Bridging Logic and Learning: Decoding Temporal Logic Embeddings via Transformers

Sara Candussio[1 [0009−0004−5198−6970]] (✉), Gaia Saveri[1 [0009−0003−2948−7705]], Gabriele Sarti[2 [0000−0001−8715−2987]], and Luca Bortolussi[1 [0000−0001−8874−4001]]

[1] AILab, MIGe, University of Trieste, 34127 Trieste, IT
{sara.candussio,gaia.saveri}@phd.units.it, lbortolussi@units.it
[2] Center for Language and Cognition (CLCG), University of Groningen
g.sarti@rug.nl

Abstract. Continuous representations of logic formulae allow us to integrate symbolic knowledge into data-driven learning algorithms. If such embeddings are semantically consistent, i.e. if similar specifications are mapped into nearby vectors, they enable continuous learning and optimization directly in the semantic space of formulae. However, to translate the optimal continuous representation into a concrete requirement, such embeddings must be invertible. We tackle this issue by training a Transformer-based decoder-only model to invert semantic embeddings of Signal Temporal Logic (STL) formulae. STL is a powerful formalism that allows us to describe properties of signals varying over time in an expressive yet concise way. By constructing a small vocabulary from STL syntax, we demonstrate that our proposed model is able to generate valid formulae after only 1 epoch and to generalize to the semantics of the logic in about 10 epochs. Additionally, the model is able to decode a given embedding into formulae that are often simpler in terms of length and nesting while remaining semantically close (or equivalent) to gold references. We show the effectiveness of our methodology across various levels of training formulae complexity to assess the impact of training data on the model's ability to effectively capture the semantic information contained in the embeddings and generalize out-of-distribution. Finally, we deploy our model for solving a requirement mining task, i.e. inferring STL specifications that solve a classification task on trajectories, performing the optimization directly in the semantic space.

Keywords: Transformers · Neuro-Symbolic Embeddings · Temporal Logic.

1 Introduction

Integrating learning algorithms and symbolic reasoning is an increasingly prominent research direction in modern Artificial Intelligence (AI), towards striking a balance between the high efficiency of black box data-driven Machine Learning (ML) models and the interpretability of logical languages. The cornerstone of such efforts, within the Neuro-Symbolic (NeSy) computing paradigm [36,37], is knowledge representation by means of formal languages rooted on logic. A

promising approach to combine logic with machine learning is that of mapping logic formulae into continuous vectors, i.e. embeddings, which can be natively integrated into ML algorithms [7,11,17,28,33].

Having real-valued representation of logic specifications preserving their semantic, i.e. mapping similar formulae to nearby vectors, enables continuous learning and optimization directly in the semantic space of formulae, e.g. for conditioning generative models on producing data compliant to some background knowledge [28] or finding a requirement able to discriminate among regular and anomalous points [27]. However, to translate the devised optimal continuous representation into a concrete requirement, hence promoting interpretability and reliability of the resulting system, such embeddings must be invertible.

In this work we focus on temporal data and on a dialect of Linear Temporal Logic, namely Signal Temporal Logic (STL) [19,23], which is emerging as the defacto standard language for stating specifications of continuous systems varying over time, in various domains such as biological or cyber-physical systems [2]. Indeed, STL is powerful enough to describe many phenomena, yet easily interpretable, as it avoids the vagueness and redundancy of natural language, still being easy to translate in common words [12]. For example, in STL one can state properties like "within the next 10 minutes, the temperature will reach at least 25 degrees, and will stay above 22 degrees for the next hour". Notably, there exists a well-defined procedure for consistently embedding STL specifications in a real vector space, grounded directly in the semantics of the logic via kernel-based methods [5,28]. This embedding, however, is not invertible, essentially due to the fact that semantically equivalent formulae with different syntactic structure are mapped in the same vector.

On the other hand, Language Models (LMs) based on the Transformer architecture [32] have reached astonishingly high performance on a wide range of applications and for different data modalities [35]. Thanks to their effective and efficient self-attention mechanism, LMs have proven to be the most-powerful general-purpose representation learning models, setting new standards on various domain, beyond the traditional Natural Language Processing (NLP). For this reason, we believe that a decoder-only Transformer-based model could prove itself an effective choice for inverting continuous representations of STL formulae. Indeed, we tackle such decoding task as a translation from semantic vectors to strings representing formulae, constructing a small vocabulary from STL syntax.

Our contributions consist in: (i) end-to-end training of a decoder-only Transformer model for the downstream task of reconstructing STL specifications from a continuous representation encoding their semantics (Section 5.1); (ii) extensive experimental analysis on the effectiveness of our methodology across various levels of training formulae complexity to assess the impact of training data on the model's ability to effectively capture the semantic information contained in the embeddings and generalize out-of-distribution, as well as comparisons to related approaches (Section 5.1); (iii) leveraging the proposed architecture for requirement mining, i.e. inferring formally specified system properties from observed behaviors, by integrating our architecture inside a Bayesian Optimization (BO)

loop, proving that the resulting model is able to extract interpretable information form the input data, promoting knowledge discovery about the system (Section 5.2). Data, code and trained models presented in this work can be found at can be found at this[3] link.

2 Related Work

The ability of LM to understand formal languages in general, and temporal logic in particular, has been explored in the literature under different perspectives. A number of works exploits Transformed-based architectures to translate informal natural language statements to formal specifications, towards aiding the error-prone and time-consuming process of writing requirements [22]. In this context, either an encoder-decoder LM is trained from scratch as accurate translator from unstructured natural language sentences to STL formulae [12], or off-the-shelf LMs are finetuned on an analogous translation task for first-order logic (FOL) or linear-time temporal logic (LTL) [6,10]. Notably, Transformers are also trained to solve tasks typically pertaining to the formal methods realm: in [9] a LM is trained end-to-end for solving the LTL verification problem of generating a trace satisfying a given formula, showing generalization abilities of the model to the semantics of the logic; taking a dual perspective, in [14] the LTL requirement mining problem is framed as an auto-regressive language modeling task, in which an encoder-decoder LM is trained as a translator having as source language the input trace and as target language LTL.

