Falsification using Reachability of Surrogate Koopman Models

Stanley Bak
stanley.bak@stonybrook.edu
Stony Brook University
New York, USA

Sergiy Bogomolov
bogom.s@gmail.com
Newcastle University
Newcastle upon Tyne, UK

Abdelrahman Hekal
b6062805@ncl.ac.uk
Newcastle University
Newcastle upon Tyne, UK

Niklas Kochdumper
niklas.kochdumper@gmail.com
Stony Brook University
New York, USA

Ethan Lew
elew@galois.com
Galois, Inc
Portland, Oregon, USA

Andrew Mata
andrew.mata@stonybrook.edu
Stony Brook University
New York, USA

Amir Rahmati
amir@rahmati.com
Stony Brook University
New York, USA

Abstract

Black-box falsification problems are most often solved by numerical optimization algorithms. In this work, we propose an alternative approach, where simulations are used to construct a surrogate model for the system dynamics using data-driven Koopman operator linearization. Since the dynamics of the Koopman model are linear, the reachable set of states can be computed and combined with an encoding of the signal temporal logic specification in a mixed-integer linear program (MILP). To determine the next sample, an MILP solver computes the least robust trajectory inside the reachable set of the surrogate model. The trajectory's initial state and input signal are then executed on the original black-box system, where the specification is either falsified or additional simulation data is generated that we use to retrain the surrogate Koopman model and repeat the process.

The proposed method is highly effective. Evaluation on the complete set of benchmarks taken from the 2022 ARCH falsification competition demonstrates superior performance—fewer expected simulations—over all participating tools on 16 out of 19 benchmarks. Further, on three benchmarks where no tool consistently reports a falsifying trace, our method reliably uncovers a counterexample.

CCS Concepts

• **Computer systems organization** → **Embedded and cyber-physical systems**; • **Mathematics of computing** → *Mixed discrete-continuous optimization*; *Numerical analysis.*

Keywords

Cyber-Physical Systems, Falsification, Signal Temporal Logic, Koopman Operator Linearization, Reachability Analysis

ACM Reference Format:
Stanley Bak, Sergiy Bogomolov, Abdelrahman Hekal, Niklas Kochdumper, Ethan Lew, Andrew Mata, and Amir Rahmati. 2024. Falsification using Reachability of Surrogate Koopman Models. In *27th ACM International Conference on Hybrid Systems: Computation and Control (HSCC '24), May 14–16, 2024, Hong Kong SAR, China.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3641513.3650141

HSCC '24, May 14–16, 2024, Hong Kong SAR, China

ACM ISBN 979-8-4007-0522-9/24/05
https://doi.org/10.1145/3641513.3650141

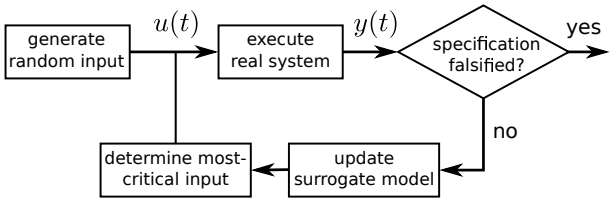

Figure 1: Our falsification approach uses formal analysis of a surrogate model to determine the next input signal to try.

1 Introduction

In recent years, vast effort has been focused on embedding computing and communication capabilities in objects and structures in the physical environment [29, 40]. Such cyber-physical systems (CPS), including autonomous vehicles, aerospace systems, and medical applications, are often safety-critical, where human life could be endangered if these systems fail. Consequently, reasoning about the safety of such systems has become a topic of substantial interest.

Although formal verification approaches for CPS, such as reachability analysis [5, 11] and theorem proving [39, 42] have made great progress, many real-world CPS are still too complex or modeled in frameworks that preclude analysis by existing formal methods. Thus, there is an urgent need for alternative analysis techniques.

One such method is falsification, which is an orthogonal approach to verification: while verification aims to prove that a system satisfies all safety properties, falsification tries to find counterexamples that violate such properties. The corresponding safety properties are often represented in signal temporal logic (STL) [33], which can model complex behaviors of continuous system states over time.

In this work, we present a new approach for falsification, whose high-level concept is visualized in Fig. 1. In contrast to most existing falsification methods, which aim to falsify systems by solving a bounded optimization problem numerically, our approach is based on the construction of a surrogate model using Koopman operator linearization [26, 28]. Koopman operator linearization is a data-driven modeling approach. It uses machine learning to create a system model approximation from execution trajectories. It has been successfully applied to predict behaviors in a set of diverse domains [28], including epidemiology, neuroscience, and financial trading. It has also been used in CPS for model predictive control [27] and state estimation [38]. Our approach is, to the best of our knowledge, the first that uses it for falsification.

One of the advantages of the Koopman operator approach is that it results in a *linear* representation of the system dynamics, while

being more accurate than other linearization techniques such as Taylor series expansion [25] or state-space linearization [30]. The linear nature of the Koopman model allows us to determine witness initial state and input signals that falsify the surrogate model efficiently via mixed-integer linear programming. However, a counterexample of the surrogate model may not correspond to a counterexample for the original black-box system. Therefore, if the counterexample proves to be spurious, we use it to further refine the learned Koopman model. This iterative process continues until a valid counterexample for the black-box system is found.

The main contributions of this paper are as follows:

- We present the first approach that uses Koopman operator linearization for black-box falsification of CPS. Our framework iteratively constructs a surrogate model that gets analyzed using MILP to select the next trajectory.
- We optimize our approach in several ways, using (1) reachable set information to simplify the MILP problem, (2) periodic data resets to improve the model's local accuracy, and (3) a specification offset approach to counteract observed errors in the learned Koopman model.
- We demonstrate the effectiveness of our approach by a comparison to state-of-the-art falsification tools using the complete set of benchmarks from the 2022 ARCH falsification competition [18]. Our approach records a lower number of average simulations required to find a falsifying trace for 16 out of 19 benchmarks.

2 Related Work

The most common approach for falsification is to apply quantitative robustness semantics for STL [20], which maps a trajectory of the system to a scalar robustness value that describes how robustly the trajectory satisfies a temporal logic specification. Input signals are usually parameterized by the values of the input signal at a finite set of control points that are distributed over the considered time horizon. The continuous-time input signal is then obtained by interpolating between the values at the control points using, for example, piecewise constant, linear, or polynomial interpolation. Falsification thus becomes a numerical optimization problem, with the goal of finding the global minimum robustness over the bounded search space.

While some deterministic optimization strategies such as gradient-decent have been considered [2], most falsification approaches apply probabilistic approaches for global optimization, such as hit-and-run sampling in combination with Monte Carlo techniques [37], simulated annealing [1], Bayesian optimization [13], the cross-entropy method [44], ant-colony optimization [7], stochastic optimization with adaptive restarts [35], or Tabu search [14]. These probabilistic methods usually aim to either sample points more frequently from regions that are expected to have low robustness (e.g., cross-entropy method), or from regions where the uncertainty on the robustness values is large (e.g., Bayesian optimization). One common challenge with numerical optimization falsification approaches is that they usually perform poorly if the search space is high-dimensional due to the large number of required sample points. Moreover, quantitative STL semantics reduce the system behavior to a scalar robustness value, and so optimization-based falsification approaches ignore the dynamics of the system to a large extent. In contrast, our learned surrogate models—and therefore our sample selection strategy—incorporate relationships about all of the system states over time.

Improvements and extensions for falsification include tailored approaches for falsifying conjunctions of multiple requirements [34], and using different robustness semantics such as additive robustness [17] or QB-robustness [51]. Some approaches adapt the space of possible input signals by optimizing the number of control points [3] or by refining the temporal and spacial granularity of the input signals [19]. In addition, there exist multiple shooting approaches [52] that aim to minimize the gaps between multiple smaller trajectory segments. Reinforcement learning [49] has been applied for falsification. Other approaches [8, 10][24, Sec. 4.1] extract falsifying trajectories from reachable sets, but are restricted to white-box systems with simple reach-avoid specifications.

Simple surrogate models have been considered for falsification. One approach [47] learns a Mealy automaton as a surrogate model and then falsifies this via model checking. Another method [36] uses data-driven system identification [45] to replace a compute-intensive black-box CPS simulator, but then still relies on numerical optimization for falsification. Our approach, in contrast, learns a continuous surrogate model and computes its least robust trajectory using a combination of reachability analysis and MILP.

Common tools for falsification are ARIsTEO [36], Breach [15], FalCAuN[1], falsify[2], FalStar[3], ForeSee[4], S-TaLiRo [6], and Ψ-TaLiRo [46]. ARIsTEO is build on top of S-TaLiRo and tailored toward systems where simulations are computationally expensive, FalCAuN uses the Mealy automaton approach [47], falsify applies reinforcement learning [49], FalStar is based on input signal refinement [19], ForeSee uses QB-robustness [51], Breach and S-TaLiRo support several different optimization techniques such as simulated annealing and stochastic optimization with adaptive restarts, and Ψ-TaLiRo is the Python version of S-TaLiRo.

3 Problem Formulation

We represent a CPS system as a black-box model $\mathcal{M}$, which takes as input an initial state $x_0 \in \mathbb{R}^n$ and an input signal $w(t) : [0,T] \to \mathbb{R}^m$ with time horizon T and produces an output signal $y(t) : [0,T] \to \mathbb{R}^s$:

$$y(t) \leftarrow \mathcal{M}\big(x_0, w(t)\big). \tag{1}$$

A trajectory or trace of system (1) is represented by a tuple $\tau = (x_0, w(t), y(t))$ consisting of the initial state x_0 as well as the input signal $w(t)$ and the corresponding output signal $y(t)$.

Specifications are used to describe the desired system behavior and are represented as signal temporal logic (STL) formulas [33]:

Definition 1 (Signal Temporal Logic). *Given a set of atomic predicates $p \in \mathcal{A}$ which are defined as $p := f(y) > 0$, where $f : \mathbb{R}^s \to \mathbb{R}$ is a nonlinear function, the syntax for a signal temporal logic formula is*

$$\varphi := \text{True} \mid p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \Box_{[a,b]}\varphi \mid \Diamond_{[a,b]}\varphi \mid \varphi_1 U_{[a,b]} \varphi_2,$$

where a and b with $b \geq a$ are non-negative scalars denoting time bounds. For a signal $y(t) : \mathbb{R}_{\geq 0} \to \mathbb{R}^s$, the semantics of STL is defined as follows:

$$y(t) \models p \quad \Leftrightarrow \quad f(y(0)) > 0$$

[1] https://github.com/MasWag/FalCAuN
[2] https://github.com/yoriyuki-aist/Falsify/
[3] https://github.com/ERATOMMSD/falstar
[4] https://github.com/choshina/ForeSee

$$
\begin{aligned}
y(t) \models \neg\varphi \quad &\Leftrightarrow \quad \neg(y(t) \models \varphi) \\
y(t) \models \varphi_1 \wedge \varphi_2 \quad &\Leftrightarrow \quad (y(t) \models \varphi_1) \wedge (y(t) \models \varphi_2) \\
y(t) \models \varphi_1 U_{[a,b]} \varphi_2 \quad &\Leftrightarrow \quad \exists c \in [a,b] : y(t+c) \models \varphi_2 \\
&\qquad\qquad \wedge\ \forall d \in [0,c) : y(t+d) \models \varphi_1 .
\end{aligned}
$$

Moreover, the semantics for the finally operator $\Diamond_{[a,b]}\varphi := \text{True}\, U_{[a,b]}\varphi$ and the globally operator $\Box_{[a,b]}\varphi := \neg\Diamond_{[a,b]}\neg\varphi$ directly follows from the semantics of the until operator $\varphi_1 U_{[a,b]}\varphi_2$.

