Automatica 195 (2027) 113325

Contents lists available at ScienceDirect

Automatica

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

Brief paper

Learning low-dimensional embeddings for black-box optimization☆

Riccardo Busetto [a], Manas Mejari [a], Marco Forgione [a], Alberto Bemporad [b], Dario Piga [a,*]

[a] IDSIA Dalle Molle Institute for Artificial Intelligence, SUPSI, Lugano-Viganello, Switzerland
[b] IMT School for Advanced Studies, Lucca, Italy

ARTICLE INFO

Article history:
Received 1 October 2025
Received in revised form 11 June 2026
Accepted 14 August 2026

Keywords:
Black-box optimization
Meta learning
Machine learning
Model predictive control

ABSTRACT

Black-box optimization (BBO) is a valuable alternative to gradient-based methods when differentiating the objective function is not feasible. However, BBO often struggles with high-dimensional decision spaces and limited evaluation budgets. In this work, we propose a meta-learning approach that pre-computes a reduced-dimensional manifold capturing the optimal solutions of a class of optimization problems. For new problem instances drawn from the same class, BBO is then performed in the latent space, substantially reducing the computational effort required to reach near-optimal solutions. We further establish probabilistic upper bounds on the performance gap between solving the original problem and its reduced-dimensional counterpart. The effectiveness of the method is demonstrated both on benchmark optimization problems and on the automatic tuning of model predictive controllers.

© 2026 The Author(s). Published by Elsevier Ltd. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

1. Introduction

1.1. Black-box optimization

Black-box optimization (BBO) aims to find the solution of an optimization problem where the objective function lacks an explicit mathematical formulation. Instead, the function can only be evaluated through queries, such as physical experiments or simulations through complex computational models. However, in many real-world scenarios, these evaluations are expensive, noisy, or time-consuming. Therefore, a key goal of black-box optimization is to find near-optimal solutions while limiting the number of costly function evaluations. Due to these challenges, BBO relies on sample-efficient strategies that select query points to balance *exploration* (searching unexplored regions of the solution space) and *exploitation* (searching where an optimal solution is estimated to be). Various BBO techniques have been proposed in the literature, including Kriging-based methods (Jeong et al., 2005), Bayesian Optimization (BO) (Garnett, 2023), Mesh adaptive direct search (Audet & Dennis, 2006), Set-Membership (SM) approaches (Sabug et al., 2022), and global optimization using inverse distance weighting and surrogate radial basis functions (GLIS) (Bemporad, 2020).

The main limitation of the BBO algorithms mentioned above is their scalability with respect to the number of decision variables, restricting their applicability to small- to medium-size problems. Black-box optimization in high-dimensional spaces still remains an open challenge, as the number of function evaluations required to properly sample the domain of decision variables increases exponentially with the number of dimensions. Existing approaches to high-dimensional BBO either rely on a function structure fixed a priori or adapt to a single problem instance online.

By leveraging the meta-learning paradigm (Vanschoren, 2019), the goal of this paper is to learn a reduced search space where BBO is performed for an entire given class of related optimization problems. We use the word meta-learning in its broad sense: knowledge is acquired offline from a class of related tasks and then reused to improve the solution of new tasks from the same class. In our case, this knowledge is not a model of a single problem, but a representation of the solution structure shared across the class, used to learn a low-dimensional manifold where new optima are expected to lie. To obtain this manifold, we train an *autoencoder* (AE) to reconstruct optimal solutions from the problem class and use its latent space as the reduced decision space for BBO.

☆ This research was partly supported by the Eurostars programme, project "E3093 - SLIMPEC: A Software suite for LearnIng-based embedded Model PrEdictive Control" and by the European Research Council (ERC), Advanced Research Grant COMPACT (Grant Agreement No. 101141351). Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or the European Commission. Neither the European Union nor the granting authority can be held responsible for them. The material in this paper was not presented at any conference. This paper was recommended for publication in revised form by Associate Editor Dominic Liao-McPherson under the direction of Editor Florian Dorfler.

* Corresponding author.
E-mail addresses: riccardo.busetto@supsi.ch (R. Busetto), manas.mejari@supsi.ch (M. Mejari), marco.forgione@supsi.ch (M. Forgione), alberto.bemporad@imtlucca.it (A. Bemporad), dario.piga@supsi.ch (D. Piga).

https://doi.org/10.1016/j.automatica.2026.113325
0005-1098/© 2026 The Author(s). Published by Elsevier Ltd. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

Table 1
Comparison of representative high-dimensional BBO methods.

	REMBO (Wang et al., 2016)	Add. decomp. (Kandasamy et al., 2015; Rolland et al., 2018; Ziomek & Bou Ammar, 2023)	SAASBO (Eriksson & Jankowiak, 2021)	TuRBO (Eriksson et al., 2019)	Latent-space BO (Eismann et al., 2018; Gómez-Bombarelli et al., 2018; Grosnit et al., 2021; Maus et al., 2022; Tripp et al., 2020)	Antonova etal (Antonova et al., 2020)	Morita etal (Morita et al., 2023)	Meta-BO (Maraval et al., 2023)	Learn-to-warm start (Banerjee et al., 2025; Sambharya et al., 2023)	**This work**
Embedding	Fixed	Fixed	None	None	Learned	Learned	Learned	None	None	**Learned**
Problem class	Single	Single	Single	Single	Single	Single	Single	Parametric	Parametric	**Parametric**
Theor. bound	Worst-case	Regret	–	–	–	–	–	–	Solver-tied	**Distr.-free**

1.2. Related work

High-dimensional black-box optimization has been addressed in the literature by exploiting different structural assumptions on the objective function or the decision space. In Table 1, we summarize the main differences in terms of the type of embedding, single-problem versus parametric problem-class setting, and analysis of theoretical bounds on the performance loss.

Linear embeddings and additive decompositions. REMBO (Wang et al., 2016) projects the decision space onto a random low-dimensional linear subspace, under a low-effective-dimensionality assumption, although such projections may distort the problem geometry (Letham et al., 2020). Additive BO methods instead assume that the objective decomposes into low-dimensional components (Kandasamy et al., 2015; Rolland et al., 2018; Ziomek & Bou Ammar, 2023), an idea also used in controller tuning by iteratively optimizing subsets of parameters (Roveda et al., 2020). These methods rely on a structure fixed a priori, rather than on solution data from a problem class.

Surrogate-based high-dimensional BO. SAASBO (Eriksson & Jankowiak, 2021) identifies relevant input dimensions online through sparsity-inducing GP priors, while TuRBO (Eriksson et al., 2019) restricts the search to adaptive local trust regions. Both operate on a single problem instance. Our approach instead learns a low-dimensional search manifold offline, so that online optimization is always performed in the compressed space.

Latent-space BBO with generative models. Several works perform BBO in the latent space of a trained AE or VAE, for instance, for structured objects such as molecules (Gómez-Bombarelli et al., 2018; Maus et al., 2022), simulated robot trajectories (Antonova et al., 2020), or accelerator tuning (Morita et al., 2023). Other methods use the objective values collected during optimization to update the generative model, so that new latent samples are more likely to decode into high-performing candidate solutions (Eismann et al., 2018; Grosnit et al., 2021; Tripp et al., 2020). These methods learn or refine a representation for the optimization domain of a given task, whereas our latent representation is trained offline on near-optimal solutions sampled across a parametric class of problems.

Amortized optimization and meta-BO. Amortized optimization and learning-to-optimize methods exploit parametric problem families to predict a solution or warm-start for a new instance (Amos, 2023; Banerjee et al., 2025; Sambharya et al., 2023). Meta-BO methods similarly use past optimization runs to inform new tasks (Maraval et al., 2023). These approaches exploit the knowledge from related problems, but typically operate in the original decision space or learn a parameter-conditioned solution. In contrast, our method learns a reduced search space shared across an entire class, while online BBO is still performed in the reduced space.

1.3. Positioning of the contribution

Our approach combines four elements which are not jointly present in any of the methods reviewed above. First, the reduced space is learned offline from near-optimal solutions of a parametric class of problems, rather than fixed a priori or inferred online for a single instance. Second, unlike latent-space BO methods that use autoencoders to turn discrete or structured objects, such as molecules or trajectories, into continuous optimization variables, our variables are already continuous and the autoencoder is used to learn the low-dimensional region where near-optimal solutions of a parametric problem class concentrate. Third, the learned embedding is not conditioned on the problem parameters and does not predict the solution of a new instance. Instead, the autoencoder defines a low-dimensional search space shared across the problem class, and online optimization is still performed in this reduced domain. Fourth, we provide a distribution-free, algorithm-agnostic probabilistic bound on the sub-optimality gap incurred by optimizing in the latent space (Theorem 1). These distinctions are summarized in Table 1.

