ECAI 2024
U. Endriss et al. (Eds.)

doi:10.3233/FAIA240638

stl2vec: Semantic and Interpretable Vector Representation of Temporal Logic

Gaia Saveri[a, b], **Laura Nenzi**[a], **Luca Bortolussi**[a] **and Jan Křetínský**[c, d]

[a]University of Trieste, Italy
[b]University of Pisa, Italy
[c]Masaryk University of Brno, Czech Republic
[d]Technical University of Munich, Germany

Abstract. Integrating symbolic knowledge and data-driven learning algorithms is a longstanding challenge in Artificial Intelligence. Despite the recognized importance of this task, a notable gap exists due to the discreteness of symbolic representations and the continuous nature of machine-learning computations. One of the desired bridges between these two worlds would be to define semantically grounded vector representation (feature embedding) of logic formulae, thus enabling to perform continuous learning and optimization in the semantic space of formulae. We tackle this goal for knowledge expressed in *Signal Temporal Logic (STL)* and devise a method to compute continuous embeddings of formulae with several desirable properties: the embedding (i) is finite-dimensional, (ii) faithfully reflects the semantics of the formulae, (iii) does not require any learning but instead is defined from basic principles, (iv) is *interpretable*. Another significant contribution lies in demonstrating the efficacy of the approach in two tasks: learning model checking, where we predict the probability of requirements being satisfied in stochastic processes; and integrating the embeddings into a neuro-symbolic framework, to constrain the output of a deep-learning generative model to comply to a given logical specification.

1 Introduction

The need for integrating Artificial Intelligence (AI) and symbolic (i.e. logical) knowledge has been claimed for a long time [27], with logic being closely related to the way in which humans represent knowledge and reasoning [19]. However, a remarkable gap burdens on the integration of Machine Learning (ML) algorithms and symbolic representations: the latter are discrete objects, while ML models typically work in continuous domains. In this context, Neuro-Symbolic AI (NeSy) is emerging as a paradigm for the principled integration of sub-symbolic connectionist systems and logic knowledge [7]. As an example, NeSy models might address the following: leveraging logic knowledge for aiding the ML system improve its performance and/or learn with less data, using background knowledge expressed in symbolic form to constraint the behaviour of the ML system [12]. *Temporal logic* is a formalism suitable and since [28] widely used for describing properties and requirements of time-series related task, in particular of *dynamical systems*. Here, we specifically consider stochastic processes, such as epidemiological models or cyber-physical systems, where *Signal Temporal Logic (STL)* [26] emerges as the de-facto standard language, being concise yet rich and expressive for stating specifications of systems evolving over time [5]. For example in STL one can state properties like "the temperature of the room will reach 25 degrees within the next 10 minutes and will stay above 22 degrees for the successive hour". In this area, one is typically interested in understanding or verifying which properties the system under analysis is compliant to (or more precisely, in the probability of observing behaviour satisfying the property). Such analysis is often tackled by formal methods, via algorithms belonging to the world of quantitative model checking [4]. **In this work, we address the challenge of incorporating knowledge in the form of temporal logic formulae inside data-driven learning algorithms.** The key step is to devise a *finite-dimensional* embedding (feature mapping) of logical formulae into *continuous space*, yielding their representation as vectors of real numbers. In this way, symbolic knowledge can be seamlessly integrated into distance-based or neural-based architectures, and eventually doors are opened towards gradient-based optimization techniques. To make these techniques truly effective, we additionally require that semantically similar formulae are mapped to nearby representations. We call such embeddings *semantic*, allowing the efficient continuous optimization to happen in the “semantic” feature space of formulae.

Our contribution consists in formulating a way for computing such *finite-dimensional continuous semantic* embeddings of formulae of STL that are *interpretable*, and proving their effectiveness in integrating logical knowledge and machine-learning algorithms. In detail, we make the following contributions:

(i) We construct *finite-dimensional* semantic embeddings of STL formulae starting from the kernel defined in [8]: kernel methods are indeed suitable in this context, since they efficiently allow to implicitly define a rich feature space, without the need of manually constructing it. Kernel PCA [33] then allows us to construct suitable finite-dimensional approximations;
(ii) We give an *interpretable description* of the geometry of such embeddings, up to a certain quantified extent, differently from state-of-art logical embedding methods. Notably, the embeddings are not learnt but defined from basic principles, and, as we show, the characterization is resilient w.r.t. the parameters of the embedding construction method, indicating the revealed structure is inherent to the logic. The extracted features foster human-understandability of the formulae representation, and thus also of the optimization;

(iii) We prove that the computed representations meaningfully capture the *semantic* similarity of formulae, by using our finite-dimensional logical embeddings for *learning model checking*, i.e. for predicting the probability of a given requirement being satisfied by a stochastic process, given a set of observed properties with their probabilities;
(iv) We demonstrate the efficacy of the representations in preserving the semantic information carried by the formulae by using them as semantic *conditioning inside a NeSy deep generative framework*. We show that this improves the deep-learning process and model, critically relying in the form of our embeddings.

2 Preliminaries

Kernel methods leverage a positive semi-definite kernel function k to map input datapoints, e.g. vectors in $\mathbb{R}^m$, to a feature space $\mathbb{R}^D$, usually of higher dimension, i.e. $D \gg m$ [29]. Let $\Phi : \mathbb{R}^m \to \mathbb{R}^D$ denote this feature map, a key characteristic of kernel functions is that Φ is not explicitly calculated, but instead it is implicitly defined by computing its inner product in $\mathbb{R}^D$, formally $k : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}$ such that $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \langle \Phi(\boldsymbol{x}_i), \Phi(\boldsymbol{x}_j) \rangle$. The kernel trick hence allows to perform learning tasks in a feature space of higher dimension without explicitly constructing it, enabling the encoding of nonlinear manifolds without knowing the explicit feature maps, with a computational cost independent of the amount of features but only on the number of training points.

Kernel Principal Component Analysis (PCA) is a nonlinear dimensionality reduction technique that involves performing PCA [18] in the manifold identified by a kernel function. Given a dataset with points described in $\mathbb{R}^D$ and an integer number $d \ll D$, PCA consists in finding the set of d orthogonal directions, called Principal Components (PC), preserving the highest amount of information (i.e. variance) of the original dataset, and projecting the datapoints along these vectors, reducing their dimension. In kernel PCA, such directions are provably the eigenvectors of the centered kernel matrix of the dataset, corresponding to its d highest eigenvalues.

