Boolformer: Symbolic Regression of Logic Functions with Transformers

Stéphane d'Ascoli* [1] **Arthur Renard*** [2] **Vassilis Papadopoulos** [2] **Josh Susskind** [3] **Samy Bengio** [3]
Emmanuel Abbé [4][3]

Abstract

We introduce Boolformer, a Transformer-based model trained to perform end-to-end symbolic regression of Boolean functions. First, we show that it can predict compact formulas for complex functions not seen during training, given their full truth table. Then, we demonstrate that even with incomplete or noisy observations, Boolformer is still able to find good approximate expressions. We evaluate Boolformer on a broad set of real-world binary classification datasets, demonstrating its potential as an interpretable alternative to classic machine learning methods. Finally, we apply it to the widespread task of modeling the dynamics of gene regulatory networks and show through a benchmark that Boolformer is competitive with state-of-the-art genetic algorithms, with a speedup of several orders of magnitude. Our code and models are available publicly.

1. Introduction

Deep neural networks, in particular those based on the Transformer architecture (Vaswani et al., 2017), have led to breakthroughs in computer vision (Dosovitskiy et al., 2021) and language modelling (Brown et al., 2020), and have fuelled the hopes to accelerate scientific discovery (Jumper et al., 2021). However, their ability to perform simple logic tasks remains limited (Delétang et al., 2022). These tasks differ from traditional vision or language tasks in the combinatorial nature of their input space, which makes representative data sampling challenging.

Reasoning tasks have thus gained major attention in the deep learning community, either (i) with explicit reasoning in the logical domain, e.g., tasks in the realm of arithmetic (Saxton et al., 2019; Lewkowycz et al., 2022), algebra (Zhang et al.,

2022) or algorithmics (Velickovic et al., 2022), or (ii) implicit reasoning in other modalities, e.g., benchmarks such as Pointer Value Retrieval (Zhang et al., 2021) and Clevr (Johnson et al., 2017) for vision models, or LogiQA (Liu et al., 2020) and GSM8K (Cobbe et al., 2021) for language models. Reasoning also plays a key role in tasks which can be tackled via Boolean modelling, particularly in the fields of biology (Wang et al., 2012) and medicine (Hemedan et al., 2022).

As these endeavours remain challenging for current Transformer architectures, it is natural to examine whether they can be handled more effectively with different approaches, e.g., by better exploiting the Boolean nature of the task. In particular, when learning Boolean functions with a 'classic' approach based on minimizing the training loss on the outputs of the function, Transformers learn potentially complex interpolators as they focus on minimizing the degree profile in the Fourier spectrum, which is not the type of bias desirable for generalization on domains that are not well sampled (Abbe et al., 2022). In turn, the complexity of the learned function makes its interpretability challenging. This raises the question of how to improve generalization and interpretability of such models.

In this paper, we tackle Boolean function learning with Transformers, but we rely directly on 'symbolic regression': our Boolformer is tasked to directly predict a Boolean formula, i.e., a symbolic expression of the Boolean function in terms of the three fundamental logical gates (AND, OR, NOT) such as those of Figs. 1,2. As illustrated in Fig. 3, this task is framed as a sequence prediction problem: each training example is a synthetically generated function whose truth table is the input and whose formula is the target.

By moving to this setting, we decouple the symbolic task of inferring the logical formula and the numerical task of evaluating it on new inputs: the Boolformer only has to handle the first part. We show that this approach can give surprisingly strong performance both in abstract and real-world settings, and discuss how this lays the ground for future improvements and applications.

---

*Equal contribution [1]EPFL [2]CSFT, EPFL [3]Apple [4]SB/IC, EPFL. Correspondence to: Arthur Renard <arthurenard@icloud.com>.
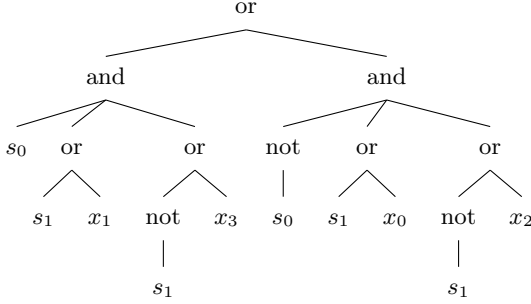
Figure 1. **Example Boolean formula predicted by our model.** Depicted: the multiplexer, a function commonly used in electronics to select one out of four sources x_0, x_1, x_2, x_3 based on two selector bits s_0, s_1.

1.1. Contributions

1. We train Transformers over synthetic datasets to perform end-to-end symbolic regression of Boolean formulas. The synthetic functions are generated by simplifying formulas whose tree have either low width or low depth (see Section 2.1). We show that given the full truth table of an unseen function, Boolformer is able to predict a compact formula, as illustrated in Fig. 1.
2. We show that Boolformer is robust to noisy and incomplete observations, by training it on incomplete truth tables with flipped bits and irrelevant variables. This is a necessary condition for its applicability to real-word data.
3. We evaluate Boolformer on various real-world binary classification tasks from the PMLB database (Olson et al., 2017) and show that it is competitive with classic machine learning approaches such as Random Forests while being more interpretable as illustrated in Fig. 2.
4. We apply Boolformer to the well-studied task of modeling Gene Regulatory Networks (GRNs) in biology. Using a recent benchmark (Pušnik et al., 2022), we show that our model is competitive with state-of-the-art methods with orders of magnitude faster inference.

Reproducibility All code used for the paper is available at https://github.com/arthurenard/Boolformer.