Our approach results in the following workflow. Near-optimal solutions are first computed offline, e.g., from simulators of related problem instances, and used to train an autoencoder. Then, for a given new instance, BBO is performed in the learned, reduced-dimensional latent space and candidate points are decoded before evaluating the original objective. In this paper, we adopt the BBO solver GLIS, although the embedding and the bounds on the suboptimality gap we provide are independent of the specific BBO algorithm used. The approach is motivated by multiparametric programming (Gal, 2010), where the optimizer can, under suitable assumptions, be expressed as a function of the problem parameters and therefore traces a solution manifold.

The rest of the paper is organized as follows. Section 2 introduces the considered class of black-box optimization problems and their reduced-dimensional counterparts. Section 3 presents the proposed latent-space learning procedure and the resulting Meta-GLIS algorithm. Section 4 provides theoretical bounds on the sub-optimality gap between optimization in the original and latent domains. Section 5 reports numerical results on a nonconvex benchmark and on automated MPC hyperparameter tuning. Section 6 concludes the paper.

2. Problem description

We consider the following *class* of optimization problems parameterized by θ

$$f_x^\star(\theta) = \min_x f(x;\theta) \qquad \text{s.t.} \quad x \in \mathcal{X} \tag{1}$$

where, for a given parameter $\theta \in \mathbb{R}^{n_\theta}$, $f(x;\theta) : \mathbb{R}^{n_x} \to \mathbb{R}$ is a function of the optimization variable $x \in \mathcal{X} \subseteq \mathbb{R}^{n_x}$. The parameter θ represents a set of conditions that affect the optimization problem. These conditions may correspond to simulation settings,

system configurations, or operational scenarios in real-world applications, which in turn influence the structure of the function $f(x;\theta)$.

We assume that an analytical expression of $f(x;\theta)$ is not available, but the function can be evaluated at any feasible query input $x \in \mathcal{X}$ and for any parameter θ. This may be the case, for instance, when θ describes different configurations of the environment and x is a design variable to be computed in order to optimize a metric f, which in turn can be evaluated only through the execution of a real experiment or high-fidelity simulation. The set $\mathcal{X}$ represents known constraints on x. For the theoretical analysis of our approach, we will denote by $p(\theta)$ the distribution from which the values of the parameter θ are sampled. For the practical implementation, as clarified in Section 3, we only assume access to a finite set of samples drawn from $p(\theta)$.

Problem Statement. *Given the class of optimization problems (1) in n_x variables and a desired reduced dimension $n_z < n_x$, learn a map $\mathrm{D} : \mathbb{R}^{n_z} \to \mathbb{R}^{n_x}$ such that the optimization of f can be carried out effectively with respect to the lower-dimensional embedding $z \in \mathbb{R}^{n_z}$:*

$$z^\star(\theta) \in \arg\min_{z \in \mathcal{Z}} f(\mathrm{D}(z);\theta), \tag{2}$$

with

$$f(\mathrm{D}(z^\star(\theta));\theta) \approx f_x^\star(\theta), \quad \forall \theta \in \mathbb{R}^{n_\theta},$$

where the feasible set $\mathcal{Z} = \{z \in \mathbb{R}^{n_z} : \mathrm{D}(z) \in \mathcal{X}\}$. In (2), $\in$ is used to account for the possible non-uniqueness of the minimizer over the latent feasible set $\mathcal{Z}$.

In other words, our goal is to find a map D such that we can retrieve an approximation $\mathrm{D}(z^\star(\theta))$ of an optimizer $x^\star(\theta)$ of the original problem (1) from the solution $z^\star(\theta)$ of the simpler problem (2).

3. Methodology

3.1. Meta-dataset creation

We generate a set of N parameters $\{\theta^{(i)}\}_{i=1}^N$ by sampling from the parameter distribution $p(\theta)$, *i.e.*, $\theta^{(i)} \sim p(\theta)$. For each $\theta^{(i)}$, we solve (1) by using any sample-based optimization algorithm (such evolutionary learning, MCMC with simulated annealing, Bayesian Optimization, GLIS) and we gather a set $\{x_k^{(i)}, f_k^{(i)}\}_{k=1}^K$ of K *near-optimal* vectors $x_k^{(i)}$, and the corresponding function values $f_k^{(i)} = f(x_k^{(i)}, \theta^{(i)})$, with $x_l^{(i)} \neq x_m^{(i)}, \forall l \neq m$, $l, m = 1, \dots, K$. For a given $\theta^{(i)}$, we refer to as near-optimal vectors as the best K candidate points explored during the iterations of the numerical algorithms used to solve (1). We denote the meta-dataset as

$$\mathcal{D} := \left\{\theta^{(i)}, \{x_k^{(i)}, f_k^{(i)}\}_{k=1}^K\right\}_{i=1}^N. \tag{3}$$

Remark 1. The meta-dataset $\mathcal{D}$ is generated offline, e.g., from simulators or inexpensive proxies. This cost is a one-time investment, amortized over future instances of the same problem class, and is distinct from the online evaluation budget used to compute the optimal solution of a new problem instance.

3.2. Learning a latent representation via autoencoders

An *autoencoder* (AE) (Hinton & Salakhutdinov, 2006) is trained to map an input vector $x \in \mathbb{R}^{n_x}$ to itself, while passing through a *bottleneck* layer of size $n_z < n_x$. Let us consider an autoencoder $\mathcal{AE}_\phi : \mathbb{R}^{n_x} \to \mathbb{R}^{n_x}$ with learning parameters ϕ, which is composed of an encoder $\mathrm{E}_{\phi_e} : \mathbb{R}^{n_x} \to \mathbb{R}^{n_z}$ mapping x to a lower dimensional latent space vector z and a decoder $\mathrm{D}_{\phi_d} : \mathbb{R}^{n_z} \to \mathbb{R}^{n_x}$ mapping the latent vector z to reconstruct the original vector x. Given the meta-dataset $\mathcal{D}$ in (3), we train the autoencoder $\mathcal{AE}_\phi$ by minimizing the following weighted reconstruction error over the parameters $\phi = \{\phi_e, \phi_d\}$:

$$\mathcal{L}(\phi) = \min_\phi \frac{1}{N} \sum_{i=1}^N \sum_{k=1}^K w_k^{(i)} \|x_k^{(i)} - \mathcal{AE}_\phi(x_k^{(i)})\|_2^2, \tag{4}$$

where the weights $w_k^{(i)}$ are chosen to give relatively more importance to the best candidate points $x_k^{(i)}$. In particular, given a hyperparameter $\lambda \in [0, 1)$, we define the weights as follows:

$$(j_1^{(i)}, \dots, j_K^{(i)}) := \mathrm{argsort}\left(\{f_k^{(i)}\}_{k=1}^K\right),$$
$$w_k^{(i)} := \lambda^{j_k^{(i)}}, \quad k = 1, \dots, K,$$

where the function argsort$(\cdot)$ sorts the argument values in ascending order.

After training the autoencoder, black-box optimization is performed over z in a lower-dimensional space. The decoder D_{ϕ_d} then maps the latent variable z back to the original high-dimensional space, where the function $f(x;\theta)$ is evaluated. The latent dimension n_z is a hyperparameter selected from the reconstruction error on a held-out validation set. Since the meta-dataset is generated offline, validation can be performed without consuming the online evaluation budget. The choice of n_z reflects the usual trade-off of latent-space methods: decreasing n_z degrades reconstruction, whereas increasing n_z weakens the dimensionality-reduction benefit. Notice that, when standard activation functions and finite weights are used, the decoder is continuous and locally Lipschitz, although the regularity of the induced latent objective $f(D_{\phi_d}(z);\bar{\theta})$ still depends on the black-box function f.

Remark 2. Penalty terms can be added to the loss (4) to penalize violations of the constraints defining $\mathcal{X}$, although this only encourages feasible decoder outputs and does not strictly enforce feasibility. For box constraints $\mathcal{X} := [l, q]^{n_x}$, scaled and shifted sigmoid activations may be used in the decoder output layer to restrict the reconstruction between l and q. This, however, excludes boundary points, so applications where optima are expected to lie on the boundary of the feasible set may require alternative parameterizations or projection-based constraint handling. Similarly, bounds on the latent variables can be imposed by using a sigmoid activation function in the last layer of the encoder, which rescales z to the box $\mathcal{Z} = [0, 1]^{n_z}$, restricting the search space over the latent domain, as discussed in the next section.

In the following paragraph, we provide more details on the application of the proposed approach for the implementation of GLIS in a low-dimensional space. Nevertheless, the same ideas can also be applied to other BBO algorithms, such as Bayesian Optimization.

