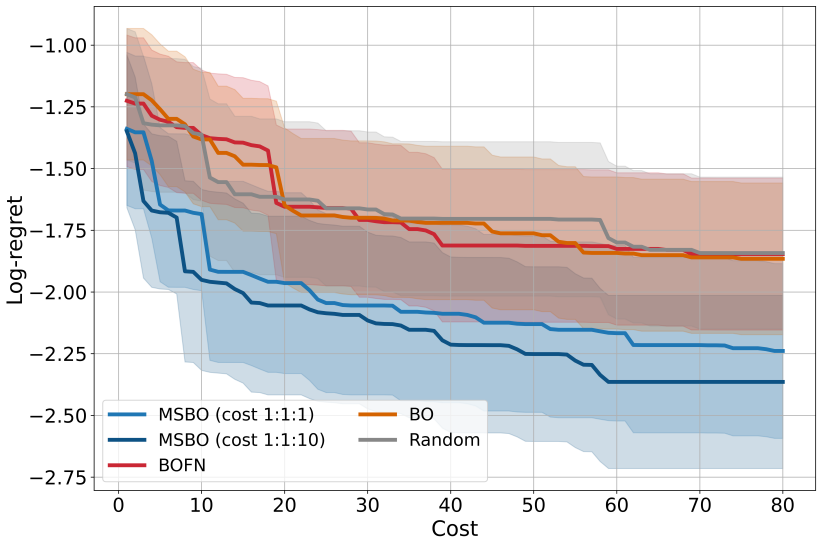

Fig. S5 Performance on a partially observable three-stage cascade process. Aggregated log-regret plotted against cost for a synthetic three-stage process with controllable parameters $\boldsymbol{x}_1 \in \mathbb{R}^4$, $\boldsymbol{x}_2 \in \mathbb{R}^2$, and $\boldsymbol{x}_3 \in \mathbb{R}^1$. The latent process outputs are $\boldsymbol{h}_1 \in \mathbb{R}^4$, $\boldsymbol{h}_2 \in \mathbb{R}^2$, and $y \in \mathbb{R}^1$. Partial observability is simulated using a masking matrix $\boldsymbol{M}$ applied to the latent outputs, resulting in accessible intermediate observation dimensions of 2, and 1, respectively. The stage complexities ($\mathscr{D}_{seed}$ sizes) are 50, 15, and 2. The curves display MSBO performance under uniform (1:1:1) and heterogeneous (1:1:10) cost ratios, compared to BO, BOFN, and random baselines. The trajectories represent the aggregated results over 10 optimisation runs. Shaded regions indicate a 1 standard deviation interval. Cost and regret are considered unitless here.