To guide the search toward signals that violate the specification, we will use the quantitative robustness for temporal logic [20], which specifies how robustly a signal satisfies an STL formula:

Definition 2 (Robustness). *The quantitative robustness semantics of STL is represented by a function $Q(\varphi, y(t))$ that maps an STL formula φ and a signal $y(t)$ to a scalar value. This function is recursively defined as follows:*

$$
\begin{aligned}
Q(\text{True}, y(t)) &:= \infty \\
Q(p, y(t)) &:= f(y(t)) \\
Q(\neg\varphi, y(t)) &:= -Q(\varphi, y(t)) \\
Q(\varphi_1 \wedge \varphi_2, y(t)) &:= \min\big(Q(\varphi_1, y(t)), Q(\varphi_2, y(t))\big) \\
Q(\varphi_1 \vee \varphi_2, y(t)) &:= \max\big(Q(\varphi_1, y(t)), Q(\varphi_2, y(t))\big) \\
Q(\Box_{[a,b]}\varphi, y(t)) &:= \min_{c \in [t+a, t+b]} Q(\varphi, y(c)) \\
Q(\Diamond_{[a,b]}\varphi, y(t) &:= \max_{c \in [t+a, t+b]} Q(\varphi, y(c)) \\
Q(\varphi_1 U_{[a,b]} \varphi_2, y(t)) &:= \max_{c \in [t, t+b]} \min\Big(Q(\varphi_2, y(c)), \\
&\qquad\qquad \min_{d \in [c-a, c]} Q(\varphi_1, y(d)) \Big).
\end{aligned}
$$

The robustness measures the extent to which a signal $y(t)$ satisfies an STL formula φ, where $Q(\varphi, y(t)) \geq 0$ entails satisfaction $y(t) \models \varphi$, and larger values for $Q(\varphi, y(t)$ indicate stronger satisfaction.

With the approach presented in this paper, we aim to solve falsification tasks, which are defined as follows:

Problem 1 (Falsification). *Given a CPS model $\mathcal{M}$ as in* (1)*, a system specification in form of a signal temporal logic formula φ as in Def. 1, a set of uncertain initial states $\mathcal{X}_0 \subset \mathbb{R}^n$, and a set of uncertain inputs*[5] *$w(t) \in \mathcal{W}$, the goal of falsification is to find an initial state $x_0 \in \mathcal{X}_0$ and an input signal $w(t) \in \mathcal{W}$ such that the corresponding output $y(t) \leftarrow \mathcal{M}(x_0, w(t))$ violates the specification $y(t) \not\models \varphi$.*

For ease of notation in the remainder of the paper, we combine the initial state x_0 and the input signal $w(t)$ into a tuple $u(t) = (x_0, w(t))$ that represents all inputs to the system. Consequently, (1) simplifies to $y(t) \leftarrow \mathcal{M}(u(t))$, and we write $u(t) \in \mathcal{U}$ with $\mathcal{U} = \mathcal{X}_0 \times \mathcal{W}$ to denote $x_0 \in \mathcal{X}_0$ and $w(t) \in \mathcal{W}$.

To better illustrate our approach, we will use the following running example throughout the paper [22], taken from the 2022 ARCH falsification competition [18]:

Example 1 (**Running Example**). *An automatic transmission controller selects a discrete gear from 1 to 4. The system has two inputs $w(t) = [throttle, brake]^T$ that are uncertain within the set $\mathcal{W} = [0,100] \times [0,325]$, and three outputs $y(t) = [g, v, \omega]^T$, where g is the gear, v is the speed of the vehicle, and ω is the angular velocity of the engine. The goal of the falsification problem is to find a trace that violates the STL formula $\varphi = \Diamond_{[0,30]}\omega \geq 3000 \vee \Box_{[0,4]} v < 35$ over a simulation time of $T = 30s$. That is, finding a trace where the angular velocity is less than 3000 for the whole duration of the simulation and the speed exceeds 35 in the first 4 seconds.*

[5] We use $w(t) \in \mathcal{W}$ as a shorthand for $\forall t \in [0,T] : w(t) \in \mathcal{W}$.

Algorithm 1: Koopman Surrogate Falsification

Data: Black-box CPS model $\mathcal{M}$, STL formula φ, input space $\mathcal{U}$, time horizon T, maximum number of simulations $N_{\max}$
Result: Counterexample trajectory τ

1 $\tau \leftarrow$ RunSimulation($\mathcal{M}$, RandomInput($\mathcal{U}$), T)
2 $\mathcal{T} \leftarrow \{\tau\}$
3 **for** $i = 1$ **to** N_{max} **do**
4 $\quad \mathcal{M}_K \leftarrow$ LearnKoopmanModel($\mathcal{T}$)
5 $\quad \mathcal{R}(t) \leftarrow$ ComputeReachableSet($\mathcal{M}_K, \mathcal{U}, T$)
6 $\quad u^* \leftarrow \underset{u(t) \in \mathcal{U}}{\text{argmin}}\ Q(\varphi, y(t))$ s.t. $\begin{cases} y(t) \leftarrow \mathcal{M}_K(u(t)) \\ \forall t \in [0,T] :\ y(t) \in \mathcal{R}(t) \end{cases}$
7 $\quad \tau \leftarrow$ RunSimulation($\mathcal{M}, u^*, T$)
8 $\quad$ **if** $Q(\varphi, \tau) < 0$ **then**
9 $\quad\quad$ **return** τ `// counterexample found`
10 $\quad$ **end**
11 $\quad \mathcal{T} \leftarrow \mathcal{T} \cup \{\tau\}$
12 **end**

4 Koopman Surrogate Falsification

In this paper, we solve the falsification problem described in Sec. 3 by constructing and analyzing a surrogate model for the black-box CPS. The details of our framework shown in Fig. 1 are outlined in Alg. 1: First, we run a single simulation of the real system with random inputs and extract the corresponding trajectory τ (Line 1). We use this trajectory to learn a surrogate model of the system using Koopman operator linearization (Line 4). Next, we compute the reachable set of the Koopman model for the set of all possible inputs $\mathcal{U}$ over the time horizon T (Line 5). Afterward, we determine the input u^* to the Koopman model that violates the specification the most, by minimizing the robustness in the optimization problem in Line 6, where the reachable set is used to encode constraints on the system states. Finally, the determined input is used to generate a trajectory of the real system, which is again checked against the STL specification. If the counterexample violates the STL spec on the real system, the loop terminates; otherwise, the extracted trajectory is used to further refine the Koopman model. In the following, we describe each of the steps in detail, as well as several enhancements that improve performance.

4.1 Koopman Operator Linearization

Koopmean operator linearization [26, 28] is a machine learning technique that infers a symbolic system model from trajectory data. Compared with other competing methods like neural networks, the Koopman operator approach can work with relatively little training data—a single trajectory is often enough to produce a useful model. Further, the learned model is a high-dimensional linear model. Efficient

analysis methods for linear systems, such as reachability analysis, can therefore be used to reason over the learned model.

Koopman operator linearization works by applying a *nonlinear* state space transformation defined by an observable function $g(x)$: $\mathbb{R}^n \rightarrow \mathbb{R}^q$ to the system state x, with usually $q > n$. This transforms the system to a higher-dimensional space where the dynamics is represented by a *linear* discrete-time system.

$$\begin{aligned} g(x(t_{i+1})) &= A\, g(x(t_i)) + B\, w(t_i) \\ y(t_i) &= C\, g(x(t_i)), \end{aligned} \tag{2}$$

where $A \in \mathbb{R}^{q \times q}$, $B \in \mathbb{R}^{q \times m}$, and $C \in \mathbb{R}^{s \times q}$ are the corresponding system matrices. The time points $t_{i+1} = t_i + \Delta t$ are obtained via discretization with time step size Δt, where Δt is a user-defined parameter.

While, at least for systems with purely continuous dynamics, it is in theory possible to construct a Koopman model that represents the dynamics of the real system *exactly* [26], this is in general computationally infeasible. Instead, we aim to construct a Koopman model that instead closely *approximates* the dynamic behavior of the real system, which may still be useful for falsification. In Alg. 1, the behavior of the real system is captured by a set $\mathcal{T}$ of system trajectories τ, which are obtained by simulating the real system for a specific input. Koopman operator linearization uses this data to determine an observable function $g(x)$ as well as system matrices A, B, and C.

The standard approach for Koopman operator linearization first selects a set of observables $g(x)$, and then computes the system matrices A, B, and C that yield the best approximation of the trajectories from the real system [48]. The observable function $g(x)$ could be polynomials [43], random Fourier features [12] or neural networks [21, 50]. One challenge with Koopman operator linearization is that the model's accuracy depends on many parameters such as the type and number of observables, the matrix rank used in a dynamic mode decomposition step, or parameters for describing the observables $g(x)$. We bypass such issues by using the recently-developed AutoKoopman framework [31], which automates hyper-parameter tuning to obtain the best-fitting Koopman model.

In Alg. 1, we rely on an iterative process to construct the Koopman model: First, a single random trajectory is used to learn the initial Koopman model. Afterward, we append the set of trajectories $\mathcal{T}$ used to construct the Koopman model in each iteration with the newly obtained trajectory in case the system is not falsified. Fig. 2 shows a comparison of the trajectory from the real system and the prediction from the Koopman model for our running example after one iteration of Alg. 1. Despite the small amount of training data, the trajectories are qualitatively similar, although some quantitative differences are apparent.

Often, Koopman operator linearization performs very well for learning local models that are accurate in a specific region of the state space but struggles to identify a good global model that is accurate for the overall state space. Based on this observation, we investigate an enhancement that resets the training set $\mathcal{T}$ to the empty set after N_{reset} iterations, where N_{reset} is a user-defined parameter. This process intuitively focuses Koopman operator linearization to learn a good local model for the region of the state space that is currently explored rather than aiming to learn an accurate global model for the whole state space. For the same reason, we also consider removing the initial random trajectory from the training set. The benefit of this enhancement is analyzed later in Sec. 5.3. In Alg. 1, this is done in place of appending the new trajectory on line 11.

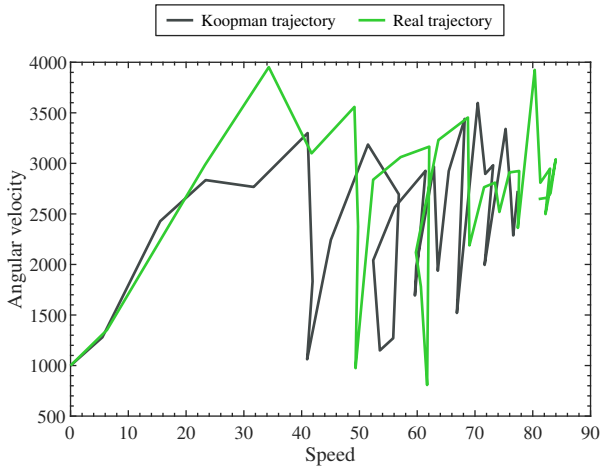

Figure 2: Comparison between the trajectory from the real system and the prediction from the surrogate Koopman model for the system in Example 1.

4.2 Reachability Analysis

Line 5 of Alg. 1 performs reachability analysis on the Koopman linearized model. Although the Koopman dynamics matrix is linear, the nonlinear initialization using $g(x)$ makes the initial set of states non-convex. We use a recently-developed reachability algorithm [9], where polynomial zonotopes [23] represent non-convex sets[6]:

Definition 3 (Polynomial Zonotope). *Given a constant offset $c \in \mathbb{R}^n$, a generator matrix $G \in \mathbb{R}^{n \times h}$, and an exponent matrix $E \in \mathbb{N}_{\geq 0}^{p \times h}$, a polynomial zonotope $\mathcal{PZ} \subset \mathbb{R}^n$ is defined as*

$$\mathcal{PZ} := \left\{ c + \sum_{j=1}^{h} \left(\prod_{k=1}^{p} \alpha_k^{E_{(k,j)}} \right) G_{(\cdot,j)} \;\middle|\; \alpha_k \in [-1,1] \right\},$$

where the scalars α_k are called factors, $G_{(\cdot,j)}$ denotes the j-th column of matrix G, and $E_{(k,j)}$ denotes the k-th entry in column j of matrix E.