3.3. Meta-GLIS: GLIS over a low-dimensional space

In this section, we describe how to apply the proposed methodology to implement *Meta-GLIS*, a lightweight version of GLIS (Bemporad, 2020) based on meta learning that operates in a low-dimensional latent space. First, we recall that the encoder E_{ϕ_e} and decoder D_{ϕ_d} are pretrained (e.g., in a simulation environment) using the meta-dataset $\mathcal{D}$ defined in (3). Given a query parameter $\bar{\theta}$ (which represents, for instance, the conditions of a real-world problem, rather than a simulated one), the goal is to solve the original optimization problem (1) for $\theta = \bar{\theta}$. Outside the simulation environment, evaluating the objective function

$f(x, \bar{\theta})$ is assumed to be costly. Therefore, Meta-GLIS is expected to provide a near-optimal solution within a limited number of function evaluations.

The Meta-GLIS algorithm, summarized in Algorithm 1, follows a standard surrogate-based black-box optimization strategy. First, an initial set of M_{init} samples is drawn from the latent space domain $\mathcal{Z} = [0,\ 1]^{n_z}$ using, e.g., Latin Hypercube Sampling (LHS) (Step 2). The function $f(\mathrm{D}_{\phi_d}(z); \bar{\theta})$ is then evaluated at these initial points, forming the initial dataset $\mathcal{D}_M$ (Step 3). In the iterative optimization loop (Step 4), a *surrogate model* $\hat{f} : \mathcal{Z} \to \mathbb{R}$ is constructed to approximate the objective function based on the dataset $\mathcal{D}_M = \{z_j, f(\mathrm{D}_{\phi_d}(z_j); \bar{\theta})\}_{j=1}^{M}$ (Step 4.1). As in the original formulation of GLIS, a linear combination of Radial Basis Functions (RBFs) (Gutmann, 2001) is used to create the surrogate function. Specifically,

$$\hat{f}(z) = \sum_{k=1}^{M} \beta_k \phi(\mu d(z, z_k)), \tag{5}$$

where $\phi : \mathbb{R} \to \mathbb{R}$ is an RBF, $d : \mathcal{Z} \times \mathcal{Z} \to \mathbb{R}$ is the squared Euclidean distance $d(z, z_k) = \|z - z_k\|_2^2$, $\mu > 0$ is a scalar hyper-parameter defining the shape of the RBF. The unknown coefficients $\beta = [\beta_1 \ \dots \ \beta_M]^T$ are computed by fitting the surrogate $\hat{f}(z)$ to the dataset $\mathcal{D}_M$ through the minimization of the regularized squared error:

$$\sum_{j=1}^{M} \Big\| f(\mathrm{D}_{\phi_d}(z_j; \bar{\theta})) - \underbrace{\sum_{k=1}^{M} \beta_k \phi(\mu d(z_j, z_k))}_{\hat{f}(z_j)} \Big\|^2 + \gamma \|\beta\|^2, \tag{6}$$

where the quadratic regularization term (weighted by the hyperparameter $\gamma > 0$) is added to guarantee strict convexity of (6). Some RBFs commonly used are $\phi(\mu d) = \frac{1}{1+(\mu d)^2}$ (*inverse quadratic*) and $\phi(\mu d) = e^{-(\mu d)^2}$ (*squared exponential kernel*). Both the hyperparameter μ and γ are tuned through cross-validation.

The *acquisition function* $a(z)$ is then constructed (Step 4.2) as:

$$a(z) = \hat{f}(z) - \delta h(z), \tag{7}$$

where $h(z)$ is an exploration-promoting function, and $\delta > 0$ is a tuning hyperparameter that balances exploitation and exploration. According to GLIS (Bemporad, 2020, Section 4), the exploration term is defined by the *Inverse Distance Weighting* (IDW) function:

$$h(z) = \begin{cases} 0 & \text{if } z \in \mathcal{D}_M, \\ \frac{2}{\pi} \tan^{-1}\left(\frac{1}{\sum_{i=1}^{M} \frac{1}{\|z - z_i\|^2}} \right) & \text{otherwise.} \end{cases} \tag{8}$$

The acquisition function (7) is then minimized to determine the next sampling point z_{M+1} and the corresponding decoded value $x_{M+1} = \mathrm{D}_{\phi_d}(z_{M+1})$, at which the objective function is evaluated (Steps 4.3 and 4.4). The process continues until the maximum number of iterations is reached. The best solution is then returned.

4. Sub-optimality quantification

The goal of this section is to establish probabilistic theoretical bounds that quantify the gap between solving the original problem (1) and the reduced-dimensional problem (2) over the latent set $\mathcal{Z}$.

Algorithm 1 Meta-GLIS

Input: number M_{init} of initial samples; maximum number $M_{\max} \geq M_{\text{init}}$ of function evaluations; pre-trained encoder E_{ϕ_e} and decoder D_{ϕ_d}; query parameter $\bar{\theta}$.

1: Set $M \leftarrow M_{\text{init}}$;
2: Generate M random initial samples $Z = [z_1, \dots, z_M]$ from $\mathcal{Z}$;
3: Create dataset $\mathcal{D}_M \leftarrow \{(z_j, f(\mathrm{D}_{\phi_d}(z_j; \bar{\theta})))\}_{j=1}^{M}$;
4: **while** $M < M_{\max}$ **do**:
  4.1 Compute surrogate $\hat{f}(z)$ of $f(\mathrm{D}_{\phi_d}(z); \bar{\theta})$ from $\mathcal{D}_M$;
  4.2 Compute acquisition function $a(z)$ (see Eq. (7));
  4.3 Compute next input $z_{M+1} = \arg\min_{z \in \mathcal{Z}} a(z)$;
  4.4 Evaluate $f(\mathrm{D}_{\phi_d}(z_{M+1}); \bar{\theta})$;
  4.5 Augment the dataset:
  $$\mathcal{D}_{M+1} \leftarrow \mathcal{D}_M \cup \{(z_{M+1}, f(\mathrm{D}_{\phi_d}(z_{M+1}); \bar{\theta}))\}$$
  4.6 $M \leftarrow M + 1$;
5: **end while**;
6: Compute best solution:
$$z^* = \arg\min_{z \in \mathcal{D}_M} f(\mathrm{D}_{\phi_d}(z); \bar{\theta}),$$
$$x^* \leftarrow \mathrm{D}_{\phi_d}(z^*),\ f^* \leftarrow f(x^*; \bar{\theta}).$$

Output: Best function value f^* and corresponding input x^*.

4.1. Definitions

For a given parameter θ taking values in a measurable set $\Theta \subseteq \mathbb{R}^{n_\theta}$ and a random variable $\xi \in \Xi \subseteq \mathbb{R}^{n_\xi}$, let $\mathcal{A}_x : \Theta \times \Xi \to \mathcal{X}$ be a deterministic map from (θ, ξ) to a value $x = \mathcal{A}_x(\theta, \xi)$, $x \in \mathcal{X}$. In our work, $\mathcal{A}_x$ represents a numerical optimization algorithm (e.g., gradient descent, particle swarm optimization, Bayesian optimization), ξ the random realizations characterizing the algorithm (e.g., initial conditions, mini-batches in stochastic gradient descent, etc.), and $\mathcal{A}_x(\theta, \xi)$ the solution returned by $\mathcal{A}_x$ when applied to problem (1) with parameters θ and for random algorithm's parameters ξ. Alternatively, $\mathcal{A}_x$ can be the mapping that returns a global minimizer of problem (1). In this case, $x^\star(\theta) = \mathcal{A}_x(\theta, \xi)$ depends only on the parameters θ characterizing the objective function f.[1]

Analogously, let $\mathcal{A}_z : \Theta \times \Omega \to \mathcal{Z}$ be a deterministic mapping such that $\mathcal{A}_z(\theta, \omega)$ represents the solution of the reduced-dimensional problem (2) returned by algorithm $\mathcal{A}_z$, with $\omega \in \Omega$ capturing the random settings involved in the reduced-space optimization procedure. Note that $\mathcal{A}_x$ and $\mathcal{A}_z$ do not need to be the same numerical algorithms.

We measure the relative gap in performance when optimizing in the reduced space via the *relative performance gap* defined as:

$$\Psi(\theta, \xi, \omega) := \frac{f(\mathrm{D}(\mathcal{A}_z(\theta, \omega)); \theta) - f(\mathcal{A}_x(\theta, \xi); \theta)}{|f(\mathcal{A}_x(\theta, \xi); \theta)| + \epsilon}, \tag{9}$$

where $\epsilon > 0$ is a small constant introduced to avoid possible divisions by zero. In the special case

$$\mathcal{A}_x(\theta, \xi) = x^\star(\theta), \qquad \mathcal{A}_z(\theta, \omega) = z^\star(\theta), \tag{10}$$

the performance gap $\Psi(\theta, \xi, \omega)$ measures the theoretical loss between the optimal solution of problem (1) and that of the reduced-dimensional problem (2). However, this value can be computed only if problems (1) and (2) can be solved exactly at

[1] In case of multiple global optima, $x^\star(\theta)$ may also depend on some random variables ξ if $x^\star(\theta)$ is selected at random from the set of global optimizers.

the global optimum. More generally, when $\mathcal{A}_x$ and $\mathcal{A}_z$ denote numerical optimization algorithms, the performance gap may also take negative values as the global optimum of problem (1) might not be reached by the algorithm $\mathcal{A}_x$.