1.2. Related Work

Logical reasoning in deep learning Several papers have studied the ability of deep neural networks to solve logic tasks. Evans & Grefenstette (2018) introduce differential inductive logic as a method to learn logical rules from noisy data, and a few subsequent works attempted to craft dedicated neural architectures to improve this ability (Ciravegna et al., 2023; Shi et al., 2020b; Dong et al., 2019). Large language models (LLMs) such as ChatGPT, however, have

been shown to perform poorly at simple logical tasks such as basic arithmetic (Delétang et al., 2022; Jelassi et al., 2023), and tend to rely on approximations and shortcuts (Liu et al., 2022). Although some reasoning abilities seem to emerge with scale (Wei et al., 2022a) and can be enhanced via several procedures such as scratchpads (Nye et al., 2021) and chain-of-thought prompting (Wei et al., 2022b), achieving holistic and interpretable reasoning in LLMs remains a challenge.

Learning Boolean functions Learning Boolean functions has been an active area in theoretical machine learning, mostly under the probably approximately correct (PAC) and statistical query (SQ) learning frameworks (Hellerstein & Servedio, 2007; Reyzin, 2020). More recently, Abbe et al. (2023) shows that regular neural networks learn by gradually fitting monomials of increasing degree, in such a way that the sample complexity is governed by the 'leap complexity' of the target function, i.e. the largest degree jump the Boolean function sees in its Fourier decomposition. In turn, Abbe et al. (2022) shows that this leads to a 'min-degree bias' limitation: Transformers tend to learn interpolators having least 'degree profile' in the Boolean Fourier basis, which typically lose the Boolean nature of the target and often produce complex solutions with poor out-of-distribution generalization.

Inferring Boolean formulas A few works have explored the paradigm of inferring Boolean formulas in symbolic form, using SAT solvers (Narodytska et al., 2018), ILP solvers (Wang & Rudin, 2015; Su et al., 2015) or LP-relaxation (Malioutov et al., 2017). However, all these works predict the formulas in conjunctive or disjunctive normal forms (CNF/DNF), which typically amounts to exponentially long formulas. In contrast, the Boolformer is biased towards predicting compact expressions[1], which is more akin to logic synthesis – the task of finding the shortest circuit to express a given function, also known as the Minimum Circuit Size Problem (MCSP). While a few heuristics (e.g. Karnaugh maps (Karnaugh, 1953)) and algorithms (e.g. ESPRESSO (Rudell & Sangiovanni-Vincentelli, 1987)) exist to tackle the MCSP, its NP-hardness (Murray & Williams, 2017) remains a barrier towards efficient circuit design. Given the high resilience of computers to errors, approximate logic synthesis techniques have been introduced (Scarabottolo et al., 2018; Venkataramani et al., 2012; 2013; Boroumand et al., 2021; Oliveira & Sangiovanni-Vincentelli, 1993; Rosenberg et al., 2023), with the aim of providing approximate expressions given incomplete data –

---

[1]Consider for example the comparator of Fig. 1: since the truth table has roughly as many positive and negative outputs, the CNF/DNF involves $\mathcal{O}(2^D)$ terms where D is the number of input variables. For $D = 10$ this is several thousand binary gates, versus 17 for our model.
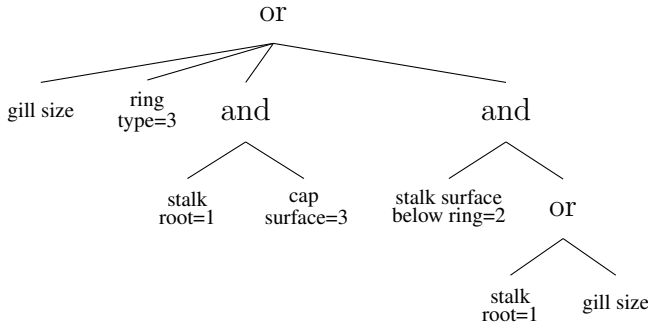
Figure 2. **A Boolean formula predicted to determine whether a mushroom is poisonous.** We considered the "mushroom" dataset from the PMLB database (Olson et al., 2017), and this formula achieves an F1 score of 0.96.

Figure 3. **Summary of our approach.** We feed N points $(\boldsymbol{x}, f(\boldsymbol{x})) \in \{0,1\}^{D+1}$ to a seq2seq Transformer, and supervise the prediction to f via cross-entropy loss.

this is similar in spirit to what we study in Section 4.

Symbolic regression   Symbolic regression (SR), i.e. the search of mathematical expressions underlying a set of numerical values, is still today a rather unexplored paradigm in the ML literature. Since this search cannot directly be framed as a differentiable problem, the dominant approach for SR is genetic programming (see (La Cava et al., 2021) for a recent review). A few recent publications applied Transformer-based approaches to SR (Biggio et al., 2021; Valipour et al., 2021; Kamienny et al., 2022; Tenachi et al., 2023), yielding comparable results but with a significant advantage: the inference time rarely exceeds a few seconds, several orders of magnitude faster than existing methods. Indeed, while the latter needs to be run from scratch on each new set of observations, Transformers are trained over synthetic datasets, and inference simply consists of a forward pass. Such efficiency opens the possibility of merging the two approaches, by using the trained model to suggest meaningful mutations in the context of genetic algorithm (e.g. (Romera-Paredes et al., 2024)). This may boost both the convergence speed and the quality of the results, especially in search spaces where most random mutations would result in invalid candidates.

2. Methods

Our task is to infer Boolean functions of the form $f : \{0,1\}^D \to \{0,1\}$, by predicting a Boolean formula built from the basic logical operators: AND, OR, NOT, as illustrated in Figs. 1,2. We train Transformers (Vaswani et al., 2017) on a large dataset of synthetic examples, following the seminal approach of (Lample & Charton, 2020). For each example, the input Ω_{fit} is a set of pairs $\{(\boldsymbol{x}_i, y = f(\boldsymbol{x}_i))\}_{i=1\dots N}$, and the target is the function f

as described above. Our general method is summarized in Fig. 3. Examples are generated by first sampling a random function f, then generating the corresponding $(\boldsymbol{x}, y)$ pairs as described in the following sections.