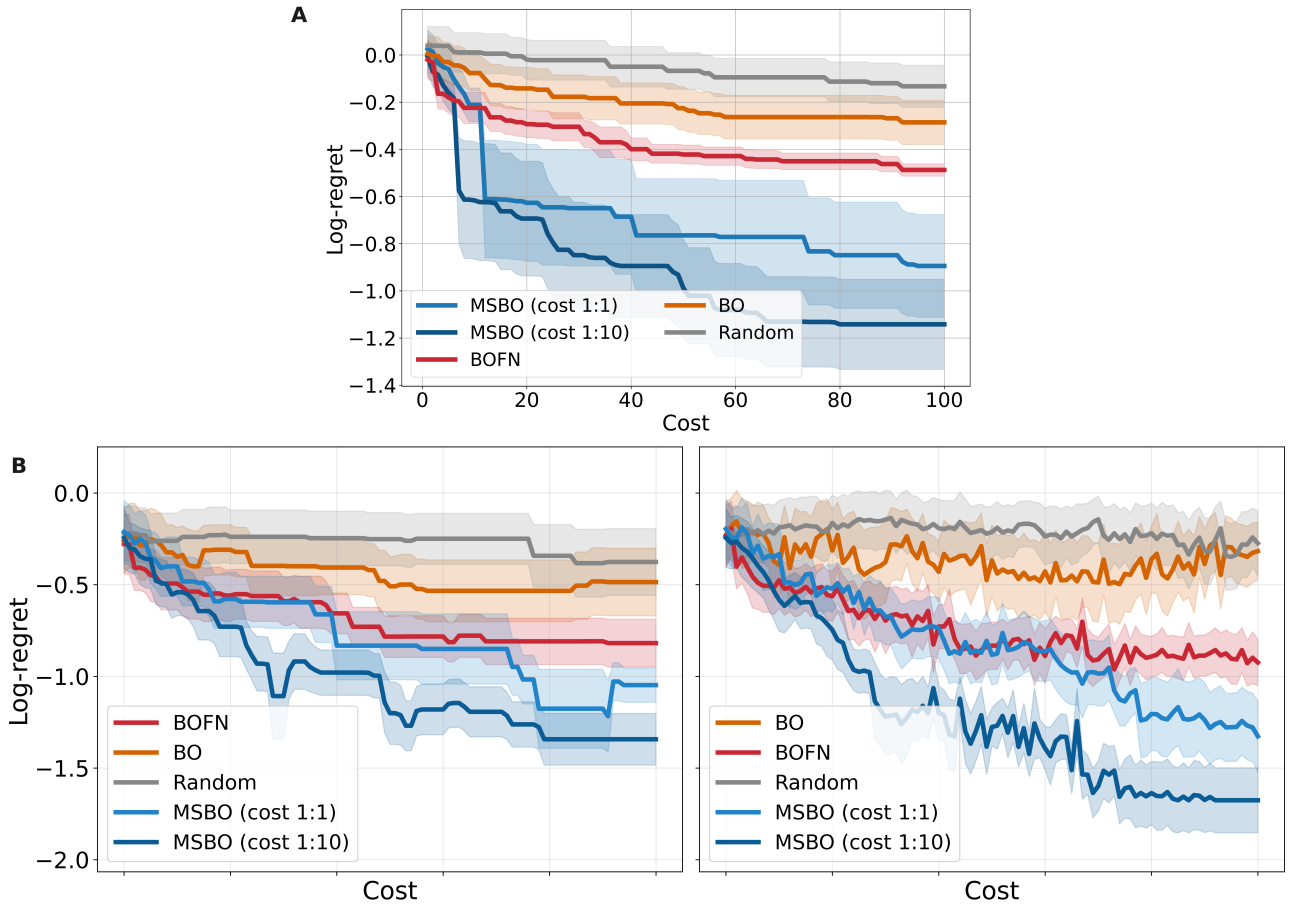

Fig. S6 Assessment of optimisation robustness under process noise and the impact of model-based candidate selection. We evaluate a two-stage process with controllable parameters $\boldsymbol{x}_1 \in \mathbb{R}^4$ and $\boldsymbol{x}_2 \in \mathbb{R}^2$, intermediate state $\boldsymbol{h}_1 \in \mathbb{R}^2$, complexities $(50, 2)$, and Gaussian process noise with standard deviations $(0.05, 0.1)$. (A) Simple regret calculated with respect to the noisy observations $(y_{opt} + 3\sigma)$, where σ is the estimated noise at the output. (B) Regret calculated with respect to y_{opt} of the denoised output. The left panel displays the denoised performance of the parameter set corresponding to the single best observed value (same parameters as in Panel A). The right panel displays the denoised performance of the parameter set selected at each iteration step by maximising the surrogate model's predicted mean among the collected samples.