The procedure for computing the reachable set is as follows: First, Taylor model arithmetic [32] is used to map the initial set $\mathcal{X}_0$ through the observable function $g(x)$, which yields the initial reachable set $\mathcal{R}_g(0)$ in the high-dimensional Koopman space. Next, the resulting Taylor model is converted to a polynomial zonotope [23, Prop. 4]. Since the dynamics of the Koopman model is represented by a discrete-time linear system (2), the reachable set $\mathcal{R}(t)$ can be computed with the following propagation rule:

$$\begin{aligned} \mathcal{R}_g(t_{i+1}) &= A\, \mathcal{R}_g(t_i) \oplus B\, \mathcal{W} \\ \mathcal{R}(t_i) &= C\, \mathcal{R}_g(t_i). \end{aligned}$$

The two set operations that are required are linear map $H\, \mathcal{S} := \{H\, s \mid s \in \mathcal{S}\}$ with $H \in \mathbb{R}^{m \times n}$, $\mathcal{S} \subset \mathbb{R}^n$ and Minkowski sum $\mathcal{S}_1 \oplus \mathcal{S}_2 := \{s_1 + s_2 \mid s_1 \in \mathcal{S}_1, s_2 \in \mathcal{S}_2\}$ with $\mathcal{S}_1, \mathcal{S}_2 \subset \mathbb{R}^n$, which can be computed efficiently for polynomial zonotopes [23, Prop. 8 and 9]. To speed-up subsequent computations, we enclose the polynomial zonotopes that represent the reachable sets $\mathcal{R}(t_i)$ by regular zonotopes [23, Prop. 5]:

[6] In contrast to [23, Def. 1], we do not include c into G, and we omit the independent generators and the unique identifiers for simplicity

Definition 4 (Zonotope). *Given a constant offset $c \in \mathbb{R}^n$ and a generator matrix $G \in \mathbb{R}^{n \times p}$, a zonotope $\mathcal{Z} \subset \mathbb{R}^n$ is defined as*

$$\mathcal{Z} := \left\{ c + \sum_{k=1}^{p} G_{(\cdot,k)}\, \alpha_k \;\middle|\; \alpha_k \in [-1,1] \right\},$$

where $G_{(\cdot,k)}$ denotes the k-th column of matrix G.

One key property of this reachability algorithm is that it **preserves dependencies across time** [24]. It is not the case that every point in the reachable set $\mathcal{R}(t_i)$ at some time t_i can get to every point in the reachable set $\mathcal{R}(t_j)$ at some later time t_j. Dependency preservation can track which pairs of states are actually possible between the two set of states (or in general, across longer trajectories of the system). This is done by requiring factors α_k that are common in the polynomial zonotope representation of the reachable set at different points in time to be equal.For proving that a system avoids a fixed unsafe set, dependency preservation does not matter. However, with temporal logic specifications, single parts of the formula at different times can interact with each other. Dropping dependencies by considering only, for example, the box bounds of the reachable states at different points in time, would be a large overapproximation of the actual possible set of system trajectories, and could lead to more spurious counterexamples during falsification.

4.3 Robustness Minimization with MILP

We now describe how to solve the optimization problem in Line 6 of Alg. 1. One of the advantages of using Koopman operator linearization is that the dynamic behavior of the system is linear. This allows us to formulate the optimization problem as an MILP consisting of two parts: (1) the encoding of the dynamic behavior of the Koopman model and (2) the encoding of the robustness of the STL formula.

Dynamic Behavior. Since the reachability algorithm we used to compute the reachable set in Sec. 4.2 preserves dependencies, the dynamic behavior of the system defined by (2) is captured by the zonotopes that represent the reachable sets $\mathcal{R}(t_i)$ at the discrete time points t_i. Further, we computed the reachable set for the set of all possible initial states $x_0 \in \mathcal{X}_0$ and the set of all possible input signals $w(t) \in \mathcal{W}$, and so the constraint $u(t) \in \mathcal{U}$ on the system input is implicitly represented by the reachable set. To encode the dynamic behavior of model, it therefore suffices to add the constraint $y(t_i) \in \mathcal{R}(t_i)$ for all time points t_i to the optimization problem. Since reachable sets are represented by zonotopes, we can represent this set by the constraints

$$\begin{aligned} y_i &= c + G[\alpha_{i,1},\ldots,\alpha_{i,p}]^T \\ -1 &\le \alpha_{i,k} \le 1, \quad k = 1,\ldots,p, \end{aligned}$$

where the optimization problem variables are the system outputs $y_i \in \mathbb{R}^s$ at time t_i and the factors $\alpha_{i,k}$ of the reachable set zonotopes.

It is also possible to encode the dynamic behavior of the Koopman model directly into the optimization problem. However, using reachability analysis first has two big advantages: (1) without computing the reachable set, the dynamic constraints would be encoded in the higher-dimensional observable space. This would increase the number of decision variables in the MILP problem, reducing solver efficiency; (2) the mapping $g(x)$ from states to observables is typically nonlinear. For falsification problems with uncertain initial states $x_0 \in \mathcal{X}_0$, a direct encoding would result in a nonlinear constraint

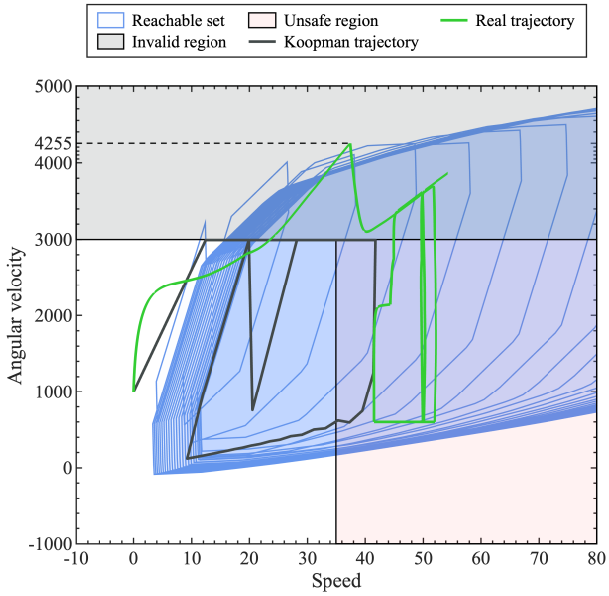

Figure 3: MILP computes a trace that violates the specification in the Koopman linearized system, but the corresponding trajectory in the original black-box system enters the invalid region and therefore this is not a valid counterexample.

that is not allowed in an MILP. With reachability analysis, we can avoid this issue since zonotope enclosures remove the nonlinearities. These advantages are underlined by a numerical performance comparison between reachability analysis and direct encoding in Sec. 5.3.

STL Robustness. To encode the robustness of an STL formula in MILP formulation, we build off a prior approach [41, Sec. V]. The quantitative robustness $Q(\varphi, y(t))$ of STL, as in Def. 2, is computed recursively using min() and max() operators. The main concept of the approach is to translate these operators into an MILP formulation by using additional binary variables and the Big-M method. In particular, we use the same time-discretization as for reachability analysis and represent the robustness of an STL formula φ at time t_i by a variable r_i^{φ} in the optimization problem. If the STL formula represents a single predicate $p := f(y(t)) > 0$, the robustness is according to Def. 2 simply given by the value of the function that defines the predicate $r_i^{\varphi} = f(y_i)$. The MILP encoding requires the predicates to be linear or affine [7]

The robustness of $\varphi = \varphi_1 \wedge \varphi_2$ according to Def. 2 is $Q(\varphi, y(t)) = \min(Q(\varphi_1, y(t)), Q(\varphi_2, y(t)))$. It can encoded by adding the following MILP constraints for all time steps i:

$$\begin{aligned} r_i^{\varphi_1} - (1 - z_i)M \le r_i^{\varphi} \le r_i^{\varphi_1} + (1 - z_i)M \\ r_i^{\varphi_2} - z_i M \le r_i^{\varphi} \le r_i^{\varphi_2} + z_i M \\ r_i^{\varphi} \le r_i^{\varphi_1}, \quad r_i^{\varphi} \le r_i^{\varphi_2}, \quad z_i \in \{0,1\}, \end{aligned}$$

where we introduce an additional binary variable z_i, and $M \in \mathbb{R}_{\ge 0}$ is a sufficiently large number. Here, the combination of the Big-M method and the binary variable z_i ensures that only one of the first two constraints is active, and the third constraint forces it to be the

[7] Alternatively, predicates can be any convex or even nonlinear function for which corresponding solvers can be leveraged.

Algorithm 2: Specification Offset Strategy

Data: STL formula φ, real system trajectory τ
Result: Modified STL formula φ_{offset}

1 $\mathcal{A} \leftarrow$ ExtractAtomicPredicates(φ)
2 $\varphi_{\text{offset}} \leftarrow$ RecursiveOffset($\varphi, \tau, \mathcal{A}$)
3 **Function** RecursiveOffset($\varphi, \tau, \mathcal{A}$):
4 $Q \leftarrow Q(\varphi, \tau)$ // get current robustness
5 **foreach** $p_j \in \mathcal{A}$ **do**
6 **foreach** $s \in \{-1, 1\}$ **do**
7 $p_j^* \leftarrow p_j + s \cdot Q$
8 $\varphi^* \leftarrow \varphi[p_j^* \rightarrow p_j]$ // replace p_j with p_j^* in φ
9 $Q^* \leftarrow Q(\varphi^*, \tau)$ // get updated robustness
10 **if** $Q^* < Q$ **then**
11 **if** $Q^* \leq 0$ **then**
12 **return** φ^*
13 **else**
14 $\varphi^* \leftarrow$ RecursiveOffset($\varphi^*, \tau, \mathcal{A}$)
15 **end**
16 **end**
17 **end**
18 **end**
19 **end**

one corresponding to the minimum. In a similar fashion, one can encode disjunctions as well as all discrete-time temporal operators.

Fig. 3 shows the result of the optimization problem for our running example after one iteration of Alg. 1. The MILP solver computes the lowest robustness Koopman trajectory inside the reachable set. The returned Koopman trajectory has negative robustness, indicating that it falsifies the system. However, the corresponding trajectory of the original black-box system violates the first requirement of the falsification problem as the angular velocity exceeds 3000. In order to try to reduce the error in the model approximation, the real trajectory is added to the set of simulation data that is used to retrain the Koopman model, and the process is repeated.

4.4 Specification Offset

In Figure 3, approximation error in the surrogate model caused a spurious counterexample. In this section, we consider a second way to prevent spurious counterexamples (other than retraining) where we use the observed model error to adjust the optimization target. In the figure, the real trajectory has both higher angular velocity and higher speed than the Koopman trajectory. A modified optimization target could try to keep the angular velocity lower to avoid the invalid region, even if it means the speed cannot go as far into the unsafe region. We outline this process below, calling it *specification offset*.