2.1. Generating formulas

To sample Boolean formulas[2], we construct random unary-binary tree with mathematical operators at the internal nodes and variables at the leaves. We rely on the tree generator of (Lample & Charton, 2020), whose distribution is biased towards trees which are either relatively narrow (and possibly deep) or relatively shallow (and possibly wide). Moreover, once operators and variables are sampled inside the tree, we further simplify the formula using algebraic rules in order to make the formula as simple as possible, encouraging the model to predict maximally compact formulas. The full sampling procedure is detailed in App. A.

The distribution of functions generated in this way spans the whole space of possible Boolean functions (of size 2^{2^D}), but in a non-uniform fashion with a bias towards functions described by a relatively simple formula[3]. As discussed quantitatively in App. B, the diversity of functions generated in this way is such that throughout the whole training procedure, functions of dimension $D \geq 7$ are typically encountered at most once.

---

[2]A Boolean formula is a tree where input variables can appear more than once, and differs from a Boolean circuit, which is a directed graph which can feature cycles, but where each input bit appears once at most.

[3]Indeed, for uniformly random function, the task would be hopeless as it is known to be NP-hard (Murray & Williams, 2017)

2.2. Generating inputs

Once the function f is generated, we select N points $\boldsymbol{x}$ in the Boolean hypercube following the procedure detailed below, and compute the corresponding outputs $y = f(\boldsymbol{x})$. Optionally, we may flip the bits of the inputs and outputs independently with probability σ_{flip}; we consider the two following setups.

Noiseless regime The noiseless regime, studied in Sec. 3, is defined as follows:

- **Noiseless data:** there is no bit flipping, i.e. $\sigma_{\text{flip}} = 0$.
- **Full support:** all the input variables are present in the Boolean formula.
- **Full observability:** the model has access to the whole truth table of the Boolean function, i.e. $N = 2^D$. This limits us to a maximum of 10 input variables.

Noisy regime In the noisy regime, studied in Sec. 4, the model must determine which variables affect the output, while also being able to cope with corruption of the inputs and outputs. During training, we vary the amount of noise for each sample so that the model can handle a variety of noise levels:

- **Noisy data:** the probability of each bit (both input and output) being flipped σ_{flip} is sampled uniformly in $[0, 0.1]$.
- **Partial support:** the model can handle functions with up to 120 input variables, but only up to 6 of these are "active", i.e. appear in the Boolean formula – all the other variables are "inactive".
- **Partial observability:** a subset of the hypercube is observed: the number of input points N is sampled uniformly in $[30, 300]$, which is typically much smaller that 2^D. Additionally, instead of sampling uniformly (which would cause distribution shifts if the inputs are not uniformly distributed at inference), we generate the input points via a random walk in the hypercube. Namely, we sample an initial point $\boldsymbol{x}_0$ then construct the following points by flipping independently each coordinate with a probability sampled uniformly in $[0.05, 0.25]$.

2.3. Model

Tokenization To represent Boolean formulas as sequences processed by the decoder, we enumerate the nodes of the trees in prefix order, i.e., direct Polish notation as in (Lample & Charton, 2020): operators and variables are represented as single autonomous tokens, e.g. $[\texttt{AND}, x_1, \texttt{NOT}, x_2]$. The evaluations fed to the encoder are embedded using $\{\texttt{0}, \texttt{1}\}$ tokens. In the noiseless regime, we shrink the input length by providing less than half of the truth table, namely only the entries corresponding to the less frequent output of the boolean function. Using a special token, we indicate

whether this value is 0 or 1 which effectively provides the information of the full truth table, albeit implicitly. Formulas requiring more than 200 tokens are discarded, as we are limited by the attention size of the decoder.

Token Embeddings Our model is provided N input points $(\boldsymbol{x}, y)$, each of which is represented by $D + 1$ tokens of dimension D_{emb}, where D is the dimension of $\boldsymbol{x}$. As D and N become large, this would result in very long input sequences (ND tokens) which are suboptimal given the quadratic complexity of Transformers in the input length. To mitigate this, we introduce compressed embeddings to map each input pair $(\boldsymbol{x}, y)$ to a single embedding, following (Kamienny et al., 2022). To do so we pad the empty input dimensions to D_{max}, enabling our model to handle variable input dimension, then concatenate all the tokens and feed the $(D_{\text{max}} + 1) \times D_{\text{emb}}$-dimensional result into a linear layer which projects it down to dimension D_{emb}. The resulting N embeddings of dimension D_{emb} are then fed to the Transformer.

Transformer We use the Transformer architecture (Vaswani et al., 2017) where the encoder and decoder use 8 and 8 layers, 16 attention heads and an embedding dimension of 512, for a total of $\sim$ 60M parameters. A notable property of this task is the permutation invariance of the N input points. As such, we remove positional embeddings from the encoder, encoding this invariance in the model. The decoder uses standard learnable, absolute positional embeddings.

We have attempted scaling up the model, testing 110M and 450M parameter versions. Interestingly, there were few if any improvements, both on the cross-entropy loss, and the benchmark evaluations. This is peculiar, as transformers have shown to yield predictable improvements with scaling (Kaplan et al., 2020). We leave to future work the investigation of this phenomenon and its potential relation to the specific task of symbolic regression.

2.4. Training and evaluation

Training We optimize a cross-entropy loss with the AdamW optimizer and a batch size of 1024, warming up the learning rate from 10^{-7} to 2×10^{-4} over the first 5,000 steps. It is then kept constant for 60,000 steps, after which we perform a linear cooldown back to 0 (Hägele et al., 2024). On 4 H100 GPUs, this takes about 1 day.

Inference At inference time, we find that beam search does not provide measurable improvements compared to standard sampling. Therefore, in most results presented in this paper, we generate 10 candidates by sampling, then rank them according to how well they fit the input data (to assess this, we use the fitting error defined below). Note that

when the data is noiseless, the model will often find several candidates which have an accuracy close to 100%.

