Falsification of Cyber-Physical Systems using Bayesian Optimization

FALSIFICATION OF CYBER-PHYSICAL SYSTEMS USING BAYESIAN OPTIMIZATION

Zahra Ramezani
Chalmers University of Technology
Gothenburg, Sweden
rzahra@chalmers.se

Kenan Šehić
Lund University, Lund, Sweden,
University of Sarajevo, Bosnia and Herzegovina
kenan.sehic@cs.lth.se, ksehic@etf.unsa.ba

Luigi Nardi
Lund University and Stanford University
DBtune, Sweden
luigi.nardi@cs.lth.se, lnardi@stanford.edu

Knut Åkesson
Chalmers University of Technology
Gothenburg, Sweden
knut@chalmers.se

ABSTRACT

Cyber-physical systems (CPSs) are often complex and safety-critical, making it both challenging and crucial to ensure that the system's specifications are met. Simulation-based falsification is a practical testing technique for increasing confidence in a CPS's correctness, as it only requires that the system be simulated. Reducing the number of computationally intensive simulations needed for falsification is a key concern. In this study, we investigate Bayesian optimization (BO), a sample-efficient approach that learns a surrogate model to capture the relationship between input signal parameterization and specification evaluation. We propose two enhancements to the basic BO for improving falsification: (1) leveraging local surrogate models, and (2) utilizing the user's prior knowledge. Additionally, we address the formulation of acquisition functions for falsification by proposing and evaluating various alternatives. Our benchmark evaluation demonstrates significant improvements when using local surrogate models in BO for falsifying challenging benchmark examples. Incorporating prior knowledge is found to be especially beneficial when the simulation budget is constrained. For some benchmark problems, the choice of acquisition function noticeably impacts the number of simulations required for successful falsification.

***K*eywords** Cyber-Physical Systems · Testing · Falsification · Bayesian Optimization

1 Introduction

Cyber-physical systems (CPS) [1] are frequently complex and safety-critical. Testing is a common approach for evaluating correctness, with the primary goal of identifying inputs that falsify given specifications, known as *counterexamples*.

In many industrial systems, explicit mathematical models for analysis are unavailable, and only system simulations are possible. In this paper, we assume the availability of a black-box model representing the system under test (SUT), which can be simulated.

For numerous industrial CPSs, simulations can be computationally expensive, making it desirable to minimize the number of simulations required during falsification. *Simulation-based falsification using optimization* aims to reduce the number of tests, i.e., simulations, by employing an optimization method to determine the next set of input signals based on previous simulation results. A key challenge lies in selecting an efficient optimization method and identifying the information an optimizer should use to decide on the next input signals.

When the SUT is represented as a black-box model, the optimization approach is limited to gradient-free optimization methods [2]. Optimization-based methods are generally divided into two categories: *direct-search* [3] and *model-based* methods [2]. Direct-search methods evaluate the objective function directly without sharing information between consecutive evaluations, while model-based searches create a surrogate model of the objective function to explore and exploit the search space [2, 4].

Model-based methods are suitable when the SUT is expensive to simulate, and it might be worth the additional cost of building a surrogate model. Bayesian optimization (BO) [4] is a model-based optimization method that has been successfully applied to various problems, such as machine learning hyperparameter optimization [5, 6] and robotics [7, 8].

Often BO outperforms conventional optimization when addressing costly-to-evaluate nonconvex functions with multiple local optima [4]. As falsification of CPSs is typically a high-dimensional problem, the computational requirements for vanilla BO (i.e., the standard Bayesian optimization approach) are not well suited.

In [9], it is demonstrated that BO is comparable with respect to other optimization methods, such as CMA-ES [10], for falsification of CPSs. However, [9] is using a dimensionality reduction method REMBO [11]. REMBO and its extensions, such as HeSBO [12] and ALEBO [13], project the search space to a low-dimensional subspace using a linear embedding. In these methods, a Gaussian process model is trained on a low-dimensional space, from which the original high-dimensional input space for evaluations is derived via inverse random projection. Consequently, points outside the search domain are projected to the boundary, upper or lower bound, resulting in over-exploitation of the boundary region. However, determining the optimal size of a linear embedding is a challenging task in practice. Creating an adequate linear embedding with the optimal size for different falsification problems is an open research question. In general, it is impractical for practitioners to specify linear embeddings in real-world applications without having multiple intensive trials.

Moreover, in [9], the authors use different linear embeddings for each example, and there is no clear suggestion on which embedding is optimal. Furthermore, the benchmark problems used in [9] have been updated to more recent benchmarking problems used in the falsification community [14, 15].

A different and more practical approach would be to use an approach that does not require input from the practitioner. TuRBO [16] is such a method that is based on trust regions [17]. A trust region (TR) is a subset of the input space centered at the current-best solution where the objective function is approximated locally. Instead of focusing on a linear embedding, TuRBO searches for the objective function locally with a sequence of local optimization runs. Using TRs in which a probabilistic model is trained similarly to the global BO framework helps to avoid overexploiting and thus provides a balance between exploration and exploitation. In some practical applications of falsification, the objective function can be constant in large regions or may have discontinuities. In these situations, vanilla BO will have difficulty learning anything credible. However, this limitation can be easily omitted by shrinking the search locally within a TR as done in TuRBO. Although, the final performance of BO depends crucially on the selection of the acquisition function. The acquisition function determines how exploration and exploitation are balanced in BO [18, 19, 20], an aspect that is not discussed in [9]. One acquisition function can be used to emphasize model uncertainty more than prediction, while another acquisition function can be used to accelerate convergence but might get trapped in local optima easily. To address this issue, in addition to the default choice with Thompson Sampling (TS) [18], TuRBO is modified, as part of this work, to use the lower confidence bound (LCB) [19], and an adjusted version of the probability of improvement (PI) [20] that emphasizes more failure events as our primary goal is to find configurations that falsify the SUT.

Despite being a popular method for optimizing expensive black-box functions, plain BO does not incorporate the expertise of domain experts. For certain falsification problems, there is prior knowledge about where a falsified point can potentially be located. Such an example is the corner points, that is, values at the boundaries of the allowed input ranges, that are shown to be likely to falsify many falsification problems; this is further discussed in [21]. In such situations, prior knowledge can be incorporated into the model-based method to increase efficiency. The recent works, BOPrO [22] and πBO [5] propose how to incorporate the prior injection about the optimal solution in BO, which allows the practitioner to emphasize certain regions that potentially a falsified point can be located. The methods can forget incorrect prior knowledge and eventually converge to an optimal solution. To our knowledge, injecting a prior belief about the falsifiable area has never been evaluated for falsification of CPSs. In [22], a user-provided prior distribution is combined with a data-driven model to form a pseudo-posterior. Still, this approach does not allow for arbitrary priors to be integrated. Furthermore, this method does not allow for different acquisition functions. A generalized approach, πBO, is proposed in [5] to address these issues. In contrast to other works, while being conceptually simple, πBO can easily be integrated with existing BO works and different acquisition functions. Furthermore, [5] provides a theoretical guarantee proving convergence at regular rates independently of the prior.

In this paper, we investigate how TuRBO, a BO method for high-dimensional problems, and πBO, a BO method for including prior knowledge, can be used for falsification. We propose the modification of the TuRBO method with different acquisition functions, evaluate them on benchmark problems, and compare them with other state-of-the-art methods.

These mentioned limitations and shortcomings can be one of the reasons why BO has not yet received much attention in the falsification community. Moreover, since the initial work [9] on BO was proposed a few years ago, the new standard benchmark problems for falsification have emerged [14, 15]. Therefore, a new study on BO optimization for falsification is necessary. We have formulated the following research questions in this paper. (1) How can recent contributions from the BO community be exploited to efficiently solve falsification problems? (2) Which formulation of the acquisition functions is suitable for the falsification process? (3) How can guesses about falsification points be included in the BO approach? The contributions in this paper are: (i) A discussion of recently proposed BO optimization methods that include trust regions and allow the possibility to inject prior knowledge in the problem formulation. In particular, we discuss how these new features are affecting the falsification process. (ii) A new acquisition function that is tailor-made for falsification problems is proposed. (iii) An extensive evaluation of the proposed methods and acquisition functions on a representative set of benchmark problems. The evaluations demonstrate that recent trust-region-based Bayesian optimization is, for hard problems to falsify, outperforming traditional optimization-based approaches.

This paper is organized as follows: Section 2 reviews the related work. Section 3 introduces the first falsification process briefly, then the BO method. Section 4 introduces the methods used in this paper. The evaluation results on the benchmark problems are discussed in Section 5. Finally, Section 6 summarizes the contributions.

2 Related Work

Different optimization methods have been evaluated for falsification of CPSs. In [23], the optimization methods, Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [10], Nelder-Mead (NM) [24], SNOBFIT [25], and Simulated Annealing [26] were evaluated on benchmark problems [14] to evaluate the effect of different optimization techniques on falsification problems. The evaluation shows that there is a significant difference in the efficiency of the falsification process. SNOBFIT [25], a model-based approach, demonstrated the best performance for falsification in [23]. Still some benchmark problems were not possible to falsify using this approach. On the basis of the observation of SNOBFIT's performance, it was discovered that SNOBFIT often explores new parameters at the extremes of parameter ranges (known as corner points) when it does not receive any hints from the quantitative semantics as to which direction to continue its search. This motivated the paper [21] in which a new falsification method, Line-Search falsification (LSF) [21], is proposed. LSF is a direct-search method specifically developed for falsification problems and has demonstrated better performance than SNOBFIT as described in [21]. An explanation of the good performance of LSF is the ability of LSF to merge random exploration with local search by creating random lines in the parameter space and optimizing over line segments as well as the ability to investigate corner points while searching. Although LSF outperforms previous approaches in falsifying benchmark problems, it is a direct-search method that does not attempt to learn a surrogate model of the objective function. LSF will, for this reason, be compared with BO methods since it has exhibited strong performance in solving falsification problems, as discussed in [21].

BO has been previously explored for falsification of CPSs in [9, 27, 28, 29]. In [27], BO was adapted for use with conjunctive requirements, focusing on solving problems with numerous requirements. In [28], the Gaussian process upper confidence bound (GP-UCB) was employed for falsifying conditional safety properties, which require a safety property to hold whenever an antecedent condition is met. Gaussian process regression was utilized to identify the input search space region where the antecedent condition holds. The GP-UCB algorithm for conditional safety was further enhanced in [29] by concentrating on points satisfying the antecedent. The GP-UCB approach is suitable for moderate-dimensional spaces, up to approximately 10 dimensions. However, for high-dimensional parameter spaces, this method is not feasible.

ALEBO [13] addresses REMBO's limitations, such as nonlinear distortion in the objective function and a low probability of containing an optimum within a linear embedding, by using a Mahalanobis kernel and sampling a linear embedding from the unit hypersphere. HeSBO [12] offers an alternative by defining a linear embedding through hashing and sketching. Nevertheless, the performance of both methods still heavily depends on the optimal selection of the linear embedding size, which is difficult to determine in practice.