STL formulae are often composed of multiple sub-formulae, which may exhibit varying levels of scale and complexity. For instance, in our running example the specification is $\varphi = \Diamond_{[0,30]} \omega \geq 3000 \vee \Box_{[0,4]} v < 35$. There exists a mismatch in values for the angular velocity of the engine ω and the speed of the vehicle v, where the former is typically on the order of thousands and the latter is on the order

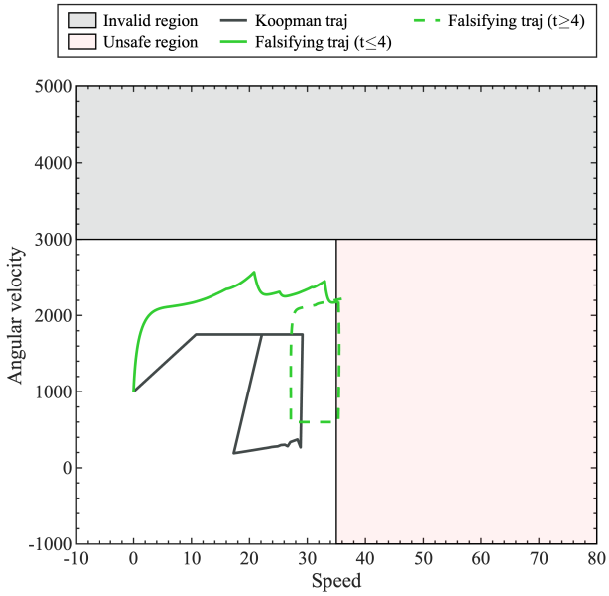

Figure 4: By applying the specification offset strategy described in Sec. 4.4, the witness inputs for the lowest robustness trajectory of the Koopman linearized system violate the original STL specification on the original black-box system.

of tens. This mismatch can be linked to the *scale problem*, a commonly recognized issue in falsification [51] where one sub-formula masks the effect of another. In contrast to numerical falsification, the scale problem in our approach mainly arises due to approximation error in the surrogate model. In Fig. 3, for example, the absolute error of the angular velocity in the learned Koopman model is significantly larger than the absolute error of speed. This mismatch in error, driven by the scale difference, results in the real trajectory failing to falsify the system, as ω does not stay below 3000.

To counteract the error in model learning and the effect of scale, we propose an offset technique, presented in Alg. 2. This enhancement modifies the STL formula φ used in the MILP optimization step, and can be done after each iteration of the main falsification algorithm in Alg. 1 (after line 11). In particular, if the trajectory of the real system fails to falsify the specification and φ is composed of more than one atomic predicate p, we modify φ by applying the offset strategy described in Alg. 2. This algorithm identifies and modifies the predicate(s) responsible for the failure in falsification. In the main loop of the algorithm (Line 5-19), each atomic predicate p is offset by the current robustness value of φ (Line 7), in both the positive and negative direction, until a responsible predicate p is found, which is the case if the value for the robustness decreases (Line 10). The formula is then updated with the offset predicate (Line 8), and the process is repeated until all responsible predicates are identified, which is the case if the robustness reaches zero (Line 11). The modified STL formula that is returned by Alg. 2 is then used to bias the optimizer in Alg. 1 when searching for a counterexample. This strategy modifies the constraints imposed by φ, guiding the system towards a counterexample while considering the inaccuracies in the learned model and the challenges posed by scale issues.

Fig. 4 shows the effect of the offset strategy on the running example. Previously in Fig. 3, after the first iteration of Alg. 1, the

robustness of the trajectory from the real system is $Q = 4255 - 3000 = 1255$, and the predicate $\omega \geq 3000$ is responsible for the failure of falsification. Alg. 2 returns the modified STL formula $\varphi_{\text{offset}} = \square_{[0,30]}\omega \geq (3000 - Q) \vee \square_{[0,4]} v < 35 = \square_{[0,30]}\omega \geq 1745 \vee \square_{[0,4]} v < 35$. As before, the real trajectory still has both higher angular velocity and higher speed than the Koopman trajectory, but the stricter bounds compensate for the large approximation error of the Koopman model for the angular velocity ω. Consequently, although the Koopman trajectory of minimum robustness does not falsify the modified spec φ_{offset}, the corresponding trajectory of the real system falsifies the original spec φ.

5 Numerical Evaluation

We implemented our approach [8] in MATLAB/Simulink, using AutoKoopman [31] to learn the Koopman surrogate model, CORA [4] for set-based computing and reachability analysis, Breach [15] for efficient computation of the quantitative robustness of an STL formula, BLuSTL [16] to encode STL robustness in MILP formulation, and Gurobi[9] for solving the MILP optimization problems. All presented computations were done on a 3.4GHz AMD Ryzen 9 5950X 16-core processor with 64GB memory. Moreover, for AutoKoopman we use random Fourier features as observables $g(x)$, reset the training set after $N_{\text{reset}} = 5$ trajectories, and apply grid-search for hyper-parameter optimization, where we use $q = 20$ as an upper bound for the number of observables. A detailed analysis that justifies our choice for these parameters is provided in Sec. 5.3. We use $N_{\max} = 5000$ as the limit on the number of iterations of Algorithm 1, although most systems are falsified in fewer than 10 simulations.

5.1 ARCH Competition Benchmarks

We compare our approach to other state-of-the-art falsification tools using the complete set of benchmarks and participating tools from the 2022 ARCH competition[10] [18]. The ARCH competition featured six systems: the *Automatic Transmission* (AT) system that we used for our running example, a *Neural-Network Controller* (NN) that aims to keep a magnet which hovers in an electromagnetic field at a given reference position, a *Chasing Cars* (CC) benchmark that considers a platoon of 5 vehicles, an *Aircraft Ground Collision Avoidance System* (F16) that examines the auto-pilot of an F16 fighter jet, a *Fuel Control of an Automotive Powertrain* (AFC) model, and a *Steam Condenser with Recurrent Neural Network Controller* (SC). The parameters for the systems are summarized in Table 1 with further details available in the competition report [18, Sec. 2.2]. Each system can have multiple temporal logic specifications with different constraints [18, Tab. 1].

The ARCH competition features two problem instances. For the first instance, arbitrary input signals $w(t)$ are allowed and participants can freely choose the number of control points and interpolation method that works best for their tool. The parameters we used for our approach are documented in Table 1. For the second instance type, the input signals parameters are given on a per model basis. The chasing cars model, for example, requires that the input is a piecewise constant signal with 20 segments. We generally use the number of control points to determine the time step size Δt. Then, we use the same step size for the time step in Koopman operator linearization, reachability analysis and the encoding of the STL formula in the MILP. The only exception is the AFC benchmark, where a time step size of $\Delta t = 5$ was found to be too coarse for effective analysis, so we used $\Delta t = 1$ instead.

Table 1: Properties for the models from the ARCH competition, consisting of the number of inputs m, number of outputs s, number of uncertain initial states n, and time horizon T. Moreover, we report the time step size Δt that defines the number of control points as well as the interpolation method that we used to construct $w(t)$ for the problem instance with arbitrary input signals.

Model	m	s	n	T	Δt	Interpolation
AT	2	3	-	30	1	piecewise constant
NN	1	1	-	40	3.33	piecewise constant
CC	2	5	-	100	10	polynomial
F16	-	16	3	15	0.1	piecewise constant
AFC	2	1	-	50	1	piecewise constant
SC	1	1	-	15	0.1	polyonmial

Since many falsification tools include some randomness and are therefore non-deterministic, the results reported in the ARCH competition are averaged over 10 falsification attempts. The main metrics used to evaluate the performance of the tools are the **falsification rate** FR, which specifies for how many of the 10 attempts the tool succeeded in falsifying the specification, and the **average and median number of simulations**, $\overline{S}$ and $\widetilde{S}$, of the real system that a tool needs to falsify the specification.

To ensure a meaningful comparison, we independently validate the results reported by other tools. We use the input signals reported by each tool [11] to run simulations of the models and then use Breach to compute the corresponding robustness. In our analysis, we discovered that some of the trajectories submitted for the ARCH competition violate input constraints or do not strictly falsify the specification ($Q \not< 0$). For this reason, we specify for the falsification rate both the number of trajectories we could successfully validate as well as the number reported in the ARCH competition. Details of the issues we discovered are provided in Appendix A.

The results for instance 1 with arbitrary input signals and instance 2 with constrained input signals are shown in Tables 2 and 3, respectively. We omitted the results for the tool FalCAuN from the tables for space reasons, since this tool is outperformed on all benchmarks. As visible in Table 2, for the problem instance with arbitrary inputs our approach achieves the lowest number of average simulations for 16 of the 19 benchmarks. For benchmark SC, no other tool reports a falsifying trace, and in CC4 and $\text{NN}_{\beta=0.04}$, only our algorithm consistently falsifies the model across all 10 runs. For several specifications of the automatic transmission benchmarks (AT52,AT53,AT54), a single simulation proves sufficient to identify a falsifying trace, which implies that the random simulation generated to learn the model already falsifies the requirement. The same behavior can be observed for the tools ARIsTEO and falsify. For benchmarks AT6a, AT6b, AT6c, and AT6abc, our approach demonstrates a significant performance improvement over the state of the art. In fact, for benchmark AT6b we require on average 150 fewer simulations than the best-performing tool, which corresponds to 2343% improvement in performance.

Another aspect of the results that is not apparent from the tables is that other tools often have many more hyper-parameters that have

[8] https://github.com/Abdu-Hekal/KoopmanFalsification

[9] https://www.gurobi.com/

[10] Note that the report for the 2023 edition of ARCHCOMP came out a few weeks before the submission deadline for this work.

[11] https://gitlab.com/goranf/ARCH-COMP/-/tree/master/2022/FALS

Table 2: Comparison of the performance of different falsification tools on the benchmarks from the ARCH competition (instance 1), where the results are averaged over 10 executions. The evaluation metrics are the mean $\overline{S}$ and median $\widetilde{S}$ number of simulations, simulation time percent $R = \frac{\text{Simulation Time}}{\text{Total Time}} * 100(\%)$, as well as the falsification rate FR reported in the format $^V/_A$, where V is number of validated falsification results and A is the number of falsifications reported in the ARCH competition.

