GENERATING SYMBOLIC REASONING PROBLEMS WITH TRANSFORMER GANS

Anonymous authors
Paper under double-blind review

ABSTRACT

Constructing training data for symbolic reasoning domains is challenging: Existing instances are typically hand-crafted and too few to be trained on directly and synthetically generated instances are often hard to evaluate in terms of their meaningfulness. We study the capabilities of GANs and Wasserstein GANs equipped with Transformer encoders to generate sensible and challenging training data for symbolic reasoning domains. We conduct experiments on two problem domains where Transformers have been successfully applied recently: symbolic mathematics and temporal specifications in verification. Even without autoregression, our GAN models produce syntactically correct instances. We show that the generated data can be used as a substitute for real training data when training a classifier, and, especially, that training data can be generated from a real dataset that is too small to be trained on directly. Using a GAN setting also allows us to alter the target distribution: We show that by adding a classifier uncertainty part to the generator objective, we obtain a dataset that is even harder to solve for a classifier than our original dataset.

1 INTRODUCTION

Deep learning is increasingly applied to more untraditional domains that involve complex symbolic reasoning. Examples include the application of deep neural network architectures to SAT (Selsam et al., 2019; Selsam & Bjørner, 2019; Ozolins et al., 2021), SMT (Balunovic et al., 2018), temporal specifications in verification (Hahn et al., 2021; Schmitt et al., 2021), symbolic mathematics (Lample & Charton, 2020), or theorem proving (Loos et al., 2017; Bansal et al., 2019; Huang et al., 2019; Urban & Jakubuv, 2020).

The acquisition of training data for symbolic reasoning domains, however, is a challenge. Existing instances, such as benchmarks in competitions (Biere & Claessen, 2010; Froleyks et al., 2021; Jacobs et al., 2017) are typically hand-crafted, for example, in a “bring your own benchmarks” setting (Balyo et al., 2017). Since the instances are too few to be trained on, training data is, thus, typically generated synthetically. For example by random sampling (Selsam et al., 2019; Lample & Charton, 2020), or by randomly re-combining parts of existing instances (Schmitt et al., 2021). Although these data generation methods already lead to good results, training on randomly generated data carries the risk of training on meaningless data or the risk of introducing unwanted biases.

In this paper, we study the generation of symbolic reasoning problems with Generative Adversarial Networks (GANs) (Goodfellow et al., 2014) and show that they can be used to construct large amounts of meaningful training data from a significantly smaller data source. GANs, however, can not immediately be applied: Symbolic reasoning problems reside typically in a discontinuous domain and, additionally, training data is typically sequential and of variable length. We show that training directly in the one-hot encoding space is possible when adding Gaussian noise to each position. We, furthermore, use a Transformer (Vaswani et al., 2017) encoder to cope with the sequential form of the data and the variable length of the problem instances.

We provide experiments to show the usefulness of a GAN approach for the generation of reasoning problems. The experiments are based around two symbolic reasoning domains where recent studies on the applicability of deep learning relied on large amounts of artificially generated data: symbolic mathematics and linear-time temporal logic (LTL) specifications in verification. We report our experimental results in three sections. We first provide details on how to achieve a stable training of a standard GAN and a Wasserstein GAN (Arjovsky et al., 2017) both equipped with Transformer encoders. We analyze the particularities of their training behavior, such as the effects of adding different amounts of noise to the one hot embeddings. Secondly, we show for an LTL satisfiability classifier that the generated data can be used as a substitute for real training data, and, especially, that training data can be generated from a real dataset that is too small to be trained on directly. In particular, we show that out of 10K real training instances, a dataset consisting of 400K instances can be generated, on which a classifier can successfully be trained on. Lastly, we show that generating symbolic reasoning problems in a GAN setting has a specialty: We can alter the target distribution by adding a classifier uncertainty part to the generator objective. By doing this, we show that we can obtain a dataset that is even harder to solve than the original dataset which has been used to generate the data from.

The remainder of this paper is structured as follows. In Section 2, we give a short introduction to the problem domains considered in this paper and describe how the origin training data has been constructed. In Section 3, we present our Transformer GAN architecture(s), before providing experimental results in Section 4. We give an overview over related work in Section 5 before concluding in Section 6.

2 Problem domain and base datasets

In this section, we introduce the two problem domains on which we base our experiments on: satisfiability of temporal specifications for formal verification and function integration and ordinary differential equations (ODEs) for symbolic mathematics. We furthermore give an overview over the data generation processes of these base datasets.

2.1 Hardware Specifications in Linear-time Temporal Logic (LTL)

Linear-time Temporal Logic (LTL) (Pnueli, 1977) is the basis for industrial hardware specification languages like the IEEE standard PSL (IEEE-Commission et al., 2005). It is an extension of propositional logic with temporal modalities, such as the Next-operator ($\bigcirc$) and the Until-operator ($\mathcal{U}$). There also exist derived operators, such as "eventually" $\diamondsuit\varphi$ ($\equiv$ $true\,\mathcal{U}\,\varphi$) and "globally" $\square\varphi$ ($\equiv \neg\diamondsuit\neg\varphi$). For example, mutual exclusion can be expressed as the following specification: $(\neg\square(access_{p0} \wedge access_{p1})$, stating that processes $p0$ and $p1$ should have no access to a shared resource at the same time. The base problem of any logic is its satisfiability problem. It is the problem to decide whether there exists a solution to a given formula. The satisfiability problem of LTL is a hard problem, in fact, it is PSPACE-hard Sistla & Clarke (1982). The full syntax, semantics and additional information on the satisfiability problem can be found in Appendix A.

So far, the construction of datasets for LTL formulas has been done in two ways (Hahn et al., 2021): Either by obtaining LTL formulas from a fully random generation process, which likely results in unrealistic formulas, or by sampling conjunctions of LTL specification patterns (Dwyer et al., 1999). To obtain a healthy amount of unsatisfiable and satisfiable instances in this artificial generation process, we slightly refined the pattern-based generation method with two operations. Details can be found in Appendix B. Since the formula length correlates to unsatisfiability, we filter for equal proportions of classes per formula length. We restrict the tree size of the formulas to 50. We call this dataset `LTLbase`.

2.2 Symbolic Mathematics

Lample & Charton (2020) showed that Transformer models perform surprisingly well on symbolic mathematics. More precisely, they applied the models to function integration and ordinary differential equations (ODEs).

We consider the function integration problem and use the forward generated dataset (`https://github.com/facebookresearch/SymbolicMathematics`). Random functions with up to n operators are generated and their integrals are calculated with computer algebra systems. Functions that the system cannot integrate are discarded. Mathematical expressions are generated randomly. The dataset is cleaned, with equation simplification, coefficients simplification, and filtering out invalid expressions (Lample & Charton, 2020). We restrict the tree size to 50.

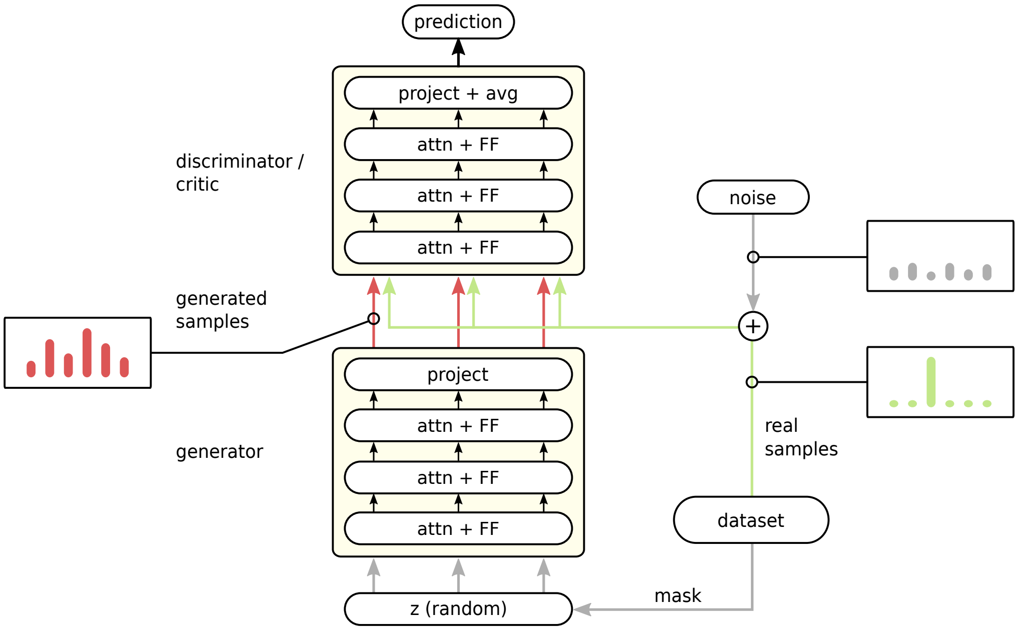


Figure 1: TGAN-SR: Transformer GAN for generating symbolic reasoning problems with visualizations of the per-position one-hot space.

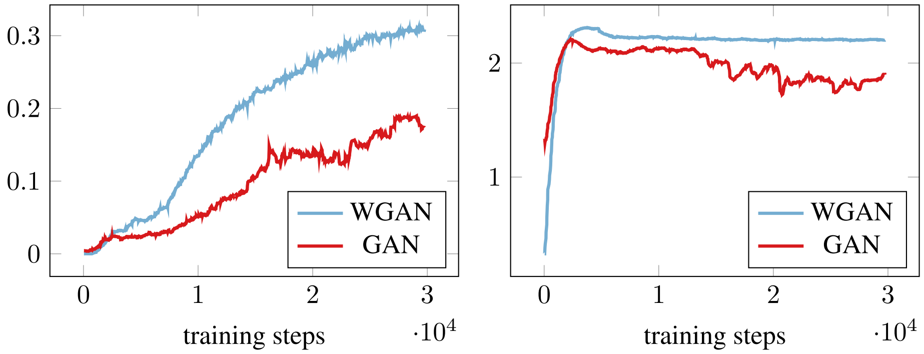
3 Architecture

The Transformer GAN architecture for generating symbolic reasoning problems (TGAN-SR) is depicted in Figure 1. It consists of two Transformer encoders as discriminator/critic and generator, respectively. The inner layers of the encoders are largely identical to standard transformers (Vaswani et al., 2017), but their input and output processing is adjusted to the GAN setting. We use an embedding dimension of $d_{emb} = 128$, $n_h = 8$ attention heads, and a feed-forward network dimension of $d_{FF} = 1024$ for both encoders as default.