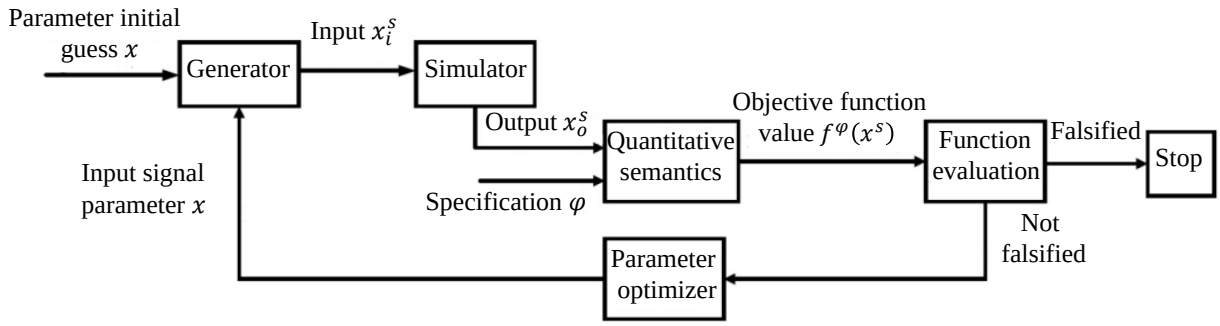

Figure 1: The process of optimization-based falsification [30].

3 Background

3.1 Falsification of Cyber-Physical Systems

The process of optimization-based falsification is shown in Figure 1. Initially, a generator creates input signals to the system based on an input parametrization. The input parameters $x \in \mathbb{R}^n$ with n, as the number of parameters, are used to generate an input trace describing a sequence of input vectors, $x_i^s[k]$, where k ranges from the start to the end of the simulation, for the input (i) signals (s). For example, a sinusoidal signal can be parameterized using the amplitude and the period or a piecewise constant signal by the values and time instants at which the signal changes value. Next, a simulator generates simulation traces of output signals x_o^s, where the SUT is simulated with the x_i^s as inputs. The combination of both x_i^s and x_o^s, i.e. x^s, are used with the specification φ, possibly containing temporal operators, to evaluate the specification using a quantitative semantics.

A quantitative semantics determines whether the specification is satisfied and a measure of to what extent the specification is fulfilled. If the specification is falsified, the process ends. However, if it is not falsified, a parameter optimizer generates a new set of parameters for the input generator, and a new simulation of the system takes place. Mapping between the input parametrization and the objective value $y \in \mathbb{R}$ of the selected quantitative semantics is taken as a black-box function $f^{\varphi}(x^s) : \mathcal{X} \longrightarrow \mathbb{R}$. In falsification, $y = f^{\varphi}(x^s) < 0$ denotes that the specification is falsified. Thus, minimizing the objective function $f^{\varphi}(x^s)$ should guide us to a point with $f^{\varphi}(x^s_{\text{falsified}}) < 0$ if the system is possible to falsify. With slight abuse of notation, we will write $f(x)$ as shorthand for f^{φ}.

In falsification, the specification is often modeled using signal temporal logic (STL) or metric interval temporal logic (MITL) with their quantitative semantics defined in [31] and [32], respectively. In this paper, STL specifications are used.

3.2 Signal Temporal Logic

The syntax of STL [33] is defined as follows:

$$\varphi ::= \mu \,|\, \neg\mu \,|\, \varphi \wedge \psi \,|\, \varphi \vee \psi \,|\, \Box_{[a,b]}\psi \,|\, \Diamond_{[a,b]}\psi \,|\, \varphi \mathcal{U}_{[a,b]}\psi,$$

where the predicate μ is $\mu \equiv \mu(x^s) > 0$ and x^s is a signal; φ and ψ are STL formulas; $\Box_{[a,b]}$ denotes the *globally* (*always*) operator between times a and b (with $a < b$); $\Diamond_{[a,b]}$ denotes the *eventually* operator between a and b; and $\mathcal{U}_{[a,b]}$ denotes the *until* operator between a and b.

The satisfaction of the formula φ for the discrete signal x^s, consisting of both inputs and outputs to the SUT, at the discrete-time instant k is defined:

$$
\begin{aligned}
(x^s, k) &\models \mu && \Leftrightarrow\ \mu(x^s[k]) > 0 \\
(x^s, k) &\models \neg\mu && \Leftrightarrow\ \neg((x^s, k) \models \mu) \\
(x^s, k) &\models \varphi \wedge \psi && \Leftrightarrow\ (x^s, k) \models \varphi \wedge (x^s, k) \models \psi \\
(x^s, k) &\models \varphi \vee \psi && \Leftrightarrow\ (x^s, k) \models \varphi \vee (x^s, k) \models \psi \\
(x^s, k) &\models \Box_{[a,b]}\varphi && \Leftrightarrow\ \forall k' \in [k+a, k+b], (x^s, k') \models \varphi \\
(x^s, k) &\models \Diamond_{[a,b]}\varphi && \Leftrightarrow\ \exists k' \in [k+a, k+b], (x^s, k') \models \varphi \\
(x^s, k) &\models \varphi\, \mathcal{U}_{[a,b]}\psi && \Leftrightarrow\ \exists k' \in [k+a, k+b]\ \ (x^s, k') \models \psi \\
& && \quad \wedge \forall k'' \in [k, k'), (x^s, k'') \models \varphi
\end{aligned}
$$

Two quantitative semantics defined for STL are *Max* and *Additive*. Both of these can be expressed in terms of VBools [30], but in this paper, we focus on using *Max* since it is the most widely used quantitative semantics in falsification. A VBool $\langle v, z\rangle$ is a combination of a Boolean value v (true $\top$, or false $\bot$) together with a real number z that is a measure of how true or false the VBool is. This real value will estimate how convincingly a test passed or how severely it failed. The quantitative semantics defines this value.

3.3 Quantitative Semantics

For *Max* semantics, the *and* ($\wedge$), *or* ($\vee$), *always* ($\Box$), *eventually* ($\Diamond$), and *until* ($\mathcal{U}$) operators are introduced.

The *and*-operator is defined as

$$
\begin{aligned}
(\top, s) \wedge (\top, z) &= \Big(\top, \min(s, z)\Big), \\
(\top, s) \wedge (\bot, z) &= (\bot, z), \\
(\bot, s) \wedge (\top, z) &= (\bot, s), \\
(\bot, s) \wedge (\bot, z) &= \Big(\bot, \max(s, z)\Big).
\end{aligned}
$$

Using the de Morgan laws, the *or* operator can be defined in terms of *and*, as: $(v_s, s) \vee (v_z, z) = \neg_{\mathrm{v}}(\neg_{\mathrm{v}}(v_s, s) \wedge \neg_{\mathrm{v}}(v_z, z))$, where VBool negation is defined as $\neg_{\mathrm{v}}(v_s, s) = (\neg v_s, s)$.

The *always* operator over an interval $[a, b]$ is straightforwardly defined in terms of *and*-operator, as

$$
\Box_{[a,b]}\varphi = \bigwedge_{k=a}^{b} \varphi\,[k]\,,
$$

where φ is a finite sequence of VBools defined for all the discrete-time instants in $[a, b]$.

Furthermore, the *eventually* operator is for both semantics defined over an interval $[a, b]$ in terms of *always*-operator, as: $\Diamond_{[a,b]}\varphi = \neg(\Box_{[a,b]}(\neg_{\mathrm{v}}\,\varphi))$.

Finally, the *Max until*-operator as:

$$
\varphi\, \mathcal{U}_{[a,b]}\ \psi = \bigvee_{k=a}^{b} \left(\psi[k] \wedge \left(\bigwedge_{k'=0}^{a-1} \varphi[k'] \right) \right).
$$

3.4 Bayesian Optimization

BO [4] assumes a probabilistic belief about the mapping between the input parametrization x and the objective value y. Designing an acquisition function guides the optimization procedure to the optimal solution by selecting adequate configurations for evaluation. While different regression models such as random forest can work as a surrogate model, a probabilistic model in BO is typically based on Gaussian processes regression [34]. In principle, Gaussian process regression defines the output of a simulation-based falsification as $p(y) = \mathcal{GP}(y; \mu', K)$, where μ' is a prediction mean value, and K is a covariance that returns similarity between points where $\sigma(x) = \sqrt{K(x, x)}$ is the marginal standard deviation of $f(x)$.

Algorithm 1 Bayesian Optimization for falsification

1: **Input:** Input space $\mathcal{X} \subseteq \mathbb{R}^n$, the initial design size M, the max number of simulations N.
2: **Output:** A falsified design - if possible, to find within the given simulation budget.
3: $\{x_i\}_{i=1}^{M} \sim \mathbb{U}(x), \{y_i \longleftarrow f(x_i)\}_{i=1}^{M}$, {*Sample random configurations from the uniform distribution and evaluate using simulation.*}
4: **if** the specification is falsified at one of $\{x_i\}_{i=1}^{M}$ **then**
5: **return** The falsified point {*Falsification is successful.*}
6: **else**
7: $\mathcal{D}_0 \longleftarrow \{(x_i, y_i)\}_{i=1}^{M}$ {*Collect the initial design set.*}
8: **for** (j = 1, 2, ..., N) **do**
9: $x^* \longleftarrow \text{argmin}_{x \in \mathcal{X}}\ \alpha(x, \mathcal{D}_{j-1})$ {*Train a probabilistic model as* $p(y|\mathcal{D}) = \mathcal{GP}(y; \mu'_{y|\mathcal{D}}, K_{y|\mathcal{D}})$ *with* $\mathcal{D}_{j-1}$ *and find the next configuration* x^* by minimization of the acquisition function α.}
10: $y^* \longleftarrow f(x^*)$ {*Evaluate the objective function by using simulation followed by the quantitative semantics.*}
11: **if** $y^* < 0$ **then**
12: **return** x^* {*Falsification is successful.*}
13: **else**
14: $\mathcal{D}_j = \mathcal{D}_{j-1} \cup \{(x^*, y^*)\}$ {*Update the sample set.*}
15: **end if**
16: **end for**
17: **end if**

We summarize the main steps of BO for falsification in Algorithm 1. At the start, we need to evaluate the objective function $f(x)$ on an initial number of samples (points) that is noted here as M numbers, line 1. Defining the initial points in the search space $\mathcal{X} \subseteq \mathbb{R}^n$, where n is the dimensional parameter space, is typically done by randomly selecting parameters within the allowed range, line 3. After evaluating $f(x)$, if the specification is falsified at any of the initial samples, lines 4-5, the algorithm terminates with the falsified point. Otherwise, we create the initial sample set $\mathcal{D}_0 \longleftarrow \{(x_i, y_i)\}_{i=1}^{M}$ based on an initial probabilistic model, line 7. Using the trained probabilistic model, we can decide which configuration to evaluate next. This process is done by minimization of the used acquisition function which will be introduced in 4.1.