Bench.	Our Approach				Uniform			ARIsTEO			falsify			FalStar			ForeSee			Ψ−TaLiRo			
	FR	$\overline{S}$	$\widetilde{S}$	R	FR	$\overline{S}$	$\widetilde{S}$	FR	$\overline{S}$	$\widetilde{S}$	FR	$\overline{S}$	$\widetilde{S}$	FR	$\overline{S}$	$\widetilde{S}$	FR	$\overline{S}$	$\widetilde{S}$	FR	$\overline{S}$	$\widetilde{S}$	R
AT1	$^{10}/_-$	8.8	8.0	55.8	-	-	-		-	-	-	-	-	$^-/_{10}$	26.2	18.5	$^{10}/_{10}$	338	348	$^{10}/_{10}$	106	106	72.7
AT2	$^{10}/_-$	2.1	2.0	60.6	$^9/_{10}$	7.6	5.0	$^{10}/_{10}$	5.4	2	-	-	-	$^-/_{10}$	3.5	2	$^{10}/_{10}$	108	33.5	$^{10}/_{10}$	15.5	12.5	57.5
AT51	$^{10}/_-$	7.9	7.0	31.8	$^1/_1$	923	923	-	-	-	$^0/_{10}$	4.5	3.0	$^-/_{10}$	65.2	22	$^{10}/_{10}$	19.3	10.5	$^1/_1$	22.0	22	92.0
AT52	$^{10}/_-$	1.1	1.0	64.1	$^{10}/_{10}$	4.1	2.0	$^{10}/_{10}$	1.7	1.5	$^0/_{10}$	1.1	1.0	$^0/_{10}$	253	241	$^{10}/_{10}$	65.1	36.5	$^{10}/_{10}$	3.2	2.5	61.7
AT53	$^{10}/_-$	1.0	1.0	81.1	$^{10}/_{10}$	18.6	15.0	$^{10}/_{10}$	12.8	8.5	$^0/_{10}$	1.2	1.0	$^-/_{10}$	99.6	81.5	$^{10}/_{10}$	4.0	2.0	$^{10}/_{10}$	28.0	21.0	60.2
AT54	$^{10}/_-$	1.0	1.0	81.2	$^4/_4$	1121	1056	-	-	-	$^0/_{10}$	1.4	1.0	$^-/_{10}$	86.3	52.5	$^{10}/_-$	24.8	23.5	$^7/_7$	715	554	90.6
AT6a	$^{10}/_-$	9.1	10.0	45.4	$^{10}/_{10}$	74.4	41.5	$^{10}/_{10}$	93.4	70.0	-	-	-	$^-/_{10}$	74.4	79.0	$^{10}/_{10}$	128	136	$^{10}/_{10}$	76.6	89.0	58.1
AT6b	$^{10}/_-$	5.0	5.0	46.1	$^{10}/_{10}$	251	189	$^4/_4$	112	127	$^2/_3$	166	186	$^-/_{10}$	92.0	81.0	$^{10}/_{10}$	228	211	$^7/_7$	426	224	58.2
AT6c	$^{10}/_-$	6.7	6.0	44.2	$^{10}/_{10}$	185	86.0	$^9/_9$	124	112	-	-	-	$^3/_{10}$	620	953	$^{10}/_{10}$	157	128	$^9/_9$	328	112	58.3
AT6abc	$^{10}/_-$	5.9	5.0	36.5	$^{10}/_{10}$	58.8	33.5	$^{10}/_{10}$	73.4	30.5	-	-	-	$^-/_{10}$	41.0	602	-	-	-	$^{10}/_{10}$	34.2	29.5	37.0
NN	$^{10}/_-$	1.9	2.0	43.5	$^{10}/_{10}$	38.6	27.5	$^0/_1$	299	299	$^7/_9$	19.7	13.0	$^{10}/_{10}$	186	126	$^7/_{10}$	51.3	46.0	$^{10}/_{10}$	36.4	35.5	84.6
$NN_{\beta}=0.04$	$^{10}/_-$	53.9	30.5	27.5	-	-	-	-	-	-	$^0/_5$	72.4	74.0	-	-	-	-	-	-	-	-	-	-
NNx	-	-	-	-	-	-	-	-	-	-	-	-	-	$^0/_5$	787.0	813.0	$^0/_{10}$	1.1	1.0	-	-	-	-
CC1	$^{10}/_-$	6.8	6.0	43.4	$^{10}/_{10}$	10.4	9.5	$^-/_{10}$	24.8	17.5	$^{10}/_{10}$	42.0	28.5	$^-/_{10}$	2.9	1.5	-	-	-	$^{10}/_{10}$	13.1	8.5	71.5
CC2	$^{10}/_-$	3.5	2.5	48.0	$^{10}/_{10}$	15.4	15.0	$^{10}/_{10}$	14.6	9.0	$^4/_4$	43.3	10.0	$^-/_{10}$	3.8	1.5	-	-	-	$^{10}/_{10}$	16.4	11.0	65.5
CC3	$^{10}/_-$	7.1	4.5	31.3	$^{10}/_{10}$	77.9	54.5	$^{10}/_{10}$	59.5	35.0	$^{10}/_{10}$	35.0	33.0	$^-/_{10}$	6.2	4.5	-	-	-	$^{10}/_{10}$	21.5	15.0	71.8
CC4	$^{10}/_-$	198	139	39.5	-	-	-	-	-	-	-	-	-	$^2/_2$	1257	1257	-	-	-	$^1/_1$	1253	1253	93.6
CC5	$^{10}/_-$	27.7	21.5	28.8	$^{10}/_{10}$	28.5	14.5	$^{10}/_{10}$	18.8	18.0	$^2/_3$	5.7	5.0	$^8/_{10}$	60.5	32.0	-	-	-	$^{10}/_{10}$	47.3	39.0	84.8
CCx	$^{10}/_-$	110	63.5	35.3	$^9/_9$	668	469	$^3/_3$	134	74.0	-	-	-	$^9/_9$	962	1000	-	-	-	$^{10}/_{10}$	211	70.0	20.8
F16	-	-	-		-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
SC	$^{10}/_-$	57.8	38.5	6.2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	

Table 3: Comparison of the performance of different falsification tools on the benchmarks from the ARCH competition (instance 2), where the results are averaged over 10 executions. The evaluation metrics are the mean $\overline{S}$ and median $\widetilde{S}$ number of simulations, simulation time percent $R = \frac{\text{Simulation Time}}{\text{Total Time}} * 100(\%)$, as well as the falsification rate FR reported in the format $^V/_A$, where V is number of validated falsification results and A is the number of falsifications reported in the ARCH competition.

Bench.	Our Approach				Uniform			ARIsTEO			falsify			FalStar			ForeSee			Ψ−TaLiRo			
	FR	$\overline{S}$	$\widetilde{S}$	R	FR	$\overline{S}$	$\widetilde{S}$	FR	$\overline{S}$	$\widetilde{S}$	FR	$\overline{S}$	$\widetilde{S}$	FR	$\overline{S}$	$\widetilde{S}$	FR	$\overline{S}$	$\widetilde{S}$	FR	$\overline{S}$	$\widetilde{S}$	R
AT1	$^{10}/_-$	4.9	4.5	57.0	-	-	-		-	-	-	-	-	-	-	-	-	-	-	$^{10}/_{10}$	105	105	86.8
AT2	$^{10}/_-$	2.5	2.5	63.0	$^{10}/_{10}$	18.8	13.5	$^{10}/_{10}$	12.9	8.0	-	-	-	-	-	-	-	-	-	$^{10}/_{10}$	11.6	11.0	50.3
AT51	$^{10}/_-$	24.1	27.5	48.7	$^9/_{10}$	20.5	16.5	$^{10}/_{10}$	19.0	10.0	$^2/_{10}$	15.5	15.5	-	-	-	-	-	-	$^{10}/_{10}$	13.7	8.5	69.3
AT52	$^{10}/_-$	6.0	4.0	46.2	$^{10}/_{10}$	74.1	65.0	$^{10}/_{10}$	74.7	46.0	$^2/_{10}$	4.7	5.0	-	-	-	-	-	-	$^{10}/_{10}$	79.1	95.0	92.9
AT53	$^{10}/_-$	1.8	1.0	57.3	$^{10}/_{10}$	1.5	1.0	$^{10}/_{10}$	1.4	1.0	$^3/_{10}$	2.4	1.5	-	-	-	-	-	-	$^{10}/_{10}$	2.7	2.0	57.1
AT54	$^{10}/_-$	85.0	62.0	46.0	$^{10}/_{10}$	47.9	42.0	$^{10}/_{10}$	44.0	40.0	$^6/_{10}$	33.4	29.0	-	-	-	-	-	-	$^{10}/_{10}$	37.7	32.0	79.6
AT6a	$^{10}/_-$	27.6	15.5	48.3	$^{10}/_{10}$	157	138	$^7/_7$	65.1	62.0	-	-	-	-	-	-	-	-	-	$^9/_9$	255	226	61.7
AT6b	$^{10}/_-$	14.1	7.0	48.5	$^{10}/_{10}$	472	588	$^5/_5$	91.2	74.0	$^5/_9$	107	76.0	-	-	-	-	-	-	$^9/_9$	580	522	49.4
AT6c	$^{10}/_-$	10.5	10.0	49.4	$^{10}/_{10}$	327	176	$^7/_7$	175	141	-	-	-	-	-	-	-	-	-	$^8/_8$	362	168	54.7
AT6abc	$^{10}/_-$	15.8	12.5	39.0	$^{10}/_{10}$	149	126	$^9/_9$	84.1	81.0	-	-	-	-	-	-	-	-	-	$^{10}/_{10}$	241	74.0	7.8
AFC27	$^{10}/_-$	9.3	5.0	30.8	-	-	-	$^8/_9$	21.9	25.0	$^4/_{10}$	1.9	1.0	-	-	-	-	-	-	$^0/_{10}$	113	110	97.8
AFC29	$^{10}/_-$	2.2	2.0	72.3	$^4/_{10}$	25.1	19.0	$^{10}/_{10}$	3.4	3.5	$^5/_{10}$	1.0	1.0	-	-	-	-	-	-	$^9/_{10}$	19.6	19.0	100
AFC33	-	-	-	-	-	-	-	-	-	-	$^0/_{10}$	1.0	1.0	-	-	-	-	-	-	-	-	-	-
NN	$^{10}/_-$	5.1	2.0	38.0	$^{10}/_{10}$	277	159	$^1/_{10}$	117	82.5	$^{13}/_{16}$	44.4	1.0	-	-	-	$^5/_{10}$	74.9	88.5	$^{10}/_{10}$	156	100	88.8
$NN_{\beta}=0.04$	$^{10}/_-$	446	477	32.5	-	-	-	-	-	-	$^{10}/_{10}$	1.0	1.0	-	-	-	-	-	-	-	-	-	-
NNx	$^{10}/_-$	155	131	17.6	$^{10}/_{10}$	713	488	-	-	-	-	-	-	-	-	-	$^0/_{10}$	1.0	1.0	$^{10}/_{10}$	46.8	48.0	37.2
CC1	$^{10}/_-$	3.7	3.0	49.5	$^{10}/_{10}$	16.4	9.5	$^{10}/_{10}$	9.1	8.0	$^{10}/_{10}$	19.9	16.5	-	-	-	$^{10}/_{10}$	27.6	28.5	$^{10}/_{10}$	10.8	8.0	73.9
CC2	$^{10}/_-$	3.9	3.5	43.3	$^{10}/_{10}$	12.4	13.0	$^{10}/_{10}$	10.8	9.0	$^9/_9$	8.4	7.0	-	-	-	$^1/_1$	148	148	$^{10}/_{10}$	9.6	7.0	68.3
CC3	$^{10}/_-$	4.7	4.0	26.1	$^{10}/_{10}$	19.6	21.0	$^{10}/_{10}$	12.8	13.5	$^{10}/_{10}$	8.2	5.0	-	-	-	$^9/_{10}$	16.4	10.0	$^{10}/_{10}$	11.7	8.0	70.5
CC4	$^{10}/_-$	888	480	27.7	-	-	-	-	-	-	$^0/_1$	15.5	15.0	-	-	-	$^9/_{10}$	586	575	$^3/_3$	1608	1580	96.9
CC5	$^{10}/_-$	18.2	15.0	9.2	$^{10}/_{10}$	37.4	22.0	$^{10}/_{10}$	21.1	11.5	$^4/_4$	16.4	16.0	-	-	-	$^9/_{10}$	95.2	30.0	$^{10}/_{10}$	28.3	27.0	75.3
CCx	$^7/_-$	3024	3492	29.9	$^7/_7$	614	291	$^3/_3$	97.0	103	-	-	-	-	-	-	-	-	-	$^{10}/_{10}$	241	74.0	35.4
SC	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	

been tuned by experts—the tool authors—such as a varying number of control points for different specifications *of the same model*. Our approach uses a consistent number of control points for all falsification problems of each model.

Our approach is outperformed on three benchmarks (CC1, CC3, CC5), which all belong to the chasing cars model. This is a difficult, highly nonlinear model, with a coarse time step size of $\Delta t = 10$. Despite these challenges, our approach still consistently finds a falsifying trace for each run for all three benchmarks and outperforms the majority of tools. In particular, for benchmark CC1, only FalStar records a lower average number of simulations (2.9 compared to 5.5 for our approach). Similarly, for CC3, FalStar requires 6.2 simulations on average, while our approach requires 7.1 simulations. For CC5, only ARIsTEO requires fewer simulations on average to find a falsifying trace. There exist two benchmarks, NNx and F16, where no tool (including our work) finds a valid falsifying trace. Note that FalStar and ForeSee report falsifying instances for NNx in the ARCH competition [18], however we discovered that the reported instances for each tool violate the input requirements for the benchmark (see Appendix A for more details).

For instance 2 with constrained input signals, we outperform state-of-the-art tools for 15 out of 21 benchmarks as shown in Table 3. The slight decrease in performance of our approach we believe is caused by the additional constraints. The restrictions on the inputs narrow the solution space, which leaves less room for error in the learned surrogate model. At the same time, the restricted input format results in a smaller search space, which helps numerical optimization falsification approaches such as Ψ−TaLiRo.