Finally, performing symbolic regression (SR) using Transformer-based models is gaining momentum in the field as an alternative to genetic programming. SR is indeed the problem of inferring a symbolic mathematical expression of a function of which we have collected some observations in the form of input-output pairs: in [15] a LM is trained to simultaneously predict the skeleton and the numerical constants of the searched expression, possibly augmenting the generation with a planning strategy [29]; in [8] the investigation is pushed even further, as a Transformer is trained to infer multidimensional ordinary differential equations from observations, i.e. to model a dynamical systems from data.

Mining STL specifications from data has seen a surge of interest in the last few years, as reported in [3]. Many of such works decompose the requirement mining task in two steps, i.e. as bi-level optimization problem: they first learn the structure of the specification from data, and then instantiate a concrete formula using parameter inference methods [1,4,20,21,26,31]. Both in [21,27] Bayesian optimization, and in particular GP-UCB, is used towards optimizing the searched specification. In the same spirit of this work, [27] learns simultaneously both the structure and the parameters of the STL requirement, by performing optimization in a continuous space representing the semantics of formulae.

[3] `https://github.com/gaoithee/transformers/tree/main/src/transformers/models/stldec`

3 Background

Signal Temporal Logic (STL) is a linear-time temporal logic which expresses properties on trajectories over dense time intervals [19]. Signals (or trajectories) are here defined as 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*, i.e. a function over variables $\boldsymbol{x} \in \mathbb{R}^n$ of the form $f_\pi(\boldsymbol{x}) \geq 0$; $\neg$ 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 derived. 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, while a property is *globally* satisfied if it is true continuously in $[a, b]$; finally the *until* operator captures the relationship between two conditions φ, ψ in which the first condition φ holds until, at some point in $[a, b]$, the second condition ψ becomes true. 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 $\rho(\varphi, \xi, t) \in \mathbb{R}$. The latter, also called *robustness*, is a measure of how robust is the satisfaction of φ w.r.t. perturbations of the signals. Intuitively, robustness measures how far is a signal ξ from violating a specification φ, with the sign indicating the satisfaction status. Indeed, 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.

Embeddings of STL formulae are devised with an ad-hoc kernel in [5], which yields representations that have been experimentally proven to be semantic-preserving [28], i.e. STL specifications with similar meaning are mapped to nearby vectors. Indeed, such embeddings are not learned but grounded in the semantics of the logic, moving form the observation that 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 the space of trajectories $\mathcal{T}$, a kernel function capturing similarity among STL formulae on mentioned feature representations can be defined as:

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

opening the doors to the use of the scalar product in the Hilbert space L^2 as a kernel for $\mathcal{P}$; at a high level, 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. Hence, a D-dimensional embedding $k(\varphi) \in \mathbb{R}^D$ of a formula φ is obtained from Equation 1 by fixing (possibly at random) an *anchor set* of D STL requirements $\psi_1, \ldots, \psi_D$ such that $k(\varphi) = [k(\varphi, \psi_1), \ldots, k(\varphi, \psi_D)]$.

Transformers are a class of deep learning models designed to autoregressively handle sequential data effectively by leveraging self-attention mechanism [32]. Given a sequence of tokens $t = [t_1, \ldots, t_m]$, the model learns the distribution $P(t)$ that can be decomposed as $P(t) = P(t_1) \prod_{i=1}^{m-1} P(t_{i+1}|t_1, \ldots, t_i)$ using the chain rule, so that each conditional can be parameterized using a neural network optimized via Stochastic Gradient Descent (SGD) to maximize the likelihood of a corpus used for training.

Here, we focus on the decoder-only variant of this architecture [25], whose objective is to predict the next token given a context made of the last k already generated tokens. The fundamental learning mechanism in the Transformer decoder is the attention mechanism, that allows the model to contextualize token representation at each layer, dynamically determining the importance of each token in a computationally efficient (i.e. parallelizable) way.

4 Decoding STL Embeddings via Transformers

The main research question addressed with this work is that of checking whether a Transformer-based decoder-only model is able to decode a STL formula starting from a real-valued vector representing its semantics. In this context, we frame this problem as the approximate inversion of STL kernel embeddings computed as described in Section 3, i.e. using Equation 1. A positive answer to this question can be interpreted as an experimental proof of the fact that, in this setting, ad-hoc trained Transformers are able to grasp the semantics of STL.

4.1 Data

In order to train the model, we need to collect a set of pairs $\{(\varphi_i, k(\varphi_i))\}_i$ consisting of STL formulae φ_i and their corresponding kernel embeddings $k(\varphi_i)$ computed as described in Section 1. To construct such a set of formulae φ, we can consider a distribution $\mathcal{F}$ over STL formulae, defined by a syntax-tree random recursive growing scheme, which 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. Intuitively, the higher p_{leaf}, the smaller (in terms of number of operators and depth of the resulting syntax tree) the generated formula will be.

The work of [24] spots a light on the impact that training data has in making the model learn hierarchical patterns, resulting in different generalization abilities at inference time. While they work on linear and hierarchical rules on

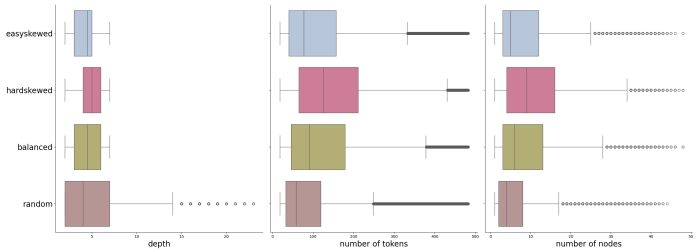

Fig. 1. Distribution of depth, number of tokens and number of nodes of the formulae composing the different training sets used.

natural language syntax trees, our case study is focused on syntactically reasonable representations of semantic vectors. We question whether, also in this different scenario, complexity level of training data (i.e. the composition of these sets in terms of formulae depths) should be reflected in some margin into the model's prediction abilities.