A key challenge is a trade-off between exploring regions of the parameter space not yet explored versus investigating areas around the most promising parameter values yet found. This trade-off is called the exploration-exploitation dilemma. Balancing adequately between exploring yet unexplored areas and exploiting promising regions determines the efficiency of the falsification process. Too much exploitation results in a greedy optimization where a surrogate model can easily be trapped in a local minimum. Vice versa, too much exploration would result in an inefficient performance where a surrogate model evolves with every new iteration without any exploitation. Therefore, an acquisition function that balances exploration vs. exploitation is used to select the next configuration for evaluation. Since an acquisition function is built on top of a surrogate model with a clear analytical form, optimizing it can be done efficiently. In general, acquisition functions come with different concepts in exploring and exploiting, which we discuss in Section. 4.

The algorithm selects the next configuration, point x^*, for evaluation by optimizing a predefined acquisition function, line 9. Once evaluated by simulating the SUT with the generated input from the parameters x^*, the corresponding objective value y^* is checked, line 10. If $y^* < 0$, we have found a falsifying configuration, and the optimization procedure ends, lines 11-12. Otherwise, the sample set $\mathcal{D}$ is updated, lines 13-14, and the previous steps are repeated. This continues until the total number of simulations, N, is exhausted, lines 8-16.

4 Method

The cost of using the Gaussian process for the surrogate model scales cubic with the number of samples evaluated, see [34]. High-dimensional applications typically require more evaluations to converge, so using vanilla BO in this setting becomes impractical [12, 35]. Furthermore, it is noted that BO searches more the edges, the lower and upper bounds of the input ranges, of the search space in high-dimensional applications, this results in suboptimal performance [11, 16]. In [16], this is mitigated by building multiple local probabilistic models in an approach called TuRBO.

4.1 TuRBO

TuRBO, a trust-region (TR) BO method, utilizes a sequence of local optimization runs using independent probabilistic models to overcome the problem of overexploiting. Furthermore, an implicit multi-armed bandit strategy at each iteration addresses the global optimization where a local run is selected for additional evaluations. It is possible to represent a TR as a sphere, a polytope, or a hyperrectangle, with its center located at the point of the lowest objective function found so far in the optimization process. TuRBO uses Gaussian process (GP) models within a hyperrectangle TR. A hyperrectangle centered at the current best solution is created with the predefined length. Within the hyperrectangle, a local surrogate model is trained. Large enough TR would be equivalent to standard global BO methods. Therefore, TR should be large enough to encompass good solutions while remaining small enough to build an accurate local model. Consequently, there are limitations for the size of TR (L_{min}, L_{max}). The TR is expanded when a new point with a better objective function is found in that region; otherwise, TuRBO shrinks when it appears stuck. At the beginning of the TuRBO process, a base side length is initialized for TR, L_{init}. An acquisition function is used at each iteration, i, to select a batch of q candidates $x_1^{(i)}, \ldots, x_q^{(i)}$ within TR. If better points are searched consecutively within TR, the size of TR is doubled, i.e., $\min(L_{min}, 2L)$. If TuRBO fails to find better points, TR is halved in size, $L/2$. If the size of TR is less than L_{min} or greater than L_{max}, the current TR is discarded, and a new TR with L_{init} is initialized. The evaluated TuRBO method in this paper uses a single local BO strategy using a TR method in each search. TuRBO uses the best current point with the lowest objective function that has been found so far as the most promising within a local optimization run instead of just doing random restarts. This leads to a more efficient use of the evaluation budget.

Using standard acquisition functions, TuRBO finds the best-next configuration x^* for evaluation. If the best-next evaluation $f(x^*)$ is better than the current best solution, then the trust region is expanded. Otherwise, it is shrunk. The default acquisition function used in TuRBO is Thompson sampling (TS). To demonstrate the usage of TuRBO in falsification and preferably w.r.t. the global BO methods, our work covers a detailed study on the selection of an acquisition function. As part of this work, TuRBO has been modified to work with Lower Confidence Bound (LCB) [19], and a version of Probability of improvement (PI) [20].

4.1.1 Thompson Sampling (TS)

TS [18] is a simple yet effective approach for handling the exploration-exploitation dilemma in Bayesian optimization. Once we have a trained surrogate model, the concept is to greedily sample a configuration from the posterior with the lowest value, and sampling from the posterior generates TS's randomness.

4.1.2 Lower Confidence Bound (LCB)

Each prediction made by a surrogate model comes with the confidence interval explained with a corresponding standard deviation. The standard deviation is a measure of uncertainty. Thus, LCB has been introduced to leverage this measure for exploration and exploitation. It refers to the lower bound of the uncertainty of the surrogate model. In this paper, we consider the minimization problem where the lower bound is of interest; for maximization problems, the Upper Confidence Bound (UCB) is used instead. The best-next configuration x^* is [19]

$$\alpha_{\mathrm{LCB}}(x^*) \in \operatorname*{argmin}_{x \in \mathcal{X}} \mu'(x) - \beta \cdot \sigma(x), \tag{1}$$

where the parameter $\beta \in \mathbb{R} \geq 0$ balances exploitation and exploration. Note that the $\arg\min$ of a black-box function returns a set since more than one value might achieve the minimum. Small β values mean more greedy exploitation, while large values mean more exploration. Defining an optimal value for β is an open question, in this work, we use a well-known formulation used in practice [36], $\beta = \sqrt{0.125 \log(2j+1)}$, where j is the number of simulations.

4.1.3 Probability of Improvement (PI)

PI [20] defines the probability that the best-next configuration x^* leads to an improvement with respect to a target value τ, which ideally can be seen as the optimal solution $f_{\min}$. Thus, we write

$$\mathbf{P}(f(x) < \tau) = \mathbf{P}\left(\frac{f(x) - \mu'(x)}{\sigma(x)} < \frac{\tau - \mu'(x)}{\sigma(x)}\right) = \Phi\left(\frac{\tau - \mu'(x)}{\sigma(x)}\right), \tag{2}$$

where Φ is the standard normal cumulative distribution function. As the optimal solution $f_{\min}$ is unknown, τ is typically defined as the current best solution, although for falsification a negative value is required. Thus, if a potential configuration has an associated predicted value larger than the current best solution, then the optimization procedure is not improving. Scaling the difference between the current best solution and a prediction value with the standard

deviation creates the exploitation nature of PI. Because of maximizing the improvement, the next-best x^* is

$$\alpha_{\mathrm{PI}}(x^*) \in \underset{x\in\mathcal{X}}{\operatorname{argmax}}\ \Phi\left(\frac{\tau-\mu'(x)}{\sigma(x)}\right). \tag{3}$$

The fraction in (3) is sometimes referred to as the *U*-function [37]. Instead of maximizing (3), we can minimize the *U*-function as

$$\alpha_{\mathrm{U}}(x^*) \in \underset{x\in\mathcal{X}}{\operatorname{argmin}} -\frac{\tau-\mu'(x)}{\sigma(x)} = \underset{x\in\mathcal{X}}{\operatorname{argmin}} \frac{\mu'(x)-\tau}{\sigma(x)}. \tag{4}$$

The formulation in (4) with the absolute value of $|\mu'(x^*)-\tau|$ is also found in reliability analysis, where the objective is to approximate the probability of failure. Using the absolute value here improves the threshold between failure and non-failure events. In reliability analysis, the best-next configuration x^* close to τ from any side is sufficient for evaluating [37, 38]. As we are only interested in falsification for CPSs (i.e., having values less than 0), defining $\tau \leq 0$ emphasizes failure events [37, 38]. In the present study, we experiment with two choices, $\tau = 0$ and $\tau = -1$.

4.2 Incorporating prior belief

In certain situations, the practitioner has an available prior belief about the potential location of the optimum [22, 5]. While this source of information might be available, vanilla BO fails to incorporate it. Therefore, the work previously done in [22, 5] proposed how to modify BO to inject this prior. In particular, the latest algorithm πBO [5] is conceptually simpler than the previous work. The objective is to modify an acquisition function by multiplying it with a predefined probability distribution $\pi(x)$ as

$$x^* \in \underset{x\in\mathcal{X}}{\operatorname{argmax}}\, \alpha(x, \mathcal{D}_{j-1}) \cdot \pi(x)^{\beta^*/j}. \tag{5}$$

for the j-th iteration with $j \in \{1, \ldots, N\}$. $\beta^* \in \mathbb{R}^+$ is a hyperparameter that is used as the practitioner's confidence about the prior knowledge. Here, a probability distribution $\pi(x)$ serves to describe our belief about the optimum. For example, in falsification, falsified points for several benchmark problems are located at the edges of the search space, making it easy to falsify if this information is known [21]. Therefore, in the present study, we defined $\pi(x)$ as a U-shaped distribution where the edges are weighted more than the inside area. Even though the prior can be wrong, BO can still converge the optimal solution as a result of the forgetting factor in (5) as proven in [5]. By raising $\pi(x)$ in (5) to a power of β^*/j, the wrong prior decays towards zero with growing j.

5 Experimental Evaluation

We evaluate the proposed BO methods on benchmark problems from [14], and [15] which are introduced briefly in appendix A. For the benchmark problems in [14], two variants of input signals are considered Instance 1 and Instance 2, respectively. Instance 1 allows arbitrary piecewise continuous input signals, but with a finite number of discontinuities in the ARCH19 competition. On the other hand, Instance 2 is restricted to constrained input signals, i.e., the input signal format is fixed, but discontinuities are allowed. The problems such as *AT*, *CC*, *NN*, and *SC* include both Instance 1 and Instance 2 type, while the problem *AFC*, *WT*, and *F16* do not include different types of instances. Additionally, three benchmark problems from [15] are included, i.e., AT', modulator $\Delta - \Sigma$, and *SS*.

5.1 Experimental Setup

The performance of the proposed BO methods compared to vanilla BO is compared against state-of-the-art methods found in the literature, such as HCR (i.e., an optimization-free method) [21] and line-search falsification LSF method (i.e., a direct-search optimization method). We set up the experiments using Breach [39], with *Max* semantics [31, 30]. As BO methods require a set of initial samples to start the process, we set the initial number of samples to $2 \cdot n$, where n is the number of input parameters (i.e., the dimensionality of the optimization problem). This method is implemented in Python. In this implementation, the optimization process is conducted in Python, while the simulation and evaluation of the objective function are performed in MATLAB. For πBO, a U-shaped distribution is used, which assigns a probability density to different points in the space: "prior:[0.4,0.1,0.1,0.4]". Similar to TuRBO, πBO is implemented in Python and interacts with MATLAB. For the LSF method, the maximum number of iterations to improve a single line is set to three. The presented results for LSF use Option 4, which works with lines extending beyond the boundaries of input ranges and thus has a higher chance of resulting in corner values; see [21] for more details. The HCR method starts with a

Table 1: Results for the problems that are easily falsified using an optimization-free method. Instances refer to different types of input parameterization and interpolation. The first number is the relative success rate of falsification in percent, and the number in parenthesis is the average number of simulations (rounded) per successful falsification out of 1000 simulations.