Evaluation Given a set of input-output pairs Ω generated by a target function $f_\star$, we compute the accuracy of a predicted function f as $\frac{1}{|\Omega|} \sum_{(\boldsymbol{x},y) \in \Omega} 1[f(\boldsymbol{x}) = f_\star(\boldsymbol{x})]$.
We can then define:

- **Fitting accuracy:** accuracy obtained when re-using the points used to predict the formula, $\Omega = \Omega_{\text{fit}}$
- **Fitting perfect recovery:** defined as 1 if the fitting error is strictly equal to 0, and 0 otherwise.
- **Test accuracy:** accuracy obtained when sampling points different than the ones used for the prediction. Thie metric does not apply in the noiseless regime, as the model observes all possible sampling points.
- **Test perfect recovery:** defined as 1 if the test error is strictly equal to 0, and 0 otherwise.

3. Noiseless regime: finding the shortest formula

The noiseless setting is akin to logic synthesis, where the goal is to find the shortest formula that implements a given function. The practical relevance of this regime is limited by the fact that it requires full observability of the function; as such, the results of this section serve as a controlled setting to probe the generalization abilities of Boolformer. Specifically, we do not claim that logic synthesis is a useful application of Boolformer in its current iteration, but rather an interesting task to examine the potential of the model, which we argue really shines in the noisy setting (Sec. 4).

In-domain performance In Fig. 4, we report the performance of the model when varying the number of input variables and the number of operators in the ground truth. Metrics are averaged over $10,000$ formulas, sampled from the generator used during training.

The model demonstrates high accuracy in predicting target functions across all cases, including for $D \geq 7$, where memorization is not feasible (samples with $D \geq 7$ have typically not been encountered during training as shown in App. B).

Of course, these results reflect the model's performance on the specific distribution of functions it was trained on, which is highly nonuniform within the 2^{2^D}-dimensional space of Boolean functions. It is important to note that for a Boolean function sampled uniformly from this space, the likelihood of achieving any meaningful accuracy is effectively negligible, as such random functions are highly unlikely to be expressible in fewer than 200 tokens. As a baseline, we also compare with ESPRESSO (Rudell &
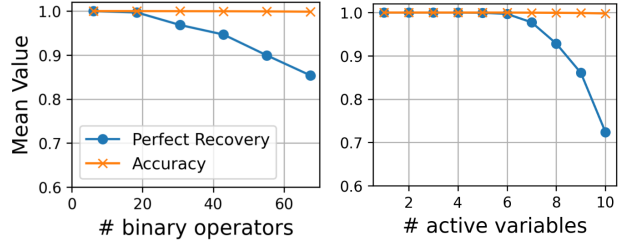

Figure 4. **Our model is able to approximate the formula of unseen functions with high accuracy.** We report perfect recovery and fitting accuracy of our model when varying the number of binary gates and input variables. Metrics are averaged over 10,000 samples from the function generator.

Sangiovanni-Vincentelli, 1987), which is a heuristic logic synthesizer that, despite its age, is still relevant today. Comparing to Boolformer (more details in App. C), the two methods yield comparable average formula lengths, with Boolformer yielding shorter formulas more often, especially when more variables are involved. The comparison is not perfect, as ESPRESSO is constrained to output formulas in sum-of-products form. Still, the ability to generate concise formulas is confirmation that Boolformer is at least learning non-trivial representations that generalize to unseen data.

Success and failure cases In Fig. 1, we show two examples of Boolean functions where our model successfully predicts a compact formula for: the 4-to-1 multiplexer (which takes 6 input variables) and the 5-bit comparator (which takes 10 input variables). In App. E, we provide more examples: addition and multiplication, as well as majority and parity functions. By increasing the dimensionality of each problem up to the point of failure, we show that in all cases our model typically predicts exact and compact formulas as long as the function can be expressed with less than 100 binary gates (which is the largest size seen during training, as larger formulas exceed the 200 tokens limit) and fails beyond. Still, even in this cases, the model is still able to approximate the formula well, as the accuracy still remains high.

Hence, the failure point depends on the intrinsic difficulty of the function: for example, Boolformer can predict an exact formula for the comparator function up to $D = 10$, but only $D = 6$ for multiplication, $D = 5$ for majority and $D = 4$ for parity as well as typical random functions (whose outputs are independently sampled from $\{0, 1\}$). Parity functions are well-known to be the most difficult functions to learn for SQ models due to their leap-complexity, and are also the hardest to learn in our framework because they require the most operators to be expressed (the XOR operator being excluded in this work).
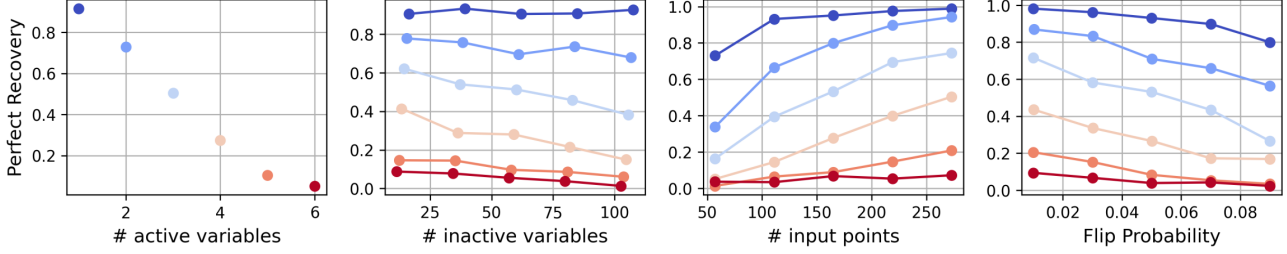
Figure 5. **Our model is robust to data incompleteness, bit flipping and noisy variables.** We display the test perfect recovery of our model when varying the four factors of difficulty described in Sec. 2. The colors depict different number of active variables (which appear in the Boolean formula), as shown in the first panel. Metrics are averaged over 10k samples from the random generator.