Specifically, we compose 4 different training sets, by sampling from the distribution $\mathcal{F}$ varying the parameter p_{leaf} and possibly filtering the obtained formulae based on the desired depth, each containing 78000 examples as follows:

- `easyskewed`: 52000 easier formulae (i.e. of depths 2, 3, 4), 26000 harder formulae (i.e. of depths 5, 6, 7);
- `hardskewed`: 26000 easier formulae (i.e. of depths 2, 3, 4), 52000 harder formulae (i.e. of depths 5, 6, 7);
- `balanced`: 13000 formulae of each considered depth level (i.e. depths 2, 3, 4, 5, 6, 7);
- `random`: 78000 formulae randomly sampled from $\mathcal{F}$ with $p_{\text{leaf}} = 0.45$, without filtering on the minimum or maximum depths; the result is a broadly spread distribution, comprising formulae from depth 2 to depth 23.

A glance on the distribution of the depth and number of nodes of the syntax tree of the formulae in the different training sets is shown in Figure 1.

The idea behind the exploitation of multiple training sets is to test whether the model can effectively learn to reconstruct a formula corresponding to a given embedding, or if the learning process is limited to replicating a superficial pattern observed in the training data [24]. To assess the generalization abilities acquired during the training stage, we test each model on a balanced test set consisting in formulae with various depths (from 2 to 7).

4.2 Model

The model follows a decoder-only architecture consisting in 12 layers (as it was originally proposed by [34]), each of which comprises 16 attention heads, and a feed-forward layer of dimension 4096 with Gaussian Error Linear Unit (GELU) [13] activation function. Each attention layer is followed by a residual

connection and a normalization layer to enhance training stability. This is also applied after the feed-forward layer. In order to prevent overfitting, we apply dropout with rate 0.1 before each residual connection and normalization operations.

The vocabulary is customized on the STL syntax, thus is limited to 35 tokens corresponding to the numbers, the (temporal and propositional) operators, the parentheses and the blank space separating the different logical structures. Additionally, the `unk` (i.e. unknown token), `pad` (i.e. pad token), `bos` (i.e. beginning-of-sentence) and `eos` (i.e. end-of-sentence) special tokens are added to the vocabulary.

The semantic information contained in the embedding of a STL formula is integrated through the cross-attention mechanism into the generating process: during the inference phase, each auto-regressively generated token is conditioned on the information contained in this semantic embedding.

Training Our experiments aim at assessing the ability of a decoder-only Transformer architecture to reconstruct a plausible formula starting from a semantic embedding. In this direction, we train from scratch the formerly described architecture using different training sets, as detailed in Section 4.1 and obtained different models, which we will refer to using the same name of the used training set (namely `random, hardskewed, easyskewed, balanced`).

Another point of interest in our analysis consists in the impact of the richness of the semantic representations, i.e. the chosen embedding dimension. Indeed, we both train models with hidden dimensions of 512 and 1024 and studied the differences in the generalization abilities of the resulting models. This is achieved by embedding both training and test sets using the STL kernel of Equation 1, fixing a set of either 1024 or 512 anchor formulae sampled from $\mathcal{F}$. As expected, the training time doubles when the hidden dimension is duplicated. To refer to the model trained with embedding of size 512, we append the suffix `small` to the name of the model.

We train all the models for 10 epochs (corresponding to ~ 24000 steps) on those training sets, all of size 78000, with a batch size of 32. We further elongate the training stage for the best models (according to the criteria described in Section 5.1) for 10 more epochs, in order to test whether or not we could observe a more refined behavior, when allowing for a greater training time. The optimization is performed using the AdamW optimizer [18], which decouples weight decay from gradient updates for improved generalization. A linear learning rate scheduler is applied, starting with a warm-up phase of 5000 steps before gradually decreasing over 50000 total training steps. The initial learning rate is set to $5 \cdot 10^{-5}$, with weight decay of 0.01 to mitigate possible overfitting.

At inference time the semantic embedding of a formula is fed to the trained model along with the `bos` (begin of sentence) starting token. The model auto-regressively infers the next formula elements starting from this NeSy representation.

5 Experiments

We claim and experimentally prove the effectiveness of our methodology in capturing the semantic information contained in the STL kernel representations in two different settings: (i) approximately inverting embeddings of STL formulae, see Section 5.1 and (ii) performing the requirement mining task in several benchmark datasets, as shown in Section 5.2.

5.1 Approximately Inverting STL Kernel Embeddings

The goal of this suite of experiments is twofold: verifying that a Transformer-based model is able to effectively grasp the semantics of STL, as encoded by the STL kernel embeddings, and compare it to a related approach based on Information Retrieval (IR) techniques; investigating if and how much the model size and the training set distribution influence such capabilities.

We test these abilities on a set of formulae with balanced depth levels, in order to robustly assess the aforementioned *desiderata*. The depths of the formulae range from 2 to 7, and it is worth noting that depth-4 formulae can already involve up to three different temporal operators, making them quite complex.

As an example, we can consider 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", which translates in STL as $F_{[0,10]}(\tau \geq 25 \wedge G_{[0,60]}\tau \geq 22)$ (i.e. a requirement with depth 4 and 3 logical operators).