Specifications	Instances	Number of Dimensions	vanilla BO	TuRBO TS	TuRBO LCB	TuRBO PI ($\tau = 0$)	TuRBO PI ($\tau = -1$)	πBO	LSF	HCR
φ_6^{AT}	Instance 1	8	65 (322)	100 (90)	100 (84)	100 (143)	100 (68)	**100 (55)**	100 (63)	100 (117)
φ_7^{AT}	Instance 1	8	100 (165)	95 (166)	100 (38)	100 (32)	100 (59)	100 (85)	**100 (13)**	100 (130)
φ_8^{AT}	Instance 1	8	95 (143)	100 (32)	100 (79)	100 (64)	100 (70)	100 (242)	**100 (26)**	100 (178)
φ_2^{AT}	Instance 2	40	100 (13)	95 (167)	100 (87)	100 (161)	100 (90)	100 (62)	100 (187)	**100 (3)**
$\varphi_3^{AT'}(T = 4.5)$	-	10	**100 (15)**	85 (300)	85 (312)	90 (289)	100 (176)	100 (82)	100 (173)	100 (21)
$\varphi_4^{AT'}(T = 1)$	-	10	100 (54)	65 (345)	100 (308)	70 (504)	95 (335)	75 (363)	60 (376)	**100 (1)**
$\varphi_5^{AT'}(T = 1)$	-	10	**100 (5)**	100 (301)	100 (219)	100 (174)	95 (323)	100 (29)	100 (37)	100 (120)
$\varphi_8^{AT'}(\bar{\omega} = 3500)$	-	10	100 (47)	35 (511)	30 (679)	25 (576)	20 (411)	100 (83)	55 (324)	**100 (5)**
φ_2^{CC}	Instance 2	40	100 (6)	100 (151)	90 (104)	75 (242)	100 (137)	100 (30)	90 (304)	**100 (1)**
φ_2^{NN}	Instance 1	12	100 (283)	25 (445)	10 (415)	15 (502)	20 (345)	35 (232)	80 (253)	**100 (83)**
$\varphi_1^{SS}(\gamma = 0.7)$	-	2	100 (18)	90 (393)	95 (405)	85 (235)	95 (435)	100 (18)	100 (66)	**100 (3)**
$\varphi_1^{SS}(\gamma = 0.8)$	-	2	100 (24)	60 (17)	65 (523)	70 (202)	50 (274)	100 (15)	100 (60)	**100 (3)**
$\varphi_1^{SS}(\gamma = 0.9)$	-	2	**100 (52)**	35 (803)	55 (282)	0 (-)	30 (257)	100 (16)	100 (121)	100 (121)

Table 2: Results for the problems that are not easily falsified using an optimization-free method and require a large number of simulations for successful falsification. Instances refer to different types of input parameterization and interpolation. The first number is the relative success rate of falsification in percent, and the number in parenthesis is the average number of simulations (rounded) per successful falsification out of 1000 simulations.

Specifications	Instances	Number of Dimensions	vanilla BO	TuRBO TS	TuRBO LCB	TuRBO PI ($\tau = 0$)	TuRBO PI ($\tau = -1$)	πBO	LSF	HCR
φ_7^{AT}	Instance 2	40	0 (-)	100 (200)	100 (242)	100 (185)	**100 (184)**	5 (645)	100 (121)	0 (-)
φ_8^{AT}	Instance 2	40	0 (-)	100 (287)	100 (300)	95 (359)	100 (334)	0 (-)	**100 (127)**	0 (-)
φ_9^{AT}	Instance 2	40	0 (-)	100 (248)	100 (225)	100 (215)	100 (195)	0 (-)	**100 (75)**	0 (-)
$\varphi_6^{AT'}(T = 10)$	-	10	0 (-)	95 (126)	**100 (161)**	90 (191)	85 (196)	90 (293)	30 (543)	0 (-)
$\varphi_6^{AT'}(T = 12)$	-	10	15 (768)	100 (67)	**100 (43)**	95 (78)	100 (64)	100 (228)	95 (173)	60 (297)
$\varphi_7^{AT'}$	-	10	0 (-)	45 (632)	25 (507)	45 (337)	35 (708)	35 (525)	25 (568)	30 (621)
φ_4^{CC}	Instance 1	8	0 (-)	75 (431)	**80 (348)**	55 (365)	65 (351)	30 (629)	40 (381)	0 (-)
φ_4^{CC}	Instance 2	40	0 (-)	45 (620)	**95 (717)**	0 (-)	60 (808)	0 (-)	10 (620)	0 (-)
φ_2^{NN}	Instance 2	3	35 (230)	10 (343)	15 (570)	5 (135)	5 (191)	25 (299)	**45 (400)**	0 (-)
$\varphi_1^{\Delta-\Sigma} U \in [-0.35, 0.35]$	-	4	**100 (108)**	95 (156)	90 (124)	95 (122)	95 (149)	100 (149)	100 (249)	0 (-)
φ_1^{F16}	-	3	25 (685)	70 (515)	70 (303)	**85 (337)**	80 (356)	60 (350)	65 (530)	5 (767)

corner point, and the next point is a uniform random (UR) point. It switches between the corners and UR points until the maximum number of simulations, $N = 1000$, is reached or a falsified point is found. The number of corners is limited and depends on the dimensionality of the SUT since there are 2^n corners. If the maximum number of corners is reached, HCR continues using only random input points. Both LSF and HCR are implemented in MATLAB.

In Tables 1-2, we present and discuss the benchmark problems where the choice of the optimization method affects the falsification process. The results for the benchmark problems that are easily falsified with a few simulations regardless of which used optimization method are shown in Appendix B. We also include the results for the specifications that are hard to falsify regardless of the optimization methods, in Appendix C.

In these tables, the first column denotes the specifications; the second refers to which instance is used to evaluate the specification. The third column shows the number of dimensions, and the rest of the columns contain the evaluation results for vanilla BO, TuRBO with TS, LCB, and PI; πBO, LSF, and HCR, respectively. Two different target values are considered for PI, $\tau = 0$, and $\tau = -1$. Each falsification is set to have 1000 maximum number of simulations. Since the falsification process contains random elements, we repeat the falsification process 20 times. Two values are presented for each specification; the first is the relative success rate of falsification in percent. There are 20 falsification runs for each parameter value and specification; thus, the success rate is a multiple of 5%. The second value, inside parentheses, is the average number of simulations (rounded) *per successful falsification*.

5.2 Results

While Table 1 includes the benchmark problems that can be easily falsified using a straightforward approach such as HCR, Table 2 covers the hard problems and specifications where the number of simulations to falsify easily exceeds the maximum simulation budget [21]. In particular, in Table 1, these selected specification are φ_6^{AT}-φ_8^{AT} for Instance 1 of AT problem; Instance 2 of φ_2^{AT}; $\varphi_3^{AT'}(T=4.5)$, $\varphi_4^{AT'}(T=1)$, $\varphi_5^{AT'}(T=1)$, $\varphi_8^{AT'}(\bar{\omega}=3500)$; Instance 2 of φ_2^{CC}; Instance 1 of φ_2^{NN} and all specifications of SS problem.

For *AT* problem, TuRBO (regardless of a selected acquisition function), πBO, and LSF perform better than vanilla BO. Vanilla BO is not successful in falsifying φ_6^{AT} and φ_8^{AT} in each run. However, in Instance 2 of *AT* problem, vanilla BO for φ_2^{AT} of this problem performs as well as HCR and better than other optimization-based methods. When vanilla BO is used for high-dimensional applications, exploration of the edges is more frequent. Search spaces expand faster than sampling budgets resulting in regions with high posterior uncertainty which are typically located at the edges due to the extrapolation of a model. Hence, a typical acquisition function would overemphasize the edges and fail to exploit promising areas. Hence, vanilla BO searches for more edges and likely more corners. As a result, vanilla BO falsifies φ_2^{AT} within a few simulations. On the other hand, πBO also shows a good performance here because it emphasizes more the corners following our prior U-shaped distribution. TuRBO is prone to explore fewer corner points as the search is limited to local trust regions. In particular, TS is not always successful, while PI and LCB require more simulations than HCR and vanilla BO.

For the specifications of AT' problem, vanilla BO and πBO, except for $\varphi_4^{AT'}(T=1)$, also perform quite well with a 100% success rate. Further, the TuRBO method and LSF are not always successful for the benchmark problems $\varphi_3^{AT'}(T=4.5)$, $\varphi_4^{AT'}(T=1)$, $\varphi_5^{AT'}(T=1)$, $\varphi_8^{AT'}(\bar{\omega}=3500)$. In contrast, vanilla BO falsifies them with only a few simulations. Also, for $\varphi_5^{AT'}(T=1)$, vanilla BO and πBO beats HCR with fewer evaluations. While vanilla BO required 5 simulations on average to falsify, πBO needed 29 simulations. The better performance of vanilla BO and πBO results in $\varphi_8^{AT'}(\bar{\omega}=3500)$ performing among the top optimization-based methods, but not better than HCR that only requires 5 simulations.

For φ_2^{CC} of instance 2, πBO and vanilla BO work similarly to HCR and much better than other optimization-based methods. This specification can be falsified at some of the corner points. Among TuRBO results, PI ($\tau=-1$) and TS perform better than LCB and PI ($\tau=0$), which are successful only 90% and 75%, respectively.

The benchmark problem φ_2^{NN} is falsified with HCR with 100% and using 83 simulations on average. In contrast, vanilla BO is the only optimization-based method that is successful in each independent trial. Based on the evaluation of falsified points, we observe that this specification is falsifiable where at least some input parameters are at the upper or lower bound of input ranges. Hence, vanilla BO and HCR performed better than TuRBO. TuRBO never evaluates the corner points, or the points are on the bound of the input ranges. However, πBO does not falsify efficiently, only 35% are successfully falsified; it might be because the number of dimensions is increased, and hence the efficiency of πBO drops.

SS is a two-dimensional synthetic problem that tricks the optimizing algorithms that try to estimate gradients to search in the wrong direction. For all specifications of SS, HCR, vanilla BO, LSF, and importantly πBO provide better results than TuRBO. Both input parameters are defined within the range $[-1,1]$ in this special case. The target specification is falsified at the corner point $x^T=[1,1]$ and close to it. The gradient cannot point toward the falsification area if the initial samplings of *First Input* and *Second Input* are approximately in the ranges $(-1,1)$ and $(-1,0.8)$, respectively when $\gamma=0.7$, which is a threshold parameter in this problem. Hence, the three methods HCR, vanilla BO, LSF, and πBO that can search the corner points which lead that all falsify this problem better than TuRBO. On the other hand, TuRBO searches more within the input ranges. Thus, it is difficult for TuRBO to approach the failure area with a local hyperrectangle trust region that is centered at the best solution found. A larger value for γ such as $(\gamma=0.8)$ and $(\gamma=0.9)$ results in a smaller failure surface. Thus, the performance of TuRBO drops additionally with the best performance with PI ($\tau=0$) as 70% and LCB as 55% for $(\gamma=0.8)$ and $(\gamma=0.9)$, respectively.