The generator's input is a real scalar random value with uniform distribution $[0, 1]$ for each position in the sequence. It is mapped to d_{emb} by an affine transformation before being processed by the first layer. The position-wise padding mask is copied from the real data during training, so the lengths of real and generated formulas at the same position in a batch are always identical. During inference, the lengths can either be sampled randomly or copied from an existing dataset similar to training. Either way, the generator encoder's padding mask is predetermined so it has to adequately populate the unmasked positions. With V being the vocabulary, and $|V|$ being the size of the vocabulary, an affine transformation to dimensionality $|V|$ and a softmax is applied after the last layer. The generator's output lies, thus, in the same space as one-hot encoded tokens. We use $n_{lG} = 6$ layers for our default model's generator.

A GAN discriminator and WGAN critic are virtually identical in terms of their architecture. The only difference is that a critic outputs a real scalar value where a discriminator is limited to the range $[0, 1]$, which we achieve by applying an additional logistic sigmoid in the end. To honor their differences regarding the training scheme, we use both terms when referring to exchangeable properties and make no further distinctions between them. For input processing, their $|V|$-dimensional (per position) input is mapped to d_{emb} by an affine transformation. After the last layer, the final embeddings are aggregated over the sequence by averaging and a linear projection to a scalar value (the prediction logit) is applied. Our default model uses $n_{lD} = 4$ layers. We achieved best results with slightly more generator than discriminator/critic layers. A full hyperparameter study can be found in AppendixC.2.

Working in the $|V|$-sized one-hot domain poses harsh constraints on the generator's output. Contrary to continuous domains were GANs are usually employed, each component of a real one-hot vector is, by definition, either 0 or 1. If the generator were to identify this distribution and use it as criterion to



(a) fully correct fraction

(b) sequence entropy

Figure 2: Quality measures for the GAN and WGAN variant when generating temporal specifications.

tell real and generated instances apart, this would pose a serious difficulty for training. We therefore sample a $|V|$-sized vector of Gaussian noise $N(0, \sigma^2_{\text{real}})$ for each position (see Figure 1). We add it to the real samples' one-hot encoding and re-normalize it to sum 1 before handing them to the discriminator/critic. By default, we use a value of $\sigma_{\text{real}} = 0.1$ for all models to get comparable results. We study the effect of different amounts of noise more closely in Section 4.1.2.

4 Experiments

In this section, we report our experimental findings. We structure our results in three sections. We first report on the performance of the TGAN-SR architecture in constructing syntactically correct instances of temporal specifications and mathematical expressions. Secondly, we show, exemplary for LTL formulas, that the newly generated dataset can be used as a substitute for the origin dataset. Lastly, we show, by altering the target distribution, that the network can generate a dataset that is harder to solve for a classifier. We trained the models on an NVIDIA DGX A100 system for around 8 hours. We begin each subsection with a short preamble on the training setting.

4.1 Producing Syntactically Correct Symbolic Reasoning Problems

The goal of the experiments in this section is to asses the generator's capability in creating valid symbolic reasoning problems as objectively as possible. If not stated otherwise, in plots and tables, we report results from our default model averaged across three runs and with an exponential smoothing ($\alpha = 0.95$) applied. For temporal specifications, we use `LTLbase` as training set and for symbolic math the dataset described in section 2.2.

4.1.1 Training setting

For the GAN variant, we employ the standard GAN training algorithm (Goodfellow et al., 2014). For our default model, we use $n_c = 2$ discriminator training steps per generator training step and a batch size of $bs = 1024$. Notably, we use the alternative generator loss $-\mathbb{E}_{z\sim p_z}[\log D(G(z))]$ instead of the theoretically more sound $\mathbb{E}_{z\sim p_z}[\log(1 - D(G(z)))]$. The WGAN variant uses the WGAN-GP training with gradient penalty as proposed by Gulrajani et al. (2017) with $\lambda_{GP} = 10$. Standard WGAN losses are used and the training loop parameters n_c and bs are identical to the GAN variant. To calculate the gradient penalty of intermediate data points according to Gulrajani et al. (2017), we make use of the fact that for each batch element, real and generated samples share the padding mask. After the gradient with respect to an intermediate point is calculated, the gradient's squared components are masked out at padded positions before being summed up over the sequence length. For both variants, both discriminator and generator are trained with the Adam (Kingma & Ba, 2015) optimizer ($\beta_1 = 0, \beta_2 = 0.9$) and constant learning rate $lr = 1e-4$, similar to Gulrajani et al. (2017).

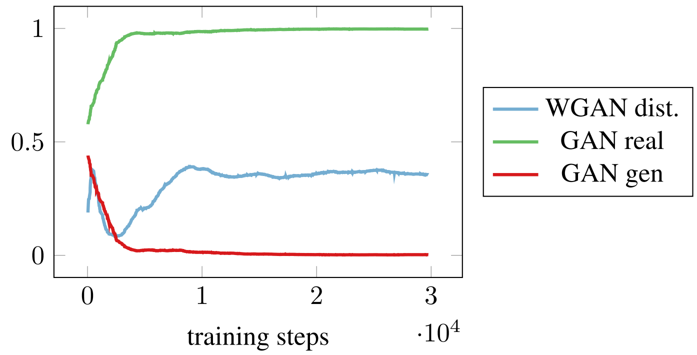


Figure 3: GAN real/generated predictions and WGAN Wasserstein distance estimate when generating temporal specifications.

4.1.2 Results

Generating valid symbolic reasoning problems. During training we periodically sample several generated instances and convert them to their text representation, which involves taking the *argmax* at every position. We then try to parse a prefix-encoded tree from the resulting tokens. If the parsing of a problem is successful and no tokens remain in the sequence, we note this problem as *fully correct*. The fraction over the course of training to generate temporal specifications is depicted in Figure 2a. Both GAN and WGAN variants increase the measure relatively continuously, but eventually reach their limit around 30K training steps. Still, both generators are able to produce a large fraction of fully correct temporal specifications, despite the length of the instances (up to 50 tokens) and the non-autoregressive, fixed-step architecture. We list some examples below:

$$\neg(h \rightarrow h)\,\mathcal{W}\bigcirc(g \vee h) \wedge (g \wedge g) \wedge \Diamond\square\neg\square j \wedge\neg\square j\,\mathcal{W}\neg b \wedge \square(\square h \wedge \bigcirc j \rightarrow \square j) \wedge \Diamond\Diamond j \quad ,$$

$$\begin{gathered}\bigcirc\bigcirc\bigcirc(c \vee i) \wedge \neg d \wedge \neg\Diamond\bigcirc c\,\mathcal{W}\neg c \wedge \square(\square d \wedge \neg((b \leftrightarrow c) \hookleftarrow \\ \leftrightarrow \square c) \rightarrow \bigcirc(c \leftrightarrow \square d)) \wedge \square(b \wedge d \rightarrow \Diamond\Diamond\square\square d) \wedge c \quad .\end{gathered}$$

The network also produces correct symbolic mathematical expressions when training on the forward generated mathematical dataset of Lample & Charton (2020). After 30K steps, on average 30% are fully correct. We list some examples below:

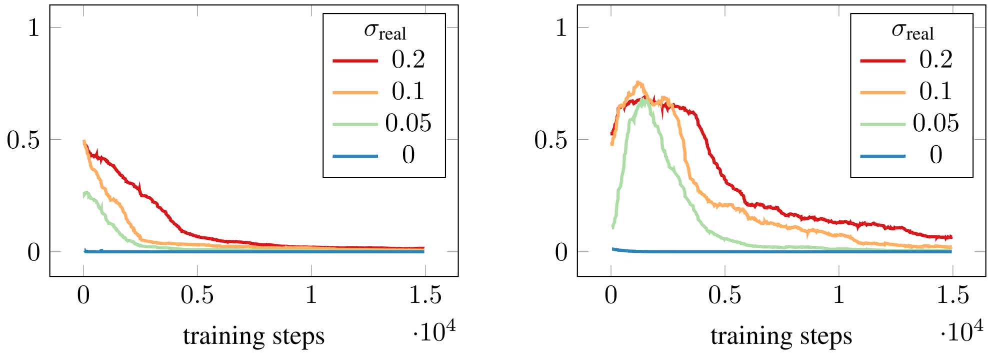
$$\begin{gathered}x^3 \cdot ((-1) \cdot (\ln x)^3 + 2 \cdot x \cdot (\text{acosh}\,(5) + 1 + (-1) \cdot x \cdot (2 + x)^{44})) \quad , \\ 1 \div 2 \cdot 81264 \div x \cdot 1 \div 5 \cdot x \cdot \ln 4 + 2 \quad , \\ x \cdot (3 \cdot (x^3) + x \cdot 2) + (2 + x) \cdot 4 \cdot x \cdot (1 \div 201 + \text{acos}(44)) \quad .\end{gathered}$$

Differences in homogeneity. Comparing the valid generated formulas from the WGAN and GAN variants, we find that often, the latter would produces formulas in the likes of

$$\begin{gathered}\bigcirc\bigcirc\bigcirc\bigcirc\bigcirc\bigcirc\bigcirc\bigcirc\bigcirc\bigcirc\Diamond i \wedge \bigcirc\bigcirc i \wedge \bigcirc\bigcirc\bigcirc\bigcirc\neg\bigcirc\square\neg\neg\Diamond i \wedge \neg\neg(g \wedge g \wedge i) \quad \text{or} \\ \tanh 12225556676677996655766669 \cdot x \quad ,\end{gathered}$$

which contains repetitions (of the $\bigcirc$-operator) or easily stringed together sequences (for example of numbers). In fact, some GAN runs achieved fully correct fractions above 30% (higher than WGAN), but these exclusively produced formulas with such low internal variety. To quantify this, we calculated a *sequence entropy* which treats the number of occurrences of the same token in the sequence relative to the sequences length as probability. Figure 2b shows that indeed this metric decreases for the GAN variant during training but remains stable for WGANs. We therefore speculate that the discriminator/critic indeed learns to check syntactic validity to some extend and some generators "exploit" this fact by producing correct, but repetitive formulas. For further experiments that use generated instances, we therefore exclusively stick to the WGAN variant.