Ideally, if we query our model with the embedding $k(\varphi)$ of a known formula φ, then we expect to decode a specification $\hat{\varphi}$ which has the same semantics of φ, with possibly a different syntax. To verify this property, we can consider the robustness vector of a formula, i.e. $\boldsymbol{\rho}(\varphi) = [\rho(\varphi, \xi_i)]_{i=1}^{M}$ for an arbitrary, but fixed, set of trajectories $\Xi = \{\xi_i\}_{i=1}^{M}$, and compute the quantity $d(\boldsymbol{\rho}(\varphi), \boldsymbol{\rho}(\hat{\varphi})) = ||\boldsymbol{\rho}(\varphi) - \boldsymbol{\rho}(\hat{\varphi})||_2$: given that two STL formulae are semantically similar if they behave similarly on the same set of signals, a lower value of $d(\boldsymbol{\rho}(\varphi), \boldsymbol{\rho}(\hat{\varphi}))$ indicates a good approximation of the inverse of $k(\varphi)$. Following the same reasoning line, as additional metrics we can consider: (a) the cosine similarity between robustness vectors, namely $\cos(\boldsymbol{\rho}(\varphi), \boldsymbol{\rho}(\hat{\varphi})) = \frac{\boldsymbol{\rho}(\varphi) \cdot \boldsymbol{\rho}(\hat{\varphi})}{||\boldsymbol{\rho}(\varphi)|| ||\boldsymbol{\rho}(\hat{\varphi})||} \in [-1, 1]$, with $\cos(\boldsymbol{\rho}(\varphi), \boldsymbol{\rho}(\hat{\varphi})) = 1$ if the original φ and reconstructed $\hat{\varphi}$ are (un)satisfied on the same set of trajectories of Ξ, possibly with different robustness degrees and (b) average number of signals in which φ and $\hat{\varphi}$ have opposite satisfaction status, i.e. $\text{diff}(\boldsymbol{s}(\varphi), \boldsymbol{s}(\hat{\varphi})) = \frac{\sum_{i=1}^{M} \mathbb{I}(s(\varphi, \xi_i) \neq s(\hat{\varphi}, \xi_i))}{M}$ being $\mathbb{I}$ the indicator function. When $\text{diff}(\boldsymbol{s}(\varphi), \boldsymbol{s}(\hat{\varphi})) = 0$ the the original φ and reconstructed $\hat{\varphi}$ are (un)satisfied on exactly the same set of trajectories.

Besides comparing all the trained models on the above mentioned metrics, we also analyze their performance w.r.t. the IR-based approach devised in [27], in which a dense vector database (hereafter denoted as DB) containing STL kernel embeddings of millions of formulae is built, so that an approximate inverse $\hat{\varphi}$ of the embedding $k(\varphi)$ of a specification φ is obtained with approximate nearest neighbors by querying the DB with $k(\varphi)$.

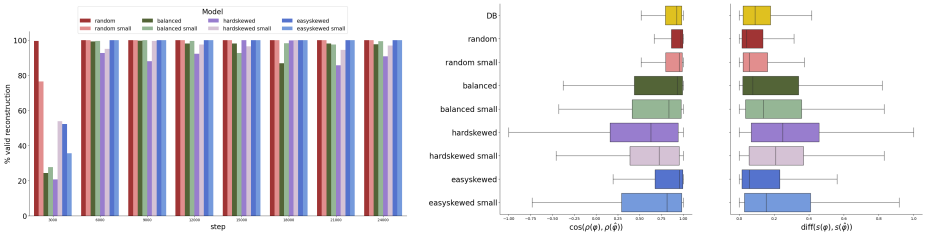

Fig. 2. Considering all the trained models: (left) percentage of syntactically valid formulae generated across the training steps and (right) comparison of results between transformer-based after 24000 steps of training models and semantic vector database.

Table 1. Comparison of results of different checkpoints of the best Transformer-based models and of the DB. On parenthesis the percentile of the distribution of the corresponding metric on a random set of formulae.

	$d(\boldsymbol{\rho}(\varphi), \boldsymbol{\rho}(\hat{\varphi}))$				$\cos(\boldsymbol{\rho}(\varphi), \boldsymbol{\rho}(\hat{\varphi}))$				$\text{diff}(\boldsymbol{s}(\varphi), \boldsymbol{s}(\hat{\varphi}))$			
Model	1quart	median	3quart	99perc	1quart	median	3quart	99perc	1quart	median	3quart	99perc
DB	16.51 (1)	28.13 (4)	43.87 (12)	81.46 (75)	0.795 (91)	0.9262 (98)	0.9802 (99)	0.9995 (100)	0.0225 (1)	0.0908 (4)	0.1797 (11)	0.4831 (47)
`random` (step 24K)	8.447 (1)	17.63 (2)	36.03 (7)	123.4 (100)	0.8617 (95)	0.9718 (99)	0.9939 (100)	1.000 (100)	0.0159 (1)	0.0423 (2)	0.1354 (7)	0.7571 (85)
`random small` (step 24K)	11.66 (1)	23.06 (2)	44.51 (12)	113.4 (100)	0.7985 (91)	0.9518 (99)	0.9885 (100)	1.000 (100)	0.0224 (1)	0.0586 (3)	0.1631 (9)	0.7154 (81)
`random` (step 48K)	6.054 (0)	12.71 (1)	29.89 (4)	110.1 (100)	0.9012 (96)	0.9854 (100)	0.9971 (100)	1.000 (100)	0.0073 (0)	0.0274 (1)	0.1085 (6)	0.6625 (75)
`random small` (step 48K)	7.462 (1)	16.01 (1)	31.22 (5)	101.1 (100)	0.8944 (96)	0.9774 (99)	0.9952 (100)	1.000 (100)	0.0106 (1)	0.0359 (2)	0.1159 (6)	0.5563 (61)