4. Noisy regime: applications to real-world data

We now turn to the noisy regime, which is defined at the end of Sec. 2.2. We begin by studying the robustness of Boolformer to incomplete and corrupted observations, then demonstrate its practical relevance by studying two real-world applications: interpretable binary classification and efficient Gene Regulatory Network (GRN) inference.

4.1. Results on noisy data

In Fig. 5, we show how the performance of our model depends on the various factors of difficulty of the problem. The different colors correspond to different numbers of active variables appearing in the formula, as shown in the leftmost panel: in this setting with multiple sources of noise, we see that accuracy drops much faster with the number of active variables than in the noiseless setting.

As could be expected, performance improves as the number of input points N increases, and degrades as the amount of random flipping and the number of inactive variables increase. However, our model copes relatively well with noise in general, as it displays nontrivial generalization even when we add up to 120 inactive variables and up to 10% random flipping.

4.2. Application 1: interpretable binary classification

In this section, we show that our noisy model can be applied to binary classification tasks, providing an interpretable alternative to classic machine learning methods on tabular data.

Method   We consider the tabular datasets from the Penn Machine Learning Benchmark (PMLB) from (Olson et al., 2017). These contain a wide variety of real-world problems, such as predicting chess moves, toxicity of mushrooms, credit scores, and heart diseases. Since our model only takes binary features as input, we discard continuous features and

binarize the categorical features with $C > 2$ classes into C binary variables. Note that this procedure can greatly increase the total number of features, we only keep datasets for which this results in less than 120 features (the maximum our model can handle). We randomly sample 25% of the examples for testing and report the F1 score obtained on this set.

We compare our model with two classic machine learning methods: logistic regression and random forests, using the default hyperparameters from `sklearn`. For random forests, we test two values for the number of estimators: 1 (in which case we obtain a simple decision tree as for Boolformer) and 100.

Results   Results are reported in Fig. 6, where for readability we only display the datasets where the random forest with 100 estimators achieves an F1 score above 0.75. The performance of Boolformer is similar on average to that of logistic regression : logistic regression typically performs better on "hard" datasets where there is no exact logical rule, for example medical diagnosis tasks such as `heart_h`, but worse on logic-based datasets where the data is not linearly separable such as `xd6`.

The F1 score of our model is slightly below that of a random forest of 100 trees, but slightly above that of the random forest with a single tree. This is remarkable considering that the Boolean formula it outputs usually contains a few dozen nodes, whereas the trees of random forest use up to several hundreds. As an example, we display a formula predicted for the mushroom toxicity dataset in Fig. 2, and a more extensive collection of formulas in App. F.

4.3. Application 2: efficient inference of gene regulatory networks

A Boolean network is a dynamical system composed of D bits whose transition from one state to the next is governed
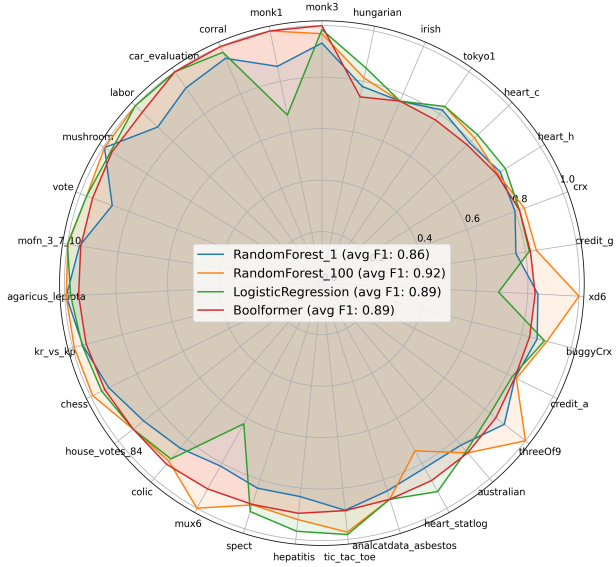
Figure 6. **Our model is competitive with classic machine learning methods while providing highly interpretable results.** We display the F1 score obtained on various binary classification datasets from the Penn Machine Learning Benchmark (Olson et al., 2017). We compare the F1 score of the Boolformer with random forests (using 1 and 100 estimators) and logistic regression.

by a set of D Boolean functions[4]. These types of networks have attracted a lot of attention in the field of computational biology as they can be used to model gene regulatory networks (GRNs) (Zhao et al., 2021) – see App. G for a brief overview of this field. In this setting, each bit represents the (discretized) expression of a gene (on or off) and each function represents the regulation of a gene by the other genes. In this section, we investigate the applicability of our symbolic regression-based approach to this task.

Benchmark   We use the recent benchmark for GRN inference introduced by (Pušnik et al., 2022). This benchmark compares 5 methods for Boolean network inference on 30 Boolean networks inferred from biological data, with sizes ranging from 16 to 64 genes, and assesses both dynamical prediction (how well the model predicts the dynamics of the network) and structural prediction (how well the model predicts the Boolean functions compared to the ground truth). Structural prediction is the binary classification task of predicting whether variable i influences variable j, and can be evaluated by many binary classification metrics; we report here the structural F1 and Youden's J statistic metrics, which are the most holistic, and defer other metrics to App. G.

---

[4]The i-th function f_i takes as input the state of the D bits at time t and returns the state of the i-th bit at time $t + 1$.

Method   Our model predicts each component f_i of the Boolean network independently, by taking as input the whole state of the network at times $[0 \ldots t-1]$ and as output the state of the ith bit at times $[1 \ldots t]$. Once each component has been predicted, we can build a causal influence graph, where an arrow connects node i to node j if j appears in the update equation of i: an example is shown in Fig. 7c. Note that since the dynamics of the Boolean network tend to be slow, an easy way to get rather high dynamical accuracy would be to simply predict the trivial fixed point $f_i = x_i$. In concurrent approaches, the function set explored excludes this solution; in our case, we simply mask the ith bit from the input when predicting f_i.