Discriminator / critic predictions. We observe a quick identification of real and generated instances by the GAN discriminator as depicted in Figure 3. Predictions reach values above 0.99 and



(a) generated, original

(b) generated, arg-maxed

Figure 4: GAN discriminator predictions for generated samples with different noise level σ_{real} on real samples when generating temporal specifications.

below 0.01, respectively, and never change directions. Similarly, the WGAN critic's Wasserstein distance estimate soon reaches a value of around 0.4 at which it remains for the rest of training. For this behavior, one would expect the generator to not improve significantly, which is contrary to the observed improvements in quality.

Effects of additive noise on one-hot representation. We also studied the effect of adding different amounts of noise to the one-hot representation of real temporal specification instances (see Table 1). It strongly affects the performance of the GAN scheme, which is unable to work without added noise. Stronger noise however improves this variants performance. WGAN models on the other hand were not significantly influenced by added noise and are able to be trained without it.

Additionally, we compare how the GAN discriminator rates unmodified generated instances and $argmaxed$ versions thereof (see Figure 4). For this, we also evaluate $argmaxed$ instances during each step of training without changing the training regime. While the score for unmodified instances immediately decreases at the start of the training, it initially rises for the $argmaxed$ ones. After a while of training, though, the scores of the $argmaxed$ samples quickly deteriorate and, at least for lower values of σ_{real}, approach their soft-valued counterparts. A possible interpretation is that the discriminator first identifies generated samples by their different distributions in the one-hot domain, which, naturally, is eased with low noise on the real samples, before shifting its focus from this low-level criterion to more global features.

4.2 Substituting Training Data with Generated Instances

In this subsection, we show that the origin training data can be substituted with generated training data when training a classifier on the LTL satisfiability problem.

Table 1: Comparison on fully correct formulas (fc) and sequence entropy (se) of GAN and WGAN with different σ_{real} when generating temporal specifications. 3-run average, smoothed ($\alpha = 0.95$), standard deviations in Table 5.

architecture	σ_{real}	fc	se
GAN	0	0%	-
	0.05	15%	1.7
	0.1	17%	1.8
	0.2	41%	1.9
	0.4	11%	2.2

architecture	σ_{real}	fc	se
WGAN	0	26%	2.2
	0.05	25%	2.2
	0.1	31%	2.2
	0.2	25%	2.2
	0.4	3%	2.4

4.2.1 Training setting

Binary classifier. We use a classifier that is similar to the GAN discriminator, consisting of a Transformer encoder followed by an averaging aggregation and linear transformation to a scalar output value. Finally, a logistic sigmoid is applied to obtain a prediction for the formula's satisfiability. The classification loss is a standard cross-entropy between real labels and predictions. Similar to the GAN discriminator, we use $n_l = 4$ layers and a batch size of $bs = 1024$. Contrary to the GAN training scheme, we use the default Transformer scheme Vaswani et al. (2017) with varying learning rate and 4000 warmup steps as well as the Adam optimizer Kingma & Ba (2015) with parameters $\beta_1 = 0.9, \beta_2 = 0.98$. This training scheme resulted in a faster improvement and higher final accuracy than adopting the settings from GAN training. We trained the classifier for 30K steps.

Generated dataset. To obtain a dataset of generated instances, we first train a WGAN with default parameters but smaller batch size of 512 on a set of 10K instances from the `LTLbase` dataset. After training for 15K steps, we collect 800K generated formulas from it and call this dataset `Generated-raw`. This set is processed similar to the original base dataset: Duplicates are removed and satisfiable and unsatisfiable instances are balanced to equal amounts per formula size. We randomly keep 400K instances and call the resulting dataset `Generated`.

4.2.2 Results

We compare the performance of similar classifiers on different training sets in Table 2. The training curves can be found in Appendix C.3. The validation accuracy is computed on the `LTLbase` dataset. Training on differently-sized subsets of `LTLbase` shows that a reduced number of training samples strongly decreases performance. 10K instances lead to immense overfitting and poor accuracy. We were not able to train a classifier on this few formulas with significantly higher accuracy.

A classifier trained on the `Generated` set however achieves almost the identical validation accuracy on the base set as the classifier that was actually trained on it. Note that the GAN that created this set was trained on only 10K instances. We therefore find that the data produced by the TGAN-SR is highly valuable as it can serve as full substitute for the complete original training data even when provided with much fewer examples.

Two instances of `LTLbase` ($(\Box \neg a) \land (\Box \bigcirc a)$ and $(\Box \Box e) \land (\Box \neg e)$), i.e. only 0.02%, reappear in the 800K large data set `Generated-raw`. Additionally, in `Generated-raw`, only 2.3K of the 800K (0.28%) generated formulas were duplicates, which displays an enormous degree of variety.

4.3 Uncertainty Objective for Generating Harder-to-classify Instances

In this experiment, we show that, by adding an uncertainty measure to a simultaneously trained classifier, the model generates instances of temporal specifications in LTL that are harder to classify. We train a model on the `LTLbase` dataset to jointly learn to imitate its formulas and classify them as satisfiable or unsatisfiable.

4.3.1 Training Setting

GAN with included classifier. For this experiment, we combine both critic and LTL satisfiability classifier into one Transformer encoder with two outputs and train them simultaneously. Both parts

Table 2: Accuracies of Transformer classifiers trained on different datasets (5-run average with standard deviations in parentheses); all are validated on the `LTLbase` dataset.

trained on	bs	train acc @ 30K	val acc @ 30K	train acc @ 50K	val acc @ 50K
`LTLbase`	1024	96.6% (0.5)	95.5% (0.4)	98.1% (0.3)	**96.1%** (0.3)
	512	92.4% (0.7)	93.0% (0.8)	95.4% (0.5)	95.0% (0.8)
`LTLbase` 100K	512	95.3% (0.7)	88.3% (0.9)	98.1% (0.3)	87.8% (1.0)
`LTLbase` 10K	512	100% (0.1)	76.4% (1.7)	100% (0.0)	75.5% (1.5)
`Generated`	1024	95.4% (0.2)	93.6% (1.0)	97.1% (0.1)	**93.9%** (0.3)

share the three lower layers of the encoder but have separate fourth layers. We found this to improve both classification accuracy and GAN performance slightly compared to sharing all layers (the linear projection layer is never shared). A comparision can be found in Section C.1 in the appendix. We stick to the WGAN training scheme from Section 4.1.1 including the optimizer settings, but add an additional classification loss term similar to Section 4.2.1. The classification loss is added to the GAN critic loss and scaled by coefficient α_{class}, which we set to 10 when training a WGAN. The resulting model achieves similar generative performance to the pure WGAN but is limited to a classification accuracy of around 92%.

Classifier uncertainty. We calculate the entropy of a class prediction s of the classifier as $H(s) = -s \cdot \log(s) - (1 - s) \cdot \log(1 - s)$, as a measure of uncertainty on a particular instance. We add a term $-\alpha_{\text{unct}} H(s)$ to the generator's loss function, which leads to the uncertainty measure being propagated back through the critic just like the standard GAN objective. $H(s)$ is maximized at $s = 0.5$ (with value $\log 2$), so the generator is encouraged to produce instances which "confuse" the classifier included in the critic. Naturally, this conflicts with the original GAN objective, so they must be carefully balanced. As default, we chose $\alpha_{\text{conf}} = 2$. Since GAN training is hindered by adding the uncertainty objective, we only apply it after pre-training for 30K steps with default WGAN and classification objectives. We then train for additional 15K steps with the uncertainty objective included. This decreases the fraction of fully correct formulas to around 10%; sequence entropy as classification accuracy remain unaffected. From the fully trained model, we obtain a dataset similar to Section 4.2.1 and call it `Uncert-e`. Additionally, we construct a dataset of 200K formulas from this set and 200K from `LTLbase` and call it `Mixed-e`.

Alternative uncertainty objective. The entropy becomes unhandy to compute for values close to 0 and 1. We therefore explore a pragmatic alternative measure for (un)certainty: the absolute value of the classification logit. Values close to zero lead to predictions around 0.5. We therefore add a generator loss of the form $\alpha_{\text{unct}}|l|$ for this variant (with l the classification logit; $s = \sigma(l)$) and use a value of $\alpha_{\text{conf}} = 0.5$ in this case. The model is trained similar to the entropy variant and behaves very similarly. We call the dataset obtained from this model `Uncert-a` and also construct a mixed set `Mixed-a`.

4.3.2 RESULTS

We compare the accuracy of classifiers trained similar to Section 4.2 (pure classifiers with optimized training schedule, *not* included GAN classifiers) on different (generated) datasets in Table 3. The classifier trained on `LTLbase` serves as reference again with 94.5% accuracy. Training on the `Uncert` sets however allows the classifier to achieve only 91% and 90.5% accuracy (for entropy and absolute variants, respectively). Also when trained longer than 30K steps, there is no significant improvement.

The datasets produced by WGANs with added uncertainty objective are indeed harder to classify than the original dataset `LTLbase`. To validate this, we also trained classifiers on `Mixed` sets and find that they also achieve 4.5 percent points higher accuracy when tested on the base set compared to the generated sets. Additionally, the performance on the original dataset is never deteriorated and even slightly higher when training on the mixed set. This approach is especially useful in the domain of symbolic reasoning, because data can, in contrast to archetypal deep learning domains, often be labeled automatically (e.g. with classical tools and algorithms). This underpins the usefulness of a GAN setting when generating new training instances for symbolic reasoning problems.