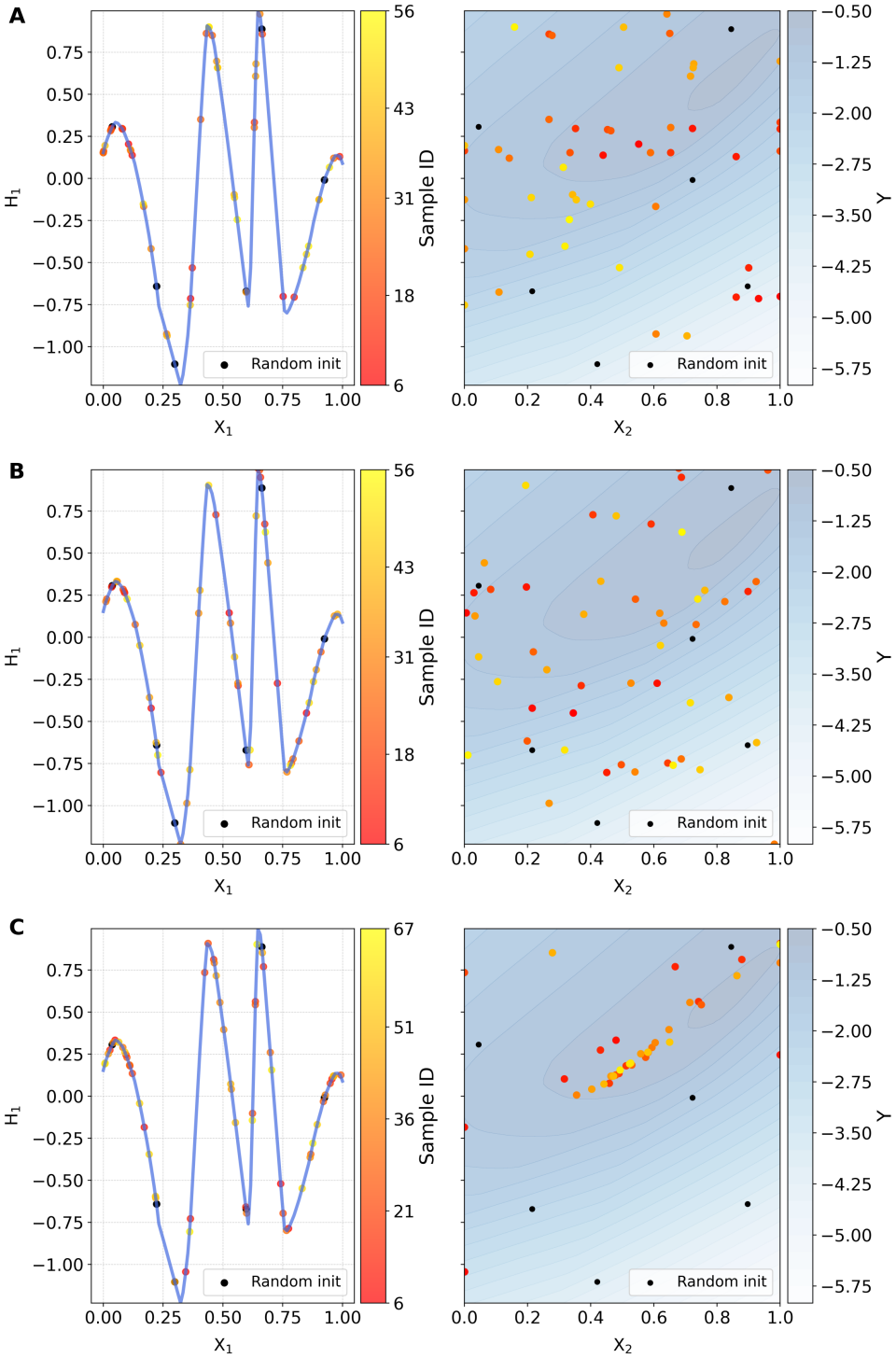

Fig. S7 Representative sampling distributions for a two-stage process $f_1(x_1) \to h_1, f_2(h_1, x_2) \to y$. Panels (A), (B), and (C) correspond to standard BO, BOFN, and MSBO, respectively. The left subplots display the first-stage intermediate outputs h_1 (dots) plotted against the input x_1, overlaid on the true first-stage function. The right subplots show the locations of the second-stage evaluations within the $h_1 \times x_2$ domain. Markers are colour-coded by the sample ID, with black dots representing random initialisation.