Results   We display the results of our model on the benchmark in Fig. 7a. Boolformer performs on par with the SOTA algorithms, GABNI (Barman & Kwon, 2018) and MIBNI (Barman & Kwon, 2017). A striking feature of our model is its inference speed, displayed in Fig. 7b: a few seconds, against up to an hour for concurrent approaches, which mainly rely on genetic programming. Note also that our model predicts an interpretable Boolean function, where the other SOTA methods (GABNI and MIBNI) only pick out the most important variables and the sign of their influence.

The fast inference speed of the Boolformer suggests that it could be used in combination with genetic approaches, to further increase the quality of the results, at the cost of inference speed. In such an approach, the fast model can be used to replace the random mutations, instead sampling the mutations from Boolformer, which should allow for a much more efficient exploration of the search space (Romera-Paredes et al., 2024). Due to the autoregressive nature of Boolformer, it is not well-suited for this task; a BERT-like architecture (Devlin et al., 2019) would be more appropriate, so we leave explorations of this approach for future work.

5. Discussion

In this work, we have shown that Transformers can be used to strongly perform the symbolic regression of logical functions, opening up a new, more interpretable framework than classical machine learning to solve certain types of classification tasks. Their ability to infer GRNs several orders of magnitude faster than existing methods offers the promise of many other exciting applications in biology, where Boolean modelling plays a key role (Hemedan et al., 2022). There are however several limitations in our current approach, which open directions for future work.

Limited Number of Input Points.   First, the number of input points is limited to a thousand during training, which limits our model's performance on high-dimensional functions (although the model does exhibit some length generalization abilities at inference, as shown in App. D). Note that
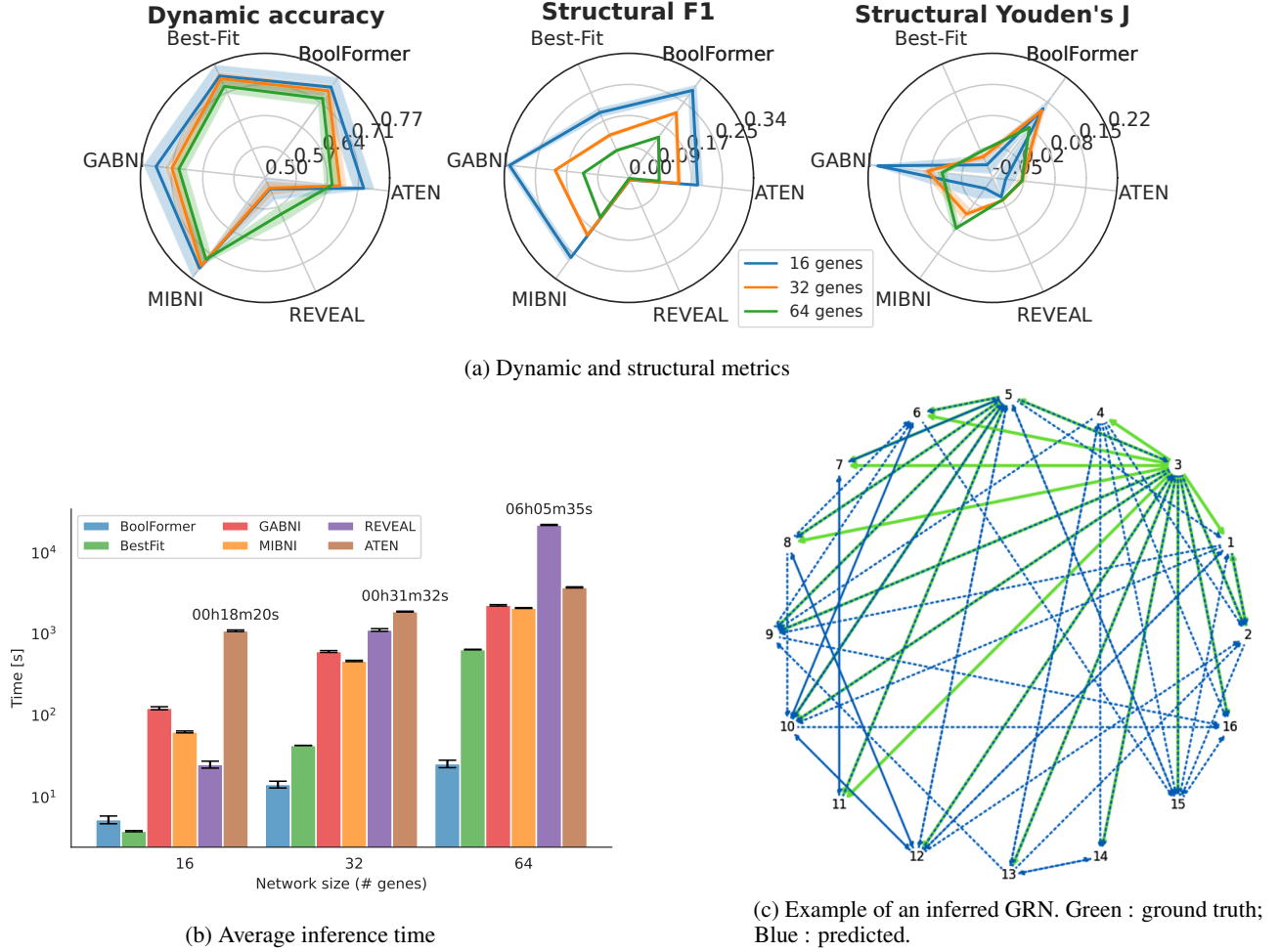
(a) Dynamic and structural metrics

(b) Average inference time

(c) Example of an inferred GRN. Green : ground truth; Blue : predicted.