To select the next sample, our approach involves learning a surrogate model (with automated hyper-parameter tuning), performing reachability analysis and then solving an MILP. While this helps our approach make better decisions for complex specifications, a legitimate concern could be that this process may take too long and become a bottleneck. We analyze this by measuring the percentage of overall computation time spent running simulations, $\mathrm{R} = \frac{\text{Simulation Time}}{\text{Total Time}} * 100(\%)$. While the computation time for our approach is apparent in the tables—on average 44.8% of the computation time is spent running simulations—it does not preclude our approach from being applied. In some cases, such as the AT2 benchmark, we even record both a better percent simulation time and a lower number of expected simulations. Further, our implementation is a prototype, with likely room for optimization.

5.2 Aircraft Model

The ARCH competition introduces a diverse set of challenging benchmarks with various behaviors and requirements. However, with the exception of the F16 model, all benchmarks only consider falsification via time-varying input signals and do not contain uncertain initial conditions $x_0 \in \mathcal{X}_0$. Therefore, to evaluate the performance of our approach for falsification tasks with both uncertain input signals and uncertain initial states, we now consider an additional aircraft model [37, Example 3.2].

The state $x(t) = [v, \phi, a]^T$ of the aircraft, which is identical to the system output $y(t) = x(t)$, consists of the velocity v, the flight path angle ϕ, and the altitude a. The input $w(t) = [f, \theta]^T$ to the system consists of the thrust f and the angle of attack θ. The initial state x_0 is uncertain within the set $\mathcal{X}_0 = [200,260] \times [-10,10] \times [120,150]$, and the input signal $w(t)$ is uncertain within the set $\mathcal{W} = [34386,53973] \times [0,16]$. For falsification, the time horizon is $T = 4s$. The benchmark considers the following two STL specifications:

$$\varphi_1 = \Diamond_{[1,1.5]} v \notin [250,260] \vee \Box_{[3,4]} v \notin [230,240]$$
$$\varphi_2 = \Box(a > 0)$$

Table 4: Falsification results for the aircraft model averaged over 10 executions, where the evaluation metrics are the falsification rate FR, the mean $\overline{S}$ and median $\widetilde{S}$ number of simulations, and the computation time t_{comp} in seconds.

Bench.	Our Approach				Uniform Random				Simulated Annealing			
	FR	$\overline{S}$	$\widetilde{S}$	t_{comp}	FR	$\overline{S}$	$\widetilde{S}$	t_{comp}	FR	$\overline{S}$	$\widetilde{S}$	t_{comp}
φ_1	10	26.7	12.5	24.7	8	1866	2036	83.7	10	859	630	39.9
φ_2	10	2.0	2.0	0.54	10	19.0	13.5	0.04	10	64.0	25.5	0.15

We compare our approach to S-TaLiRo [6], which, unlike many other tools, supports the falsification of systems with uncertainties in the initial set. S-TaLiRo employs optimization-based falsification and supports several different algorithms. For our comparison, we consider the two algorithms *uniform random exploration* and *simulated annealing*. Uniform random exploration serves as an effective metric for assessing the complexity of a benchmark since it requires few simulations for properties that are easier to falsify. On the other hand, simulated annealing represents a more sophisticated falsification approach and is the default optimization algorithm for S-TaLiRo. For a fair comparison, we consider piecewise constant input signals with 10 segments for our approach as well as for S-TaLiRo, which corresponds to a time step size of $\Delta t = 0.4s$.

As with the ARCH competition, we report the results averaged over 10 falsification attempts, which are summarized in Table 4. For the relatively complex STL specification φ_1, our approach significantly outperforms both falsification algorithms from S-TaLiRo in all evaluation metrics. For the simpler STL specification φ_2, our approach on average requires the fewest number of simulations.

Despite having fewer simulations, our method's total computation time for φ_2 is larger two algorithms in S-TaLiRo, due to the extra computation needed to select the next sample. In this case, however, the falsification time is still small—0.54 seconds for our approach.

5.3 Influence of Algorithm Parameters

We conclude our numerical evaluation by analyzing the influence certain hyper-parameters have on the performance of our falsification algorithm, for which we consider the benchmarks from the ARCH competition described in Sec. 5.1.

First, we examine the influence of the number of observables q for the Koopman model on the AT1 and CC1 benchmarks. In theory, a larger number of observables should result in a more accurate Koopman model, and therefore improve the performance of the falsification algorithm. In practice, however, we see on the left side of Fig. 5 that a larger number of observables only slightly reduces the average number of simulations required to falsify the system. At the same time, as shown on the right side of Fig. 5, a larger number of observables decreases the percent time spent on simulations R, which corresponds to an increase in the overall computation time. In summary, this means that a rather low number of observables is sufficient for our falsification algorithm, and justifies the choice of $q = 20$ as an upper bound for the search range in AutoKoopman.

As we explained in Sec. 4.3, one of the main motivations for using reachability analysis rather than directly encoding the dynamics of the Koopman model into the optimization problem is the reduced

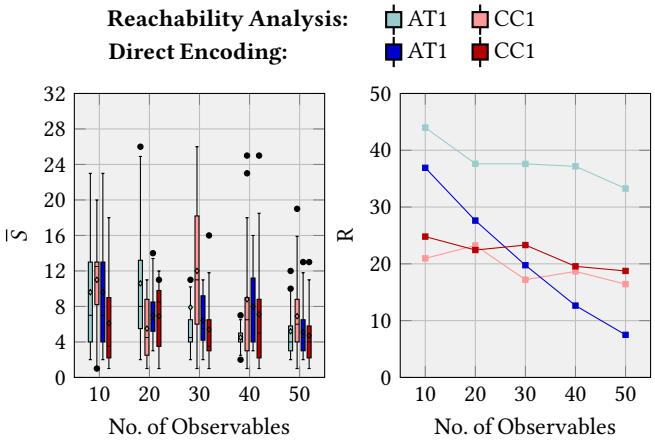

Figure 5: Mean number of simulations $\overline{S}$ (left) and simulation time ratio R (right) for the AT1 and CC1 benchmarks from the ARCH competition averaged over 10 executions, where we compare using reachability analysis to directly encoding the dynamics into the optimization problem. The black dots in the left figure represent the outliers from all executions.

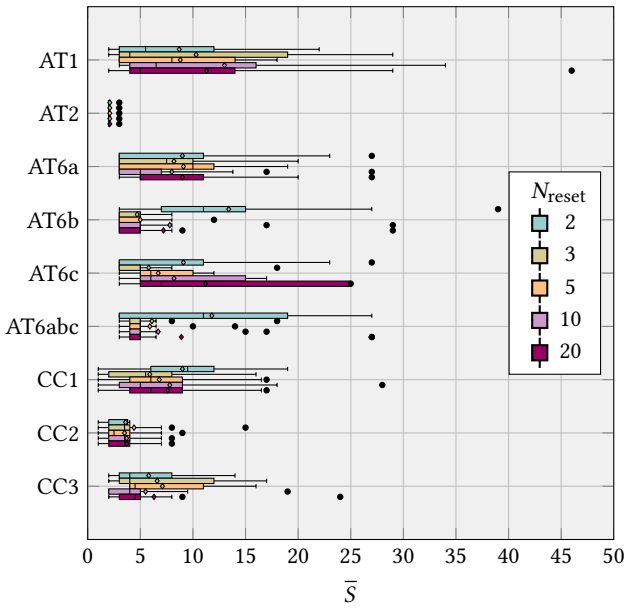

Figure 6: Average number of simulations $\overline{S}$ required to falsify the system for different values of training set resets N_{reset} for Koopman model learning evaluated on different benchmarks from the ARCH competition.

computation time. Experimental evidence for this claim is provided on the right side of Fig. 5, where using reachability analysis leads to a larger percent of simulation time R compared to direct encoding, which corresponds to a smaller total computation time. While there is a huge difference in computation time for the AT1 benchmark, especially when the number of observables is large, the difference for the CC1 benchmark is much smaller. This can be attributed to the small number of control points used for the CC1 benchmark, which results in a small amount of variables for the MILP optimization problem, even for a large number of observables. Indeed, for the CC1 benchmark with such settings, the computational cost of reachability analysis can slightly surpass the cost of direct encoding. Nevertheless, this computational gap lessens for a larger number of observables.

Finally, we examine the effect of the hyper-parameter N_{reset}, which specifies after how many trajectories the training set used to learn the Koopman model is reset to the empty set. The effect of N_{reset} on the average number of simulations is shown in Fig. 6 for several different benchmarks. We can observe that resetting the training set after $N_{\text{reset}} = 2$ trajectories leads to a higher number of simulations needed for falsification. This suggests that the data in this case is insufficient to correctly learn the dynamics of the black-box model. On the other extreme, a value of $N_{\text{reset}} = 20$ shows better overall performance, but suffers from extreme outliers. For example, in one falsification run for the AT1 benchmark, as many as 47 simulations were required to falsify the system, which is much higher than the average number. These outcomes suggest that the best value for N_{reset} is located somewhere between the two extremes, which justifies us to use $N_{\text{reset}} = 5$ as the default in our approach.

We also investigated the change in the average number of simulations and percentage of simulation time R for the ARCHCOMP benchmarks, for different choices of time step size Δt and different algorithm enhancements. This extra analysis presented in Appendix B.

6 Conclusion

In this paper we developed a new approach for the falsification of black-box cyber-physical systems. While most existing falsification algorithms apply numerical optimization to find a falsifying trace, our approach instead uses machine learning via Koopman operator linearization to create a symbolic model approximation. We then apply formal verification approaches on the inferred model—reachability analysis and MILP specification encoding—in order to find the start point and input signal that minimizes robustness. Since the formal reasoning is done on a model approximation, this does not always falsify the original black-box system. We presented strategies to improve the local accuracy of the learned model, as well as a specification offset approach to compensate for relevant model error. The developed method is highly effective. We outperform all participating tools on 16 out of 19 benchmarks from the ARCH falsification competition.

Our method incorporates earlier work from both falsification and verification methods. From falsification, we leverage quantitative semantics of STL and propose strategies to deal with the scale problem. From verification methods, we use polynomial zonotopes to represent non-convex sets arising from the nonlinear Koopman observable functions and perform dependency-preserving reachability analysis. While falsification and verification have traditionally been disperse areas, the presented approach would not have been possible without building upon research results from both.

Acknowledgments

This material is based upon work supported by the Air Force Office of Scientific Research and the Office of Naval Research under award numbers FA9550-19-1-0288, FA9550-21-1-0121, FA9550-23-1-0066 and N00014-22-1-2156, and the National Science Foundation under Award No. 2237229. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Air Force or the United States Navy.

References

[1] H. Abbas and G. Fainekos. 2012. Convergence Proofs for Simulated Annealing Falsification of Safety Properties. In *Proc. of the Annual Allerton Conference on Communication, Control, and Computing*. 1594–1601.

[2] H. Abbas, A. Winn, G. Fainekos, and A. A. Julius. 2014. Functional Gradient Descent Method for Metric Temporal Logic Specifications. In *Proc. of the American Control Conference*. 2312–2317.

[3] A. Aerts, B. T. Minh, M. R. Mousavi, and M. A. Reniers. 2018. Temporal Logic Falsification of Cyber-Physical Systems: An Input-Signal-Space Optimization Approach. In *Proc. of the International Conference on Software Testing, Verification and Validation Workshops*. 214–223.

[4] M. Althoff. 2015. An Introduction to CORA 2015. In *Proc. of the International Workshop on Applied Verification for Continuous and Hybrid Systems*. 120–151.

[5] M. Althoff, G. Frehse, and A. Girard. 2021. Set Propagation Techniques for Reachability Analysis. *Annual Review of Control, Robotics, and Autonomous Systems* 4 (2021), 369–395.

[6] Y. Annapureddy, C. Liu, G. Fainekos, and S. Sankaranarayanan. 2011. S-TaLiRo: A Tool for Temporal Logic Falsification for Hybrid Systems. In *Proc. of the International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. 254–257.

[7] Y. S. R. Annapureddy and G. Fainekos. 2010. Ant Colonies for Temporal Logic Falsification of Hybrid Systems. In *Proc. of the Annual Conference on IEEE Industrial Electronics Society*. 91–96.