Signal Temporal Logic (STL) is a linear-time temporal logic which expresses properties on trajectories over dense time intervals [26]. We define as trajectories the functions $\xi : I \to D$, where $I \subseteq \mathbb{R}_{\geq 0}$ is the time domain and $D \subseteq \mathbb{R}^k, k \in \mathbb{N}$ is the state space. The syntax of STL is given by:

$$\varphi := tt \mid \pi \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \mathbf{U}_{[a,b]} \varphi_2$$

where tt is the Boolean *true* constant; π is an *atomic predicate*, interpreted as a function of the form $f_\pi(\boldsymbol{x}) \geq 0$ over variables $\boldsymbol{x} \in \mathbb{R}^n$ (we refer to n as the number of variables of a STL formula, i.e. individual signals used as arguments of the atomic predicates); $\neg$ and $\wedge$ are the Boolean *negation* and *conjunction*, respectively (from which the *disjunction* $\vee$ follows by De Morgan's law); $\mathbf{U}_{[a,b]}$, with $a, b \in \mathbb{Q}, a < b$, is the *until* operator, from which the *eventually* $\mathbf{F}_{[a,b]}$ and the *always* $\mathbf{G}_{[a,b]}$ temporal operators can be deduced. We can intuitively interpret the temporal operators over $[a, b]$ as follows: a property is *eventually* satisfied if it is satisfied at some point inside the temporal interval; a property is *globally* satisfied if it is true continuously in $[a, b]$; the *until* operator captures the relationship between two conditions φ, ψ in which the first φ holds until, at some point in $[a, b]$, the second ψ becomes true. For example, the sentence "the temperature τ of the room will reach 25 degrees within the next 10 minutes and will stay above 22 degrees for the successive 60 minutes" translates in STL as $F_{[0,10]}(\tau \geq 25 \wedge G_{[0,60]}\tau \geq 22)$. We call $\mathcal{P}$ the set of well-formed STL formulae. STL is endowed with both a *qualitative* (or Boolean) semantics, giving the classical notion of satisfaction of a property over a trajectory, i.e. $s(\varphi, \xi, t) = 1$ if the trajectory ξ at time t satisfies the STL formula φ , and a *quantitative* semantics, denoted by $\rho(\varphi, \xi, t)$. The latter, also called *robustness*, is a measure of how robust is the satisfaction of φ w.r.t. perturbations of the signals. A recursive definition of robustness can be found in [26]. Robustness is compatible with satisfaction via the following *soundness* property: if $\rho(\varphi, \xi, t) > 0$ then $s(\varphi, \xi, t) = 1$ and if $\rho(\varphi, \xi, t) < 0$ then $s(\varphi, \xi, t) = 0$. When $\rho(\varphi, \xi, t) = 0$ arbitrary small perturbations of the signal might lead to changes in satisfaction value. For numerical stability reasons, we use a normalized robustness, rescaling the output signals using a sigmoid function, see Section A of the supplementary material [32]. When we evaluate properties at time $t = 0$, we omit t from the previous notations. A distribution $\mathcal{F}$ over STL formulae can be algorithmically defined by a syntax-tree random recursive growing scheme, that recursively generates the nodes of a formula given the probability p_{leaf} of each node being an atomic predicate, and a uniform distribution over the other operator nodes.

Stochastic Processes within this context are probability spaces defined as triplets $\mathcal{M} = (\mathcal{T}, \mathcal{A}, \mu)$ of a trajectory space $\mathcal{T}$ and a probability measure μ on a σ-algebra $\mathcal{A}$ over $\mathcal{T}$. Given a stochastic process $\mathcal{M}$, the *expected robustness* is a function $R_\mathcal{M} : \mathcal{P} \times I \to \mathbb{R}$ such that $R_\mathcal{M}(\varphi, t) = \mathbb{E}_\mathcal{M}[\rho(\varphi, \xi, t)] = \int_{\xi \in \mathcal{T}} \rho(\varphi, \xi, t) d\mu(\xi)$. Similarly, the *satisfaction probability* $S_\mathcal{M} : \mathcal{P} \times I \to \mathbb{R}$ is computed as $S_\mathcal{M}(\varphi, t) = \mathbb{E}_\mathcal{M}[s(\varphi, \xi, t)] = \int_{\xi \in \mathcal{T}} s(\varphi, \xi, t) d\mu(\xi)$. In probabilistic and statistical model checking, one is often interested in computing or estimating these quantities, see [4] for details. In this work we consider stochastic processes that can be simulated via the Gillespie Stochastic Simulation Algorithm (SSA) [11], which samples from the exact distribution μ over trajectories.

A kernel function for STL formulae is defined in [8] by leveraging the quantitative semantics of STL. Indeed, robustness allows formulae to be considered as functionals mapping trajectories into real numbers, i.e. $\rho(\varphi, \cdot) : \mathcal{T} \to \mathbb{R}$ such that $\xi \mapsto \rho(\varphi, \xi)$. Considering these as feature maps, and fixing a probability measure μ_0 on $\mathcal{T}$, a kernel function capturing similarity among STL formulae on mentioned feature representations can be defined as:

$$k(\varphi, \psi) = \langle \rho(\varphi, \cdot), \rho(\psi, \cdot) \rangle = \int_{\xi \in \mathcal{T}} \rho(\varphi, \xi) \rho(\psi, \xi) d\mu_0(\xi) \quad (1)$$

opening the doors to the use of the scalar product in the Hilbert space L^2 as a kernel for $\mathcal{P}$; intuitively this results in a kernel having high positive value for formulae that behave similarly on high-probability trajectories (w.r.t. μ_0), and viceversa low negative value for formulae that on those trajectories disagree. For what concerns the measure μ_0 on $\mathcal{T}$, it is designed in such a way that simple signals are more probable, considering total variation and number of changes in the monotonicity as metrics for measuring the complexity of trajectories, we refer to [8] for full details. Note that, although the feature space $\mathbb{R}^\mathcal{T}$ (which we call the *latent semantic space*) into which ρ (and thus Equation 1) maps formulae is infinite-dimensional, in practice the kernel trick allows to circumvent this issue. It does so by mapping each formula to a vector of dimension equal to the number of training formulae, i.e. those used to evaluate the kernel (Gram) matrix. Such embeddings are continuous representations of discrete symbolic objects, and can be used to solve tasks such as predicting the expected robustness and the satisfaction probability of a stochastic process via continuous optimization-based ML algorithms.

# var	$\tau =$ 0.95	$\tau =$ 0.98
3	10	13
4	11	16
5	14	19
6	16	22
7	18	25
8	20	28
9	22	31
10	24	35

Table 1: # PC for achieving $\mathcal{X}_d \geq 95\%$ (resp. 98%), increasing the number of variables # var.

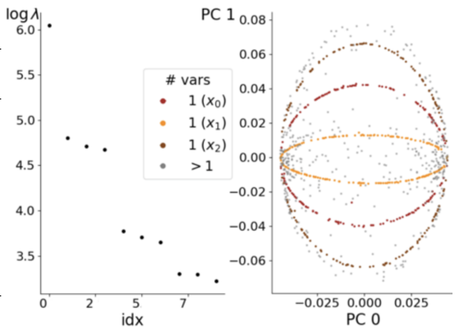

Figure 1: For a set of STL formulae with 3 variables (left) log spectrum of the covariance matrix of its Gram matrix; (right) 1st vs 2nd PC, showing formulae with only var. ● 0 ● 1 ● 2.

3 stl2vec

We are interested in **"semantic" embeddings**: intuitively, mapping formulae with similar semantics to nearby vectors; formally, given that the robustness ρ captures the considered semantics in the infinite-dimensional latent semantic space $\mathbb{R}^{\mathcal{T}}$, the new embeddings should (approximately) preserve the distances induced by the kernel in Equation 1, and thus should essentially be ρ's "almost continuous" projections. In this work, we (i) provide an algorithmic procedure, called *stl2vec*, to *construct* explicit finite-dimensional semantic embeddings of STL formulae (Section 3.1), (ii) explore the geometry of such representations, producing *human-interpretable* explanations to a vast amount of information retained by the new representation (Section 3.2), and (iii) show the effectiveness of the embeddings in integrating temporal logic knowledge inside data-driven learning algorithm (Section 4). We remark that the explainability provides more control over producing continuous STL formulae embeddings. Finally, we also recall that creating finite-dimensional representations is a crucial step to make data more manageable (reducing the risk of incurring in the so-called curse of dimensionality), and help to eliminate noise and redundant information.