Let $p(\xi)$ and $p(\omega)$ be distributions over the random variables ξ and ω, and $\{(\theta^{(i)}, \xi^{(i)}, \omega^{(i)})\}_{i=1}^{m}$ be a validation set of i.i.d. draws from product measure $p(\theta) \otimes p(\xi) \otimes p(\omega)$. The validation set thus corresponds to m realizations of the performance gap Ψ, namely, with $i = 1, \dots, m$,

$$f^{(i)} := f(x(\theta^{(i)}, \xi^{(i)}); \theta^{(i)}), \quad \Psi^{(i)} := \Psi(\theta^{(i)}, \xi^{(i)}, \omega^{(i)}). \tag{11}$$

Finally, let $(\theta^{\text{new}}, \xi^{\text{new}}, \omega^{\text{new}})$ be an independent draw from $p(\theta) \otimes p(\xi) \otimes p(\omega)$, not used during training or validation, with corresponding objective value f^{new} and performance gap Ψ^{new}.

We denote by $\mathbb{P}$ the underlying probability measure induced by the product distribution $p(\theta) \otimes p(\xi) \otimes p(\omega)$. Accordingly, for any measurable set $A \subseteq \mathbb{R}$, we write

$$\mathbb{P}_\Psi(A) := \mathbb{P}(\Psi \in A) = \mathbb{P}\left(\Psi^{-1}(A)\right) \tag{12}$$

to denote the probability distribution of the random variable Ψ. Similarly, let $S = \{\Psi^{(i)}\}_{i=1}^{m}$ denote the validation set. We write $\mathbb{P}_S$ for the probability distribution of S, induced by the i.i.d. draws $\{(\theta^{(i)}, \xi^{(i)}, \omega^{(i)})\}_{i=1}^{m} \sim p(\theta) \otimes p(\xi) \otimes p(\omega)$.

4.2. Generalization bounds on the performance gap

Theorem 1 (*Generalization Bounds*). *For any $\alpha, \delta \in (0, 1)$, let $k^\star := \left\lceil m(1 - \alpha + \varepsilon_m) \right\rceil$ and $\varepsilon_m := \sqrt{\frac{\log(2/\delta)}{2m}}$. Assume that the number of validation samples m is large enough so that $\alpha \geq \varepsilon_m$ (or equivalently, $k^\star \leq m$). Then, with probability at least $1 - \delta$ over the validation set $S = \left\{\Psi^{(i)}\right\}_{i=1}^{m}$, we have*

$$\mathbb{P}_\Psi\left(\Psi \leq \Psi_{(k^\star)}\right) \geq 1 - \alpha, \tag{13}$$

where $\Psi_{(k)}$ denotes the kth order statistic of the validation values, i.e., if

$$\Psi_{(1)} \leq \Psi_{(2)} \leq \cdots \leq \Psi_{(m)}, \tag{14}$$

then $\Psi_{(k)}$ is the kth smallest value in the ordered sequence. Equivalently,

$$\mathbb{P}_S\left(\mathbb{P}_\Psi\left(\Psi \leq \Psi_{(k^\star)} \,\middle|\, S\right) \geq 1 - \alpha\right) \geq 1 - \delta. \quad \blacksquare \tag{15}$$

The proof of the theorem is reported in Appendix.

Theorem 1 has the following interpretation. To construct the validation set, we generate m i.i.d. realizations $\{(\theta^{(i)}, \xi^{(i)}, \omega^{(i)})\}_{i=1}^{m}$ from the distribution product $p(\theta) \otimes p(\xi) \otimes p(\omega)$. For each triple $(\theta^{(i)}, \xi^{(i)}, \omega^{(i)})$, we run the two optimization algorithms $\mathcal{A}_x$ in the original space (e.g., GLIS) and $\mathcal{A}_z$ in the reduced latent space (e.g., Meta-GLIS), and we compute the corresponding performance gap

$$\Psi^{(i)} := \Psi(\theta^{(i)}, \xi^{(i)}, \omega^{(i)}), \qquad i = 1, \dots, m. \tag{16}$$

The theorem guarantees that, with probability at least $1 - \delta$ over the draw of the validation set S, the order statistic $\Psi_{(k^\star)}$ provides an upper bound on the $(1-\alpha)$-quantile of the true distribution of Ψ. Equivalently, for a new independent draw $(\theta^{\text{new}}, \xi^{\text{new}}, \omega^{\text{new}}) \sim p(\theta) \otimes p(\xi) \otimes p(\omega)$ and corresponding performance gap $\Psi^{\text{new}} = \Psi(\theta^{\text{new}}, \xi^{\text{new}}, \omega^{\text{new}})$, we have

$$\mathbb{P}_\Psi\left(\Psi^{\text{new}} \leq \Psi_{(k^\star)} \,\middle|\, S\right) \geq 1 - \alpha, \tag{17}$$

and this condition holds with probability at least $1 - \delta$ with respect to the randomness of the validation set S.

Remark 3. The performance gap Ψ in (9) is an offline validation metric, not an operational quantity to be computed at deployment. It can be evaluated once on a held-out validation set to certify the latent-space optimizer against a reference method. After validation, only the reduced-space optimizer is run on a new problem instance.

5. Numerical examples

We evaluate the performance of Meta-GLIS on two numerical examples. The first involves the Rosenbrock function (Rosenbrock, 1960), a well-established benchmark problem in nonlinear optimization. The second focuses on the automated calibration of a set of MPC tuning knobs. Computations are carried out on a server equipped with an NVIDIA RTX 3090 GPU. To guarantee reproducibility of the results, all Python scripts are made available in the GitHub repository https://github.com/buswayne/meta-glis.

5.1. Rosenbrock function benchmark

The Rosenbrock function to be minimized is defined as:

$$f(x) = \sum_{i=1}^{n_x - 1} \left[\theta_1 \left(x_{i+1} - x_i^2\right)^2 + \theta_2(\theta_{3,i} - x_i)^2\right], \tag{18}$$

where $\theta_1 \in \mathbb{R}, \theta_2 \in \mathbb{R}, \theta_3 \in \mathbb{R}^{n_x - 1}$ are random realizations sampled, respectively, from the following uniform distributions $\mathcal{U}_{[10,1000]}, \mathcal{U}_{[0.1,10]}, (\mathcal{U}_{[0.1,10]}, \dots, \mathcal{U}_{[0.1,10]})$. For the nominal case, $\theta = [100, 1, \mathbf{1}]$, where $\mathbf{1} \in \mathbb{R}^{n_x}$ is a vector with all entries equal to 1, the function is non-convex and has a narrow, curved valley containing the global minimum at $(x_1^\star, \dots, x_{n_x}^\star) = (1, \dots, 1)$.

The meta-dataset is obtained from $N = 500$ instances of the parameter θ, sampled from the uniform distributions described above, with $n_x = 20$. We solved each problem by employing a *differential evolution* algorithm (Storn & Price, 1997) with 1000 generations over the feasible input domain $\mathcal{X} = [-2.5, 2.5]^{20}$. Differential evolution is used here only to generate high-quality offline samples for the Rosenbrock benchmark, whose analytical expression makes such offline optimization inexpensive. This is distinct from the online budget used to compare GLIS and Meta-GLIS. We then created a meta-dataset $\mathcal{D} = \left\{\theta^{(i)}, \{x_k^{(i)}, f_k^{(i)}\}_{k=1}^{1000}\right\}_{i=1}^{500}$. Similarly, we generated a separate test dataset $\mathcal{D}^{\text{test}} = \left\{\theta^{(i)}, \{x_k^{(i)}, f_k^{(i)}\}_{k=1}^{1000}\right\}_{i=1}^{100}$ to evaluate the reconstruction capabilities of an autoencoder with latent dimension $n_z = 3$, when trained as discussed in Section 3.2. The encoder and decoder are fully connected neural networks, each with two hidden layers. The encoder has n_x inputs, a first hidden layer with 128 units, a second hidden layer with 64 units, and n_z outputs. The decoder has a symmetric structure. The time required to train the autoencoder is approximately 20 min.