In Figure 2 and Table 1 we report the results of tests done on a `balanced` test set of 3000 formulae, i.e. constructed as the one used to train the homonymous model described in Section 4.1. From Figure 2 (left) it is possible to notice that `random` model decodes only syntactically valid formulae after just 3000 steps (~ 1.5 epoch) of training, and that by step 24000 (~ 10 epochs) all architectures reach $\geq 85\%$ of valid generations, highlighting the ability of Transformer-based models to grasp the syntax of STL. Moreover, on Figure 2 (right) we can check that the two models trained on the `random` training set are outperforming all the others, with the bigger one surpassing also the DB approach on the tested metrics. Indeed, after 24000 steps they achieve a median $\cos(\boldsymbol{\rho}(\varphi), \boldsymbol{\rho}(\hat{\varphi})) \geq 0.95$ and median $\text{diff}(\boldsymbol{s}(\varphi), \boldsymbol{s}(\hat{\varphi})) \leq 0.06$. Pushing further the investigation, we trained these 2 top-performing model for additional 24000 steps (hence for a total of $\sim$ 20 epochs) and showed the results on Table 1. There, we not only report the quantiles of the test distribution of the 3 considered metrics, but also the corresponding percentile of the distribution of the same quantities computed on a set of 10000 random formulae. Hence, we see that the additional training steps bring a relatively small improvement on all the considered metrics, and that the Transformer-based architecture is able to grasp the semantics of STL after ~ 10 epochs of training, reporting a median ~ 0.98 cosine similarity between the vectors of the robustness of the ground truth and retrieved formula, with the two having a different satisfaction status on $\sim 3\%$ of trajectories. The median value of $d(\boldsymbol{\rho}(\varphi), \boldsymbol{\rho}(\hat{\varphi}))$, despite not being interpretable, corresponds to the 1^{st} percentile of the random distribution, hence significantly low, enforcing the observation that the `random` model effectively learns the semantics of STL as encoded by the STL kernel embeddings.

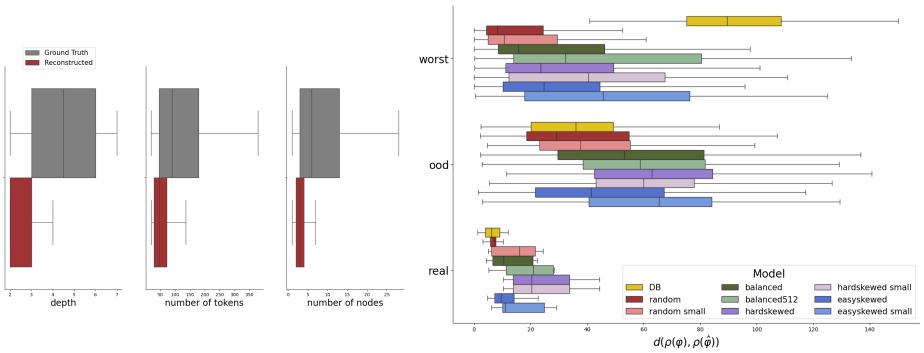

Fig. 3. (left) Distribution of depth, number of tokens and number of nodes of the ground truth and reconstructed formulae (by the `random` architecture) composing the `balanced` test set; (right) comparison of results between transformer-based models and semantic vector database on the test sets `worst, ood, real`.

Notably, from Figure 3, we observe that the `random` model tends to decode formulae that are syntactically simpler than those used to generate the test embeddings. We argue that this behavior may stem from the high syntactic variability observed in its training set, as shown in Figure 1. Since formulae with very different syntactic structures might nonetheless share very similar semantics, it is likely that the `random` training set contains pairs of specifications that are semantically close but very different in their structure, in terms e.g. of depth and number of nodes. However, given that the majority of training requirements are relatively simple, the Transformer is biased toward decoding shorter formulae. This results in a significant advantage for downstream applications in terms of interpretability: we are able to reconstruct a formula with very similar semantic meaning yet syntactically simpler.

Additionally, we tested our model on the following datasets:

- `worst`: a set of 400 formulae sampled from $\mathcal{F}$ using $p_{\text{leaf}} = 0.4$ which we experimentally found to be those on which the DB performs the worst. This set aims at testing whether the transformer-based model is able to generalize to the space of STL kernel embeddings which are scarcely covered by the formulae contained in DB;
- `ood`: a set of 500 STL formulae of depth 8, which are absent from the training sets of the `balanced`, `easyskewed`, and `hardskewed` models, and only present in a very limited amount in that of the `random` models;
- `real`: a small set of 15 formulae taken from [16], to check whether the proposed model is effective on requirements coming form real CPS applications.

The results are shown in Figure 3 (right), where all models successfully decode only valid formulae. The x-axis represents $d(\boldsymbol{\rho}(\varphi), \boldsymbol{\rho}(\hat{\varphi}))$, the distance between the robustness vectors of the gold and reconstructed formulae, and should be interpreted as *the lower, the better*. While this metric may offer limited inter-

pretability, it allows for a more fine-grained assessment of the relative differences between formulae.

For what concerns the `worst` dataset, all the trained model outperform the DB, confirming that the transformer-based decoder is able to effectively grasp the semantic of STL. Regarding the `ood` set, we notice that the DB is the best performing architecture, with the two models trained on the `random` set showing comparable performance, and the others reporting slightly worst results. This indicates that a greater variability in the structure of formulae over which the models are trained aids in length generalization. Finally, the `random` model is best performing on the `real` dataset, immediately followed by the DB.

5.2 STL Requirement Mining

The objective of the experiments described in this Section is that of checking whether our model can be leveraged for solving the requirement task, i.e. that of inferring a property (in the form of STL formula) characterizing the behavior of an observed system. We follow the approach of [20,21,27,26,1,4] and frame it as a supervised two-class classification problem, where the input consists of a set of trajectories divided in: those classified as regular (or positive) and those identified as anomalous (or negative), denoted as $\mathcal{D}_p$ and $\mathcal{D}_n$, respectively. The output is a (set of) STL formula(e) designed to distinguish between these two subsets. Additionally, we assume that these datasets originate from unknown stochastic processes, denoted as $\boldsymbol{X}_p$ and $\boldsymbol{X}_n$, respectively and adopt the approach of [21] and maximize the following function in order to mine a STL formula φ able to discriminate between positive and negative trajectories:

$$G(\varphi) = \frac{\mathbb{E}_{\mathbf{X_p}}[R_\varphi(\mathbf{X_p})] - \mathbb{E}_{\mathbf{X_n}}[R_\varphi(\mathbf{X_n})]}{\sigma_{\mathbf{X_p}}(R_\varphi(\mathbf{X_p})) + \sigma_{\mathbf{X_n}}(R_\varphi(\mathbf{X_n}))} \tag{2}$$