3.1 *Building Explicit STL Embeddings*

The starting point of our investigation are kernel embeddings for STL formulae as defined in Section 2. All reported results in this Section, unless differently specified, are obtained by keeping the default parameters used in [8]; later in the manuscript we will also report ablation studies to enforce our statements. Hence, starting from implicit infinite-dimensional embeddings constructed via Equation 1, we derive explicit finite-dimensional numerical representations of STL formulae using kernel PCA. As we will highlight in the remainder of the paper, this transformation gives us a deep insight into the geometry of these representations, to the point of making us able to give explanations for the vast majority of information captured by the embeddings.

In detail, the algorithm stl2vec proceeds as follows: given a fixed set of D STL formulae (that we call *training set*) and an integer $d \ll D$ representing the reduced dimension of the embeddings, we obtain the coordinates of the reduced dimensional space by performing the eigenvalue decomposition of the centered kernel matrix of the training set (which is D-dimensional) and retaining the top-d eigenvectors (i.e. PC), which are those corresponding to the d largest eigenvalues. These PC will be used to project the data into a lower-dimensional subspace. We remark that this procedure does not require any learning. We denote as stl2vec(d) of a STL formula φ its d-dimensional embedding computed as above.

In practical applications, given the set of eigenvalues of the kernel matrix of the data $\{\lambda_k\}_{k=1}^D$ (sorted in descending order) to select the number d of dimensions to retain, it is common to look at the so-called *proportion of variance explained*: $\mathcal{X}_d = \frac{\sum_{i=1}^d \lambda_i}{\sum_{j=1}^D \lambda_j}$, choosing the smallest d for which $\mathcal{X}_d \geq \tau$, for some threshold $\tau \in [0, 1]$. Notably, for STL kernel embeddings built from a training set of $D = 1000$ random formulae, only a few tens of components are necessary to explain more than 95% of the variability in the data, as reported in Table 1. Moreover, in Figure 1 (left) we plot the log-spectrum (first 10 eigenvalues) of a dataset of $D = 1000$ formulae with 3 variables, corresponding to the 95% of variance explained, as per Table 1.

In order to experimentally prove the independence of the individuated PC on the set of training formulae used to compute the STL kernel, we compare the coordinates found when changing the training set. In detail we sample 50 different training sets, coming from 5 different distributions, obtained by changing the parameter p_{leaf} of the formulae sampling algorithm $\mathcal{F}$ detailed in Section 2. We vary it in the set $[0.3, 0.35, 0.4, 0.45, 0.5]$ and sample 10 datasets for each value, each composed of $D = 1000$ STL formuae with 3 variables. We then reduce their dimension to $d = 13$ (hence retaining more than the 98% of information, according to Table 1). Results show that, up to permutation of coordinates, the **identified principal directions are almost the same across all datasets**. Indeed, if we compute the pairwise cosine similarity between corresponding PC of each possible pair of datasets, we get that, up to the 5^{th} PC, all datasets share a cosine similarity of at least 0.95, moreover similarity stays above 0.69 for all the 13 considered components, with both mean and median similarity being > 0.9 in every direction, for all possible pair of datasets, see also Section B of the supplementary material [32]. Hence the embeddings are robust w.r.t. the choice of training formulae, at least on their most significant components.

Finally, we check that the embeddings are semantic, by assessing linear correlation between the distance among kernel PCA embeddings (with $d = 10$) of each pair of formulae in the considered dataset, and the corresponding distance between robustness vectors, i.e. the vectors $\boldsymbol{\rho}(\varphi) = [\rho(\varphi, \xi_i)]_{i=0}^M$ of robustness of a STL formula φ computed on M (in our case $10\,000$) trajectories randomly sampled from μ_0. The Pearson correlation coefficient among the two quantities is 0.9688, and their correlation is graphically shown in Figure 2; intuitively, formulae whose quantitative robustness agrees on a high number of trajectories are mapped nearby in the continuous space of their stl2vec embeddings.

In summary, (i) the principal directions of the embeddings are inherent to the STL robustness semantics (and thus it makes sense to try and *explain* them), and (ii) our embeddings are also experimentally observed as semantic (and thus it makes sense to measure how well they *approximate* the full semantic information as defined by robustness and reflected by the kernel). We examine the former in Sec. 3.2 and the latter in Sec. 4.1.

It is worth noting that the STL kernel imposes a smoothing on the combinatorics of satisfiability, through the measure μ_0, for which the semantics of formulae is captured w.r.t. the probability distribution over trajectories (i.e. trajectories are weighted in such a way that STL formulae which only differ on few complicated signals are essentially considered equivalent), hence all the geometrical properties of the STL embeddings presented are valid up to this statistical filter. Such filter can however be changed by using a custom measure on trajectories for computing the kernel (e.g. the data generating distribution of the problem at hand), and this adds another layer of flexibility to our methodology.

3.2 Explaining Principal Directions

Having described how explicit embeddings for STL formulae are computed, and confirming their semantic character, we now delve into exploring the geometry of these representations. We substantiate our explanations by statistical evidence, namely strong correlations detailed in the remainder of this Section and in Section B.1 of the supplementary material [32]. Looking at the spectrum of the kernel matrix for formulae with 3 variables in Figure 1 (left) and recalling that clear gaps in the spectrum are an indication that dimensionality reduction including the components before the gap is meaningful, we immediately observe that, after a big gap between the first and the second eigenvalue, the spectrum is partitioned into groups of 3 eigenvalues divided by gaps. This intuitively suggests that principal directions (apart from the first one) might encode properties that hold variable-wise, possibly denoting that different variables are mapped to different sub-manifolds in the latent semantic space. Following this intuition, and having in mind the way in which embeddings are computed (i.e. starting from Equation (1)), we are able to provide an interpretable explanation for the information carried by the first principal direction and the following two sets of components, each composed of as many values as the variables appearing in the formulae. In particular, we identify statistical properties based on the robustness of STL formulae which are linearly correlated with the PC. This is intuitively meaningful since the quantitative semantics of STL is the bridge used by the STL kernel for mapping discrete formulae into a continuous space. For this reason, we also believe that further PC encode more refined properties related to the robustness profile of formulae, which we are not able to describe. We stress that a clear interpretation of projections obtained by kernel PCA is far from trivial, as seen in [30]. In this case, we work with objects and embeddings with a semantic nature, and this is reflected in the features captured by the PC, whose meaning is however not-immediate to assess.

The first principal direction PC0 describes the $\boxed{\text{median robustness}}$ of each formula φ over a random set of trajectories sampled from μ_0. For the statistical evidence refer to Section B.1 of the supplementary material [32].

Hence the first PC captures a descriptor of the satisfiability of a formula, which from a statistical point of view acts as the main source of variability of the robustness distribution computed by Equation 1.