In Fig. 1 we can see the candidate solutions generated by the differential evolution algorithm (blue points), along with the optimal solution for each problem instance (green points) and the autoencoder output for each candidate solution (orange points). We can see that the reconstructed inputs lie in a manifold with reduced dispersion, which accurately describes the region where optimal solutions lie.

The Meta-GLIS algorithm is applied in the learned $n_z = 3$-dimensional latent space, with $M_{\text{init}} = 2n_z = 6$ initial random points generated using LHS within the box $\mathcal{Z} = [0, 1]^3$. For comparison, optimization is also performed using GLIS over the original 20-dimensional input domain. In this case, $2n_x = 40$ initial points are randomly generated within the box $[-2.5, 2.5]^{20}$ using LHS. A Monte Carlo analysis is conducted on 100 newly

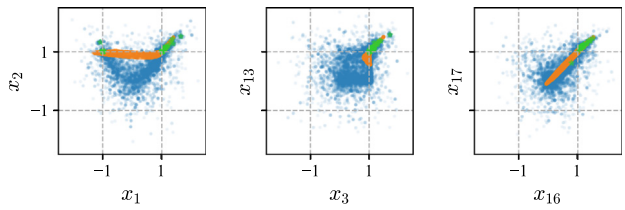


Fig. 1. Learned manifold of the trained autoencoder over test functions: candidate solutions generated by the differential evolution algorithm (blue points); optimal solution for each problem instance (green points); output of the autoencoder for each candidate solution (orange points). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

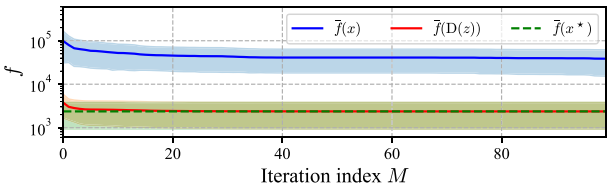


Fig. 2. Comparison of the average performance obtained with GLIS (blue) and Meta-GLIS (red) over $N^{\text{test}} = 100$ runs. Global optimum (green). Shaded areas correspond $\pm$ one standard deviation over 100 Monte Carlo runs. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

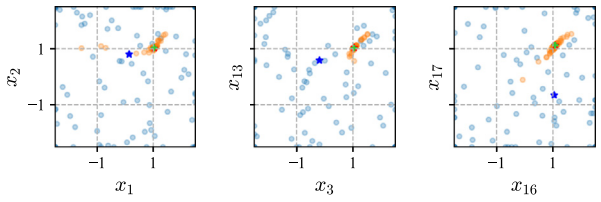


Fig. 3. Sampled points by GLIS (blue circles) and Meta-GLIS (orange circles), along with corresponding best solutions (stars). The global optimal solution (green cross) overlaps with the orange star. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

generated problem instances that were not used for training the autoencoder.

The obtained numerical results are reported in Fig. 2, which shows the best value of the Rosenbrock objective function achieved at each iteration of the GLIS and Meta-GLIS algorithm. The figure presents both the average and standard deviation over 100 Monte Carlo runs. The results highlight the advantages of Meta-GLIS, which achieves near-optimal solutions after approximately 20 iterations, whereas GLIS remains far from the global optimum even after 100 iterations. This underscores the well-known advantages of running GLIS, and more in general BBO.

This advantage can be formally quantified by applying the theoretical guarantees of Theorem 1. We generated $M = 1000$ validation instances of the optimization problem and solved them using GLIS, and Meta-GLIS, which allows us to compute the corresponding probabilistic bounds on the performance gap Ψ. For $\delta = 0.05$ and $\alpha = 0.1$, the computed bound on the performance gap between Meta-GLIS and standard GLIS (operating on the full space x) demonstrates a clear advantage: 90% of the test instances achieve at least a 68.87% relative improvement when optimization is performed in the latent space (blue dashed line in Fig. 4). This result can qualitatively be appreciated also in Fig. 3, which illustrates a single instance of the Monte Carlo runs. The figure shows the points sampled by GLIS and Meta-GLIS over the algorithm's iterations, projected onto three planes as in Fig. 1. It is

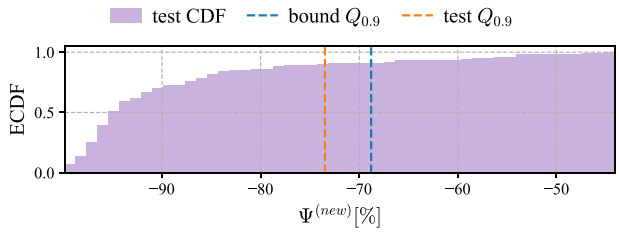


Fig. 4. Empirical CDFs of the performance gap using Meta-GLIS compared with GLIS over 100 test instances. Theoretical bound from Theorem 1 (blue line); empirical 90%-quantile on the test set (orange line). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

evident that in GLIS, which directly optimizes over x, the sampled points (blue circles) after 100 iterations almost uniformly cover the search space $\mathcal{X}$. In contrast, in Meta-GLIS, where optimization is performed over a lower-dimensional space, the sampled points (orange circles) remain within the learned manifold by construction, and the solution (orange stars) converges much closer to the optimal value.

5.2. Automated calibration of MPC hyperparameters

As another case study, we consider the same problem in Forgione et al. (2020), which uses GLIS for automatically tuning some hyperparameters of a model predictive controller. The system to be controlled is a cart–pole described by the differential equations:

$$(M+m)\ddot{p} + mL\ddot{\psi} - mL\dot{\psi}^2 \sin\psi + b\dot{p} = F, \tag{19}$$

$$L\ddot{\psi} + \ddot{p}\cos\psi - g\sin\psi + f_\psi \dot{\psi} = 0, \tag{20}$$

with p, ψ and F being, respectively, the cart position (m), the pendulum angle (rad), and the input force on the cart (N). The force F is regulated by the MPC to track a position reference p^{ref} (dashed red line in Fig. 7), while maintaining the pole in a vertical position (i.e., $\psi \approx 0$) for a pre-defined duration of the experiment T^{exp}. In addition, whenever: (*i*) the pendulum falls ($|\psi| > \frac{\pi}{6}$), (*ii*) the cart position exceeds the track length ($|p| > 1.1$ m), or (*iii*) numerical errors occur in solving the MPC problem, the controller is stopped at a time $T^{\text{stop}} < T^{\text{exp}}$.

For the implementation of the MPC, the continuous-time nonlinear dynamics (19) is discretized with a sampling time T_s^{MPC}, which must be greater than the time required for solving the MPC control law $T_{\text{calc}}^{\text{MPC}}$. At each time step t, the MPC provides, with a receding-horizon strategy, the input minimizing the objective function:

$$\begin{aligned} J_{\text{MPC}} = \sum_{k=0}^{N_p-1} \Big(& (y_{t+k|t} - y_{t+k}^{\text{ref}})^T Q_y (y_{t+k|t} - y_{t+k}^{\text{ref}}) \\ & + (u_{t+k|t} - u_{t+k}^{\text{ref}})^T Q_u (u_{t+k|t} - u_{t+k}^{\text{ref}}) \\ & + \Delta u_{t+k|t}^T Q_{\Delta u} \Delta u_{t+k|t} \Big) + Q_\epsilon \eta^2, \end{aligned} \tag{21}$$

subject to input and output constraints (see Forgione et al. (2020) for details). In (21), $y = [p,\ \psi]$; $u = F$; N_p is the prediction horizon; u_t^{ref} and y_t^{ref} are the input and output reference; respectively, at a generic time step t; η is a slack variable used to soften the input and output constraints; and Δu_t is the input variation.