denoting as $\mathbb{E}_{\boldsymbol{X}}[R_\varphi(\boldsymbol{X})]$ and $\sigma_{\boldsymbol{X}}[R_\varphi(\boldsymbol{X})]$ respectively the expected value and the standard deviation of the robustness of a formula φ on trajectories sampled from the system $\boldsymbol{X}$. Following [27], we (i) frame the learning problem as the optimization of Equation 2 in the latent semantic space of formulae, i.e. in the space of embeddings of formulae individuated by the STL kernel of Equation 1 and (ii) tackle it by means of Bayesian Optimization (BO), and more specifically of the Gaussian Process Upper-Confidence Bound (GP-UCB) algorithm [30]. Hence, the overall iterative methodology follows these steps in each iteration:

1. Optimize the acquisition function based on the current set of pairs $(k(\varphi), G(\varphi))$ of kernel embeddings and corresponding objective function values to obtain new candidate embeddings;
2. Retrieve STL formulae corresponding to such candidate embeddings by inverting the them using the transformer-based decoder model;
3. Evaluate the objective function $G(\varphi)$ on the obtained formulae and record the pair achieving the maximum value.

Table 2. Best mined formula $\hat{\varphi}$, mean and standard deviation of MCR and $Prec$ in the test set across 3 different initialization seeds.

		$\hat{\varphi}$	MCR	$Prec$
`Linear`	`random` (step 24K)	$G_{[3,\infty]}(x_0 \geq -11.44)$	0.0200 ± 0.0291	0.9650 ± 0.0489
	`random small` (step 24K)	$G(x_0 \geq -15.09)$	0.0150 ± 0.0300	0.9714 ± 0.0571
	`random` (step 48K)	$F(x_0 \geq 0.1918)$	0.0150 ± 0.0300	0.9700 ± 0.0600
	`random small` (step 48K)	$G(x_0 \geq -5.586)$	0.0150 ± 0.0201	0.9670 ± 0.0485
`HAR`	`random` (step 24K)	$G_{[5,\infty]}(x_0 \geq 39.93)$	0.1333 ± 0.2666	0.8666 ± 0.2668
	`random small` (step 24K)	$G_{[3,8]}(x_0 \geq 29.37)$	0.1368 ± 0.2736	0.8631 ± 0.2736
	`random` (step 48K)	$G_{[2,\infty]}(x_0 \geq 33.62)$	0.0947 ± 0.1894	0.9052 ± 0.1894
	`random small` (step 48K)	$G_{[5,9]}(x_0 \geq 32.92)$	0.0736 ± 0.1473	0.9304 ± 0.1392
`LP5`	`random` (step 24K)	$G(x_2 \geq -9.272)$	0.0571 ± 0.0534	0.8818 ± 0.0754
	`random small` (step 24K)	$G(x_2 \geq -12.75)$	0.1713 ± 0.2731	0.9377 ± 0.0509
	`random` (step 48K)	$G(x_2 \geq -8.142)$	0.0857 ± 0.0534	0.9043 ± 0.0830
	`random small` (step 48K)	$G(x_2 \geq -5.614)$	0.0857 ± 0.0699	0.8777 ± 0.0976
`Train`	`random` (step 24K)	$F_{[18,22]}(G_{[13,16]}(x_0 \leq 4.823))$	0.0468 ± 0.0422	0.9183 ± 0.0716
	`random small` (step 24K)	$G_{[16,19]}(F_{[12,18]}(x_0 \leq 1.894))$	0.0428 ± 0.0346	0.7979 ± 0.3113
	`random` (step 48K)	$G_{[13,\infty]}(x_0 \leq 24.64)$	0.0652 ± 0.0717	0.8968 ± 0.1046
	`random small` (step 48K)	$\neg(F_{[16,\infty]}(x_0 \geq 1.538))$	0.1101 ± 0.1593	0.7929 ± 0.3041

In our experiments, we employ Gaussian Processes with a Matérn kernel, initialized with a batch of 100 points. Due to the high-dimensional STL kernel space (either 1024 or 512, as detailed in Section 4.2), we optimize the Upper Confidence Bound (UCB) acquisition function via Stochastic Gradient Descent (SGD).

Following the related literature [20,27], we test the Linear System (`Linear`), Human Activity Recognition (`HAR`), Robot Execution Failure in Motion with Part (LP5) and Cruise Control of Train (`Train`) time series classification datasets. Results in terms of Misclassification Rate (MCR) and Precision ($Prec$), are reported in Table 2 for the most promising checkpoints of our model; recall is not shown being always equal to 1. For all datasets the `HAR` the `random` model trained for 24000 steps achieves an accuracy $\geq 90\%$ (which is always achieved by the `random` model trained for 48000 steps). Additionally, in Table 2 we show the best mined formula $\hat{\varphi}$ by each architecture, which we choose as the one obtaining the lowest MCR (namely 0 in all reported cases), using the syntactic simplicity as tiebreaker. Notably, in line with recent related works [20,27], all $\hat{\varphi}$ have at most 3 nodes in their syntax tree, hence are highly interpretable, thus promoting knowledge discovery of the resulting system.

Since our architecture is a learned model, it is highly likely that, when queried with a vector that is out-of-distribution (OOD) with respect to the STL kernel embeddings, it produces an invalid formula. This situation can arise, for example, during the exploration phase of the UCB acquisition function. In such cases, we assign an extremely low value to the objective function in Equation 2, in order to encourage the GP-UCB to explore more plausible regions of the STL embedding space. On the one hand, this has the positive effect of automatically

detecting OOD samples. A possible interpretation of why this occurs is that the embeddings of logical formulae lives in a lower dimensional manifold, hence vectors outside this manifold do not correspond to real requirements, hence embeddings corresponding to such a situation (or belonging to formulae which are semantically very different from the training data) are likely to give rise to invalid outputs. More in detail, the 1024-dimensional architectures take $\sim$ 1s to invert an embedding, while the 512-dimensional roughly 0.5s. Practically, due to the mentioned issue, we witness an computational time of $\sim$ 300s for the bigger models and $\sim$ 120s for the smaller ones, on the tests reported in Table 2.