The second group of principal components which is composed of n coordinates, when considering formulae of n variables, accounts instead for the $\boxed{\text{variability of the robustness}}$ over μ_0, being linearly correlated with the mean kernel similarity to formulae which exhibit high variance in robustness across signals sampled from μ_0. In detail, the quantity which is linearly correlated with each direction belonging to this group can be computed via the following steps, given a test dataset $\mathcal{D}$ of STL formulae with n variables:

A.1 Sample a random dataset $\mathcal{D}_i$ of STL formulae containing only variable x_i, with $i \in \mathbb{N}, 0 \leq i < n$;
A.2 Sample an arbitrary number of trajectories $\hat{\mathcal{T}}$ from μ_0; from the current trajectory distribution (e.g. μ_0);
A.3 Evaluate the robustness vector $\boldsymbol{\rho}(\varphi_j) = \{\rho(\varphi_j, \xi)\}_{\xi \in \hat{\mathcal{T}}}$ of each formula $\varphi_j \in \mathcal{D}_i$ (on the selected trajectories);
A.4 Compute the standard deviation $\sigma_j = \text{std}(\boldsymbol{\rho}(\varphi_j))$ of the robustness vector of each formula $\varphi_j \in \mathcal{D}$;
A.5 Select the indexes j of each σ_j corresponding to values above the 90^{th} percentile, to get a subset of formulae $\hat{\mathcal{D}}_i$;
A.6 Compute the vector of mean kernel similarity $\tilde{\mathbf{k}}|\mathbf{x_i} = \left\{\frac{1}{|\hat{\mathcal{D}}_i|}\sum_{k=1}^{|\hat{\mathcal{D}}_i|} k(\varphi_j, \varphi_k)\right\}_{j=1}^{|\mathcal{D}|}$ between the formulae in $\mathcal{D}$ and the ones obtained by previous steps;
A.7 $\forall i, \tilde{\mathbf{k}}|\mathbf{x_i}$ is then linearly correlated with one of the PC having index in $[1, n]$.

To give an intuitive description of the behaviour of formulae in $\hat{\mathcal{D}}_i$ obtained as per steps A.1-A.5, we have experimentally verified that most of them are properties which are robustly satisfied and robustly unsatisfied on a comparable number of trajectories sampled from μ_0.

The third group of principal components is composed of n directions as well, when considering STL formulae with n variables. The information they carry represents the $\boxed{\text{importance of each variable}}$ in determining the semantics/robustness of a formula, as it is directly proportional to the change in robustness when fixing the part of the signals involving the variable itself. In particular, the quantity which describes each of these PC can be computed with the following steps, starting with a given a test dataset $\mathcal{D}$:

B.1 Compute a set of m random trajectories $\Xi = \{\xi_k\}_{k=1}^m$ on n variables, according to the given distribution;
B.2 For each variable index $i \in \mathbb{N}, 0 \leq i < n$, compute the set of trajectories $\Xi_i = \{\xi_{ik}\}_{k=1}^m$ by replacing the i^{th} component of each signal in Ξ with the constant $\mathbf{0}$;
B.3 For each $\varphi \in \mathcal{D}$, compute the mean absolute difference $\{\tilde{\rho}_i(\varphi) = \frac{1}{m}\sum_{k=1}^m |\rho(\varphi, \xi_k) - \rho(\varphi, \xi_{ik})|\}_{i=0}^{n-1}$;
B.4 $\forall i, \tilde{\boldsymbol{\rho}}|\boldsymbol{x}_i = \{\tilde{\rho}_i(\varphi)\}_{\varphi \in \mathcal{D}}$ is then linearly correlated with one of the PC having index in $[1 + n, 2 \cdot n]$.

An intuitive understanding of the explanations can be given by considering simple requirements. If we take for example the following formulae of 1-variable: $G(x_0 \geq 0) \wedge F(x_0 < 0)$ and $G(x_0 \geq 0) \vee F(x_0 \leq 0)$ then we immediately recognise that they are a contradiction and a tautology, respectively. This is indeed reflected in the first two components of their embeddings, which are [-0.06357, 0.0025] and [0.0593, 0.0058], i.e. for both the second component is small, witnessing a little variability of their robustness across trajectories, while the first is high (positive) for the tautology and low (negative) for the contradiction (as shown in Figure 1 (right) the reference range of PC0 is ± 0.07 and of ± 0.08 for PC1). If we now take a slightly more complex formula in 2 variables, namely $\varphi = (G(x_0 \geq 0)) \wedge (G(x_1 \geq 0) \wedge F(x_1 < 0))$, then we recognize that it is a contradiction and that the most evident reason guiding our intuition only involves variable x_1, being the right conjuct of φ a contradiction in which only x_1 appears. The explainable components of φ are: [-0.03219, $-0.0272, -0.0018$, $0.0165, -0.4901$], which lead to the following observations: a high negative value (w.r.t. above mentioned ranges) for the first component together with a small value for a component belonging to the second group suggests that the formula is a contradiction, finally the fact that in the third group a component is small and positive, while the other is negative and an order of magnitude higher indicates that most of the semantic of φ only depends on a specific variable. These examples help in getting a sense of both the intuitive meaning of the explained components, and of their usefulness in grasping the semantic of a formula when the formula is too big to be understood just visually inspecting it, or when only its embedding is available (e.g. when it is the outcome of an optimization procedure).

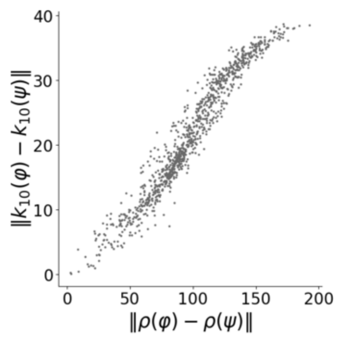

Figure 2: L_2 distance between 10-dim embeddings of random formulae vs L_2 distance among their respective robustness vectors.

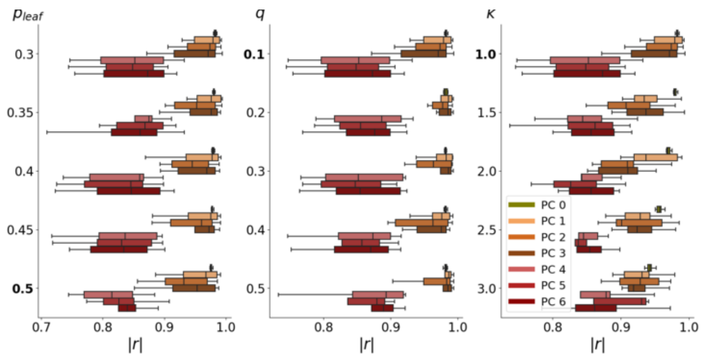

Figure 3: Resilience of the explanations of PC to changing of the parameters (from left to right) p_{leaf}, q and K in terms of absolute Pearson Correlation Coefficient (r). Bold labels represent default parameters.