Figure 7. **Our model is competitive with state-of-the-art methods for GRN inference with orders of magnitude faster inference.** (a) We compare the ability of our model to predict the next states (dynamic accuracy) and the influence graph (structural accuracy) with that of other methods using a recent benchmark (Pušnik et al., 2022) – more details in Sec. 4.3. (b) Average inference time of the various methods. (c) From the Boolean formulas predicted, one can construct an influence graph where each node represents a gene, and each arrow signals that one gene regulates another.

we did not consider linear attention mechanisms (Choromanski et al., 2021; Wang et al., 2020) because on top of potentially degrading performance[5], this would not fundamentally improve scalability, as the volume of input space grows exponentially with the input dimension.

Predefined Feature Sets. Our model is developed on binary input features. Although it is easy to binarize categorical and continuous features, this increases the input dimension significantly, and our model has a hard limit on the number of input features it can handle, which is set to a hundred in this work. This could be mitigated by considering

an extension to q-ary input features, although this requires choosing a new list of associated universal operators.

Vocabulary Limitations. The logical functions on which our model is trained do not include the XOR gate explicitly, limiting the compactness of the formulas it predicts. This limitation is due to our generation procedure that relies on expression simplification, which requires rewriting the XOR gate in terms of AND, OR and NOT. We leave it as future work to adapt the generation of simplified formulas containing XOR gates, as well as more general operators with more than two inputs as in (Rosenberg et al., 2023).

Lack of Intermediate Results and Multi-Output. The simplicity of the formulas predicted is limited in two additional ways: our model only handles (i) single-output func-

---

[5]We hypothesize that full attention span is particularly important in this specific task: the attention maps displayed in App. I are visually dense and high-rank matrices.

tions – multi-output functions are predicted independently component-wise and (ii) gates with a fan-out of one. As a result, our model cannot reuse intermediary results for different outputs or for different computations within a single output[6]. One could address this either by post-processing the generated formulas to identify repeated substructures, or by adapting the data generation process to support multi-output functions and cyclic graphs.

Finally, this paper mainly focused on investigating concrete applications and benchmarks to motivate the potential and development of Boolformers. A natural direction is to investigate theoretically and practically how the control of the data generator can influence the model simplicity and its impact on the 'generalization on the unseen' (Abbe et al., 2023) benchmarks.

References

berkeley-abc/abc, May 2025. URL https://github.com/berkeley-abc/abc. original-date: 2018-03-11T11:33:37Z.

Abbe, E., Bengio, S., Cornacchia, E., Kleinberg, J., Lotfi, A., Raghu, M., and Zhang, C. Learning to reason with neural networks: Generalization, unseen data and boolean measures. *ArXiv preprint*, abs/2205.13647, 2022. URL https://arxiv.org/abs/2205.13647.

Abbe, E., Boix-Adsera, E., and Misiakiewicz, T. Sgd learning on neural networks: leap complexity and saddle-to-saddle dynamics. *ArXiv preprint*, abs/2302.11055, 2023. URL https://arxiv.org/abs/2302.11055.

Adabor, E. S. and Acquaah-Mensah, G. K. Restricted-derestricted dynamic bayesian network inference of transcriptional regulatory relationships among genes in cancer. *Computational biology and chemistry*, 79:155–164, 2019.

Barman, S. and Kwon, Y.-K. A novel mutual information-based boolean network inference method from time-series gene expression data. *PloS one*, 12(2):e0171097, 2017.

Barman, S. and Kwon, Y.-K. A boolean network inference from time-series gene expression data using a genetic algorithm. *Bioinformatics*, 34(17):i927–i933, 2018.

Biggio, L., Bendinelli, T., Neitz, A., Lucchi, A., and Parascandolo, G. Neural symbolic regression that scales. In Meila, M. and Zhang, T. (eds.), *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pp. 936–945. PMLR, 2021. URL http://proceedings.mlr.press/v139/biggio21a.html.

Boroumand, S., Bouganis, C.-S., and Constantinides, G. A. Learning boolean circuits from examples for approximate logic synthesis. In *Proceedings of the 26th Asia and South Pacific Design Automation Conference*, pp. 524–529, 2021.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. Language models are few-shot learners. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H. (eds.), *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html.

Choromanski, K. M., Likhosherstov, V., Dohan, D., Song, X., Gane, A., Sarlós, T., Hawkins, P., Davis, J. Q., Mohiuddin, A., Kaiser, L., Belanger, D. B., Colwell, L. J., and Weller, A. Rethinking attention with performers. 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=Ua6zuk0WRH.

Ciravegna, G., Barbiero, P., Giannini, F., Gori, M., Lió, P., Maggini, M., and Melacci, S. Logic explained networks. *Artificial Intelligence*, 314:103822, 2023.

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., et al. Training verifiers to solve math word problems. *ArXiv preprint*, abs/2110.14168, 2021. URL https://arxiv.org/abs/2110.14168.

Delétang, G., Ruoss, A., Grau-Moya, J., Genewein, T., Wenliang, L. K., Catt, E., Cundy, C., Hutter, M., Legg, S., Veness, J., et al. Neural networks and the chomsky hierarchy. *ArXiv preprint*, abs/2207.02098, 2022. URL https://arxiv.org/abs/2207.02098.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019. URL https://arxiv.org/abs/1810.04805.

Dong, H., Mao, J., Lin, T., Wang, C., Li, L., and Zhou, D. Neural logic machines. 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=B1xY-hRctX.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. An image

---

[6]Consider the D-parity: one can build a formula with only $3(n-1)$ binary AND-OR gates by storing $D-1$ intermediary results: $a_1 = XOR(x_1, x_2), a_2 = XOR(a_1, x_3), \ldots, a_{n-1} = XOR(a_{D-2}, x_D)$. Our model needs to recompute these intermediary values, leading to much larger formulas, e.g. 35 binary gates instead of 9 for the 4-parity as illustrated in App. E.

is worth 16x16 words: Transformers for image recognition at scale. 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=YicbFdNTTy.

Evans, R. and Grefenstette, E. Learning explanatory rules from noisy data. *Journal of Artificial Intelligence Research*, 61:1–64, 2018.