Next, we discuss the benchmark problems noted here as the hard problems because an optimization-free method such as HCR cannot falsify them. The results are provided in Table 2. For the specifications, 7-9, of *AT* problem, HCR, vanilla BO, and πBO cannot falsify them. Based on our evaluation of these specifications, they are not falsifiable close to corners or edges using the chosen input parameters and used optimization methods. In particular, emphasizing the corners as done in πBO is wrong. However, as a result of the forgetting factor, πBO still converges to the optimal solution as it was successful in only one run out of 20 runs in φ_7^{AT}. Like any other global BO approach in a high-dimensional setting, the efficiency of πBO eventually drops significantly as the bias of a Gaussian process to exploit the edges of the search space is emphasized. In TuRBO, this shortcoming is handled, as shown in the results for the hard

problems. In TuRBO, the training of a Gaussian process is done locally within a trust region that shrinks and expands based on the performance. Hence, the inherent presence of regions with large posterior uncertainty commonly found in the global BO approach due to the curse of dimensionality is reduced. The performance of TuRBO is similar regardless of the acquisition function used. Compared to LSF, more simulations are needed.

In the AT' problem, we can see a clear advantage of TuRBO over other BO methods and state-of-the-art approaches. For $\varphi_6^{AT'}$, with both $(T = 10), (T = 12)$, LCB is successful in each run, with 161 and 43 average on simulations respectively, while LSF is only successful 30 and 95%. Another BO method, πBO, also demonstrates a good performance for $\varphi_6^{AT'}$, with 95% success rate when $(T = 10)$ and 100% when $(T = 12)$. Because the dimensionality for these specifications is moderate, πBO forgets the wrong prior and eventually converges to the falsified areas. Vanilla BO is not successful in falsifying these specifications because simulations are mostly evaluated at the edges that are not falsifiable.

$\varphi_7^{AT'}$ is one specification that is a hard problem to be falsified with a specific feature. The STL formula of this specification is $\varphi_7^{AT'} = \neg\Big((\square_{[0,1]} gear == 1) \wedge (\square_{[2,4]} gear == 2) \wedge (\square_{[5,7]} gear == 3) \wedge (\square_{[8,10]} gear == 3) \wedge (\square_{[12,15]} gear == 2)\Big)$. In general, it is impossible to determine how close we are to fulfilling equality predicates, such as $gear == 1$, $gear == 2$, or $gear == 3$. In other words, if the gear is 1, for example, the objective value using ***Max*** semantics will be a positive constant value if the specification is fulfilled, otherwise, a negative constant value. For simplicity, we assume that the positive constant value is 1 while the negative constant value is -1. In this specification, *always*-operators are combined in one big conjunction. The first *always*-operator checks to see whether $gear == 1$ between $t = 0\ sec$ and $t = 1\ sec$ or not. If it is satisfied, the *always*-operator has the total objective value $min([1, 1, \ldots, 1]) = 1$, which means that the gear is always equal to 1 in $t = [0, 1]$. Similarly, for the $gear == 2$ and $gear == 3$ at other times, the specification has a positive constant value of 1. On the other hand, if the specification is not satisfied at any time we will have something like $\min([1, -1, \ldots, 1]) = -1$. Since the ***Max*** yields the same value for different inputs, model-based methods such as BO cannot learn anything meaningful to sufficiently explore and exploit the given search space. The main assumption of BO is that the target function should be sufficiently smooth. TS and PI ($\tau = 0$) provide slightly better performance compared to LCB and PI ($\tau = -1$) with a 45% success rate. Because TS puts more emphasis on exploitation than other methods, it is expected to show better performance. Other methods balance equally between exploitation and exploration, which is useless in this specific case. Even though the learning is not adequate,TuRBO, and πBO can still falsify this problem but not always, as shown in Table 2.

For both instances in φ_4^{CC}, we see a significant advantage of using TuRBO w.r.t. other optimization-based or optimization-free methods. For Instance 1, there is not much difference between the performance of TuRBO with different acquisition functions, although LCB performs slightly better with 80%. TuRBO performs much better than other optimization methods, e.g., 40% increasing rate than LSF. For Instance 2, there is a difference between the performance of PI when the target value is specified differently. By evaluating the objective value of the falsified point in each run of this specification, we could understand that the global optimum is close to and less than -1. Hence, having a target of 0 does not provide good results. This specification was one of the hardest benchmark problems evaluated in [21] where the proposed method, LSF, was able to falsify φ_4^{CC} in 10% of the runs. As it can be seen from Table 2, more simulations are needed to falsify this specification. The acquisition function LCB in TuRBO needs fewer simulations for falsification than other methods and has a success rate of 95%. Based on the evaluation collected from this problem, we could conclude that the falsification, in this case, happens in a small region of the search space. Therefore, random approaches might not be a good choice to find this small region. A target of -1 is appropriate since the minimum achievable object value is close to -1.

In the benchmark problem φ_2^{NN} (Instance 2), LSF performs the best with the success rate of 45%, exceeding especially TuRBO with 5, 10, and 15%. Based on the evaluation of the falsified point, there are reasons why all other methods are not successful in falsifying for each trial. First, this specification is falsifiable, where at least some input parameters are at the upper or lower bound of the input interval. The falsifiable region is small, and close to the boundaries.

For the modulator problem, vanilla BO and πBO slightly work better than others in falsifying each trial. These results demonstrate that πBO works well when the number of dimensions is relatively small. For $F16$, TuRBO with PI ($\tau = 0$) provides a better result with a success rate of 85% compared to other methods.

To examine and study the performance of different optimization-based methods, an aggregated comparison is provided in Figure 2 with the hard specifications presented in Table 2. As shown in Figure 2 and evident from our early discussion, TuRBO falsifies more benchmark problems with fewer simulations if we select LCB w.r.t. other acquisition functions such as PI ($\tau = -1$), PI ($\tau = 0$) and the default TS. TuRBO with LCB falsified 5 specifications out of 11 that are specified as the hard problems in Table 2 with the success rate of 100%, while TS and PI ($\tau = -1$) could falsify four specifications similarly as LSF. PI ($\tau = 0$) falsified 2 specifications similarly to πBO. On the other hand, vanilla BO is

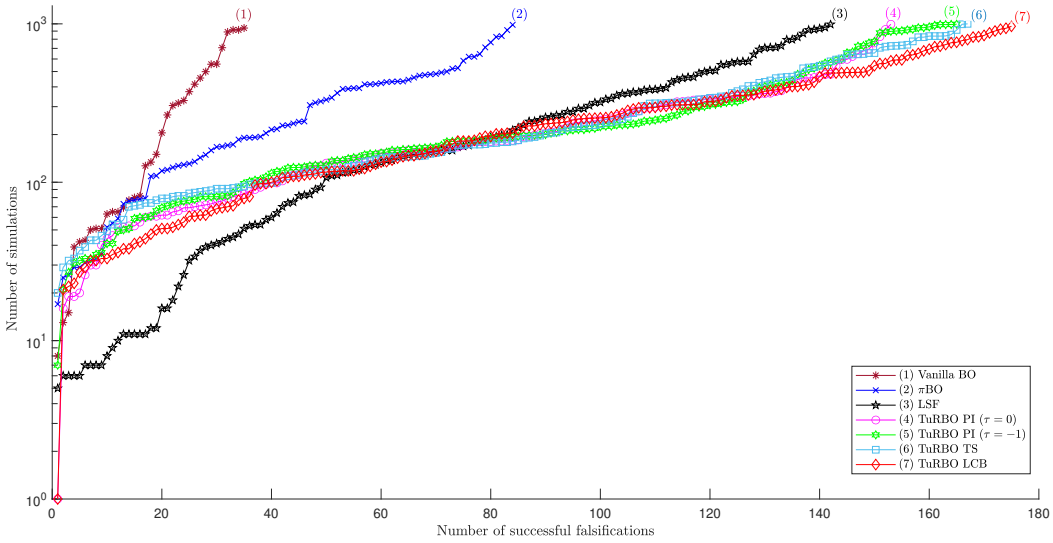

Figure 2: A cactus plot showing the performance of optimization-based methods on the hard problems in Table 2. The plotted values show how many successful falsifications (x-axis) were completed in less than always simulations (y-axis, logarithmic scale).

successful only for one specification in each trial. By taking the average of these success rates, we can see that LCB provides the best performance on average, with a success rate of 79.55%. Other acquisition functions have the average success rate as follows TS with 75.90%; PI ($\tau = -1$) with 75.45%, PI ($\tau = 0$) with 69.54%, LSF with 64.54%, πBO with 38.18%, and finally, vanilla BO with 15.90%.

5.3 Discussion

Discussion of Optimization Methods: The performance of each evaluated BO method depends on the SUT and how the falsified area or falsified points are located based on the input ranges for each benchmark problem. Vanilla BO, which is the standard global BO method, showed a good performance for those specifications and problems that can be falsified at boundaries of input spaces or corners. When the number of dimensions is high vanilla BO searches the edges more because the prediction error and the uncertainty of the surrogate model at the edges become large. On the other hand, using local regions, exploration and exploitation in TuRBO are limited to local trust regions, which results in less efficient falsification compared to vanilla BO for those easily falsifiable problems where failure points are at corners. Hence, vanilla BO showed better performance for those benchmark problems with the ability to be falsified at corners. However, the efficiency of TuRBO increases with the number of dimensions showing a remarkable performance even for hard problems that are not falsified with other methods. Compared to the other BO methods for high-dimensional settings, such as the previously proposed REMBO [11], TuRBO does not need any setup, it can be used as an out-of-box tool. The πBO allows injecting prior knowledge about a failure region. πBO has shown good performance for those problems that are easily falsified at corners due to the U-shaped distribution that was used as prior knowledge based on previous falsification experiments. Even though πBO does not scale well in a high dimensional setting, it still shows good performance for a moderate number of dimensions of the input space.

Discussion of Acquisition Functions: Comparing the performance of different acquisition functions in TuRBO, LCB showed the best performance. LCB depends on mean and variance, not on the target value, as PI does. A challenging aspect of the LCB is choosing the optimal value for β. TS, on the contrary, does not require any parameters to be selected. Choosing the best target value for PI is difficult as it depends on the application and knowing where the optimal value is. This information is, in general, unknown to us. While the default setting in BO is to use the best-current value, we here assume constant values of 0 and -1 to emphasize failure points. For those specifications with minimum optima between $(-1, 0)$, PI does not perform well because it assumes that -1 is the lowest objective value and neglects the failure points having an objective value > -1 from the surrogate model, that indeed falsifies the specification. Objective function values calculated using the *Max* semantics can be constant in large regions. It means the same objective function value for different input parameters. Hence, optimization methods cannot get any sense of direction from the

objective function. A benefit of using TS as an acquisition function is its randomness which may detect falsified points where there is no information from the objective function to guide the optimization process. In TS, once we have a trained surrogate model, the concept is to greedily sample a configuration from the posterior with the lowest value, and sampling from the posterior generates TS's randomness. As TS is focused more on exploitation, it is more efficient w.r.t. other acquisition functions.