Explanations of principal components are resilient to the measure considered in the space of trajectories. Our reference measure μ_0 (that is shown to be rather general in [8]) samples from piecewise linear functions in the interval $\mathcal{I} = [a, b]$ by: setting the number of discretization points in the trajectory and sampling the initial point from $\mathcal{N}(0, 1)$; sampling the total variation of the trajectory $tv = (\mathcal{N}(0, K))^2$; sampling the local variation between each pair of consecutive points uniformly in $[0, tv]$ and for each such a point changing the sign of the derivative (i.e. the monotonicity) with probability q. Finally consecutive points of the discretization are linearly interpolated to make the signal continuous. Hence μ_0 has the following parameters which can be tuned in order to significantly change the probability space of trajectories: (i) the mean q of the Bernoulli distribution governing the number of changes in the monotonicity of each signal and (ii) the standard deviation K of the Gaussian distribution from which the total variation of each trajectory is sampled.

We test the stability of our explanations by measuring the Pearson correlation coefficient r between the PC and the corresponding statistical quantities that we argue are their interpretation. For what concerns μ_0, by increasing q we are considering signals with an increasing number of changes in monotonocity, while by increasing K we are testing trajectories with larger total variation. Besides, considering the formulae distribution $\mathcal{F}$ (see Section 2), decreasing the parameter p_{leaf} increases the syntactic complexity of formulae. In Figure 3 we show the quantiles of the distribution of the absolute linear correlation coefficient $|r|$ between the PC and our explanations, across 50 independent datasets of STL formulae, in all the described ablation studies, verifying that it remains high in all settings, hence establishing the resilience of our interpretations. Moreover, we verify the stability of the explanations by changing the number n of variables in formulae from 3 to 10: denoting the median absolute correlation coefficient $|r|$ as $\eta_{|r|}$, we have $\eta_{|r|} > 0.97$ for the first PC, $\eta_{|r|} > 0.84$ for the second group of PC and $\eta_{|r|} > 0.8$ for the third group of PC, again proving resilience of the explanations. We remark here that, according to Table 1, when the number of variables is higher then 5 we are providing an interpretation for more than the 95% of the variance in the data. Additional results and plots are reported in Section B.1 of the supplementary material [32]. Finally, we test the stability of our explanations when replacing μ_0 with another stochastic process, namely the SIRS epidemiological model [6]: for the first component the median correlation is $\eta_{|r|} = 0.98$, for the second group of PC $\eta_{|r|} > 0.53$ and for the third group $\eta_{|r|} > 0.57$, showing moderate linear correlation, hence resilience of the explanations also for a completely different trajectory distribution. Interestingly, if we plot PC0 against PC belonging to the second group we are not only able to individuate formulae in which only a variable appears, but also identify the involved variable (i.e. its index), as reported in Figure 1 (right). Intuitively, this might depend on: (i) the fact that the explanations for the second group of components hold variable-wise (suggesting that different variables are mapped to different semantic subspaces) and (ii) the significant amount of information carried by PC0, observable from the gap after PC0 in Figure 1 (left). A similar behaviour is observed when considering PC belonging to the third group, as reported in Section B of the supplementary material [32]. From the same plot it is possible to observe a quadratic relation between PC0 and PC belonging to the second group (PC1 in the picture). Although a clear explanation for this phenomenon is still lacking, we can interpret the behavior of formulae mapped to the extreme points of the three ellipsis: PC0 $\approx$ 0 denotes formulae which neither robustly satisfy nor robustly unsatisfy any trajectory, or which robustly satisfy and unsatisfy a comparable number of trajectories, hence they are likely to have a highly variable robustness vector, explaining the fact that the (absolute) value for the second group of PC is high; viceversa, a formula whose variability is $\approx$ 0, for the opposite reason, is expected to have a high absolute median robustness value.

4 Applications

We claim and experimentally prove the high semantic expressiveness and the practical usefulness of stl2vec embeddings in two different scenarios: predicting average robustness and satisfaction probability of properties in a stochastic process (as defined in Section 2) and semantically conditioning a deep learning generative model for the generation of trajectories compliant to arbitrary temporal properties. We emphasize that in all the presented applications, the distribution used to evaluate Equation 1 is mu_0. Indeed, we experimentally prove that it is very effective in capturing semantic similarity of formulae, hence it can be used as reference distribution for all contexts in which neither the data generating distribution nor a consistent number of trajectories are available.

4.1 Predictive Power of Explicit Embeddings

In this suite of experiments, we use the embeddings of STL formulae as input for ridge regression in order to predict: robustness of formulae $\varphi \in \mathcal{F}$ on single trajectories $\xi \in \mathcal{T}$, i.e. the function $\rho : \varphi \mapsto \rho(\varphi, \xi)$; expected robustness $\mathbb{E}_{\xi\sim\mu_0}[\rho(\varphi, \xi)]$ and satisfaction probability $\mathbb{E}_{\xi\sim\mu_0}[s(\varphi, \xi)]$ of formulae $\varphi \in \mathcal{F}$, proxied by the experimental averages on a stochastic system $\{\xi_j \in \mathcal{T}\}_{j=1}^m$, i.e. respectively $R : \varphi \mapsto \frac{\sum_j \rho(\varphi,\xi_j)}{m}$ and $S : \varphi \mapsto \frac{\sum_j s(\varphi,\xi_j)}{m}$. We fix μ_0

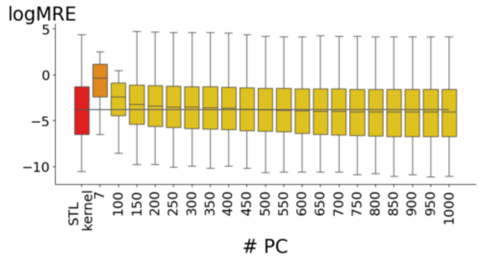

Figure 4: Mean of the quantiles for RE over 100 regression experiments for predicting average robustness of trajectories sampled from the SIRS model, varying the number of retained PC.

		relative error (RE)				absolute error (AE)			
		1quart	median	3quart	99perc	1quart	median	3quart	99perc
ρ	STL kernel	0.00772	0.02582	0.09225	1.41988	0.01362	0.04376	0.14283	0.92352
	stl2vec(250)	0.01246	0.03385	0.10293	1.26477	0.02409	0.06393	0.17317	0.83707
	stl2vec(500)	0.00917	0.02532	0.07942	1.14463	0.01769	0.04689	0.13455	0.79238
R	STL kernel	0.00629	0.02209	0.07593	1.19013	0.00608	0.02052	0.06493	0.43494
	stl2vec(250)	0.01162	0.03026	0.08979	1.28718	0.01129	0.02868	0.07669	0.38096
	stl2vec(500)	0.00822	0.02235	0.06859	1.0287	0.00797	0.021	0.05864	0.34801
S	STL kernel	0.00209	0.02762	0.85246	3.8337	0.00782	0.02634	0.08807	0.60182
	stl2vec(250)	0.00255	0.03235	1.18237	4.35775	0.01161	0.03256	0.09893	0.62148
	stl2vec(500)	0.00212	0.02821	0.87827	3.81897	0.00825	0.02679	0.08823	0.60294

Table 2: Mean of quantiles for RE and AE over 100 experiments for prediction of robustness on single trajectory ρ (top), average robustness R (middle) and satisfaction probability S (bottom), for a dataset of trajectories sampled from the SIRS model.

with its default parameters as the base measure on the space of trajectories (i.e. we use it for computing the kernel). We quantify the errors in terms both of Relative Error (RE) and Absolute Error (AE), and unless differently specified, we average results over 100 independent experiments. We denote as stl2vec(d) the embeddings obtained with our methodology, keeping the first d PC. We perform the above mentioned model checking task on different scenarios: still considering μ_0 as $\mathcal{T}$, but varying the dimensionality of signals; changing $\mathcal{T}$ considering trajectories coming from other stochastic processes, namely the SIRS epidemiological model (3-dim) and three other stochastic models (used as benchmarks also in [8]) simulated using the Python library StochPy [24] which are called Immigration (1-dim), Isomerization (2-dim) and Transcription (3-dim). We stress that in all the test cases, the STL kernel (hence the embeddings) is computed according to the base measure μ_0.