Table 3: Performance of classifiers trained and tested on datasets generated with uncertainty objectives; 30K steps, 5-run average with standard deviations, *not* smoothed.

trained on	tested on	accuracy	trained on	tested on	accuracy
`LTLbase`	`LTLbase`	94.8% (0.3)			
`Uncert-e`	`Uncert-e`	91.0% (0.5)	`Uncert-a`	`Uncert-a`	90.5% (0.5)
`Mixed-e`	`Uncert-e`	**90.2%** (0.9)	`Mixed-a`	`Uncert-a`	89.6% (0.4)
`Mixed-e`	`LTLbase`	**95.3%** (0.4)	`Mixed-a`	`LTLbase`	94.1% (0.4)

5 Related work

GANs. Generative Adversarial Networks have been applied to discrete domains especially for text generation in a reinforcement learning setting (Chen et al., 2018; Yu et al., 2017; Che et al., 2017; Lin et al., 2017; Fedus et al., 2018; Guo et al., 2018) or by using a Gumbel softmax (Kusner & Hernández-Lobato, 2016; Zhang et al.). Kumar & Tsvetkov (2020) use a continuous, pre-trained embedding. Gulrajani et al. (2017) showed that it is possible to directly use a soft one-hot representation without any sampling. Close related work is Huang et al. (2020) and Zeng et al. (2020) for adversarial text generation. They also combine Transformers and in an adversarial learning setting, where the former rely on Gumbel softmax tricks and the latter extract a style code from reference examples. Transformers and GANs have also been combined in the domain of computer vision (Vondrick & Torralba, 2017; Jiang et al., 2021; Hudson & Zitnick, 2021). GANs have been used for data augmentation, especially for images, e.g., (Antoniou et al., 2018; Bowles et al., 2018).

Temporal logics. Temporal logics have been studied in computer science since their introduction by Pnueli (1977). Since then, many extensions have been developed: e.g., computation tree logic CTL and CTL* (Clarke & Emerson, 1981; Emerson & Halpern, 1986), signal temporal logic STL (Maler & Nickovic, 2004), or temporal logics for hyperproperties, e.g., HyperLTL, (Clarkson et al., 2014). Verification methods for temporal logics have been studied extensively over the years, e.g., LTL satisfiability (Li et al., 2013; Rozier & Vardi, 2007; Schuppan & Darmawan, 2011; Li et al., 2013; 2014; Schwendimann, 1998), LTL synthesis (Finkbeiner & Schewe, 2005; 2013; Bohy et al., 2012; Faymonville et al., 2017; Meyer et al., 2018), model checking (Clarke et al., 1986), or monitoring (Clarke et al., 2001; Bauer et al., 2011; Finkbeiner & Sipma, 2004; Donzé et al., 2013).

Mathematical reasoning in machine learning. Other works have studied datasets derived from automated theorem provers (Blanchette et al., 2016; Loos et al., 2017; Gauthier et al., 2021), interactive theorem provers (Irving et al., 2016; Kaliszyk et al., 2017; Bansal et al., 2019; Huang et al., 2019; Yang & Deng, 2019; Polu & Sutskever, 2020; Wu et al., 2021b; Li et al., 2020; Lee et al., 2020; Urban & Jakubuv, 2020; Rabe et al., 2021; Paliwal et al., 2020; Rabe & Szegedy, 2021), symbolic mathematics (Lample & Charton, 2020; Zaremba et al., 2014; Allamanis et al., 2017; Arabshahi et al., 2018), and mathematical problems in natural language (Saxton et al., 2019; Schlag et al., 2019). Learning has been applied to mathematics long before the rise of deep learning. Earlier works focused on ranking premises or clauses (Cairns, 2004; Urban, 2004; 2007; Urban et al., 2008; Meng & Paulson, 2009; Schulz, 2013; Kaliszyk & Urban, 2014).

Neural architectures for logical reasoning. Wu et al. (2021a) present a reinforcement learning approach for interactive theorem proving. NeuroSAT (Selsam et al., 2019) is a graph neural network (Scarselli et al., 2009; Li et al., 2018; Gilmer et al., 2017; Wu et al., 2021c) for solving the propositional satisfiability problem. A simplified NeuroSAT architecture was trained for unsat-core predictions (Selsam & Bjørner, 2019). Neural networks have been applied to 2QBF (Lederman et al., 2020), logical entailment (Evans et al., 2018), SMT (Balunovic et al., 2018), and temporal logics (Hahn et al., 2021; Schmitt et al., 2021).

6 Conclusion

We studied the capabilities of (Wasserstein) GANs equipped with two Transformer encoders to generate sensible training data for symbolic reasoning problems. We showed that both can be trained directly on the one-hot encoding space when adding Gaussian noise. We exemplary conducted experiments in the domain of symbolic mathematics and hardware specifications in temporal logics. We showed that training data can indeed be generated and that the data can be used as a meaningful substitute when training a classifier. Furthermore, we showed that a GAN setting has a speciality: by adding an uncertainty measure to the generator's output, the models generated instances on which a classifier was harder to train on. In general, logical and mathematical reasoning with neural networks requires large amounts of sensible training data. Better datasets will lead to powerful neural heuristics and end-to-end approaches for many symbolic application domains, such as mathematics, search, verification, synthesis and computer-aided design. This novel, neural perspective on the generation of symbolic reasoning instances is also of interest to generate data for tool competitions, such as SAT, SMT, or model checking competitions.

REFERENCES

Miltiadis Allamanis, Pankajan Chanthirasegaran, Pushmeet Kohli, and Charles Sutton. Learning continuous semantic representations of symbolic expressions. In Doina Precup and Yee Whye Teh (eds.), *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pp. 80–88. PMLR, 2017. URL `http://proceedings.mlr.press/v70/allamanis17a.html`.

Antreas Antoniou, Amos J. Storkey, and Harrison Edwards. Augmenting image classifiers using data augmentation generative adversarial networks. In Vera Kurková, Yannis Manolopoulos, Barbara Hammer, Lazaros S. Iliadis, and Ilias Maglogiannis (eds.), *Artificial Neural Networks and Machine Learning - ICANN 2018 - 27th International Conference on Artificial Neural Networks, Rhodes, Greece, October 4-7, 2018, Proceedings, Part III*, volume 11141 of *Lecture Notes in Computer Science*, pp. 594–603. Springer, 2018. doi: 10.1007/978-3-030-01424-7_58. URL `https://doi.org/10.1007/978-3-030-01424-7_58`.

Forough Arabshahi, Sameer Singh, and Animashree Anandkumar. Towards solving differential equations through neural programming. In *ICML Workshop on Neural Abstract Machines and Program Induction (NAMPI)*, 2018.

Martín Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein GAN. *CoRR*, abs/1701.07875, 2017. URL `http://arxiv.org/abs/1701.07875`.

Mislav Balunovic, Pavol Bielik, and Martin T. Vechev. Learning to solve SMT formulas. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pp. 10338–10349, 2018. URL `https://proceedings.neurips.cc/paper/2018/hash/68331ff0427b551b68e911eebe35233b-Abstract.html`.

Tomáš Balyo, Marijn J.H. Heule, and Matti Järvisalo. *Proceedings of SAT Competition 2017: Solver and Benchmark Descriptions*, volume B-2017-1 of *Series of Publications B*. Department of Computer Science, University of Helsinki, Finland, 2017.

Kshitij Bansal, Sarah M. Loos, Markus N. Rabe, Christian Szegedy, and Stewart Wilcox. Holist: An environment for machine learning of higher order logic theorem proving. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pp. 454–463. PMLR, 2019. URL `http://proceedings.mlr.press/v97/bansal19a.html`.

Andreas Bauer, Martin Leucker, and Christian Schallhart. Runtime verification for LTL and TLTL. *ACM Trans. Softw. Eng. Methodol.*, 20(4):14:1–14:64, 2011. doi: 10.1145/2000799.2000800. URL `https://doi.org/10.1145/2000799.2000800`.

Armin Biere and K Claessen. Hardware model checking competition. In *Hardware Verification Workshop*, 2010.

Jasmin Christian Blanchette, Cezary Kaliszyk, Lawrence C. Paulson, and Josef Urban. Hammering towards QED. *J. Formaliz. Reason.*, 9(1):101–148, 2016. doi: 10.6092/issn.1972-5787/4593. URL `https://doi.org/10.6092/issn.1972-5787/4593`.

Aaron Bohy, Véronique Bruyère, Emmanuel Filiot, Naiyong Jin, and Jean-François Raskin. Acacia+, a tool for LTL synthesis. In *Computer Aided Verification - 24th International Conference, CAV 2012, Berkeley, CA, USA, July 7-13, 2012 Proceedings*, volume 7358 of *Lecture Notes in Computer Science*, pp. 652–657. Springer, 2012. doi: 10.1007/978-3-642-31424-7_45. URL `https://doi.org/10.1007/978-3-642-31424-7_45`.

Christopher Bowles, Liang Chen, Ricardo Guerrero, Paul Bentley, Roger N. Gunn, Alexander Hammers, David Alexander Dickie, Maria del C. Valdés Hernández, Joanna M. Wardlaw, and Daniel Rueckert. GAN augmentation: Augmenting training data using generative adversarial networks. *CoRR*, abs/1810.10863, 2018. URL `http://arxiv.org/abs/1810.10863`.

Paul A. Cairns. Informalising formal mathematics: Searching the mizar library with latent semantics. In *Mathematical Knowledge Management, Third International Conference, MKM 2004, Bialowieza, Poland, September 19-21, 2004, Proceedings*, volume 3119 of *Lecture Notes in Computer Science*, pp. 58–72. Springer, 2004. doi: 10.1007/978-3-540-27818-4_5. URL https://doi.org/10.1007/978-3-540-27818-4_5.

Tong Che, Yanran Li, Ruixiang Zhang, R. Devon Hjelm, Wenjie Li, Yangqiu Song, and Yoshua Bengio. Maximum-likelihood augmented discrete generative adversarial networks. *CoRR*, abs/1702.07983, 2017. URL http://arxiv.org/abs/1702.07983.