[8] S. Bak and P. S. Duggirala. 2017. Simulation-Equivalent Reachability of Large Linear Systems with Inputs. In *Proc. of the International Conference on Computer Aided Verification*. 401–420.

[9] S. Bak and et al. 2022. Reachability of Koopman Linearized Systems Using Random Fourier Feature Observables and Polynomial Zonotope Refinement. In *Proc. of the International Conference on Computer Aided Verification*. 490–510.

[10] S. Bogomolov and et al. 2019. Falsification of Hybrid Systems Using Symbolic Reachability and Trajectory Splicing. In *Proc. of the International Conference on Hybrid Systems: Computation and Control*. Article No. 1.

[11] Xin Chen and Sriram Sankaranarayanan. 2022. Reachability Analysis for Cyber-Physical Systems: Are We There Yet?. In *NASA Formal Methods Symposium*. Springer, 109–130.

[12] A. M. DeGennaro and N. M. Urban. 2019. Scalable Extended Dynamic Mode Decomposition Using Random Kernel Approximation. *SIAM Journal on Scientific Computing* 41, 3 (2019), 1482–1499.

[13] J. Deshmukh and et al. 2017. Testing Cyber-Physical Systems Through Bayesian Optimization. *ACM Transactions on Embedded Computing Systems* 16, 5s (2017). Article No. 170.

[14] J. Deshmukh, X. Jin, J. Kapinski, and O. Maler. 2015. Stochastic Local Search for Falsification of Hybrid Systems. In *Proc. of International Symposium on Automated Technology for Verification and Analysis*. 500–517.

[15] A. Donzé. 2010. Breach, A Toolbox for Verification and Parameter Synthesis of Hybrid Systems. In *Proc. of the International Conference on Computer Aided Verification*. 167–170.

[16] A. Donzé, V. Raman, G. Frehse, and M. Althoff. 2015. BluSTL: Controller Synthesis from Signal Temporal Logic Specifications. *Proc. of the International Workshop on Applied Verification for Continuous and Hybrid Systems* (2015), 160–168.

[17] J. L. Eddeland and et al. 2020. Enhancing Temporal Logic Falsification with Specification Transformation and Valued Booleans. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39, 12 (2020), 5247–5260.

[18] G. Ernst and et al. 2022. ARCH-COMP 2022 Category Report: Falsification with Unbounded Resources. In *Proc. of the International Workshop on Applied Verification for Continuous and Hybrid Systems*. 204–221.

[19] G. Ernst, S. Sedwards, Z. Zhang, and I. Hasuo. 2019. Fast Falsification of Hybrid Systems Using Probabilistically Adaptive Input. In *Proc. of the International Conference on Quantitative Evaluation of Systems*. 165–181.

[20] G. Fainekos and G. Pappas. 2009. Robustness of Temporal Logic Specifications for Continuous-Time Signals. *Theoretical Computer Science* 410, 42 (2009), 4262–4291.

[21] Y. Han and et al. 2020. Deep Learning of Koopman Representation for Control. In *Proc. of the International Conference on Decision and Control*. 1890–1895.

[22] B. Hoxha, H. Abbas, and G. Fainekos. 2015. Benchmarks for Temporal Logic Requirements for Automotive Systems. In *Proc. of the International Workshop on Applied Verification for Continuous and Hybrid Systems*. 25–30.

[23] N. Kochdumper and M. Althoff. 2021. Sparse Polynomial Zonotopes: A Novel Set Representation for Reachability Analysis. *IEEE Trans. Automat. Control* 66, 9 (2021), 4043–4058.

[24] N. Kochdumper, B. Schürmann, and M. Althoff. 2020. Utilizing Dependencies to Obtain Subsets of Reachable Sets. In *Proc. of the International Conference on Hybrid Systems: Computation and Control*. Article No. 1.

[25] K. Komatsu and H. Takata. 2008. Nonlinear Feedback Control of Stabilization Problem via Formal Linearization Using Taylor Expansion. In *Proc. of the International Symposium on Information Theory and Its Applications*. 1–5.

[26] B. O. Koopman. 1931. Hamiltonian Systems and Transformation in Hilbert Space. *Proceedings of the National Academy of Sciences of the United States of America* 17, 5 (1931), 315–318.

[27] M. Korda and I. Mezić. 2018. Linear Predictors for Nonlinear Dynamical Systems: Koopman Operator meets Model Predictive Control. *Automatica* 93 (2018), 149–160.

[28] J Nathan Kutz, Steven L Brunton, Bingni W Brunton, and Joshua L Proctor. 2016. *Dynamic mode decomposition: data-driven modeling of complex systems*. SIAM.

[29] E. A. Lee. 2008. Cyber Physical Systems: Design Challenges. In *Proc. of the International Symposium on Object and Component-Oriented Real-Time Distributed Computing*. 363–369.

[30] H.-G. Lee, A. Arapostathis, and S. I. Marcus. 1987. Linearization of Discrete-Time Systems. *Internat. J. Control* 45, 5 (1987), 1803–1822.

[31] E. Lew and et al. 2023. AutoKoopman: A Toolbox for Automated System Identification via Koopman Operator Linearization. In *Proc. of the International Symposium on Automated Technology for Verification and Analysis*. 237–250.

[32] K. Makino and M. Berz. 2003. Taylor Models and Other Validated Functional Inclusion Methods. *International Journal of Pure and Applied Mathematics* 4, 4 (2003), 379–456.

[33] O. Maler and D. Nickovic. 2004. Monitoring Temporal Properties of Continuous Signals. In *Proc. of the International Conference on Formal Modelling and Analysis of Timed Systems*. 152–166.

[34] L. Mathesen, G. Pedrielli, and G. Fainekos. 2021. Efficient Optimization-Based Falsification of Cyber-Physical Systems with Multiple Conjunctive Requirements. In *Prof. of the International Conference on Automation Science and Engineering*. 732–737.

[35] L. Mathesen, S. Yaghoubi, G. Pedrielli, and G. Fainekos. 2019. Falsification of Cyber-Physical Systems with Robustness Uncertainty Quantification Through Stochastic Optimization with Adaptive Restart. In *Proc. of the International Conference on Automation Science and Engineering*. 991–997.

[36] C. Menghi, S. Nejati, L. Briand, and Y. I. Parache. 2020. Approximation-Refinement Testing of Compute-Intensive Cyber-Physical Models: An Approach Based on System Identification. In *Proc. of the International Conference on Software Engineering*. 372–384.

[37] T. Nghiem and et al. 2010. Monte-Carlo Techniques for Falsification of Temporal Properties of Non-Linear Hybrid Systems. In *Proc. of the International Conference on Hybrid Systems: Computation and Control*. 211–220.

[38] Z. Pan and F. Liu. 2023. Nonlinear Set-Membership State Estimation Based on the Koopman Operator. *International Journal of Robust and Nonlinear Control* 33, 4 (2023), 2703–2721.

[39] André Platzer. 2018. *Logical foundations of cyber-physical systems*. Vol. 662. Springer.

[40] R. Rajkumar, I. Lee, L. Sha, and J. Stankovic. 2010. Cyber-Physical Systems: The Next Computing Revolution. In *Proc. of the Design Automation Conference*. 731–736.

[41] V. Raman and et al. 2014. Model Predictive Control with Signal Temporal Logic Specifications. In *Proc. of the International Conference on Decision and Control*. 81–87.

[42] A. Rashid, U. Siddique, and S. Tahar. 2020. Formal Verification of Cyber-Physical Systems Using Theorem Proving. In *Proc. of the International Workshop on Formal Techniques for Safety-Critical Systems*. 3–18.

[43] A. Rauh and et al. 2009. Carleman Linearization for Control and for State and Disturbance Estimation of Nonlinear Dynamical Processes. In *Proc. of the International Conference on Methods and Models in Automation and Robotics*. 455–460.

[44] S. Sankaranarayanan and G. Fainekos. 2012. Falsification of Temporal Properties of Hybrid Systems Using the Cross-Entropy Method. In *Proc. of the International Conference on Hybrid Systems: Computation and Control*. 125–134.

[45] T. Söderström and P Stoica. 1989. *System Identification*.

[46] Q. Thibeault and et al. 2021. PSY-TaLiRo: A Python Toolbox for Search-Based Test Generation for Cyber-Physical Systems. In *Proc. of the International Conference on Formal Methods for Industrial Critical Systems*. 223–231.

[47] M. Waga. 2020. Falsification of Cyber-Physical Systems with Robustness-Guided Black-Box Checking. In *Proc. of the International Conference on Hybrid Systems: Computation and Control*. Article No. 11.

[48] M. O. Williams and et al. 2015. A Data–Driven Approximation of the Koopman Operator: Extending Dynamic Mode Decomposition. *Journal of Nonlinear Science* 25, 6 (2015), 1307–1346.

[49] Y. Yamagata and et al. 2020. Falsification of Cyber-Physical Systems Using Deep Reinforcement Learning. *IEEE Transactions on Software Engineering* 47, 12 (2020), 2823–2840.

[50] E. Yeung and et al. 2019. Learning Deep Neural Network Representations for Koopman Operators of Nonlinear Dynamical Systems. In *Proc. of the American Control Conference*. 4832–4839.

[51] Z. Zhang and et al. 2021. Effective Hybrid System Falsification Using Monte Carlo Tree Search Guided by QB-Robustness. In *Proc. of the International Conference on Computer Aided Verification*. 595–618.

[52] A. Zutshi, J. V. Deshmukh, S. Sankaranarayanan, and J. Kapinski. 2014. Multiple Shooting, CEGAR-Based Falsification for Hybrid Systems. In *Proc. of the International Conference on Embedded Software*. Article No. 5.

A ARCHCOMP Validation Results

In the course of performing our evaluation for this work, we validated the results reported in the falsification category of ARCH competition 2022 [18] and uncovered a few issues, presented in this section. There are two main concerns for validation:

(1) Are all constraints on inputs satisfied (i.e $u \in \mathcal{U}$) and does the input signal adhere to instance requirements?
(2) Are the falsification results strictly correct (i.e. all reported falsifications exhibit $Q < 0$) ?

Regarding issue (1), it is worth noting that the primary factor contributing to any discrepancies appears to be accidental human error. For instance, in the case of the automatic transmission benchmark, the tool *falsify* mistakenly sets the input range for the *brake* signal as [0,350]. As such, a significant portion of the reported falsifying inputs turns out to be invalid.

Regarding issue (2), there exist several sources of error—both computational and human. These errors include mistakes in translation of requirements, mismatch in robustness computation, numerical issues, or even manual copy and paste errors. In general, our independent robustness computation typically matches with the robustness evaluations carried out during the ARCH competition validation process. However, unlike our strict requirement $Q < 0$, the ARCH competition allows for tolerances for each benchmark where $Q < \delta$, in order to compensate for the possibility of small numerical errors.

We also remark that while we automate the validation process, we also make an effort to manually verify the results. This allows us to alleviate some human errors in reporting of the results. For instance, in the case of the Fuel Control benchmarks (denoted as AFC^*), S-TaLiRo provides results with the input signal order as $[\omega,\theta]$ instead of the correct $[\theta,\omega]$. This leads to a significant discrepancy between computed and ARCH competition results. We manually adjust for this error and report the validation results with the corrected order.

We now list the issues found through our independent validation. We note that we use '-' where a validation result was not available due to the absence of data.