As reported in Table 2, for a dataset of $D = 1000$ STL formulae tested on trajectories sampled from the SIRS model, stl2vec embeddings of 500 components, i.e. half the original size, achieve results comparable to those of full STL kernel ridge regression. Moreover, even if we keep just 250 components, the predictive performance of the embeddings still is acceptable (median relative error $< 6\%$ when predicting ρ, $< 1\%$ when predicting R and $< 2\%$ for S). Interestingly, as shown in Figure 4, where we compare against standard kernel regression monitoring performance changes as the number of retained PC is varied, the quality of predictions in terms of both errors improves until the dimensionality of the representations is ≤ 300, then it stabilizes to values comparable to those of full STL kernel ridge regression (whose quantiles are reported in red in the figure). In the same figure, we highlight with an orange box the errors reported when doing regression just with the components that we are able to explain (7 in this case, since we are working with a dataset of 3 variables), hence in a scenario in which ridge regression can be fully interpreted. For what concerns the Immigration, Isomerization and Transcription models, under the same experimental assumptions, as well as experiments done on 10-dimensional signals sampled from μ_0, results in terms of median RE are reported in Table 3. In all cases, we observe that the difference in performance between full and reduced embeddings is limited: using stl2vec(500) instead of vanilla STL kernel brings at most 0.01% of additional error, while using stl2vec(250) brings a performance drop of at most 2.5%. In general, we can observe that results of these experiments are good: in all cases, the error when predicting ρ is $< 3.5\%$, it is $< 1.4\%$ when estimating R and $< 8.5\%$ for S. in We refer to Section C.1 of the supplementary material [32] for more detailed results, however the same observations done for the SIRS models applies in all tested cases. Hence, in summary, the dimensions required for our embeddings to capture almost complete information are reasonably small.

4.2 *Conditional Generation of Trajectories*

Another context in which stl2vec might be sensibly applied is that of conditional generation of trajectories, i.e. inside a model whose goal is to produce synthetic multivariate signals satisfying arbitrary STL properties. To the best of our knowledge, conditioning a deep learning model on temporal logic embeddings for generating time-series has not been studied before [35].

Conditional Variational Autoencoders (CVAE) [34, 17] are generative models that learn a probabilistic mapping between input data and distributions on a continuous latent space, conditioning the generation process on some given additional information. More in detail, given inputs $\boldsymbol{x}$ with associated conditioning vectors $\boldsymbol{y}$, CVAE maps $\boldsymbol{x}$ to latent representations $\boldsymbol{z}$ by simultaneously learning two parametric functions: a probabilistic generation network (decoder) $p_\theta(\boldsymbol{x}|\boldsymbol{y}, \boldsymbol{z})$ and an approximated posterior distribution (encoder) $q_\phi(\boldsymbol{z}|\boldsymbol{y}, \boldsymbol{x})$, by maximizing the evidence lower bound (given a prior $p_\psi(\boldsymbol{z}|\boldsymbol{y})$):

$$\begin{aligned}\mathcal{L}(\phi, \theta, \psi; \boldsymbol{x}, \boldsymbol{y}) =& \mathbb{E}_{\boldsymbol{z}\sim q_\phi(\boldsymbol{z}|\boldsymbol{y},\boldsymbol{x})}[\log p_\theta(\boldsymbol{x}|\boldsymbol{y}, \boldsymbol{z})] \\ &- \beta \cdot KL[q_\phi(\boldsymbol{z}|\boldsymbol{y}, \boldsymbol{x}) \| p_\psi(\boldsymbol{z}|\boldsymbol{y})]\end{aligned} \tag{2}$$

where $KL[\cdot\|\cdot]$ is the Kullback-Leibler divergence, weighted by a hyperparameter $\beta \in \mathbb{R}$ controlling the balance between the reconstruction accuracy and the regularization of the learned latent space [16]. Once trained, one might use the decoder as a generative model, by sampling vectors $\boldsymbol{z}$ from the prior distribution and adding conditional information $\boldsymbol{y}$, to obtain a point $\hat{\boldsymbol{x}}$ which should satisfy the given condition.

	ρ	R	S
Immigration	0.023/0.030/0.023	0.014/0.017/0.013	0.024/0.024/0.024
Isomerization	0.027/0.043/0.027	0.008/0.015/0.008	0.043/0.047/0.043
Transcription	0.033/0.054/0.033	0.011/0.019/0.010	0.064/0.085/0.065
μ_0 (10-dim)	0.034/0.039/0.035	0.003/0.005/0.003	0.005/0.006/0.005

Table 3: Median RE (across 100 experiments) when using STLkernel/stl2vec(250)/stl2vec(500) in learning model checking under different test trajectory distributions.

We devise a CVAE for multivariate time-series data, whose objective is to generate trajectories statistically similar to those of the data generating distribution and satisfying a given STL requirement, provided in the form of stl2vec embedding. More in detail, we encode signals using multiple stacked 1D convolutional layers, and decode them using the same number of 1D transposed convolutions; both the encoder and the decoder take as conditioning vector $\boldsymbol{y}$ the stl2vec representation of a property each input trajectory satisfies. We trained the architecture on signals sampled from the SIRS model [6] (which are 3-dimensional time-series): we randomly sampled a set $\mathcal{D}_{train}$ of 1000 formulae from $\mathcal{F}$, and $\forall\varphi \in \mathcal{D}_{train}$ we generated 200 SIRS trajectories ξ via SSA satisfying φ (we do not exclude

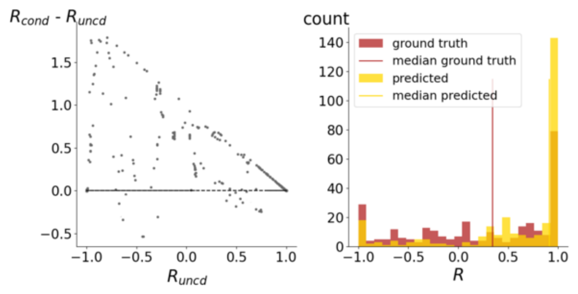

Figure 5: Results of a random experiment for the conditional generation of trajectories using CVAE, in terms of average robustness.