The goal of this case study is to tune the following $n_x = 14$ hyperparameters within the intervals visualized in Fig. 5: the positive semidefinite weight matrices of the MPC cost $Q_y = \begin{pmatrix} q_{y_{11}} & 0 \\ 0 & q_{y_{22}} \end{pmatrix}$, $Q_{\Delta u} \in \mathbb{R}$; prediction N_p and control horizon N_u;

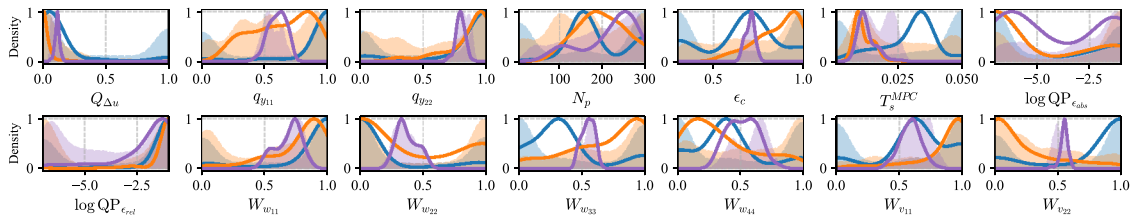


Fig. 5. Empirical distributions of the optimal solutions over 100 test system realizations (solid lines) and corresponding 3000 queries sampled during the optimization (shaded areas) for each MPC hyperparameter: GLIS (blue); Meta-GLIS with nonlinear (orange) and linear decoder (purple). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

MPC sampling time T_s^{MPC}; log of the absolute and relative tolerances of the QP solver $\log \mathrm{QP}_{\epsilon_{abs}}$, $\log \mathrm{QP}_{\epsilon_{rel}}$ used to minimize (21); process and measurement noises of diagonal covariance matrices $W_w \in \mathbb{R}^{4\times4}$ and $W_v \in \mathbb{R}^{2\times2}$ and used by a Luenberger observer to reconstruct the system state. The hyperparameter-calibration problem is formalized as the minimization of the following closed-loop performance index over the entire duration T_{exp} of the experiment:

$$\tilde{J}^{\mathrm{cl}} = \ln\left(\int_{t=0}^{T_{\mathrm{exp}}} 10|p^{\mathrm{ref}}(t) - p(t)| + 30|\psi(t)|\, dt\right) + \ell\left(T_{\mathrm{calc}}^{\mathrm{MPC}} - T_{\mathrm{s}}^{\mathrm{MPC}}\right) + \ell'(T_{\mathrm{exp}} - T_{\mathrm{stop}}), \tag{22}$$

where the first term penalizes the tracking error and oscillations of the pendulum, while ℓ and ℓ' are barrier functions penalizing, respectively, unfeasible real-time implementation of the control law (i.e., $T_{\mathrm{calc}}^{\mathrm{MPC}} \geq T_{\mathrm{s}}^{\mathrm{MPC}}$) and early stopping of the experiment (i.e., $T_{\mathrm{stop}} < T_{\mathrm{exp}}$).

To generate the meta-dataset $\mathcal{D}$, we sample the parameters $\theta = [M, m, b, f_\psi, L]^\top$ of the cart–pole dynamics equations from the uniform distribution over $\mathcal{U}_{[0.5,2.5]} \times \mathcal{U}_{[0.2,1.0]} \times \mathcal{U}_{[0.1,0.5]} \times \mathcal{U}_{[0.1,0.5]} \times \mathcal{U}_{[0.3,1.5]}$, generating a total of $N = 200$ system configurations. In the generation of $\mathcal{D}$, the cost function (22) is minimized with GLIS over 500 iterations, leading to $\mathcal{D} = \left\{\theta^{(i)}, \{x_k^{(i)}, f_k^{(i)}\}_{k=1}^{500}\right\}_{i=1}^{200}$ with $200 \times 500 = 100,000$ points. These explored points are used for training an autoencoder with the same architecture as the previous example ($n_z = 3$), with $\lambda = 0.5$.

Meta-GLIS is then applied on $N^{\mathrm{test}} = 100$ test cases, where the cart–pole dynamics are sampled from the same distributions as those used for training. As discussed in Remark 1, both the meta-dataset and the test evaluations are simulated in this proof-of-concept study. However, the reported iteration counts only refer to the online budget used to tune a new system instance.

For a fair comparison, Meta-GLIS is evaluated against standard GLIS that also leverages information from the meta-dataset $\mathcal{D}$. In particular, we initialize GLIS with the optimal hyperparameter vector associated to the central parameter $\theta_0 = [1.5, 0.6, 0.3, 0.3, 0.9]$. From Fig. 6, we can see that Meta-GLIS (orange line) starts with a higher cost with respect to GLIS, but it quickly converges to cost values $\tilde{J}^{\mathrm{cl}} \leq 0$ at around iteration 20, where the cart position p has oscillations that are very small around the reference p^{ref}. This can be visualized in the simulations in Fig. 7, where the shaded areas associated to GLIS (blue) and Meta-GLIS (orange) represent the standard deviation of the trajectories over the 100 tests. Indeed, the orange shaded area is very narrow and almost coincides with the reference signal. Thus, in this case study the main advantage of Meta-GLIS is not a substantially different final trajectory, but faster and more consistent calibration under a limited online evaluation budget.

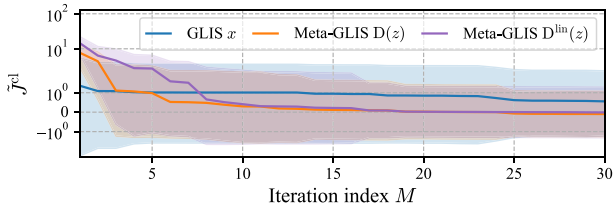


Fig. 6. Closed-loop performance index *vs* number of algorithm's iterations. GLIS (blue); Meta-GLIS (orange); Meta-GLIS with linear decoder (purple). Average (solid lines) and standard deviation (shaded areas) over 100 system realizations. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

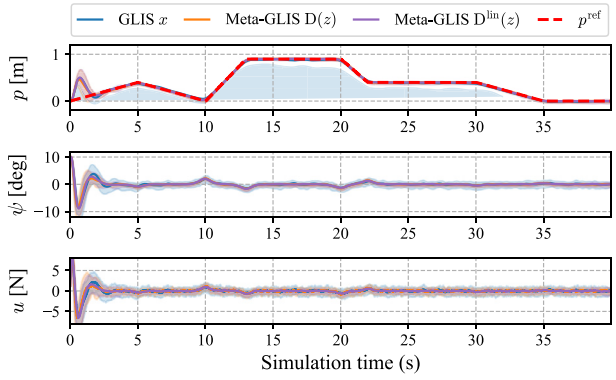


Fig. 7. MPC simulation over 100 test systems: achieved performance with hyperparameters calibrated through GLIS (blue); Meta-GLIS (orange) and Meta-GLIS with linear Decoder (purple). Average (solid lines); standard deviations (shaded areas); position reference (dashed red line). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

5.3. Interpretation via a linear decoder

The learned latent space can also provide qualitative information on which original variables are most relevant for the optimization. To illustrate this, we replace the nonlinear decoder with a linear one, $\mathrm{D}_{\phi_d}^{\mathrm{lin}}(z) = Az + b$, while keeping the encoder and $n_z = 3$ unchanged. This ablation is not part of the main method, but makes the effect of the latent variables directly readable from the column-normalized matrix A.

As shown in Figs. 6 and 7, the linear decoder still outperforms GLIS, with only moderate degradation relative to the nonlinear decoder. The largest entries of the column-normalized matrix A correspond to N_p and the QP-solver log-tolerances $\mathrm{QP}_{\epsilon_{abs}}$ and

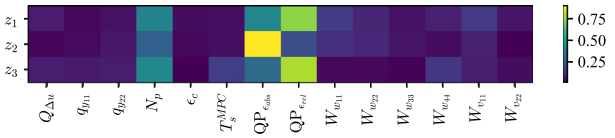


Fig. 8. Magnitude of the weights of the normalized matrix $A^\top$ characterizing the linear decoder.

$\mathrm{QP}_{\epsilon_{rel}}$ (Fig. 8), indicating that the learned manifold mainly allocates search variability to these hyperparameters. This is consistent with Fig. 5, where broader exploration is retained mainly for the most influential variables.

6. Conclusions

We presented a method for reducing the dimension of the decision variable space in black-box optimization. The key idea is to learn, via an autoencoder, a low-dimensional embedding of precomputed near-optimal solutions sampled from a class of problems, and then perform online optimization in this embedded space using experiment-driven techniques such as Bayesian Optimization or GLIS. As shown on the Rosenbrock benchmark and on automated MPC hyperparameter calibration, this substantially reduces the number of online experiments required compared to optimizing in the original space.

Future research will extend the methodology in several directions. On the methodological side, we aim to integrate the approach with preference-based optimization, where objective functions are evaluated through qualitative comparisons rather than measured directly. On the theoretical side, the generalization bounds could be sharpened and extended to broader meta-learning scenarios such as amortized optimization and reduced-space system identification.

Appendix

In this appendix we provide the proof of Theorem 1. First, a set of useful result is provided.