Liqun Chen, Shuyang Dai, Chenyang Tao, Haichao Zhang, Zhe Gan, Dinghan Shen, Yizhe Zhang, Guoyin Wang, Ruiyi Zhang, and Lawrence Carin. Adversarial text generation via feature-mover's distance. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/file/074177d3eb6371e32c16c55a3b8f706b-Paper.pdf.

Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Logics of Programs, Workshop, Yorktown Heights, New York, USA, May 1981*, volume 131 of *Lecture Notes in Computer Science*, pp. 52–71. Springer, 1981. doi: 10.1007/BFb0025774. URL https://doi.org/10.1007/BFb0025774.

Edmund M. Clarke, E. Allen Emerson, and A. Prasad Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Program. Lang. Syst.*, 8 (2):244–263, 1986. doi: 10.1145/5397.5399. URL https://doi.org/10.1145/5397.5399.

Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model checking*. MIT Press, 2001. ISBN 978-0-262-03270-4. URL http://books.google.de/books?id=Nmc4wEaLXFEC.

Michael R. Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher K. Micinski, Markus N. Rabe, and César Sánchez. Temporal logics for hyperproperties. In Martín Abadi and Steve Kremer (eds.), *Principles of Security and Trust - Third International Conference, POST 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014, Proceedings*, volume 8414 of *Lecture Notes in Computer Science*, pp. 265–284. Springer, 2014. doi: 10.1007/978-3-642-54792-8_15. URL https://doi.org/10.1007/978-3-642-54792-8_15.

Alexandre Donzé, Thomas Ferrère, and Oded Maler. Efficient robust monitoring for STL. In Natasha Sharygina and Helmut Veith (eds.), *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, volume 8044 of *Lecture Notes in Computer Science*, pp. 264–279. Springer, 2013. doi: 10.1007/978-3-642-39799-8_19. URL https://doi.org/10.1007/978-3-642-39799-8_19.

Matthew B. Dwyer, George S. Avrunin, and James C. Corbett. Patterns in property specifications for finite-state verification. In *Proceedings of the 21st International Conference on Software Engineering*, ICSE '99, pp. 411420, New York, NY, USA, 1999. Association for Computing Machinery. ISBN 1581130740. doi: 10.1145/302405.302672. URL https://doi.org/10.1145/302405.302672. ICSE '99.

E. Allen Emerson and Joseph Y. Halpern. "sometimes" and "not never" revisited: on branching versus linear time temporal logic. *J. ACM*, 33(1):151–178, 1986. doi: 10.1145/4904.4999. URL https://doi.org/10.1145/4904.4999.

Richard Evans, David Saxton, David Amos, Pushmeet Kohli, and Edward Grefenstette. Can neural networks understand logical entailment? In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL https://openreview.net/forum?id=SkZxCk-0Z.

Peter Faymonville, Bernd Finkbeiner, and Leander Tentrup. BoSy: An experimentation framework for bounded synthesis. In *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part II*, volume 10427 of *Lecture Notes in Computer Science*, pp. 325–332. Springer, 2017. doi: 10.1007/978-3-319-63390-9_17.

William Fedus, Ian J. Goodfellow, and Andrew M. Dai. Maskgan: Better text generation via filling in the _______. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL https://openreview.net/forum?id=ByOExmWAb.

Bernd Finkbeiner and Sven Schewe. Uniform distributed synthesis. In *20th IEEE Symposium on Logic in Computer Science (LICS 2005), 26-29 June 2005, Chicago, IL, USA, Proceedings*, pp. 321–330. IEEE Computer Society, 2005. doi: 10.1109/LICS.2005.53. URL https://doi.org/10.1109/LICS.2005.53.

Bernd Finkbeiner and Sven Schewe. Bounded synthesis. *Int. J. Softw. Tools Technol. Transf.*, 15(5-6):519–539, 2013. doi: 10.1007/s10009-012-0228-z. URL https://doi.org/10.1007/s10009-012-0228-z.

Bernd Finkbeiner and Henny Sipma. Checking finite traces using alternating automata. *Formal Methods Syst. Des.*, 24(2):101–127, 2004. doi: 10.1023/B:FORM.0000017718.28096.48. URL https://doi.org/10.1023/B:FORM.0000017718.28096.48.

Nils Froleyks, Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda. Sat competition 2020. *Artificial Intelligence*, 301:103572, 2021.

Thibault Gauthier, Cezary Kaliszyk, Josef Urban, Ramana Kumar, and Michael Norrish. Tactictoe: Learning to prove with tactics. *J. Autom. Reason.*, 65(2):257–286, 2021. doi: 10.1007/s10817-020-09580-x. URL https://doi.org/10.1007/s10817-020-09580-x.

Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pp. 1263–1272. PMLR, 2017. URL http://proceedings.mlr.press/v70/gilmer17a.html.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger (eds.), *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014. URL https://proceedings.neurips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf.

Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/892c3b1c6dccd52936e27cbd0ff683d6-Paper.pdf. NIPS '17.

Jiaxian Guo, Sidi Lu, Han Cai, Weinan Zhang, Yong Yu, and Jun Wang. Long text generation via adversarial training with leaked information. In Sheila A. McIlraith and Kilian Q. Weinberger (eds.), *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pp. 5141–5148. AAAI Press, 2018. URL https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16360.

Christopher Hahn, Frederik Schmitt, Jens U. Kreber, Markus Norman Rabe, and Bernd Finkbeiner. 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. URL https://openreview.net/forum?id=dOcQK-f4byz.

Daniel Huang, Prafulla Dhariwal, Dawn Song, and Ilya Sutskever. Gamepad: A learning environment for theorem proving. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=r1xwKoR9Y7.

Fei Huang, Jian Guan, Pei Ke, Qihan Guo, Xiaoyan Zhu, and Minlie Huang. A text gan for language generation with non-autoregressive generator. 2020.

Drew A Hudson and C Lawrence Zitnick. Generative adversarial transformers. *arXiv preprint arXiv:2103.01209*, 2021.

IEEE-Commission et al. Ieee standard for property specification language (psl). *IEEE Std 1850-2005*, 2005.

Geoffrey Irving, Christian Szegedy, Alexander A. Alemi, Niklas Eén, François Chollet, and Josef Urban. Deepmath - deep sequence models for premise selection. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pp. 2235–2243, 2016. URL `https://proceedings.neurips.cc/paper/2016/hash/f197002b9a0853eca5e046d9ca4663d5-Abstract.html`.

Swen Jacobs, Roderick Bloem, Romain Brenguier, Rüdiger Ehlers, Timotheus Hell, Robert Könighofer, Guillermo A. Pérez, Jean-François Raskin, Leonid Ryzhyk, Ocan Sankur, Martina Seidl, Leander Tentrup, and Adam Walker. The first reactive synthesis competition (SYNT-COMP 2014). *Int. J. Softw. Tools Technol. Transf.*, 19(3):367–390, 2017. doi: 10.1007/s10009-016-0416-3. URL `https://doi.org/10.1007/s10009-016-0416-3`.

Yifan Jiang, Shiyu Chang, and Zhangyang Wang. Transgan: Two transformers can make one strong gan. *arXiv preprint arXiv:2102.07074*, 2021.

Cezary Kaliszyk and Josef Urban. Learning-assisted automated reasoning with flyspeck. *J. Autom. Reason.*, 53(2):173–213, 2014. doi: 10.1007/s10817-014-9303-3. URL `https://doi.org/10.1007/s10817-014-9303-3`.

Cezary Kaliszyk, François Chollet, and Christian Szegedy. Holstep: A machine learning dataset for higher-order logic theorem proving. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL `https://openreview.net/forum?id=ryuxYmvel`.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2015.

Sachin Kumar and Yulia Tsvetkov. End-to-end differentiable GANs for text generation. In *Proceedings on "I Can't Believe It's Not Better!" at NeurIPS Workshops*, volume 137 of *Proceedings of Machine Learning Research*, pp. 118–128. PMLR, 12 Dec 2020. URL `http://proceedings.mlr.press/v137/kumar20a.html`.

Matt J. Kusner and José Miguel Hernández-Lobato. Gans for sequences of discrete elements with the gumbel-softmax distribution. *ArXiv*, abs/1611.04051, 2016.

Guillaume Lample and François Charton. Deep learning for symbolic mathematics. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL `https://openreview.net/forum?id=S1eZYeHFDS`.

Gil Lederman, Markus N. Rabe, Sanjit Seshia, and Edward A. Lee. Learning heuristics for quantified boolean formulas through reinforcement learning. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL `https://openreview.net/forum?id=BJluxREKDB`.

Dennis Lee, Christian Szegedy, Markus N. Rabe, Sarah M. Loos, and Kshitij Bansal. Mathematical reasoning in latent space. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL `https://openreview.net/forum?id=Ske31kBtPr`.

Jianwen Li, Lijun Zhang, Geguang Pu, Moshe Y. Vardi, and Jifeng He. LTL satisfiability checking revisited. In *2013 20th International Symposium on Temporal Representation and Reasoning, Pensacola, FL, USA, September 26-28, 2013*, pp. 91–98. IEEE Computer Society, 2013. doi: 10.1109/TIME.2013.19. URL `https://doi.org/10.1109/TIME.2013.19`.

Jianwen Li, Yinbo Yao, Geguang Pu, Lijun Zhang, and Jifeng He. Aalta: an LTL satisfiability checker over infinite/finite traces. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, (FSE-22), Hong Kong, China, November 16 - 22, 2014*, pp. 731–734. ACM, 2014. doi: 10.1145/2635868.2661669. URL https://doi.org/10.1145/2635868.2661669.

Wenda Li, Lei Yu, Yuhuai Wu, and Lawrence C. Paulson. Modelling high-level mathematical reasoning in mechanised declarative proofs. *CoRR*, abs/2006.09265, 2020. URL https://arxiv.org/abs/2006.09265.

Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL https://openreview.net/forum?id=SJiHXGWAZ.

Kevin Lin, Dianqi Li, Xiaodong He, Ming-Ting Sun, and Zhengyou Zhang. Adversarial ranking for language generation. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pp. 3155–3165, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/bf201d5407a6509fa536afc4b380577e-Abstract.html.