that the same signal might appear multiple times associated with different properties). The conditioning vector of each input ξ is then computed with stl2vec, retaining 250 components. We test the capability of the network to generate a trajectory ξ compliant with a given STL property φ. Hence, for each test formula $\varphi \sim \mathcal{F}$ in the test set $\mathcal{D}_{test}$, represented as a 250-dimensional semantic vector using stl2vec, we decode 1000 signals, and compute the satisfaction probability and the average robustness of φ on them, denoted as S_{cond} and R_{cond} respectively. Ideally, all the generated trajectories should satisfy (robustly) the corresponding φ; practically, we compare our results against the satisfaction probability and the average robustness of all $\varphi \in \mathcal{D}_{test}$ on a set of 10 000 unconstrained signals sampled from the SIRS model via SSA, denoted respectively as S_{uncd} and R_{uncd}. Results are shown in Figure 5, where we plot the difference in average robustness as a function of R_{uncd}. Comparing the distribution of R_{uncd} against that of R_{cond}, as done in Table 4 and on the histogram of Figure 5, highlights the improvement in having trajectories compliant to a given STL requirement when using a generative model. We experimented with conditioning vectors of different dimensions: retaining $[10, 50, 100, 250, 500]$ components yields a median R_{cond} of $[0.7008, 0.8134, 0.8737, 0.903, 0.9023]$ and a median S_{cond} of $[0.8305, 0.9065, 0.9363, 0.9515, 0.951]$, respectively (being R_{uncd} and S_{uncd} as in Table 4). In stark constrast, using implicit STL kernel embeddings of dimension 1000 we get median R_{cond} and S_{cond} of 0.4657 and 0.73, probably because they contain redundant information which confuses the algorithm. In conclusion, our dedicated finite-dimensional embedding are much better suited for the task than the full semantic information $\rho(\varphi, \cdot)$ even if the latter can be represented also finite-dimensionally (by the Gram matrix for the original kernel) with high enough dimension from enough data. See Section C.2 of the supplementary material [32] for more results.

	1perc	1quart	median	3quart	99perc
R_{uncd}	-0.9994 ± 0.0004	-0.5128 ± 0.0123	0.0869 ± 0.0104	0.7321 ± 0.0046	1.0 ± 0
R_{cond}	-1.0 ± 0	-0.6157 ± 0.0086	0.903 ± 0.0043	1.0 ± 0	1.0 ± 0
S_{uncd}	3.23e-04 ± 0.0003	0.229 ± 0.0076	0.5243 ± 0.0051	0.8122 ± 0.0022	1.0 ± 0
S_{cond}	0.0 ± 0	0.1923 ± 0.0045	0.9515 ± 0.0021	1.0 ± 0	1.0 ± 0

Table 4: Mean and standard deviation of quantiles of the distributions of R_{uncd} (resp. S_{uncd}) and R_{cond} (resp. S_{cond}), over 300 test formulae, averaged over 30 experiments.

5 Related Work

Finding continuous embedding of logical formulae has been an active research topic lately, with several works using Graph Neural Networks (GNN) for encoding the parsing tree of a formula to a continuous representation [20]. Most of them, however, consider propositional and/or first-order logic [9, 36, 22, 31], hence are hard to generalize to temporal logics such as STL. In [1] a Semantic Probabilistic Layer is devised to impose properties on the output of a DL model, leveraging circuit representations of formulae. Although strictly related to ours, the approach is specific for DL model. Other works such as [10, 13] devise NeSy architectures which approximates first-order logic operations with neural networks, and then implement rules as neural operators applied to tensor representations of premises, to generate tensor representation of conclusions. Finding continuous embeddings of temporal logic formulae is addressed in: [37], where a GNN is used to construct semantic-based embeddings of automata generated from Linear Temporal Logic (LTL) formulae and [14], where STL formulae are mapped to a continuous space by training a skip-gram and then used inside a neural network controller. The main difference between our method and the cited works is that stl2vec embeddings are not learnt, hence they are more controllable and robust, since they do not rely upon any training.

Using STL formalism inside machine learning algorithm has been exploited in: [21], where a STL formula is learnt which abstracts the computational graph of a neural networks trained to perform interpretable classification of time-series behaviour; [23], where STL is used as language to enhance the training of a neural network model for sequence predictions compliant to a set of predefined properties; [15], in which a tool is devised for the translation of informal requirements, given as English sentences, into STL. In all these cases, we believe that our approach can be valuably integrated for enforcing the semantics of the involved properties inside the neural architectures.

Logic-based distances between models are typically tackled in the area of formal method using branching logic, e.g. bisimulation metrics for Markov models [3, 2]; the problem of computing the distance between STL specifications is instead addressed in [25] and applied to the generation of designs that exhibit desired behaviors specified in STL, in the field of synthetic genetic circuits. Differently, our work does not focus on the (dis)similarity between formulae, but instead aims at finding a semantic-preserving continuous representation of STL properties.

6 Conclusions

In this work we propose a constructive algorithm for computing *interpretable* finite-dimensional explicit embeddings of Signal Temporal Logic (STL) formulae. We demonstrate their predictive power both as features for learning models and as semantic conditioning vectors inside other algorithms; most importantly, we provide explanations for a vast amount of information retained by the embeddings, a task which is highly non-trivial in general, but which is possible in this scenario due to the semantic nature of the objects involved. We believe that stl2vec has the potential to be a new framework for incorporating background knowledge in learning algorithms, under the umbrella of Neuro-Symbolic computing. We plan to extend this algorithm to other logics, such as Linear Temporal Logic (LTL); this requires finding a distribution (hence a sampling algorithm) for LTL discrete traces, which are typically higher dimensional than continuous signals used for STL. We also aim at using stl2vec as semantic conditioning information inside learning algorithm in other contexts, such as the synthesis of robot controllers satisfying some given (safety) properties. Most importantly, we would like to devise a way for inverting such embeddings, hence opening the doors to plenty of other applications, such as requirement mining.

Acknowledgements

This study was carried out within the PNRR research activities of the consortium iNEST (Interconnected North-Est Innovation Ecosystem) funded by the European Union Next-GenerationEU (Piano Nazionale di Ripresa e Resilienza (PNRR) – Missione 4 Componente 2, Investimento 1.5 – D.D. 1058 23/06/2022, ECS_00000043) and the MUR PRIN project 20228FT78M DREAM (modular software design to reduce uncertainty in ethics-based cyber-physical systems). This manuscript reflects only the Authors' views and opinions, neither the European Union nor the European Commission can be considered responsible for them. The work was further partially supported by the MUNI Award in Science and Humanities (MUNI/I/1757/2021) of the Grant Agency of Masaryk University.

References

[1] K. Ahmed, S. Teso, K. Chang, G. V. den Broeck, and A. Vergari. Semantic probabilistic layers for neuro-symbolic learning. In *NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022.

[2] P. Amortila, M. G. Bellemare, P. Panangaden, and D. Precup. Temporally extended metrics for markov decision processes. In *Workshop on Artificial Intelligence Safety, AAAI 2019, Honolulu, Hawaii, January 27, 2019*, volume 2301 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2019.

[3] G. Bacci, G. Bacci, K. G. Larsen, R. Mardare, Q. Tang, and F. van Breugel. Computing probabilistic bisimilarity distances for probabilistic automata. In *CONCUR 2019, August 27-30, 2019, Amsterdam, the Netherlands*, volume 140 of *LIPIcs*, pages 9:1–9:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[4] C. Baier and J.-P. Katoen. *Principles of model checking*. MIT press, 2008.

[5] E. Bartocci, J. V. Deshmukh, A. Donzé, G. Fainekos, O. Maler, D. Nickovic, and S. Sankaranarayanan. Specification-based monitoring of cyber-physical systems: A survey on theory, tools and applications. In *Lectures on Runtime Verification - Introductory and Advanced Topics*, volume 10457 of *Lecture Notes in Computer Science*, pages 135–175. Springer, 2018.