Property 1 (*i.i.d. Structure of the Performance Gap*)**.** *Let* $(\theta^{(i)}, \xi^{(i)}, \omega^{(i)})$, $i = 1, \dots, m$, *be i.i.d. draws from the product* $p(\theta) \otimes p(\xi) \otimes p(\omega)$. *Define the performance gaps*

$$\Psi^{(i)} \coloneqq \Psi\big(\theta^{(i)}, \xi^{(i)}, \omega^{(i)}\big), \qquad i = 1, \dots, m, \tag{23}$$

as in (9). *Then* $\{\Psi^{(i)}\}_{i=1}^m$ *are i.i.d. real-valued random variables.*

Proof of Property 1. By hypothesis, $(\theta^{(i)}, \xi^{(i)}, \omega^{(i)})$ are i.i.d. samples drawn from the product distribution $p(\theta) \otimes p(\xi) \otimes p(\omega)$. For each triple (θ, ξ, ω), the performance gap $\Psi(\theta, \xi, \omega)$ is obtained by applying the same deterministic transformation, consisting of the optimization algorithms $\mathcal{A}_x$ and $\mathcal{A}_z$, the decoder D, and the objective function f. Since applying a fixed deterministic transformation to i.i.d. inputs preserves both independence and identical distribution, the resulting values

$$\Psi^{(i)} = \Psi(\theta^{(i)}, \xi^{(i)}, \omega^{(i)}), \qquad i = 1, \dots, m, \tag{24}$$

are themselves i.i.d. real-valued random variables. ■

The following lemma by Dvoretzky–Kiefer–Wolfowitz (DKW), and refined by Massart in Massart (1990) establishes that the empirical cumulative distribution function $\hat{F}_m$ of m i.i.d. samples uniformly approximates the true cumulative distribution function F of the underlying random variable Ψ. Specifically, with probability at least $1-\delta$, the maximum deviation $\sup_t |\hat{F}_m(t) - F(t)|$ is bounded by $\varepsilon_m = \sqrt{\log(2/\delta)/(2m)}$. This result provides a non-asymptotic, distribution-free guarantee on the accuracy of the empirical CDF.

Lemma 1 (*DKW–Massart Inequality (Massart, 1990)*)**.** *Let* Ψ *be a real-valued random variable with* cumulative density function *(CDF)* F *and let* $\hat{F}_m$ *be the empirical CDF of i.i.d. samples* $\Psi^{(1)}, \dots, \Psi^{(m)}$ *defined as:*

$$\hat{F}_m(t) = \frac{1}{m} \sum_{i=1}^{m} \mathbf{1}[\Psi^{(i)} \le t], \quad t \in \mathbb{R}, \tag{25}$$

with **1** *denoting the indicator function.*

Then, for any $\delta \in (0, 1)$, *with*

$$\varepsilon_m \coloneqq \sqrt{\frac{\log(2/\delta)}{2m}}, \tag{26}$$

it holds that

$$\mathbb{P}_S\left(\sup_{t \in \mathbb{R}} \left|\hat{F}_m(t) - F(t)\right| \le \varepsilon_m\right) \ge 1 - \delta, \tag{27}$$

where $S \coloneqq \{\Psi^{(i)}\}_{i=1}^m$ *is the set of i.i.d. samples of* Ψ.

In the following section, we provide the proof of Theorem 1.

Proof of Theorem 1. We proceed in two steps.

Step 1. The goal of this step is to show that the true quantile $Q_{1-\alpha} \coloneqq \inf\{t : F(t) \ge 1-\alpha\}$ (for any $\alpha \in (0,1)$) can be upper bounded, with high probability (i.e., with probability at least $1-\delta$, for arbitrary $\delta \in (0,1)$), by a suitable order statistic of the validation samples $S = \{\Psi^{(i)} \coloneqq \Psi(\theta^{(i)}, \xi^{(i)}, \omega^{(i)})\}_{i=1}^m$, with

$$(\theta^{(i)}, \xi^{(i)}, \omega^{(i)}) \sim p(\theta) \otimes p(\xi) \otimes p(\omega). \tag{28}$$

Let $t^\star \coloneqq \Psi_{(k^\star)}$ be the $k^\star$-th order statistic of the validation set. By construction, at least $k^\star$ samples satisfy $\Psi^{(i)} \le t^\star$, hence by definition of the empirical CDF in Eq. (25), we have:

$$\hat{F}_m(t^\star) \ge \frac{k^\star}{m}, \tag{29}$$

where the inequality (and not equality) holds because multiple samples $\Psi^{(i)}$ may coincide with $t^\star$.

Define the event

$$\mathcal{E} \coloneqq \left\{\sup_{t \in \mathbb{R}} |\hat{F}_m(t) - F(t)| \le \varepsilon_m\right\}. \tag{30}$$

On $\mathcal{E}$, we have

$$F(t^\star) \ge \hat{F}_m(t^\star) - \varepsilon_m \ge \frac{k^\star}{m} - \varepsilon_m \ge 1 - \alpha, \tag{31}$$

where the last inequality follows from the definition of $k^\star$.

By definition of the true quantile, inequality (31) implies

$$Q_{1-\alpha} \le t^\star = \Psi_{(k^\star)}. \tag{32}$$

Let us set $\varepsilon_m \coloneqq \sqrt{\frac{\log(2/\delta)}{2m}}$. Since $\{\Psi^{(i)}\}_{i=1}^m$ are i.i.d. random variables (Property 1), the DKW–Massart inequality (Lemma 1) can be applied. This implies that for any $\delta \in (0,1)$, the event $\mathcal{E}$ holds with probability at least $1-\delta$ over the draw of the validation set S. Thus,

$$\mathbb{P}_S\big(Q_{1-\alpha} \le \Psi_{(k^\star)}\big) \ge 1 - \delta. \tag{33}$$

Step 2. We now leverage the quantile bound from Step 1 to control the performance on a new draw $(\theta^{\text{new}}, \xi^{\text{new}}, \omega^{\text{new}}) \sim p(\theta) \otimes p(\xi) \otimes p(\omega)$. By definition of the $(1-\alpha)$-quantile,

$$\mathbb{P}_\Psi\big(\Psi^{\text{new}} \le Q_{1-\alpha}\big) = 1 - \alpha. \tag{34}$$

If $\mathcal{E}$ holds, then $Q_{1-\alpha} \le \Psi_{(k^\star)}$, and hence

$$\{\Psi^{\text{new}} \le \Psi_{(k^\star)}\} \supseteq \{\Psi^{\text{new}} \le Q_{1-\alpha}\}. \tag{35}$$

Therefore,

$$\mathbb{P}_{\Psi}\left(\Psi^{\text{new}} \leq \Psi_{(k^\star)} \,\middle|\, S\right) \;\geq\; \mathbb{P}_{\Psi}\left(\Psi^{\text{new}} \leq Q_{1-\alpha}\right) = 1-\alpha. \tag{36}$$

Taking the outer probability over the validation samples S, we obtain

$$\mathbb{P}_{S}\left(\mathbb{P}_{\Psi}\left(\Psi^{\text{new}} \leq \Psi_{(k^\star)} \,\middle|\, S\right) \geq 1-\alpha\right) \;\geq\; 1-\delta, \tag{37}$$

which completes the proof. ■

References

Amos, Brandon (2023). Tutorial on amortized optimization. *Foundations and Trends in Machine Learning*, *16*(5), 592–732.

Antonova, Rika, Rai, Akshara, Li, Tianyu, & Kragic, Danica (2020). Bayesian optimization in variational latent spaces with dynamic compression. In *Proceedings of machine learning research*: *vol. 100*, *Proceedings of the conference on robot learning* (pp. 456–465).

Audet, Charles, & Dennis, John E., Jr. (2006). Mesh adaptive direct search algorithms for constrained optimization. *SIAM Journal on Optimization*, *17*(1), 188–217.

Banerjee, Somrita, Cauligi, Abhishek, & Pavone, Marco (2025). Deep learning warm starts for trajectory optimization on the international space station. In *2025 international conference on space robotics (iSpaRo)* (pp. 394–401).

Bemporad, A. (2020). Global optimization via inverse distance weighting and radial basis functions. *Computational Optimization and Applications*, *77*, 571–595.

Eismann, Stephan, Levy, Daniel, Shu, Rui, Bartzsch, Stefan, & Ermon, Stefano (2018). Bayesian optimization and attribute adjustment. In *Proceedings of the 34th conference on uncertainty in artificial intelligence* (pp. 1042–1052). AUAI Press.

Eriksson, David, & Jankowiak, Martin (2021). High-dimensional Bayesian optimization with sparse axis-aligned subspaces. In *Proceedings of the 37th conference on uncertainty in artificial intelligence*: *vol. 161*, (pp. 493–503).

Eriksson, David, Pearce, Michael, Gardner, Jacob R., Turner, Ryan D., & Poloczek, Matthias (2019). Scalable global optimization via local Bayesian optimization. *32*, In *Advances in neural information processing systems*.

Forgione, Marco, Piga, Dario, & Bemporad, Alberto (2020). Efficient calibration of embedded MPC. *IFAC-PapersOnLine*, *53*(2), 5189–5194.

Gal, Tomas (2010). *Postoptimal analyses, parametric programming, and related topics: Degeneracy, multicriteria decision making, redundancy*. Walter de Gruyter.

Garnett, Roman (2023). *Bayesian optimization*. Cambridge University Press.