6 Conclusion

In this study, we assess Bayesian Optimization (BO) for falsifying cyber-physical systems (CPSs). BO learns the system under test to strategically choose the next test to execute. We adapt BO in several ways, making it more efficient for falsification compared to the vanilla BO. Specifically, we demonstrate how trust region-based BO approaches can effectively handle the evaluated falsification problems in this paper with a high number of dimensions, and how prior knowledge can be incorporated into BO. Our work establishes a practical framework for using BO as an out-of-box tool, unlike previous studies that do not account for its practicality. We propose two notable BO methods, πBO, and TuRBO, for falsification and provide a comprehensive evaluation demonstrating their efficiency in solving the standard falsification problems available in the falsification community. In our experiments, TuRBO outperforms other state-of-the-art methods for high-dimensional and difficult-to-falsify specifications, without requiring any information from the practitioner. πBO allows the injection of prior knowledge about falsification if available. For benchmark examples with fewer input parameters for optimization, πBO performs as well as or even better than HCR when the specifications are falsifiable at corners or boundaries. A correct prior about falsification, such as a U-shaped distribution emphasizing corners, can increase efficiency and reduce the number of simulations needed for falsification. Additionally, we conduct a comprehensive evaluation of different acquisition functions in TuRBO, propose modifications for falsification, and determine that using a LCB in TuRBO is the optimal choice, as it is less challenging to set up the parameters for good performance on evaluated benchmark examples. In future work, we plan to combine various BO methods to further enhance the overall efficiency.

Acknowledgments

This work was supported by the Swedish Research Council (VR) project SyTeC VR 2016-06204 and from the Swedish Governmental Agency for Innovation Systems (VINNOVA) under project TESTRON 2015-04893, and was partly supported by the Wallenberg AI, Autonomous Systems and Software program (WASP) funded by the Knut and Alice Wallenberg Foundation. This research was also supported in part by affiliate members and other supporters of the Stanford DAWN project—Ant Financial, Facebook, Google, InfoSys, Teradata, NEC, and VMware. The evaluations were performed using resources at High Performance Computing Center North (HPC2N), Umeå University, a Swedish national center for Scientific and Parallel Computing.

The authors would like to thank Martin Fabian, Koen Claessen, and Nicholas Smallbone, for their helpful comments on this paper.

References

[1] Rajeev Alur. *Principles of cyber-physical systems*. MIT press, 2015.

[2] Charles Audet and Warren Hare. *Derivative-free and blackbox optimization*. Springer Series in Operations Research and Financial Engineering. Springer, 2017.

[3] Robert Hooke and Terry A Jeeves. 'Direct search' solution of numerical and statistical problems. *Journal of the ACM (JACM)*, 8(2):212–229, 1961.

[4] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas. Taking the human out of the loop: A review of Bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2016.

[5] Carl Hvarfner, Danny Stoll, Artur Souza, Marius Lindauer, Frank Hutter, and Luigi Nardi. πBO: Augmenting acquisition functions with user beliefs for Bayesian optimization. *arXiv preprint arXiv:2204.11051*, 2022.

[6] Kenan Šehić, Alexandre Gramfort, Joseph Salmon, and Luigi Nardi. Lassobench: A high-dimensional hyperparameter optimization benchmark suite for lasso. *arXiv preprint arXiv:2111.02790*, 2021.

[7] Matthias Mayr, Faseeh Ahmad, Konstantinos Chatzilygeroudis, Luigi Nardi, and Volker Krueger. Skill-based multi-objective reinforcement learning of industrial robot tasks with planning and knowledge integration. *arXiv preprint arXiv:2203.10033*, 2022.

[8] Felix Berkenkamp, Andreas Krause, and Angela P Schoellig. Bayesian optimization with safety constraints: safe and automatic parameter tuning in robotics. *Machine Learning*, pages 1–35, 2021.

[9] Jyotirmoy Deshmukh, Marko Horvat, Xiaoqing Jin, Rupak Majumdar, and Vinayak S. Prabhu. Testing cyber-physical systems through Bayesian optimization. *ACM Trans. Embed. Comput. Syst.*, 16(5s), sep 2017.

[10] Nikolaus Hansen. The CMA evolution strategy: a comparing review. *Towards a new evolutionary computation*, pages 75–102, 2006.

[11] Ziyu Wang, Frank Hutter, Masrour Zoghi, David Matheson, and Nando de Feitas. Bayesian optimization in a billion dimensions via random embeddings. *Journal of Artificial Intelligence Research*, 55:361–387, 2016.

[12] A. Nayebi, A. Munteanu, and M. Poloczek. A framework for Bayesian optimization in embedded subspaces. In *Proceedings of the 36th International Conference on Machine Learning*, page 4752–4761, 2019.

[13] B. Letham, R. Calandra, A. Rai, and E. Bakshy. Re-examining linear embeddings for high-dimensional Bayesian optimization. In *Advances in Neural Information Processing Systems 33 (NeurIPS 2020)*, volume 33, pages 1546–1558, 2020.

[14] Gidon Ernst, Paolo Arcaini, Alexandre Donzé, Georgios Fainekos, Logan Mathesen, Giulia Pedrielli, Shakiba Yaghoubi, Yoriyuki Yamagata, and Zhenya Zhang. ARCH-COMP 2019 category report: Falsification. In *ARCH19. 6th International Workshop on Applied Verification of Continuous and Hybrid Systems*, volume 61, pages 129–140. EasyChair, 2019.

[15] Zahra Ramezani, Johan Lidén Eddeland, Koen Claessen, Martin Fabian, and Knut Åkesson. Multiple objective functions for falsification of cyber-physical systems. *IFAC-PapersOnLine*, 53(4):417–422, 2020.

[16] David Eriksson, Michael Pearce, Jacob Gardner, Ryan D Turner, and Matthias Poloczek. Scalable global optimization via local Bayesian optimization. *Advances in Neural Information Processing Systems*, 32:5496–5507, 2019.

[17] Ya-xiang Yuan. A review of trust region algorithms for optimization. In *Iciam*, volume 99 (1), pages 271–282, 2000.

[18] William R Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294, 1933.

[19] Niranjan Srinivas, Andreas Krause, Sham Kakade, and Matthias Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 1015–1022, Madison, WI, USA, 2010. Omnipress.

[20] Harold J. Kushner. A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise. *Journal of Basic Engineering*, 86:97–106, 1964.

[21] Zahra Ramezani, Koen Claessen, Nicholas Smallbone, Martin Fabian, and Knut Åkesson. Testing cyber–physical systems using a line-search falsification method. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 41(8):2393–2406, 2022.

[22] Artur Souza, Luigi Nardi, Leonardo B Oliveira, Kunle Olukotun, Marius Lindauer, and Frank Hutter. Bayesian optimization with a prior for the optimum. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 265–296. Springer, 2021.

[23] Johan Lidén Eddeland, Sajed Miremadi, and Knut Åkesson. Evaluating optimization solvers and robust semantics for simulation-based falsification. *EPiC Series in Computing*, 74:259–266, 2020.

[24] J. A. Nelder and R. Mead. A Simplex Method for Function Minimization. *The Computer Journal*, 7(4):308–313, 01 1965.

[25] Waltraud Huyer and Arnold Neumaier. SNOBFIT–stable noisy optimization by branch and fit. *ACM Transactions on Mathematical Software (TOMS)*, 35(2):1–25, 2008.

[26] H Edwin Romeijn and Robert L Smith. Simulated annealing for constrained global optimization. *Journal of Global Optimization*, 5:101–126, 1994.

[27] Logan Mathesen, Giulia Pedrielli, and Georgios Fainekos. Efficient optimization-based falsification of cyber-physical systems with multiple conjunctive requirements. In *2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)*, pages 732–737, 2021.

[28] Takumi Akazaki. Falsification of conditional safety properties for cyber-physical systems with gaussian process regression. In *International Conference on Runtime Verification*, pages 439–446. Springer, 2016.

[29] Simone Silvetti, Alberto Policriti, and Luca Bortolussi. An active learning approach to the falsification of black box cyber-physical systems. In *International Conference on Integrated Formal Methods*, pages 3–17. Springer, 2017.

[30] Koen Claessen, Nicholas Smallbone, Johan Eddeland, Zahra Ramezani, and Knut Åkesson. Using valued booleans to find simpler counterexamples in random testing of cyber-physical systems. *IFAC-PapersOnLine*, 51(7):408–415, 2018.

[31] Alexandre Donzé and Oded Maler. Robust satisfaction of temporal logic over real-valued signals. In Krishnendu Chatterjee and Thomas A. Henzinger, editors, *Formal Modeling and Analysis of Timed Systems*, pages 92–106, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[32] Georgios E. Fainekos and George J. Pappas. Robustness of temporal logic specifications for continuous-time signals. *Theoretical Computer Science*, 410(42):4262 – 4291, 2009.

[33] Oded Maler and Dejan Nickovic. Monitoring temporal properties of continuous signals. In *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, pages 152–166, Hei2004.

[34] CE. Rasmussen and CKI. Williams. *Gaussian Processes for Machine Learning*. Adaptive Computation and Machine Learning. MIT Press, 2006.

[35] P. I. Frazier. A tutorial on Bayesian optimization. *arXiv preprint arXiv:1807.02811*, 2018.

[36] Luigi Nardi, David Koeplinger, and Kunle Olukotun. Practical design space exploration. In *2019 IEEE 27th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 347–358. IEEE, 2019.

[37] Benjamin Echard, Nicolas Gayton, and Maurice Lemaire. AK-MCS: an active learning reliability method combining kriging and monte carlo simulation. *Structural Safety*, 33(2):145–154, 2011.

[38] Roland Schöbi, Bruno Sudret, and Stefano Marelli. Rare event estimation using polynomial-chaos kriging. *ASCE-ASME Journal of Risk and Uncertainty in Engineering Systems, Part A: Civil Engineering*, 3(2):D4016002, 2017.

[39] Alexandre Donzé. Breach, a toolbox for verification and parameter synthesis of hybrid systems. In Tayssir Touili, Byron Cook, and Paul Jackson, editors, *Computer Aided Verification*, pages 167–170. Springer Berlin Heidelberg, 2010.

[40] Bardh Hoxha, Houssam Abbas, and Georgios Fainekos. Benchmarks for temporal logic requirements for automotive systems. In *ARCH@CPSWeek*, 2014.

[41] Jianghai Hu, John Lygeros, and Shankar Sastry. Towards a theory of stochastic hybrid systems. In Nancy Lynch and Bruce H. Krogh, editors, *Hybrid Systems: Computation and Control*, pages 160–173. Springer Berlin Heidelberg, 2000.

[42] Adel Dokhanchi, Shakiba Yaghoubi, Bardh Hoxha, Georgios Fainekos, Gidon Ernst, Zhenya Zhang, Paolo Arcaini, Ichiro Hasuo, and Sean Sedwards. Arch-comp18 category report: Results on the falsification benchmarks. In *ARCH@ADHS*, 2018.

[43] MathWorks. Design NARMA-L2 Neural Controller in Simulink. `https://au.mathworks.com/help/deeplearning/ug/design-narma-l2-neural-controller-in-simulink.html`, 2020. Online: accessed 1 March 2021.