- Uniform random sampling and Ψ-TaLiRo report the results for the AFC benchmark with the input signal order $[\omega,\theta]^T$ instead of the correct order $[\theta,\omega]^T$. Consequently, the automated validation attempts by ARCH classify the results as invalid. We manually adjust for this error and report the validation results accordingly.
- FalCAuN and falsify report input signals that exceed the bounds for the AT benchmarks, using 350 as an upper limit instead of 325.
- For the problem instance with restricted input signals, the results from falsify for the NN and $\text{NN}_\beta = 0.04$ benchmark are inconsistent with the results reported in the ARCH paper [18]. We fix this error by reporting the correct results provided by the tool in our paper.
- FalStar violates the requirement for the NNx benchmark for the problem instance with arbitrary input signals: While the benchmark requires discontinuities in the input signal to be at least 3 time units apart, the input signal reported by FalStar contains discontinuities that are only one time unit apart.
- ForeSee violates input requirements for NNx as it does not adhere to the tightened input constraints $Ref \in [1.95,2.05]$ for this particular benchmark. The tool mistakenly uses the more relaxed constraints for other NN benchmarks, where $Ref \in [1,3]$.
- The input signal reported by Ψ-TaLiRo for the benchmark AFC29 violates the input constraints $\theta \in [0,61.1]$, because the maximum value is $\theta = 61.2$. Since this is only a small violation, a possible explanation could be an interpolation error.

B Influence of Algorithm Parameters

In this section, we report additional analysis beyond Sec. 5.3 on the influence of algorithm parameters on the performance of our falsification algorithm, where we consider the time step size and the influence of the different algorithm enhancement that we implemented.

B.1 Time Step Size

One important parameter for our approach is the time step size Δt, which defines how many control points are used for the parameterization of the input signal $w(t)$, and in addition also determines the time-discretization for the Koopman model and the MILP encoding. We therefore now discuss the influence of Δt based on the results of experiments with different time step sizes shown in Table 5. As it is often the case for algorithms that deal with dynamic systems, one would expect that choosing an adequate time step size represents a trade-off between computational complexity and accuracy. While this holds true in most cases, the results detailed in Table 5 suggest that a finer time step can, at times, have adverse effects. For the AT1 benchmark, for example, a time step of 0.5 results in a larger number of simulations than all coarser time steps. One explanation for this counter-intuitive result could be noise amplification and worsened generalization in Koopman learning when using a finer discretization.

B.2 Algorithm Enhancements

In this paper, we introduced several enhancements to our basic framework for falsification via Koopman operator linearization. In particular, these enhancements are using reachability analysis rather than directly encoding the dynamics of the Koopman model into the optimization problem, resetting the training set for the Koopman model after N_{inter} trajectories, excluding the first trajectory that is determined randomly from the training set, and the modification of the temporal logic formula using the offset strategy from Sec. 4.4. The results for different combinations of these enhancements in comparison with the baseline approach without any enhancement are shown in Table 6. The results demonstrate that each enhancement improves the average performance by increasing the falsification rate, reducing the number of simulations, or increasing the simulation time ratio, which corresponds to a decrease in overall computation time. In particular, introducing resets improves both the falsification rate and the simulation time ratio. Moreover, the offset strategy enables the successful falsification of benchmarks AT6a, AT6b, AT6c, and AT6abc, and additionally reduces the average number of simulations required to find a falsifying trace for benchmarks CC3, CC5 and CCx. Using reachable sets instead of direct encoding in general improves the simulation time ratio. The only exception are the Chasing Cars benchmarks, where the computational cost of reachability analysis is not justified due to the coarse time step and small amount of variables for the MILP optimization problem. Finally, the last enhancement, which is to remove the initial random trajectory from the training set for the Koopman model, reduces the number of required simulations, especially for the Chasing Cars and Steam Condenser benchmarks.

Table 5: Comparison of the performance of our falsification algorithm for different time step sizes Δt on benchmarks from the ARCH competition, where the results are averaged over 10 executions. The evaluation metrics are the mean $\overline{S}$ and median $\widetilde{S}$ number of simulations as well as the falsification rate FR and the simulation time ratio R $= \frac{SimulationTime}{TotalTime} * 100(\%)$. In addition to the maximum number of simulations $N_{\max} = 5000$ we use a timeout of 1000s.

Bench.	$\Delta t = 0.1$				$\Delta t = 0.5$				$\Delta t = 1$				$\Delta t = 2.5$				$\Delta t = 5$				$\Delta t = 10$			
	FR	$\overline{S}$	$\widetilde{S}$	R	FR	$\overline{S}$	$\widetilde{S}$	R	FR	$\overline{S}$	$\widetilde{S}$	R	FR	$\overline{S}$	$\widetilde{S}$	R	FR	$\overline{S}$	$\widetilde{S}$	R	FR	$\overline{S}$	$\widetilde{S}$	R
AT1	10	4.3	2.0	22.9	10	10.3	5.5	44.1	10	8.8	8.0	55.8	10	3.7	3.5	57.7	10	4.9	4.5	58.0	10	5.8	5.0	57.9
AT2	10	2.0	2.0	35.9	10	2.0	2.0	57.8	10	2.1	2.0	60.6	10	4.5	3.0	57.4	10	2.5	2.5	64.0	10	1.9	2.0	69.3
AT51	10	1.0	1.0	83.4	10	5.2	5.5	13.6	10	7.9	7.0	31.8	10	6.3	6.5	44.5	10	24.1	27.5	49.0	10	116	78.5	51.1
AT52	10	1.0	1.0	83.4	10	1.0	1.0	82.4	10	1.1	1.0	64.1	10	2.1	2.0	48.9	10	6.0	4.0	46.4	10	280	251	46.2
AT53	10	1.0	1.0	83.5	10	1.0	1.0	82.3	10	1.0	1.0	81.1	10	1.2	1.0	66.4	10	1.8	1.0	57.7	10	16.0	6.5	47.4
AT54	10	1.0	1.0	83.4	10	1.0	1.0	82.4	10	1.0	1.0	81.2	10	5.7	3.0	45.0	10	85.0	62.0	47.0	10	50.2	41.0	48.2
AT6a	10	3.0	3.0	20.3	10	5.2	4.0	41.3	10	9.1	10.0	45.4	10	9.9	5.0	49.4	10	27.6	15.5	49.7	10	37.4	17.0	51.5
AT6b	10	7.1	4.0	14.3	10	5.7	5.0	39.3	10	5.0	5.0	46.1	10	13.2	13.0	48.1	10	14.1	7.0	49.6	10	38.0	33.5	52.4
AT6c	10	28.1	10.5	10.5	10	5.4	5.0	37.5	10	6.7	6.0	44.2	10	5.8	5.0	49.6	10	10.5	10.0	50.3	10	29.0	27.5	51.2
AT6abc	10	12.1	7.5	3.5	10	4.6	3.5	25.5	10	5.9	5.0	36.5	10	7.7	6.0	39.3	10	15.8	12.5	39.6	10	24.6	21.0	41.0
CC1	10	3.3	3.5	1.9	10	2.8	2.0	21.9	10	2.3	2.0	38.4	10	2.9	2.0	44.1	10	3.4	3.0	48.8	10	6.8	6.0	43.4
CC2	-	-	-	-	10	3.2	2.0	0.6	10	3.0	3.0	2.7	10	4.2	2.5	22.3	10	4.0	3.5	38.1	10	3.5	2.5	48.0
CC3	-	-	-	-	9	4.6	4.0	0.3	10	3.2	2.0	0.8	10	3.9	4.0	1.5	10	4.0	2.5	9.3	10	7.1	4.5	31.3
CC4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	5	109	273	19.2	10	198	139	39.5
CC5	-	-	-	-	1	7.0	7.0	0.4	1	4.0	4.0	0.4	4	12.3	12.0	0.48	8	44.8	46.0	3.2	10	27.7	21.5	28.8
CCx	-	-	-	-	7	93.4	105	5.9	1	194	194	5.9	7	173	100	18.6	4	266	87.5	26.6	10	110	63.5	35.3
SC	10	57.8	38.5	6.2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Table 6: Comparison of the performance of our falsification algorithm for different enhancements on benchmarks from the ARCH competition, where the results are averaged over 10 executions. In particular, the enhancements we consider are resetting the training set for Koopman (Reset), offsetting the STL formula (Offset), using reachability analysis (Reach), and omitting the initial random trajectory from the training set for Koopman (Rand). The evaluation metrics are the mean $\overline{S}$ and median $\widetilde{S}$ number of simulations as well as the falsification rate FR and the simulation time ratio R $= \frac{SimulationTime}{TotalTime} * 100(\%)$. In addition to the maximum number of simulations $N_{\max} = 5000$ we use a timeout of 1000s.

Bench.	Baseline				Reset				Reset + Offset				Reset + Offset + Reach				Reset + Offset + Reach + Rand			
	FR	$\overline{S}$	$\widetilde{S}$	R	FR	$\overline{S}$	$\widetilde{S}$	R	FR	$\overline{S}$	$\widetilde{S}$	R	FR	$\overline{S}$	$\widetilde{S}$	R	FR	$\overline{S}$	$\widetilde{S}$	R
AT1	5	7.6	4.0	20.5	10	8.1	8.5	33.5	10	8.1	8.5	33.5	10	8.1	8.0	49.9	10	8.8	8.0	55.8
AT2	10	2.1	2.0	41.0	10	2.1	2.0	41.0	10	2.1	2.0	41.5	10	2.1	2.0	61.3	10	2.1	2.0	60.6
AT51	10	3.0	2.0	14.9	10	2.8	2.5	14.1	10	3.1	3.0	15.0	10	6.7	5.0	28.1	10	7.9	7.0	31.8
AT52	10	1.1	1.0	59.7	10	1.1	1.0	59.0	10	1.1	1.0	59.2	10	1.1	1.0	65.6	10	1.1	1.0	64.1
AT53	10	1.0	1.0	82.1	10	1.0	1.0	82.2	10	1.0	1.0	82.4	10	1.0	1.0	82.6	10	1.0	1.0	81.1
AT54	10	1.0	1.0	82.1	10	1.0	1.0	82.4	10	1.0	1.0	82.5	10	1.0	1.0	82.6	10	1.0	1.0	81.2
AT6a	-	-	-	-	-	-	-	-	10	21.3	12.5	20.5	10	15.6	16.0	45.1	10	9.1	10.0	45.4
AT6b	-	-	-	-	-	-	-	-	10	6.1	5.0	21.2	10	8.0	5.5	44.6	10	5.0	5.0	46.1
AT6c	-	-	-	-	-	-	-	-	10	8.7	9.0	19.7	10	6.7	3.5	44.7	10	6.7	6.0	44.2
AT6abc	-	-	-	-	-	-	-	-	10	14.3	15.0	10.6	10	4.0	3.0	36.2	10	5.9	5.0	36.5
NN	10	3.9	3.5	27.7	10	4.9	3.5	33.1	10	4.9	3.5	27.1	10	1.9	2.0	41.8	10	1.9	2.0	43.5
$NN_{\beta} = 0.04$	4	34.3	31.0	10.2	8	147	82.0	21.6	8	131	118	13.6	10	37.3	19.5	25.7	10	53.9	30.5	27.5
NNx	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
CC1	10	4.5	4.0	51.2	10	5.2	6.0	53.3	10	5.2	6.0	53.0	10	4.2	3.0	47.9	10	6.8	6.0	43.4
CC2	10	3.0	3.0	55.3	10	3.0	3.0	55.0	10	3.0	3.0	54.7	10	4.5	3.5	48.0	10	3.5	2.5	48.0
CC3	10	3.2	2.0	43.6	10	3.9	3.0	43.9	10	3.3	2.5	39.5	10	4.8	3.5	33.2	10	7.1	4.5	31.3
CC4	2	12.0	12.0	32.6	10	231	103	39.0	10	231	103	38.5	10	334	223	38.6	10	198	139	39.5
CC5	1	2.0	2.0	34.3	10	49.8	34.5	40.2	10	23.4	16.5	28.2	10	35.7	22.0	27.3	10	27.7	21.5	28.8
CCx	-	-	-	-	10	113	63.5	31.5	10	81.2	56.5	27.9	10	103	60.0	35.4	10	110	63.5	35.3
F16	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
SC	2	10.5	10.5	0.2	-	-	-	-	-	-	-	-	9	126	135	5.8	10	57.8	38.5	6.2