[6] R. Beckley, C. Weatherspoon, M. Alexander, M. Chandler, A. Johnson, and G. S. Bhatt. Modeling epidemics with differential equations. 2013.

[7] T. R. Besold, A. S. d'Avila Garcez, S. Bader, H. Bowman, P. M. Domingos, P. Hitzler, K. Kühnberger, L. C. Lamb, P. M. V. Lima, L. de Penning, G. Pinkas, H. Poon, and G. Zaverucha. Neural-symbolic learning and reasoning: A survey and interpretation. In *Neuro-Symbolic Artificial Intelligence: The State of the Art*, volume 342 of *Frontiers in Artificial Intelligence and Applications*, pages 1–51. IOS Press, 2021.

[8] L. Bortolussi, G. M. Gallo, J. Kretínský, and L. Nenzi. Learning model checking and the kernel trick for signal temporal logic on stochastic processes. In *TACAS 2022, Held as Part of ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings, Part I*, 2022.

[9] M. Crouse, I. Abdelaziz, C. Cornelio, V. Thost, L. Wu, K. D. Forbus, and A. Fokoue. Improving graph neural network representations of logical formulae with subgraph pooling. *CoRR*, abs/1911.06904, 2019.

[10] H. Dong, J. Mao, T. Lin, C. Wang, L. Li, and D. Zhou. Neural logic machines. In *ICLR 2019, New Orleans, LA, USA*, 2019.

[11] D. T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *The Journal of Physical Chemistry*, 81(25):2340–2361, 1977.

[12] E. Giunchiglia, M. C. Stoian, and T. Lukasiewicz. Deep learning with logical constraints. In *IJCAI 2022, Vienna, Austria, 23-29 July 2022*, pages 5478–5485. ijcai.org, 2022.

[13] C. Glanois, Z. Jiang, X. Feng, P. Weng, M. Zimmer, D. Li, W. Liu, and J. Hao. Neuro-symbolic hierarchical rule induction. In *ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pages 7583–7615. PMLR, 2022.

[14] W. Hashimoto, K. Hashimoto, and S. Takai. Stl2vec: Signal temporal logic embeddings for control synthesis with recurrent neural networks. *IEEE Robotics Autom. Lett.*, 7(2):5246–5253, 2022.

[15] J. He, E. Bartocci, D. Nickovic, H. Isakovic, and R. Grosu. Deepstl - from english requirements to signal temporal logic. In *44th IEEE/ACM ICSE 2022, Pittsburgh, PA, USA, May 25-27, 2022*, pages 610–622. ACM, 2022.

[16] I. Higgins, L. Matthey, A. Pal, C. P. Burgess, X. Glorot, M. M. Botvinick, S. Mohamed, and A. Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. In *ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.

[17] O. Ivanov, M. Figurnov, and D. P. Vetrov. Variational autoencoder with arbitrary conditioning. In *ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019.

[18] I. T. Jolliffe and J. Cadima. Principal component analysis: a review and recent developments. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 374, 2016.

[19] R. Kowalski. *Computational Logic and Human Thinking: How to Be Artificially Intelligent*. Cambridge University Press, 2011.

[20] L. C. Lamb, A. S. d'Avila Garcez, M. Gori, M. O. R. Prates, P. H. C. Avelar, and M. Y. Vardi. Graph neural networks meet neural-symbolic computing: A survey and perspective. In *IJCAI 2020*, pages 4877–4884, 2020.

[21] D. Li, M. Cai, C. Vasile, and R. Tron. Learning signal temporal logic through neural network for interpretable classification. In *ACC 2023, San Diego, CA, USA, May 31 - June 2, 2023*, pages 1907–1914. IEEE, 2023.

[22] Q. Lin, J. Liu, L. Zhang, Y. Pan, X. Hu, F. Xu, and H. Zeng. Contrastive graph representations for logical formulas embedding. *IEEE Trans. Knowl. Data Eng.*, 35(4):3563–3574, 2023.

[23] M. Ma, J. Gao, L. Feng, and J. A. Stankovic. Stlnet: Signal temporal logic enforced multivariate recurrent neural networks. In *NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[24] T. R. Maarleveld, B. G. Olivier, and F. J. Bruggeman. Stochpy: a comprehensive, user-friendly tool for simulating stochastic biological processes. *PloS one*, 8(11):e79345, 2013.

[25] C. Madsen, P. Vaidyanathan, S. Sadraddini, C. I. Vasile, N. A. DeLateur, R. Weiss, D. Densmore, and C. Belta. Metrics for signal temporal logic formulae. In *CDC 2018, Miami, FL, USA, December 17-19, 2018*, pages 1542–1547. IEEE, 2018.

[26] O. Maler and D. Nickovic. Monitoring temporal properties of continuous signals. In *Proc. of FORMATS 2004 and FTRTFT 2004, LNCS, vol. 3253, Springer (2004), pp. 152-166*, volume 3253 of *Lecture Notes in Computer Science*, pages 152–166. Springer, 2004.

[27] J. McCarthy. Programs with common sense. Technical report, USA, 1960.

[28] A. Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, pages 46–57, 1977.

[29] C. E. Rasmussen and C. K. I. Williams. *Gaussian processes for machine learning*. Adaptive computation and machine learning. MIT Press, 2006.

[30] F. Reverter, E. Vegas, and J. M. Oller. Kernel-pca data integration with enhanced interpretability. *BMC Syst. Biol.*, 8(S-2):S6, 2014.

[31] G. Saveri and L. Bortolussi. Towards invertible semantic-preserving embeddings of logical formulae. In *NeSY 2023, Siena, Italy, July 3-5*, volume 3432 of *CEUR Workshop Proceedings*, pages 174–194, 2023.

[32] G. Saveri, L. Nenzi, L. Bortolussi, and J. Křetínský. stl2vec: Semantic and interpretable vector representation of temporal logic. *CoRR*, abs/2405.14389, 2024. Full version of this paper.

[33] B. Schölkopf, A. J. Smola, and K. Müller. Kernel principal component analysis. In *ICANN '97, 7th International Conference, Lausanne, Switzerland, October 8-10, 1997, Proceedings*, volume 1327 of *Lecture Notes in Computer Science*, pages 583–588. Springer, 1997.

[34] K. Sohn, H. Lee, and X. Yan. Learning structured output representation using deep conditional generative models. In *NeurIPS 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 3483–3491, 2015.

[35] Q. Wen, L. Sun, F. Yang, X. Song, J. Gao, X. Wang, and H. Xu. Time series data augmentation for deep learning: A survey. In *IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021*, pages 4653–4660. ijcai.org, 2021.

[36] Y. Xie, Z. Xu, K. S. Meel, M. S. Kankanhalli, and H. Soh. Embedding symbolic knowledge into deep networks. In *NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 4235–4245, 2019.

[37] Y. Xie, F. Zhou, and H. Soh. Embedding symbolic temporal knowledge into deep sequential models. In *IEEE International Conference on Robotics and Automation, ICRA 2021, Xi'an, China, May 30 - June 5, 2021*, pages 4267–4273. IEEE, 2021.