[44] Goran Frehse, Alessandro Abate, Dieky Adzkiya, Lei Bu, Mirco Giacobbe, Muhammad Syifa'Ul Mufid, and Enea Zaffanella. Arch-comp18 category report: Hybrid systems with piecewise constant dynamics. In *ARCH18. 5th International Workshop on Applied Verification of Continuous and Hybrid Systems*, volume 54, pages 1–13. EasyChair, 2018.

[45] Shakiba Yaghoubi and Georgios Fainekos. Gray-box adversarial testing for control systems with machine learning components. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, HSCC '19, page 179–184, 2019.

[46] Simone Schuler, Fabiano Daher Adegas, and Adolfo Anta. Hybrid modelling of a wind turbine. In Goran Frehse and Matthias Althoff, editors, *ARCH16. 3rd International Workshop on Applied Verification for Continuous and Hybrid Systems*, volume 43 of *EPiC Series in Computing*, pages 18–26. EasyChair, 2017.

[47] Xiaoqing Jin, Jyotirmoy V. Deshmukh, James Kapinski, Koichi Ueda, and Ken Butts. Powertrain control verification benchmark. In *Proceedings of the 17th International Conference on Hybrid Systems: Computation and Control*, HSCC '14, page 253–262. Association for Computing Machinery, 2014.

[48] Thao Dang, Alexandre Donzé, and Oded Maler. Verification of analog and mixed-signal circuits using hybrid system techniques. In *International Conference on Formal Methods in Computer-Aided Design*, pages 21–36. Springer, 2004.

[49] Adel Dokhanchi, Aditya Zutshi, Rahul T Sriniva, Sriram Sankaranarayanan, and Georgios Fainekos. Requirements driven falsification with coverage metrics. In *Proceedings of the 12th International Conference on Embedded Software*, pages 31–40, 2015.

[50] Shakiba Yaghoubi and Georgios Fainekos. Gray-box adversarial testing for control systems with machine learning components. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, pages 179–184, 2019.

[51] Zahra Ramezani, Alexandre Donze, Martin Fabian, and Knut Åkesson. Temporal logic falsification of cyber-physical systems using input pulse generators. In Goran Frehse and Matthias Althoff, editors, *8th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH21)*, volume 80 of *EPiC Series in Computing*, pages 195–202. EasyChair, 2021.

A Benchmark Problems

All evaluated benchmark problems [14, 15] are introduced here briefly. Table 3 presents the STL specifications for all these benchmark problems.

Automatic Transmission (AT) There are two inputs for this problem, $0 \leq throttle \leq 100$, and $0 \leq brake \leq 325$, which can be active at the same time [40]. This problem has two instances. In Instance 1, both input signals are piecewise constant with a *previous* interpolation between them, corresponds to the previous sample value, while Instance 2 has constrained input signals with discontinuities at most every 5 time units.

Chasing Cars (CC) This model has two inputs $0 \leq throttle \leq 1$ and $0 \leq brake \leq 1$ [41] with two instances. For Instance 1, the input specifications allow any piecewise continuous signals to be distributed equally, with 8 segments. However, in Instance 2, the input signals are piecewise constant signals with a *previous* interpolation with 20 segments.

Fuel Control of an Automotive Power Train (AFC) This system has two inputs of $0 \leq \theta \leq 61.1$ and $900 \leq \omega \leq 1100$ [42]. The input signal θ is piecewise constant with 10 uniform segments, i.e., a *previous* interpolation while ω is constant.

Neural Network Controller (NN) This benchmark problem has a reference value $1 \leq Ref \leq 3$ for the position [43] as input. For Instance 1 of this problem, the input signal needs discontinuities to be at least 3 time-units long, 12 segments, while Instance 2 requires exactly 3 constant segments.

Aircraft Ground Collision Avoidance System (F16) The system [44] is required to start with initial conditions $0.2\pi \leq roll \leq 0.2833\pi$, $-0.5\pi \leq pitch \leq -0.54\pi$, and $0.25\pi \leq yaw \leq 0.375\pi$.

Steam Condenser with Recurrent Neural Network Controller (SC) The only input of this system is $3.99 \leq Fs \leq 4.01$, [45], and the input signal should be piecewise constant with 12 and 20 evenly spaced segments for Instance 1 and Instance 2, respectively.

Wind Turbine (WT) A simplified wind turbine model from [46] with only one input $8 \leq v \leq 16$ is considered. The input signal of this problem is piecewise with *spline* interpolation, a cubic polynomial interpolation between the control points, each with specified derivatives.

Automatic Transmission ($\mathbf{AT'}$) The two inputs to the model are $0 \leq throttle \leq 100$ and $0 \leq brake \leq 500$ [47]. This problem has different specifications and a different input range for the $brake$ compared to the ARCH problem presented, AT. There are 7 control points for $throttle$ and 3 for $brake$ distributed uniformly with *pchip* interpolation. To distinguish this problem from AT, it is called $\mathbf{AT'}$.

Third Order $\Delta - \Sigma$ Modulator The third order $\Delta - \Sigma$ modulator has one input U, three states x_1, x_2, x_3, and three initial conditions $x_1^{init}, x_2^{init}, x_3^{init}$, all defined in $[-0.1, 0.1]$ [48]. Three different ranges are considered for the input, $-0.35 \leq U \leq 0.35$, $-0.40 \leq U \leq 0.40$, and $-0.45 \leq U \leq 0.45$.

Table 3: Specifications to falsify for all benchmark problems

Spec.	Formula
φ_1^{AT}	$\square_{[0,20]}(v < 120)$
φ_2^{AT}	$\square_{[0,10]}(\omega < 4750)$
φ_3^{AT}	$\square_{[0,30]}\Big((\neg g_1 \wedge \circ g_1) \implies \circ \square_{[0,2.5]} g_1\Big)$
φ_4^{AT}	$\square_{[0,30]}\Big((\neg g_2 \wedge \circ g_2) \implies \circ \square_{[0,2.5]} g_2\Big)$
φ_5^{AT}	$\square_{[0,30]}\Big((\neg g_3 \wedge \circ g_3) \implies \circ \square_{[0,2.5]} g_3\Big)$
φ_6^{AT}	$\square_{[0,30]}\Big((\neg g_4 \wedge \circ g_4) \implies \circ \square_{[0,2.5]} g_4\Big)$
φ_7^{AT}	$\big(\square_{[0,30]}\omega < 3000\big) \implies \big(\square_{[0,4]} v < 35\big)$
φ_8^{AT}	$\big(\square_{[0,30]}\omega < 3000\big) \implies \big(\square_{[0,8]} v < 50\big)$
φ_9^{AT}	$\big(\square_{[0,30]}\omega < 3000\big) \implies \big(\square_{[0,20]} v < 65\big)$
φ_1^{AFC}	$\square_{[11,50]}\Big(((\theta < 8.8) \wedge (\lozenge_{[0,0.05]}(\theta > 40))$ $\vee(\theta > 40) \wedge (\lozenge_{[0,0.05]}(\theta < 8.8)) \implies (\square_{[1,5]}\lvert\mu\rvert < 0.008)\Big)$
φ_2^{AFC}	$\square_{[11,50]}\lvert\mu\rvert < 0.007$
φ_1^{NN}	$\square_{[1,37]}\Big(\neg(\lvert Pos - Ref\rvert > 0.005 + 0.03\lvert Ref\rvert)$ $\implies \lozenge_{[0,2]}\square_{[0,1]}\big(0.005 + 0.03\lvert Ref\rvert \leq \lvert Pos - Ref\rvert\big)\Big)$
φ_2^{NN}	$\square_{[1,37]}\Big(\neg(\lvert Pos - Ref\rvert > 0.005 + 0.04\lvert Ref\rvert)$ $\implies \lozenge_{[0,2]}\square_{[0,1]}\big(0.005 + 0.04\lvert Ref\rvert \leq \lvert Pos - Ref\rvert\big)\Big)$
φ_1^{WT}	$\square_{[30,630]}\theta \leq 14.2$
φ_2^{WT}	$\square_{[30,630]} 21000 \leq M_{g,d} \leq 47500$
φ_3^{WT}	$\square_{[30,630]}\Omega \leq 14.3$
φ_4^{WT}	$\square_{[30,630]}\lozenge_{[0,5]}\lvert\theta - \theta_d\rvert \leq 1.6$
φ_1^{CC}	$\square_{[0,100]}(y_5 - y_4 \leq 40)$
φ_2^{CC}	$\square_{[0,100]}\lozenge_{[0,30]} y_5 - y_4 \geq 15$
φ_3^{CC}	$\square_{[0,80]}\Big((\square_{[0,20]} y_2 - y_1 \leq 20) \vee (\lozenge_{[0,20]} y_5 - y_4 \geq 40)\Big)$
φ_4^{CC}	$\square_{[0,65]}\lozenge_{[0,30]}\square_{[0,20]}(y_5 - y_4 \geq 8)$
φ_5^{CC}	$\square_{[0,72]}\lozenge_{[0,8]}\Big((\square_{[0,5]} y_2 - y_1 \geq 9) \implies (\square_{[5,20]} y_5 - y_4 \geq 9)\Big)$
φ^{F16}	$\square_{[0,15]} altitude > 0$
φ^{SC}	$\square_{[30,35]}\Big(87 \leq pressure \wedge pressure \leq 87.5\Big)$
$\varphi_1^{AT'}$	$\lozenge_{[0,T]}(\omega \geq 2000)$
$\varphi_2^{AT'}$	$\square\lozenge_{[0,T]}(\omega \leq 3500 \vee \omega \geq 4500)$
$\varphi_3^{AT'}$	$\square_{[0,T]}(\neg(gear == 4))$
$\varphi_4^{AT'}$	$\lozenge(\square_{[0,T]}(gear == 3))$
$\varphi_5^{AT'}$	$\bigwedge_{i=1,\ldots,4}\square((\neg(gear == i) \wedge \lozenge_{[0,\epsilon]}(gear == i)$ $\implies (\square_{[\epsilon,T+\epsilon]}(gear == i)))$
$\varphi_6^{AT'}$	$\square_{[0,T]}(v \leq 85) \vee \lozenge(\omega \geq 4500)$
$\varphi_7^{AT'}$	$\neg\Big((\square_{[0,1]} gear == 1) \wedge (\square_{[2,4]} gear == 2)$ $\wedge(\square_{[5,7]} gear == 3) \wedge (\square_{[8,10]} gear == 3)$ $\wedge(\square_{[12,15]} gear == 2)\Big)$
$\varphi_8^{AT'}$	$\square_{[0,20]}\big((gear == 4 \wedge throttle > 45$ $\wedge throttle < 50) \implies \omega < \bar{\omega}\big)$
$\varphi^{\Delta-\Sigma}$	$\square\left(\bigwedge_{i=1}^{3}(-1 \leq x_i \wedge x_i \leq 1)\right).$
φ^{SS}	$\square(y \geq 0)$

Static Switched (SS) The static switched system is a model without any dynamics inspired by [49] with two inputs in the range $[-1, 1]$. Three different values are considered for parameter $\gamma = 0.7, 0.8, 0.9$.