6 Conclusion

In this work, we investigate the possibility of using a decoder-only Transformer-based architecture to invert continuous representation of Signal Temporal Logic formulae into valid requirements, which are semantically similar to the ground truth searched formulae. Our experiments prove that such model is able to generate valid formulae after only 1 epoch of training and to generalize to the semantics of the logic in about 10 epochs. Indeed, not only it is able to decode a given embedding into formulae which are semantically close to gold references, but that are often simpler in terms of length and nesting. It is worth noting that, due to the nature of Transformer-based architectures (and neural models in general), deriving meaningful theoretical bounds with practical applicability is extremely challenging. There is an inherent trade-off between speed and accuracy on one hand, and strong theoretical guarantees on the other.

These results leave open the question of whether Transformer-based models can be leveraged in other related Neuro-Symbolic tasks, possibly involving different formal languages, such as first-order logic. We indeed envision that such powerful generative capabilities of Language Models might be leveraged both, as we do here, to decode formulae from a continuous vector representing their semantic and, in an encoder-decoder setting, even to devise invertible by-design continuous semantic-preserving representations, opening the doors to a whole new range of applications. Finally, a further direction that could be explored involves the interpretability of these models; in particular, it could be interesting to study their internals, and check e.g. how the attention heads contribute in the decoding process when provided with the semantic embedding.

Acknowledgments. 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). This manuscript reflects only the Authors' views and opinions, neither the European Union nor the European Commission can be considered responsible for them. Gabriele Sarti is supported by the Dutch Research Council (NWO) for the project InDeep (NWA.1292.19.399).

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Aydin, S.K., Gol, E.A.: Synthesis of monitoring rules with STL. J. Circuits Syst. Comput. **29**(11), 2050177:1–2050177:26 (2020)
2. Bartocci, E., Deshmukh, J.V., Donzé, A., Fainekos, G., Maler, O., Nickovic, D., Sankaranarayanan, S.: Specification-based monitoring of cyber-physical systems: A survey on theory, tools and applications. In: Lectures on Runtime Verification - Introductory and Advanced Topics, Lecture Notes in Computer Science, vol. 10457, pp. 135–175. Springer (2018)
3. Bartocci, E., Mateis, C., Nesterini, E., Nickovic, D.: Survey on mining signal temporal logic specifications. Inf. Comput. **289**(Part), 104957 (2022)
4. Bombara, G., Vasile, C.I., Penedo, F., Yasuoka, H., Belta, C.: A decision tree approach to data classification using signal temporal logic. In: Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control, HSCC 2016. pp. 1–10. ACM (2016)
5. Bortolussi, L., Gallo, G.M., Kretínský, J., Nenzi, L.: Learning model checking and the kernel trick for signal temporal logic on stochastic processes. In: Tools and Algorithms for the Construction and Analysis of Systems - 28th International Conference, TACAS 2022,Proceedings, Part I (2022)
6. Cosler, M., Hahn, C., Mendoza, D., Schmitt, F., Trippel, C.: nl2spec: Interactively translating unstructured natural language to temporal logics with large language models. In: Enea, C., Lal, A. (eds.) Computer Aided Verification - 35th International Conference, CAV 2023, Paris, France, July 17-22, 2023, Proceedings, Part II. Lecture Notes in Computer Science, vol. 13965, pp. 383–396. Springer (2023)
7. Crouse, M., Abdelaziz, I., Cornelio, C., Thost, V., Wu, L., Forbus, K.D., Fokoue, A.: Improving graph neural network representations of logical formulae with subgraph pooling. CoRR **abs/1911.06904** (2019)
8. d'Ascoli, S., Becker, S., Schwaller, P., Mathis, A., Kilbertus, N.: Odeformer: Symbolic regression of dynamical systems with transformers. In: The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024. OpenReview.net (2024)
9. Hahn, C., Schmitt, F., Kreber, J.U., Rabe, M.N., Finkbeiner, B.: Teaching temporal logics to neural networks. In: 9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021. OpenReview.net (2021)
10. Hahn, C., Schmitt, F., Tillman, J.J., Metzger, N., Siber, J., Finkbeiner, B.: Formal specifications from natural language. CoRR **abs/2206.01962** (2022)
11. Hashimoto, W., Hashimoto, K., Takai, S.: Stl2vec: Signal temporal logic embeddings for control synthesis with recurrent neural networks. IEEE Robotics Autom. Lett. **7**(2), 5246–5253 (2022)
12. He, J., Bartocci, E., Nickovic, D., Isakovic, H., Grosu, R.: Deepstl - from english requirements to signal temporal logic. In: 44th IEEE/ACM 44th International Conference on Software Engineering, ICSE 2022, Pittsburgh, PA, USA, May 25-27, 2022. pp. 610–622. ACM (2022)
13. Hendrycks, D., Gimpel, K.: Gaussian error linear units (gelus). arXiv: Learning (2016), `https://api.semanticscholar.org/CorpusID:125617073`
14. Isik, I., Gol, E.A., Cinbis, R.G.: Learning to estimate system specifications in linear temporal logic using transformers and mamba. CoRR **abs/2405.20917** (2024)
15. Kamienny, P., d'Ascoli, S., Lample, G., Charton, F.: End-to-end symbolic regression with transformers. In: Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D.,