Sarah M. Loos, Geoffrey Irving, Christian Szegedy, and Cezary Kaliszyk. Deep network guided proof search. In *LPAR-21, 21st International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Maun, Botswana, May 7-12, 2017*, volume 46 of *EPiC Series in Computing*, pp. 85–105. EasyChair, 2017. URL https://easychair.org/publications/paper/ND13.

Oded Maler and Dejan Nickovic. Monitoring temporal properties of continuous signals. In *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems, Joint International Conferences on Formal Modelling and Analysis of Timed Systems, FORMATS 2004 and Formal Techniques in Real-Time and Fault-Tolerant Systems, FTRTFT 2004, Grenoble, France, September 22-24, 2004, Proceedings*, volume 3253 of *Lecture Notes in Computer Science*, pp. 152–166. Springer, 2004. doi: 10.1007/978-3-540-30206-3_12. URL https://doi.org/10.1007/978-3-540-30206-3_12.

Jia Meng and Lawrence C. Paulson. Lightweight relevance filtering for machine-generated resolution problems. *J. Appl. Log.*, 7(1):41–57, 2009. doi: 10.1016/j.jal.2007.07.004. URL https://doi.org/10.1016/j.jal.2007.07.004.

Philipp J. Meyer, Salomon Sickert, and Michael Luttenberger. Strix: Explicit reactive synthesis strikes back! In *Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part I*, volume 10981 of *Lecture Notes in Computer Science*, pp. 578–586. Springer, 2018. doi: 10.1007/978-3-319-96145-3_31.

Emils Ozolins, Karlis Freivalds, Andis Draguns, Eliza Gaile, Ronalds Zakovskis, and Sergejs Kozlovics. Goal-aware neural SAT solver. *CoRR*, abs/2106.07162, 2021. URL https://arxiv.org/abs/2106.07162.

Aditya Paliwal, Sarah M. Loos, Markus N. Rabe, Kshitij Bansal, and Christian Szegedy. Graph representations for higher-order logic and theorem proving. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pp. 2967–2974. AAAI Press, 2020. URL https://aaai.org/ojs/index.php/AAAI/article/view/5689.

Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, pp. 46–57. IEEE Computer Society, 1977. doi: 10.1109/SFCS.1977.32. URL https://doi.org/10.1109/SFCS.1977.32.

Stanislas Polu and Ilya Sutskever. Generative language modeling for automated theorem proving. *CoRR*, abs/2009.03393, 2020. URL https://arxiv.org/abs/2009.03393.

Markus N. Rabe and Christian Szegedy. Towards the automatic mathematician. In *Automated Deduction - CADE 28 - 28th International Conference on Automated Deduction, Virtual Event, July 12-15, 2021, Proceedings*, volume 12699 of *Lecture Notes in Computer Science*, pp. 25–37. Springer, 2021. doi: 10.1007/978-3-030-79876-5_2. URL https://doi.org/10.1007/978-3-030-79876-5_2.

Markus Norman Rabe, Dennis Lee, Kshitij Bansal, and Christian Szegedy. Mathematical reasoning via self-supervised skip-tree training. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=YmqAnY0CMEy.

Kristin Y. Rozier and Moshe Y. Vardi. LTL satisfiability checking. In *Model Checking Software, 14th International SPIN Workshop, Berlin, Germany, July 1-3, 2007, Proceedings*, volume 4595 of *Lecture Notes in Computer Science*, pp. 149–167. Springer, 2007. doi: 10.1007/978-3-540-73370-6_11. URL https://doi.org/10.1007/978-3-540-73370-6_11.

David Saxton, Edward Grefenstette, Felix Hill, and Pushmeet Kohli. Analysing mathematical reasoning abilities of neural models. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=H1gR5iR5FX.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Trans. Neural Networks*, 20(1):61–80, 2009. doi: 10.1109/TNN.2008.2005605.

Imanol Schlag, Paul Smolensky, Roland Fernandez, Nebojsa Jojic, Jürgen Schmidhuber, and Jianfeng Gao. Enhancing the transformer with explicit relational encoding for math problem solving. *CoRR*, abs/1910.06611, 2019. URL http://arxiv.org/abs/1910.06611.

Frederik Schmitt, Christopher Hahn, Markus N Rabe, and Bernd Finkbeiner. Neural circuit synthesis from specification patterns. *arXiv preprint arXiv:2107.11864*, 2021.

Stephan Schulz. System description: E 1.8. In *Logic for Programming, Artificial Intelligence, and Reasoning - 19th International Conference, LPAR-19, Stellenbosch, South Africa, December 14-19, 2013. Proceedings*, volume 8312 of *Lecture Notes in Computer Science*, pp. 735–743. Springer, 2013. doi: 10.1007/978-3-642-45221-5_49. URL https://doi.org/10.1007/978-3-642-45221-5_49.

Viktor Schuppan and Luthfi Darmawan. Evaluating LTL satisfiability solvers. In *Automated Technology for Verification and Analysis, 9th International Symposium, ATVA 2011, Taipei, Taiwan, October 11-14, 2011. Proceedings*, volume 6996 of *Lecture Notes in Computer Science*, pp. 397–413. Springer, 2011. doi: 10.1007/978-3-642-24372-1_28. URL https://doi.org/10.1007/978-3-642-24372-1_28.

Stefan Schwendimann. A new one-pass tableau calculus for PLTL. In *Automated Reasoning with Analytic Tableaux and Related Methods, International Conference, TABLEAUX '98, Oisterwijk, The Netherlands, May 5-8, 1998, Proceedings*, volume 1397 of *Lecture Notes in Computer Science*, pp. 277–292. Springer, 1998. doi: 10.1007/3-540-69778-0_28. URL https://doi.org/10.1007/3-540-69778-0_28.

Daniel Selsam and Nikolaj Bjørner. Guiding high-performance SAT solvers with unsat-core predictions. In *Theory and Applications of Satisfiability Testing - SAT 2019 - 22nd International Conference, SAT 2019, Lisbon, Portugal, July 9-12, 2019, Proceedings*, volume 11628 of *Lecture Notes in Computer Science*, pp. 336–353. Springer, 2019. doi: 10.1007/978-3-030-24258-9_24. URL https://doi.org/10.1007/978-3-030-24258-9_24.

Daniel Selsam, Matthew Lamm, Benedikt Bünz, Percy Liang, Leonardo de Moura, and David L. Dill. Learning a SAT solver from single-bit supervision. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=HJMC_iA5tm.

A. Prasad Sistla and Edmund M. Clarke. The complexity of propositional linear temporal logics. In Harry R. Lewis, Barbara B. Simons, Walter A. Burkhard, and Lawrence H. Landweber (eds.), *Proceedings of the 14th Annual ACM Symposium on Theory of Computing, May 5-7, 1982, San Francisco, California, USA*, pp. 159–168. ACM, 1982. doi: 10.1145/800070.802189. URL https://doi.org/10.1145/800070.802189.

Josef Urban. MPTP - motivation, implementation, first experiments. *J. Autom. Reason.*, 33(3-4): 319–339, 2004. doi: 10.1007/s10817-004-6245-1. URL https://doi.org/10.1007/s10817-004-6245-1.

Josef Urban. Malarea: a metasystem for automated reasoning in large theories. In *Proceedings of the CADE-21 Workshop on Empirically Successful Automated Reasoning in Large Theories, Bremen, Germany, 17th July 2007*, volume 257 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2007. URL http://ceur-ws.org/Vol-257/05_Urban.pdf.

Josef Urban and Jan Jakubuv. First neural conjecturing datasets and experiments. In *Intelligent Computer Mathematics - 13th International Conference, CICM 2020, Bertinoro, Italy, July 26-31, 2020, Proceedings*, volume 12236 of *Lecture Notes in Computer Science*, pp. 315–323. Springer, 2020. doi: 10.1007/978-3-030-53518-6_24. URL https://doi.org/10.1007/978-3-030-53518-6_24.

Josef Urban, Geoff Sutcliffe, Petr Pudlák, and Jirí Vyskocil. Malarea SG1- machine learner for automated reasoning with semantic guidance. In *Automated Reasoning, 4th International Joint Conference, IJCAR 2008, Sydney, Australia, August 12-15, 2008, Proceedings*, volume 5195 of *Lecture Notes in Computer Science*, pp. 441–456. Springer, 2008. doi: 10.1007/978-3-540-71070-7_37. URL https://doi.org/10.1007/978-3-540-71070-7_37.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. 2017.

Carl Vondrick and Antonio Torralba. Generating the future with adversarial transformers. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1020–1028, 2017.

Minchao Wu, Michael Norrish, Christian Walder, and Amir Dezfouli. Tacticzero: Learning to prove theorems from scratch with deep reinforcement learning. *CoRR*, abs/2102.09756, 2021a. URL https://arxiv.org/abs/2102.09756.

Yuhuai Wu, Albert Jiang, Jimmy Ba, and Roger Baker Grosse. INT: an inequality benchmark for evaluating generalization in theorem proving. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021b. URL https://openreview.net/forum?id=O6LPudowNQm.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Trans. Neural Networks Learn. Syst.*, 32 (1):4–24, 2021c. doi: 10.1109/TNNLS.2020.2978386. URL https://doi.org/10.1109/TNNLS.2020.2978386.

Kaiyu Yang and Jia Deng. Learning to prove theorems via interacting with proof assistants. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pp. 6984–6994. PMLR, 2019. URL http://proceedings.mlr.press/v97/yang19a.html.

Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. Seqgan: Sequence generative adversarial nets with policy gradient. In Satinder P. Singh and Shaul Markovitch (eds.), *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, pp. 2852–2858. AAAI Press, 2017. URL http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14344.

Wojciech Zaremba, Karol Kurach, and Rob Fergus. Learning to discover efficient mathematical identities. In Zoubin Ghahramani, Max Welling, Corinna Cortes, Neil D. Lawrence, and Kilian Q. Weinberger (eds.), *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pp. 1278–1286, 2014. URL https://proceedings.neurips.cc/paper/2014/hash/08419be897405321542838d77f855226-Abstract.html.

Kuo-Hao Zeng, Mohammad Shoeybi, and Ming-Yu Liu. Style example-guided text generation using generative adversarial transformers. *arXiv preprint arXiv:2003.00674*, 2020.