Gómez-Bombarelli, Rafael, Wei, Jennifer N., Duvenaud, David, Hernández-Lobato, José Miguel, Sánchez-Lengeling, Benjamín, Sheberla, Dennis, Aguilera-Iparraguirre, Jorge, Hirzel, Timothy D., Adams, Ryan P., & Aspuru-Guzik, Alán (2018). Automatic chemical design using a data-driven continuous representation of molecules. *ACS Central Science*, *4*(2), 268–276.

Grosnit, Antoine, Tutunov, Rasul, Maraval, Alexandre Max, Griffiths, Ryan-Rhys, Cowen-Rivers, Alexander I., Yang, Lin, Zhu, Lin, Lyu, Wenlong, Chen, Zhitang, Wang, Jun, Peters, Jan, & Bou-Ammar, Haitham (2021). High-dimensional Bayesian optimisation with variational autoencoders and deep metric learning. arXiv preprint arXiv:2106.03609.

Gutmann, H. M. (2001). A radial basis function method for global optimization. *Journal of Global Optimization*, *19*, 201–227.

Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *Science*, *313*(5786), 504–507.

Jeong, Shinkyu, Murayama, Mitsuhiro, & Yamamoto, Kazuomi (2005). Efficient optimization design method using kriging model. *Journal of Aircraft*, *42*(2), 413–420.

Kandasamy, K., Schneider, J., & Póczos, B. (2015). High dimensional Bayesian optimisation and bandits via additive models. In *Proc. of the 32nd international conference on machine learning* (pp. 295–304). JMLR.org.

Letham, B., Calandra, R., Rai, A., & Bakshy, E. (2020). Re-examining linear embeddings for high-dimensional Bayesian optimization. *33*, In *Advances in neural information processing systems* (pp. 1546–1558). Curran Associates, Inc..

Maraval, Alexandre, Zimmer, Matthieu, Grosnit, Antoine, & Bou Ammar, Haitham (2023). End-to-end meta-Bayesian optimisation with transformer neural processes. *36*, In *Advances in neural information processing systems* (pp. 11246–11260).

Massart, Pascal (1990). The tight constant in the Dvoretzky–Kiefer–Wolfowitz inequality. *The Annals of Probability*, *18*(3), 1269–1283.

Maus, Natalie, Jones, Haydn, Moore, Juston, Kusner, Matt J., Bradshaw, John, & Gardner, Jacob R. (2022). Local latent space Bayesian optimization over structured inputs. *35*, In *Advances in neural information processing systems* (pp. 34505–34518).

Morita, Yasuyuki, Washio, Takashi, & Nakashima, Yuta (2023). Accelerator tuning method using autoencoder and Bayesian optimization. *Nuclear Instruments and Methods in Physics Research Section A*, *1056*, Article 168730.

Rolland, P., Scarlett, J., Bogunovic, I., & Cevher, V. (2018). High-dimensional Bayesian optimization via additive models with overlapping groups. In *Proc. of the 21st intl. conf. on artificial intelligence and statistics*: *vol. 84*, (pp. 298–307).

Rosenbrock, H. H. (1960). An automatic method for finding the greatest or least value of a function. *The Computer Journal*, *3*(3), 175–184.

Roveda, Loris, Forgione, Marco, & Piga, Dario (2020). Robot control parameters auto-tuning in trajectory tracking applications. *Control Engineering Practice*, *101*.

Sabug, Lorenzo, Jr., Ruiz, Fredy, & Fagiano, Lorenzo (2022). SMGO-Δ: balancing caution and reward in global optimization with black-box constraints. *Information Sciences*, *605*, 15–42.

Sambharya, Rajiv, Hall, Georgina, Amos, Brandon, & Stellato, Bartolomeo (2023). End-to-end learning to warm-start for real-time quadratic optimization. In *Learning for dynamics and control conference* (pp. 220–234). PMLR.

Storn, Rainer, & Price, Kenneth (1997). Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, *11*, 341–359.

Tripp, Austin, Daxberger, Erik, & Hernández-Lobato, José Miguel (2020). Sample-efficient optimization in the latent space of deep generative models via weighted retraining. *33*, In *Advances in neural information processing systems* (pp. 11259–11272).

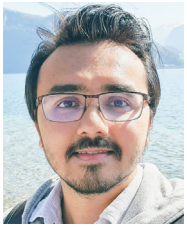
Vanschoren, Joaquin (2019). Meta-learning. In *Automated machine learning: Methods, systems, challenges* (pp. 35–61). Springer.

Wang, Z., Hutter, F., Zoghi, M., Matheson, D., & De Freitas, N. (2016). Bayesian optimization in a billion dimensions via random embeddings. *Journal of Artificial Intelligence Research*, [ISSN: 1076-9757] *55*(1), 361–387.

Ziomek, J. K., & Bou Ammar, H. (2023). Are random decompositions all we need in high dimensional Bayesian optimisation? In *Proc. of the 40th international conference on machine learning*: *vol. 202*, (pp. 43347–43368).



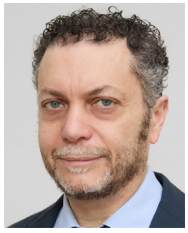
Riccardo Busetto was born in Treviso, Italy, in 1996. He received the B.Sc. and M.Sc. degrees (cum laude) in Automation and Control Engineering from Politecnico di Milano, Milan, Italy, in 2018 and 2020, respectively, and the Ph.D. degree in Information Technology — Systems and Control, also from Politecnico di Milano, in 2025. He is currently a Researcher at the IDSIA Dalle Molle Institute for Artificial Intelligence, Lugano, Switzerland. His research interests include data-driven control, system identification, and meta-learning, with applications to mechatronic, robotic, and financial systems.



Manas Mejari received his bachelors degree in Electronics and Telecommunication engineering in 2011 and his masters degree in Control Systems in 2013 from the University of Mumbai, India. He received his Ph.D. in Control Systems from IMT School for Advanced Studies Lucca, Italy, in 2018. In 2018–2019, he held a postdoctoral position at the University of Lille, France. Currently, he is a Researcher at IDSIA Dalle Molle Institute for Artificial Intelligence, SUPSI-USI, Switzerland. His research interests include system identification, control theory, machine learning and artificial intelligence.



Marco Forgione received his M.Sc. degree in Computer Engineering from the University of Pavia, Italy, in 2009, and his Ph.D. in Systems and Control Engineering from Delft University of Technology, The Netherlands, in 2014. He was a Postdoctoral Researcher at École Centrale de Lyon, France (2014–2015), and a Control Engineering Consultant at Whirlpool EMEA, Italy (2015–2018). He is currently a Senior Researcher at the IDSIA Dalle Molle Institute for Artificial Intelligence, Lugano, Switzerland. His main research interests are at the intersection of system identification, meta-learning, and control.



Alberto Bemporad received his M.S. degree cum laude in Electrical Engineering (1993) and his Ph.D. in Control Engineering (1997) from the University of Florence, Italy. He held research positions at Washington University in St. Louis (1996–1997) and ETH Zurich (1997–2002). He served at the University of Siena (1999–2009) and at the University of Trento (2010–2011). Since 2011, he has been Full Professor at the IMT School for Advanced Studies Lucca, where he was Director from 2012 to 2015. He co-founded ODYS S.r.l. in 2011, a company specialized in industrial model predictive control systems, and has held visiting appointments at Stanford University, the University of Michigan, and Zhejiang University. He has authored over 400 publications and 21 patents in model predictive control, hybrid systems, numerical optimization, nonlinear system identification, and automotive control, and co-developed the Model Predictive Control Toolbox (MathWorks) for MATLAB. He was an Associate Editor of the IEEE Transactions on Automatic Control during 2001–2004 and Chair of the IEEE CSS Technical Committee on Hybrid Systems during 2002–2010. He is an IEEE Fellow (since 2010) and IFAC Fellow (since 2025). His awards include the IFAC High-Impact Paper Award (2011–14), IEEE CSS Transition to Practice Award (2019), SAE Environmental Excellence in Transportation Award (2021), the Beale–Orchard-Hays Prize (2024), and an ERC Advanced Grant (2024).



Dario Piga received his Ph.D. in Systems and Control Engineering from Politecnico di Torino, Italy. He is currently Professor at the Dalle Molle Institute for Artificial Intelligence (IDSIA USI-SUPSI) in Lugano, Switzerland, where he is the founder and head of the LEarning for Optimization and coNtrol (LEON) research group. His research interests lie at the intersection of system identification, control theory, machine learning, and nonlinear optimization. He coordinates and serves as Principal Investigator in several research projects and collaborates with international companies on the development of innovative AI-based modeling and control solutions for applications in manufacturing, transportation, robotics, biomedical engineering, and the chemical industry. He is an Associate Editor of Automatica and Chair of the IEEE Control Systems Society Technical Committee on System Identification and Adaptive Control.