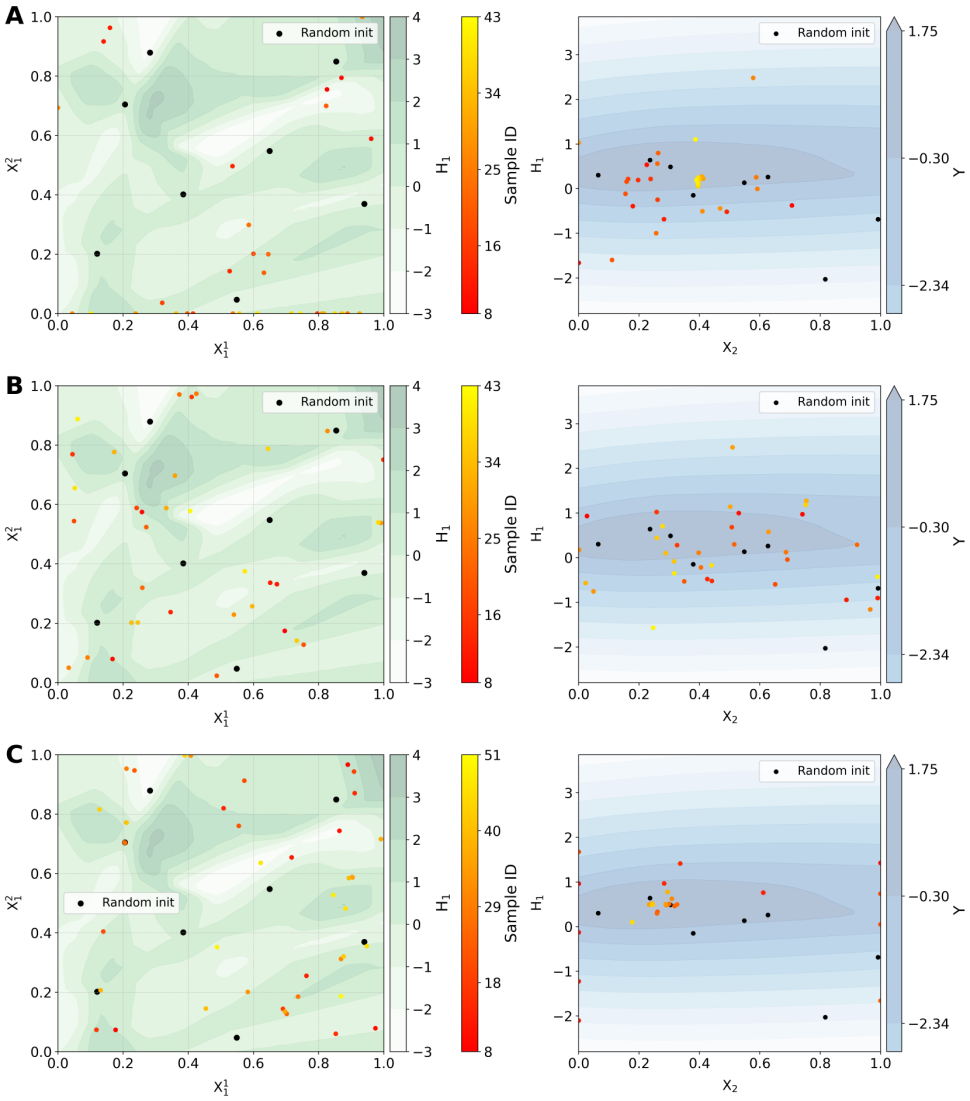

Fig. S8 Representative sampling distributions for a two-stage process $f_1(x_1^1, x_2^2) \to h_1, f_2(h_1, x_2) \to y$. Panels (A), (B), and (C) correspond to standard BO, BOFN, and MSBO, respectively. The subplots display the sample locations in the 2D input space for the first stage (left) and the second stage (right), overlaid on the true function landscapes. Markers are colour-coded by the sample ID, with black dots representing random initialisation.