B Other Experiments on ARCH Benchmark

The results for the benchmark examples that can be falsified easily with a few simulations regardless of which optimization method are presented in Table 4-6. Table 4 and 5 refers to result for the specifications of AT, CC, and NN systems with Instance 1 and Instance 2, respectively. On the other hand, Table 6 includes the results for the specifications of AT', $\varphi_1^{\Delta-\Sigma}$, WT and AFC systems that only one input instance are evaluated on them.

Table 4: Results for the specifications of AT, CC, and NN systems, Instance 1 which are not included in the tables 1- 2.

Specifications	Number of Dimensions	vanilla BO	TuRBO TS	TuRBO LCB	TuRBO PI ($\tau = 0$)	TuRBO PI ($\tau = -1$)	πBO	LSF	HCR
φ_1^{AT}	8	100 (7)	100 (86)	100 (81)	100 (79)	100 (80)	100 (37)	100 (34)	**100 (5)**
φ_2^{AT}	8	**100 (3)**	100 (13)	100 (12)	100 (9)	100 (9)	100 (8)	100 (9)	**100 (3)**
φ_3^{AT}	8	100 (70)	100 (17)	100 (43)	100 (46)	100 (22)	**100 (12)**	100 (37)	100 (27)
φ_4^{AT}	8	100 (41)	100 (16)	100 (17)	100 (22)	100 (14)	**100 (11)**	100 (31)	100 (25)
φ_5^{AT}	8	100 (52)	100 (22)	100 (11)	100 (18)	100 (18)	**100 (11)**	100 (21)	100 (23)
φ_9^{AT}	8	100 (104)	100 (28)	100 (30)	**100 (21)**	100 (40)	100 (77)	100 (22)	100 (61)
φ_1^{CC}	8	**100 (2)**	100 (6)	100 (9)	100 (7)	100 (9)	100 (4)	100 (8)	100 (5)
φ_2^{CC}	8	**100 (3)**	**100 (3)**	100 (4)	100 (6)	100 (5)	100 (6)	100 (14)	100 (5)
φ_3^{CC}	8	**100 (2)**	100 (13)	100 (11)	100 (11)	100 (12)	100 (6)	100 (14)	100 (5)
φ_5^{CC}	8	100 (6)	100 (59)	100 (55)	100 (52)	100 (61)	100 (18)	100 (32)	**100 (5)**
φ_1^{NN}	12	100 (145)	100 (32)	100 (21)	100 (32)	**100 (19)**	100 (32)	100 (25)	100 (39)

Table 5: Results for the specifications of AT, CC, and NN systems, Instance 2 which are not included in the tables 1- 2.

Specifications	Number of Dimensions	vanilla BO	TuRBO TS	TuRBO LCB	TuRBO PI ($\tau = 0$)	TuRBO PI ($\tau = -1$)	πBO	LSF	HCR
φ_3^{AT}	40	**100 (2)**	100 (5)	100 (4)	100 (5)	100 (4)	100 (3)	100 (10)	100 (7)
φ_4^{AT}	40	100 (2)	**100 (1)**	**100 (1)**	**100 (1)**	100 (2)	**100 (1)**	100 (3)	100 (3)
φ_5^{AT}	40	**100 (1)**	**100 (1)**	**100 (1)**	**100 (1)**	**100 (1)**	**100 (1)**	100 (2)	100 (2)
φ_6^{AT}	40	**100 (1)**	100 (2)	100 (2)	100 (2)	**100 (1)**	100 (2)	100 (4)	100 (3)
φ_1^{CC}	40	100 (6)	100 (59)	100 (55)	100 (52)	100 (61)	100 (18)	100 (32)	**100 (5)**
φ_3^{CC}	40	**100 (2)**	100 (18)	100 (28)	100 (24)	100 (26)	100 (8)	100 (13)	100 (5)
φ_5^{CC}	40	100 (56)	100 (49)	100 (47)	100 (34)	100 (70)	**100 (31)**	100 (67)	100 (38)
φ_1^{NN}	3	**100 (7)**	100 (48)	100 (37)	100 (42)	100 (58)	100 (11)	100 (65)	100 (125)

C Unfalsifiable benchmark examples

For some benchmark examples, falsification is challenging. We show here the result for φ_1^{AT} with Instance 2 and φ_1^{SC} with both instances in Table 7. While φ_1^{AT}, with 8 dimensions, shown in Table 4, can be easily falsified, its high-dimensional version with 40 dimensions is hard to falsify, as seen in Table 7. For the *SC* problem, [50] demonstrated that by combining a Simulated Annealing global search with an optimal control-based local search on the infinite-dimensional input space, it is possible to falsify this specification. However, this is not a black-box approach. In [51], the *SC* problem is shown to be falsified by using a pulse generator as the input generator. By optimizing over the period, it is possible to find the right period that falsifies this specification.

Table 6: Results for the specifications of AT', $\Delta - \Sigma$, WT and AFC systems which are not included in the tables 1- 2.

Specifications	Number of Dimensions	vanilla BO	TuRBO TS	TuRBO LCB	TuRBO PI ($\tau = 0$)	TuRBO PI ($\tau = -1$)	πBO	LSF	HCR
$\varphi_1^{AT'}(T = 20)$	10	100 (27)	100 (26)	100 (27)	100 (23)	100 (21)	90 (82)	100 (63)	**100 (1)**
$\varphi_1^{AT'}(T = 30)$	10	100 (73)	100 (40)	100 (50)	100 (43)	100 (37)	100 (79)	100 (172)	**100 (1)**
$\varphi_1^{AT'}(T = 40)$	10	100 (146)	95 (128)	100 (80)	100 (109)	100 (60)	100 (190)	100 (241)	**100 (1)**
$\varphi_2^{AT'}(T = 10)$	10	100 (4)	100 (13)	100 (16)	100 (14)	100 (12)	100 (11)	100 (18)	**100 (3)**
$\varphi_3^{AT'}(T = 5)$	10	**100 (11)**	100 (104)	100 (60)	100 (133)	100 (85)	100 (36)	100 (81)	100 (21)
$\varphi_4^{AT'}(T = 2)$	10	100 (12)	100 (22)	100 (18)	100 (27)	100 (29)	100 (12)	100 (29)	**100 (1)**
$\varphi_5^{AT'}(T = 2)$	10	**100 (1)**	100 (4)	100 (3)	100 (3)	100 (3)	100 (2)	100 (6)	100 (10)
$\varphi_8^{AT'}(\bar{\omega} = 3000)$	10	**100 (4)**	100 (9)	100 (13)	100 (10)	100 (7)	100 (5)	100 (11)	100 (5)
$\varphi_1^{\Delta-\Sigma}\ U \in [-0.40, 0.40]$	4	100 (11)	100 (57)	100 (79)	100 (39)	100 (115)	100 (14)	100 (52)	**100 (5)**
$\varphi_1^{\Delta-\Sigma}\ U \in [-0.45, 0.45]$	4	**100 (11)**	100 (29)	100 (54)	100 (42)	100 (64)	100 (21)	100 (39)	**100 (11)**
φ_1^{WT}	126	**100 (1)**	100 (2)	**100 (1)**	100 (2)	**100 (1)**	**100 (1)**	100 (3)	100 (3)
φ_2^{WT}	126	**100 (1)**	**100 (1)**	**100 (1)**	**100 (1)**	**100 (1)**	**100 (1)**	100 (2)	100 (2)
φ_3^{WT}	126	**100 (1)**	**100 (1)**	**100 (1)**	**100 (1)**	**100 (1)**	**100 (1)**	100 (2)	100 (2)
φ_4^{WT}	126	100 (37)	100 (85)	100 (112)	100 (125)	100 (111)	100 (37)	100 (66)	**100 (30)**
φ_1^{AFC}	11	**100 (1)**	100 (74)	100 (81)	100 (91)	100 (72)	100 (24)	100 (6)	100 (5)
φ_1^{AFC}	11	**100 (1)**	100 (15)	100 (20)	100 (15)	100 (10)	100 (8)	100 (2)	100 (5)

Table 7: Results for the specifications of φ_1^{AT}, Instance 2 and φ_1^{SC} for both instances.

Specifications	Instances	Number of Dimensions	vanilla BO	TuRBO TS	TuRBO LCB	TuRBO PI ($\tau = 0$)	TuRBO PI ($\tau = -1$)	πBO	LSF	HCR
φ_1^{AT}	Instance 2	40	0 (-)	0 (-)	0 (-)	0 (-)	0 (-)	0 (-)	0 (-)	0 (-)
φ_1^{SC}	Instance 1	12	0 (-)	0 (-)	0 (-)	0 (-)	0 (-)	0 (-)	0 (-)	0 (-)
φ_1^{SC}	Instance 2	20	0 (-)	0 (-)	0 (-)	0 (-)	0 (-)	0 (-)	0 (-)	0 (-)

D An Aggregated Comparison

An aggregated comparison among all methods, including benchmark examples from the appendix, is shown in Fig. 3. In general, TuRBO shows good performance w.r.t. state-of-the-art methods. In particular, selecting LCB provides better results than LSF. However, PI and TS are less efficient than LSF, which was specifically developed to solve falsification problems.

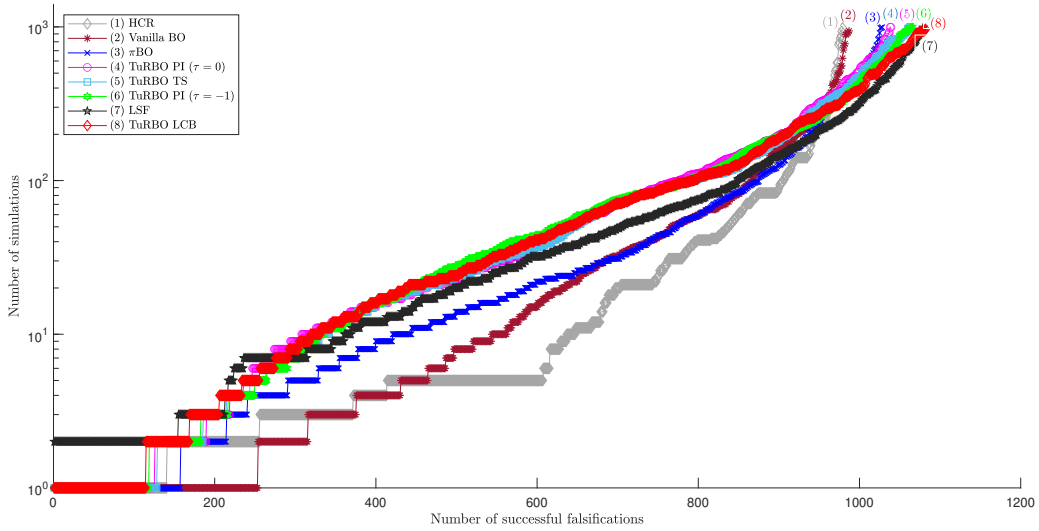

Figure 3: A cactus plot showing the performance of all examples. The plotted values show how many successful falsifications (x-axis) were completed for a given number of simulations (y-axis, logarithmic scale). A maximum of 1000 simulations are evaluated.