Yizhe Zhang, Zhe Gan, Kai Fan, Zhi Chen, Ricardo Henao, Dinghan Shen, and Lawrence Carin. Adversarial feature matching for text generation.

A SYNTAX AND SEMANTICS OF LINEAR-TIME TEMPORAL LOGIC (LTL)

In this section, we provide the formal syntax and semantics of Linear-time Temporal Logic (LTL). The formal syntax of LTL is given by the following grammar:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \land \varphi \mid \bigcirc\varphi \mid \varphi\,\mathcal{U}\,\varphi,$$

where $p \in AP$ is an atomic proposition. Let AP be a set of *atomic propositions*. A (*explicit*) *trace* t is an infinite sequence over subsets of the atomic propositions. We define the set of traces $TR := (2^{AP})^{\omega}$. We use the following notation to manipulate traces: Let $t \in TR$ be a trace and $i \in \mathbb{N}$ be a natural number. With $t[i]$ we denote the set of propositions at i-th position of t. Therefore, $t[0]$ represents the starting element of the trace. Let $j \in \mathbb{N}$ and $j \geq i$. Then $t[i, j]$ denotes the sequence $t[i]\ t[i + 1] \ldots t[j - 1]\ t[j]$ and $t[i, \infty]$ denotes the infinite suffix of t starting at position i.

Let $p \in AP$ and $t \in TR$. The semantics of an LTL formula is defined as the smallest relation $\models$ that satisfies the following conditions:

$$\begin{array}{lll}
t \models p & \text{iff} & p \in t[0] \\
t \models \neg\varphi & \text{iff} & t \not\models \varphi \\
t \models \varphi_1 \land \varphi_2 & \text{iff} & t \models \varphi_1 \text{ and } t \models \varphi_2 \\
t \models \bigcirc\varphi & \text{iff} & t[1, \infty] \models \varphi \\
t \models \varphi_1\,\mathcal{U}\,\varphi_2 & \text{iff} & \text{there exists } i \geq 0 : t[i, \infty] \models \varphi_2 \\
 & & \text{and for all } 0 \leq j < i \text{ we have } t[j, \infty] \models \varphi_1
\end{array}$$

There are several derived operators, such as $\Diamond\varphi \equiv true\,\mathcal{U}\,\varphi$ and $\Box\varphi \equiv \neg\Diamond\neg\varphi$. $\Diamond\varphi$ states that φ will *eventually* hold in the future and $\Box\varphi$ states that φ holds *globally*. Operators can be nested: $\Box\Diamond\varphi$, for example, states that φ has to occur infinitely often.

In contrast to propositional logic (SAT), where a solution is a variable assignment, the solution to the satisfiability problem of an LTL formula is a computation trace. Traces are finitely represented in the form of a "lasso" uv^{ω}, where u, called prefix, and v, called period, are finite sequences of propositional formulas. For example the mutual exclusion formula above is satisfied by a trace $(\{access_{p0}\}\{access_{p1}\})^{\omega}$ that alternates indefinitely between granting process 0 ($p0$) and process 1 ($p1$) access. There are, however, infinite solutions to an LTL formula. The empty trace $\{\}^{\omega}$, where no access is granted at all, is also a solution. In our data representation, both, the LTL formula and the solution trace are represented as a finite sequence.

B DATA GENERATION DETAILS

B.1 RICH LTL PATTERN CONCATENATION

Previously, LTL formula generation based on patterns worked by concatenating random instantiations of a fixed set of typical specification patterns (Hahn et al., 2021). The instantiations were single variables, i.e. the response pattern $S \rightarrow \Diamond T$ could be used like $d \rightarrow \Diamond a$. We keep the concept of concatenating such patterns, but extend the process by mainly two concepts: rich pattern instantiations and groundings.

Dwyer et al. (1999) analyzed typical specifications constructed a system of frequently occurring patterns. They are grouped into different types such as *absence* ($\neg S$, something does not occur) or response ($S \rightarrow \Diamond T$, if S occurred, T must eventually respond). These patterns can again appear in different scopes such as globally, before or between some events. The global absence pattern is then $\Box\neg S$; the absence before Q pattern reads $Q\,\mathcal{R}\,\neg S$. When generating a new pattern for concatenation, we sample both a type and a scope and assign different probabilities to account for more common and exotic combinations. Additionally, we instantiate patterns not with single variables, but full subformulas, which results in much more reasonable and interesting patterns such as $\Box\neg(a \land b)$ or $((\neg d \lor \bigcirc b) \rightarrow \Diamond(c \land f))\,\mathcal{U}\,e$. These subformulas may still contain temporal operators, but are strongly biased towards pure boolean operators.

During concatenating the different parts of a formula, we also distinguish between adding instantiated patterns and *groundings*. The problem with complex patterns and especially complex scopes is that they must be "activated" to have some effect: If some constraint must only hold between Q and R but these events never happen, the whole pattern is effectively useless. A grounding is a term that is likely to activate scopes, such as $\bigcirc\bigcirc a \wedge \neg b$ or $\square\diamondsuit c$. The variables used here are also biased to coincide with the ones already used in previous patterns to further increase the change for dependencies. Groundings are added with 45% probability instead of a specification pattern.

We observe that these changes indeed lead to a much higher chance of unsatisfiability. Consider the code in `data_generation/spec_patterns.py` for exact reference of the individual steps in the generation process.

B.2 Temporal Relaxation for Formula Inspection

We inspect the unsatisfiable formulas obtained by our generation process more closely. Concretely, we want to make sure that unsatisfiabilities do not stem from simple boolean contradictions, but actually require temporal reasoning to some extend. For example, the formula $(a \vee b) \wedge \neg b \wedge \square \neg a$ can be found to be unsatisfiable without considering multiple time steps. In contrast, this would be required for a formula like $\neg a \,\mathcal{U}\, b \wedge \square \neg b \wedge \diamondsuit a$.

We therefore introduce a *temporal relaxation* that transforms a LTL formula into a purely boolean formula. This allows us to check whether the relaxed version is already unsatisfiable (so, no temporal reasoning is required) or if it is only temporally unsatisfiable, which is the desired outcome. The relaxation is defined as follows:

$$\begin{aligned} \mathit{Rel}(\varphi * \psi) &= \mathit{Rel}(\varphi) * \mathit{Rel}(\psi) \quad \text{for } * \in \{\wedge, \vee, \rightarrow, \leftrightarrow, \oplus\} \\ \mathit{Rel}(\varphi * \psi) &= \mathit{Rel}(\varphi) \vee \mathit{Rel}(\psi) \quad \text{for } * \in \{\mathcal{U}, \mathcal{W}\} \\ \mathit{Rel}(\varphi \,\mathcal{R}\, \psi) &= \mathit{Rel}(\psi) \\ \mathit{Rel}(\bigcirc \varphi) &= \top \\ \mathit{Rel}(\square \varphi) &= \varphi \\ \mathit{Rel}(\diamondsuit \varphi) &= \top \\ \mathit{Rel}(\alpha) &= \alpha \quad \text{for } \alpha \in AP \cup \{\top, \bot\} \\ \mathit{Rel}(\neg\alpha) &= \neg\alpha \quad \text{for } \alpha \in AP \cup \{\top, \bot\} \end{aligned} \tag{1}$$

Notably, negation is only allowed at the level of atoms. Each LTL formula can be rewritten in a negation normal form (NNF), where only operators $\wedge, \vee, \mathcal{U}, \mathcal{R}, \bigcirc$ occur anywhere and negations only before atoms. Consequently, the relaxation can be applied to each LTL formula by first bringing it to NNF.

B.3 Base Dataset

We generated a raw dataset of 1.6M instances (see reproducibility section for details) up to size 50. To determine satisfiability, we use the tool `aalta` (Li et al., 2014). Its length distribution and satisfiability distribution is shown in Figures 5 and 6.

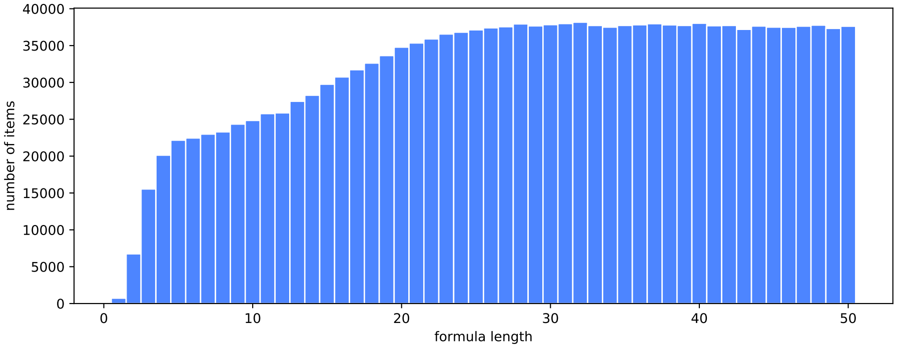


Figure 5: Raw dataset size distribution

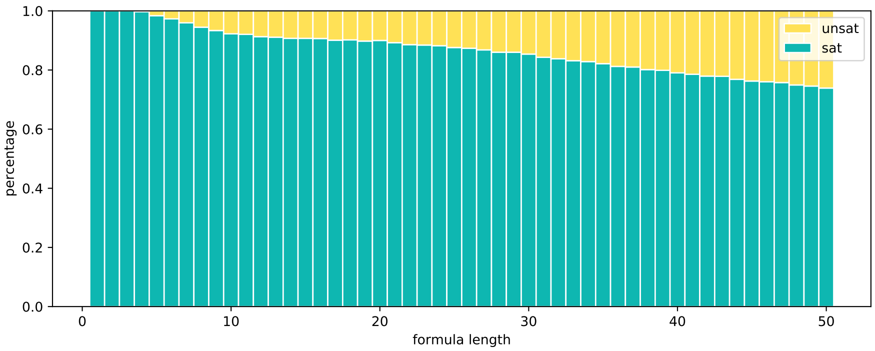


Figure 6: Raw dataset satisfiability proportions

We filter out duplicates and balance satisfiable and unsatisfiable instances per size (Figure 7). Additionally, we apply the temporal relaxation and determine the satisfiability of relaxed unsatisfiable instances. This distinction is included in Figure 8. Finally, the dataset is split into a training set (80%) and validation set (10%). The resulting training set contains around 380K instances.