Cho, K., Oh, A. (eds.) Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022 (2022)
16. Khandait, T., Formica, F., Arcaini, P., Chotaliya, S., Fainekos, G., Hekal, A., Kundu, A., Lew, E., Loreti, M., Menghi, C., Nenzi, L., Pedrielli, G., Peltomäki, J., Porres, I., Ray, R., Soloviev, V., Visconti, E., Waga, M., Zhang, Z.: Arch-comp 2024 category report: Falsification. In: Frehse, G., Althoff, M. (eds.) Proceedings of the 11th Int. Workshop on Applied Verification for Continuous and Hybrid Systems. EPiC Series in Computing, vol. 103, pp. 122–144. EasyChair (2024). https://doi.org/10.29007/hgfv, /publications/paper/fKVR
17. Lin, Q., Liu, J., Zhang, L., Pan, Y., Hu, X., Xu, F., Zeng, H.: Contrastive graph representations for logical formulas embedding. IEEE Trans. Knowl. Data Eng. **35**(4), 3563–3574 (2023)
18. Loshchilov, I., Hutter, F.: Decoupled weight decay regularization (2019), https://arxiv.org/abs/1711.05101
19. Maler, O., Nickovic, D.: Monitoring temporal properties of continuous signals. In: Proc. of FORMATS and FTRTFT, LNCS, vol. 3253, Springer (2004), pp. 152-166. Lecture Notes in Computer Science, vol. 3253, pp. 152–166. Springer (2004)
20. Mohammadinejad, S., Deshmukh, J.V., Puranic, A.G., Vazquez-Chanlatte, M., Donzé, A.: Interpretable classification of time-series data using efficient enumerative techniques. In: HSCC '20: 23rd ACM International Conference on Hybrid Systems: Computation and Control. pp. 9:1–9:10. ACM (2020)
21. Nenzi, L., Silvetti, S., Bartocci, E., Bortolussi, L.: A robust genetic algorithm for learning temporal specifications from data. In: Quantitative Evaluation of Systems - 15th International Conference, Beijing, China, Proceedings. Lecture Notes in Computer Science, vol. 11024, pp. 323–338. Springer (2018)
22. Norheim, J.J., Rebentisch, E., Xiao, D., Draeger, L., Kerbrat, A., de Weck, O.L.: Challenges in applying large language models to requirements engineering tasks. Design Science **10** (2024)
23. Pnueli, A.: The temporal logic of programs. In: 18th Annual Symposium on Foundations of Computer Science (sfcs 1977). pp. 46–57 (1977)
24. Qin, T., Saphra, N., Alvarez-Melis, D.: Sometimes i am a tree: Data drives fragile hierarchical generalization. In: NeurIPS 2024 Workshop on Scientific Methods for Understanding Deep Learning (2024), https://openreview.net/forum?id=AHakCjAPOh
25. Radford, A., Narasimhan, K., Salimans, T., Sutskever, I.: Improving language understanding by generative pre-training. OpenAI Blog (2018), https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf
26. Saglam, I., Gol, E.A.: Cause mining and controller synthesis with STL. In: 58th IEEE Conference on Decision and Control, CDC 2019. pp. 4589–4594. IEEE (2019)
27. Saveri, G., Bortolussi, L.: Retrieval-augmented mining of temporal logic specifications from data. In: Machine Learning and Knowledge Discovery in Databases. Research Track - European Conference, ECML PKDD 2024, Vilnius, Lithuania, September 9-13, 2024, Proceedings, Part VII. Lecture Notes in Computer Science, vol. 14947, pp. 315–331. Springer (2024)
28. Saveri, G., Nenzi, L., Bortolussi, L., Kretínský, J.: stl2vec: Semantic and interpretable vector representation of temporal logic. In: 27th European Conference on Artificial Intelligence, 19–24 October 2024, Santiago de Compostela, Spain. Frontiers in Artificial Intelligence and Applications, vol. 392, pp. 1381 – 1388. IOS Press (2024)

29. Shojaee, P., Meidani, K., Farimani, A.B., Reddy, C.K.: Transformer-based planning for symbolic regression. In: Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., Levine, S. (eds.) Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023 (2023)
30. Srinivas, N., Krause, A., Kakade, S.M., Seeger, M.W.: Information-theoretic regret bounds for gaussian process optimization in the bandit setting. IEEE Trans. Inf. Theory **58**(5), 3250–3265 (2012)
31. Vaidyanathan, P., Ivison, R., Bombara, G., DeLateur, N.A., Weiss, R., Densmore, D., Belta, C.: Grid-based temporal logic inference. In: 56th IEEE Annual Conference on Decision and Control, CDC 2017. pp. 5354–5359. IEEE (2017)
32. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is all you need. In: Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems. pp. 5998–6008 (2017)
33. Xie, Y., Xu, Z., Meel, K.S., Kankanhalli, M.S., Soh, H.: Embedding symbolic knowledge into deep networks. In: NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada. pp. 4235–4245 (2019)
34. Yenduri, G., M, R., G, C.S., Y, S., Srivastava, G., Maddikunta, P.K.R., G, D.R., Jhaveri, R.H., B, P., Wang, W., Vasilakos, A.V., Gadekallu, T.R.: Generative pre-trained transformer: A comprehensive review on enabling technologies, potential applications, emerging challenges, and future directions (2023), `https://arxiv.org/abs/2305.10435`
35. Zhao, W.X., Zhou, K., Li, J., Tang, T., Wang, X., Hou, Y., Min, Y., Zhang, B., Zhang, J., Dong, Z., Du, Y., Yang, C., Chen, Y., Chen, Z., Jiang, J., Ren, R., Li, Y., Tang, X., Liu, Z., Liu, P., Nie, J.Y., Wen, J.R.: A survey of large language models. CoRR **abs/2303.18223** (2025)
36. Garcez, A.d'A., Lamb, L.C.: Neurosymbolic AI: the 3rd wave. Artificial Intelligence Review, Kluwer Academic Publishers, USA. **56**(11) (2023)
37. Marra, G., Manhaeve, R., Tiddi, I., De Raedt, L.: From statistical relational to neurosymbolic artificial intelligence: A survey. Artificial Intelligence **328**, 104062 (2024)