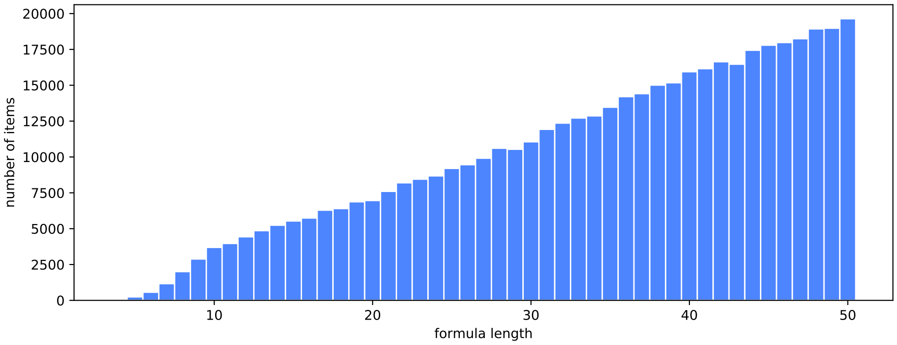


Figure 7: Final dataset size distribution (average size 34.6)

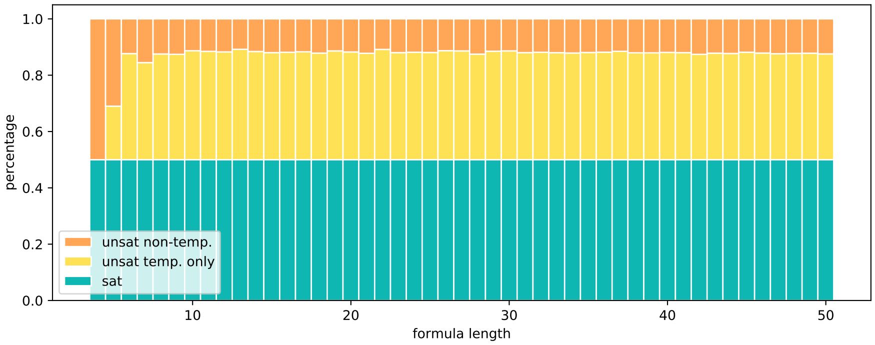


Figure 8: Final dataset satisfiability proportions

B.4 WGAN-Generated Datasets

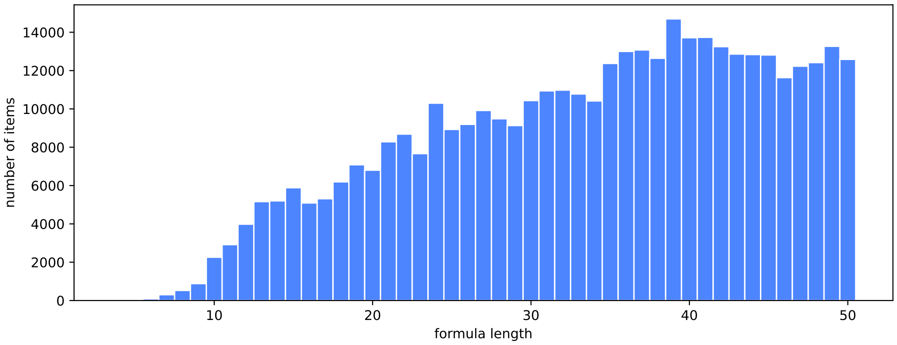


Figure 9: `Generated` dataset size distribution (average size 33.6)

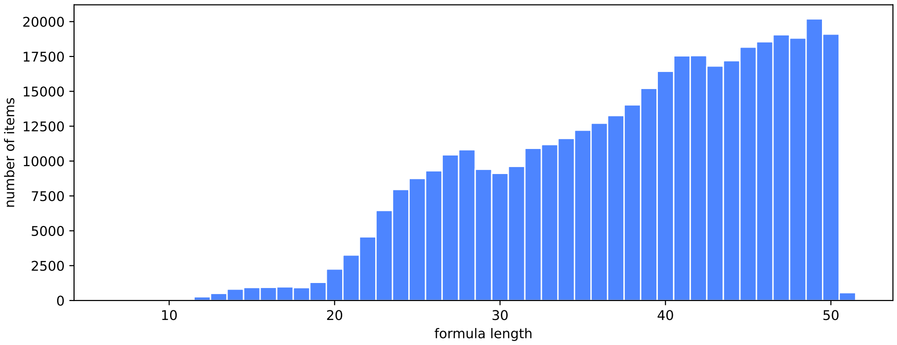


Figure 10: `Uncert-e` dataset size distribution (average size 38.0)

C Additional experiments and information

C.1 Shared Layers for Classifier Included in Critic

Table 4: Different number of shared layers for WGAN with included classifier, 2 runs each, 30K steps

shared layers	se	fc	val acc
0 / 4	2.2	31.8% (0.2)	89.9% (2.3)
2 / 4	2.2	26.6% (1.7)	92.5% (0.1)
3 / 4	2.2	24.9% (0.3)	92.1% (0.6)
4 / 4	2.2	24.0% (1.2)	90.5% (0.5)

Table 4 shows classification benefits for sharing only some layers between classifier and critic. Also note that not sharing any layers, while yielding the highest fraction of fully correct formulas in the joint GAN and classification objective, degrades performance in the uncertainty setting, where a loss is backpropagated through the classifier part.

C.2 Hyper-parameter Comparison

A hyper-parameter comparison with a 2-run average at 15K training steps.

n_{lG}	n_{lD}	n_c	bs	variant	fully correct
2	2	1	1024	GAN	6%
2	2	1	1024	WGAN	4%
2	2	2	512	GAN	6%
2	2	2	512	WGAN	4%
2	2	2	1024	GAN	7%
2	2	2	1024	WGAN	5%
2	2	2	2048	GAN	5%
2	2	2	2048	WGAN	6%
2	2	3	1024	WGAN	5%
2	4	2	1024	GAN	8%
2	4	2	1024	WGAN	9%
3	3	2	1024	GAN	8%
3	3	2	1024	WGAN	13%
4	2	2	1024	GAN	7%
4	2	2	1024	WGAN	14%
4	4	2	1024	GAN	10%
4	4	2	1024	WGAN	16%
6	4	2	1024	GAN	17%
6	4	2	1024	WGAN	20%
6	6	2	1024	WGAN	15%
8	6	2	1024	WGAN	18%

C.3 Training Curves for Data Substitution Experiments

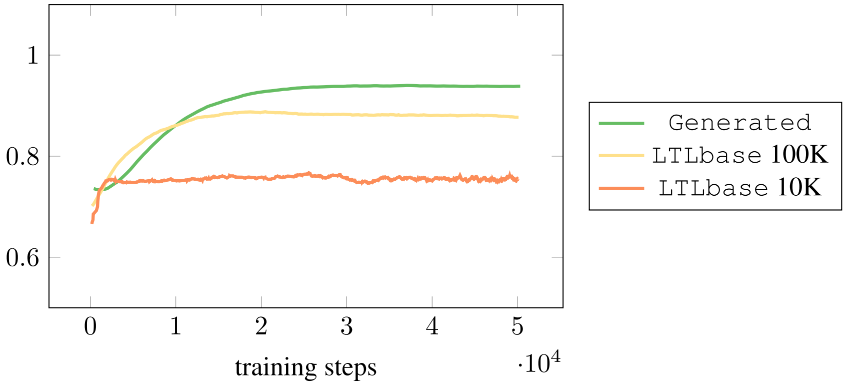


Figure 11: Validation accuracy during training of Transformer classifiers on different datasets. 5-run average, smoothed ($\alpha = 0.9$). Complements Table 2.

We provide the training curves for the data substition experiment (see Figure 11).

C.4 Standard Deviations for Table 1

Table 5: Standard deviations for Table 1, 3-run average, smoothed ($\alpha = 0.95$)

architecture	σ_{real}	fc sd	se sd	architecture	σ_{real}	fc sd	se sd
GAN	0.00	0.00	-	WGAN	0	1.51	0.00
	0.05	4.10	0.04		0.05	2.52	0.00
	0.1	6.50	0.03		0.1	1.08	0.01
	0.2	3.92	0.04		0.2	0.47	0.00
	0.4	0.14	0.02		0.4	0.14	0.03

We provide the standard deviations for Table 1 across 3 runs (see Table 5).

C.5 GAN WITH UNIFORM NOISE

Table 6: GAN variant with uniform instead of Gaussian noise. 2-run average with standard deviations, smoothed ($\alpha = 0.99$)

min	max	fc	se
0	0.1	19.2% (2.94)	1.6 (0.19)
0	0.2	33.3% (1.04)	1.8 (0.01)
0	0.4	11.2% (3.32)	2.0 (0.07)

As evident from Table 6 in comparison with Table 1, a uniform noise has no benefit over Gaussian noise.

C.6 OUT-OF-DISTRIBUTION CLASSIFICATION EXPERIMENTS

Table 7: Classifiers trained on different datasets tested out-of-distribution. 5-run average, not smoothed

trained on	training steps	tested on	accuracy
`LTLbase`	30K	`Benchmarks`	85.2% (2.2)
`Uncert-e`	30K	`Benchmarks`	85.9% (2.9)
`Uncert-e`	30K	`LTLbase`	87.5% (0.9)
`Mixed-e`	30K	`Mixed-e`	92.7% (0.6)
`LTLbase`	50K	`Benchmarks`	86.0% (5.0)
`Generated`	50K	`Benchmarks`	94.1% (1.2)

A synthetic dataset that is designed to bring classical solvers to their limits is a portfolio dataset (Schuppan & Darmawan, 2011), of which around 750 formulas fit into our encoder token and size restrictions. We conducted an out-of-distribution test on these scalable benchmarks (Table 7). Note that almost all of the instances are satisfiable.