Haury, A.-C., Mordelet, F., Vera-Licona, P., and Vert, J.-P. Tigress: trustful inference of gene regulation using stability selection. *BMC systems biology*, 6(1):1–17, 2012.

Hellerstein, L. and Servedio, R. A. On pac learning algorithms for rich boolean function classes. *Theoretical Computer Science*, 384(1):66–76, 2007.

Hemedan, A. A., Niarakis, A., Schneider, R., and Ostaszewski, M. Boolean modelling as a logic-based dynamic approach in systems medicine. *Computational and Structural Biotechnology Journal*, 20:3161–3172, 2022.

Huynh-Thu, V. A. and Geurts, P. dyngenie3: dynamical genie3 for the inference of gene networks from time series expression data. *Scientific reports*, 8(1):3384, 2018.

Huynh-Thu, V. A., Irrthum, A., Wehenkel, L., and Geurts, P. Inferring regulatory networks from expression data using tree-based methods. *PloS one*, 5(9):e12776, 2010.

Hägele, A., Bakouch, E., Kosson, A., Allal, L. B., Werra, L. V., and Jaggi, M. Scaling laws and compute-optimal training beyond fixed training durations, 2024. URL https://arxiv.org/abs/2405.18392.

Jelassi, S., d'Ascoli, S., Domingo-Enrich, C., Wu, Y., Li, Y., and Charton, F. Length generalization in arithmetic transformers. *ArXiv preprint*, abs/2306.15400, 2023. URL https://arxiv.org/abs/2306.15400.

Johnson, J., Hariharan, B., van der Maaten, L., Fei-Fei, L., Zitnick, C. L., and Girshick, R. B. CLEVR: A diagnostic dataset for compositional language and elementary visual reasoning. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pp. 1988–1997. IEEE Computer Society, 2017. doi: 10.1109/CVPR.2017.215. URL https://doi.org/10.1109/CVPR.2017.215.

Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., Tunyasuvunakool, K., Bates, R., Žídek, A., Potapenko, A., et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021.

Kamienny, P.-A., d'Ascoli, S., Lample, G., and Charton, F. End-to-end symbolic regression with transformers. In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K. (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=GoOuIrDHG_Y.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. Scaling laws for neural language models. *CoRR*, abs/2001.08361, 2020. URL https://arxiv.org/abs/2001.08361.

Karnaugh, M. The map method for synthesis of combinational logic circuits. *Transactions of the American Institute of Electrical Engineers, Part I: Communication and Electronics*, 72(5): 593–599, 1953.

La Cava, W., Orzechowski, P., Burlacu, B., de Franca, F. O., Virgolin, M., Jin, Y., Kommenda, M., and Moore, J. H. Contemporary symbolic regression methods and their relative performance. *ArXiv preprint*, abs/2107.14351, 2021. URL https://arxiv.org/abs/2107.14351.

Lähdesmäki, H., Shmulevich, I., and Yli-Harja, O. On learning gene regulatory networks under the boolean network model. *Machine learning*, 52(1-2):147, 2003.

Lample, G. and Charton, F. 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.

Lewkowycz, A., Andreassen, A., Dohan, D., Dyer, E., Michalewski, H., Ramasesh, V., Slone, A., Anil, C., Schlag, I., Gutman-Solo, T., et al. Solving quantitative reasoning problems with language models. *Advances in Neural Information Processing Systems*, 35:3843–3857, 2022.

Liang, S., Fuhrman, S., and Somogyi, R. Reveal, a general reverse engineering algorithm for inference of genetic network architectures. In *Biocomputing*, volume 3, 1998.

Liu, B., Ash, J. T., Goel, S., Krishnamurthy, A., and Zhang, C. Transformers learn shortcuts to automata. *ArXiv preprint*, abs/2210.10749, 2022. URL https://arxiv.org/abs/2210.10749.

Liu, J., Cui, L., Liu, H., Huang, D., Wang, Y., and Zhang, Y. Logiqa: A challenge dataset for machine reading comprehension with logical reasoning. In Bessiere, C. (ed.), *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, pp. 3622–3628. ijcai.org, 2020. doi: 10.24963/ijcai.2020/501. URL https://doi.org/10.24963/ijcai.2020/501.

Malioutov, D. M., Varshney, K. R., Emad, A., and Dash, S. Learning interpretable classification rules with boolean compressed sensing. *Transparent Data Mining for Big and Small Data*, pp. 95–121, 2017.

Murray, C. D. and Williams, R. R. On the (non) np-hardness of computing circuit complexity. *Theory of Computing*, 13(1): 1–22, 2017.

Narodytska, N., Ignatiev, A., Pereira, F., and Marques-Silva, J. Learning optimal decision trees with SAT. In Lang, J. (ed.), *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, pp. 1362–1368. ijcai.org, 2018. doi: 10.24963/ijcai.2018/189. URL https://doi.org/10.24963/ijcai.2018/189.

Nye, M., Andreassen, A. J., Gur-Ari, G., Michalewski, H., Austin, J., Bieber, D., Dohan, D., Lewkowycz, A., Bosma, M., Luan, D., et al. Show your work: Scratchpads for intermediate computation with language models. *ArXiv preprint*, abs/2112.00114, 2021. URL https://arxiv.org/abs/2112.00114.

Oliveira, A. and Sangiovanni-Vincentelli, A. Learning complex boolean functions: Algorithms and applications. *Advances in Neural Information Processing Systems*, 6, 1993.

Olson, R. S., La Cava, W., Orzechowski, P., Urbanowicz, R. J., and Moore, J. H. Pmlb: a large benchmark suite for machine learning evaluation and comparison. *BioData Mining*, 10(1):36, 2017. ISSN 1756-0381. doi: 10.1186/s13040-017-0154-4. URL https://doi.org/10.1186/s13040-017-0154-4.

Pušnik, Ž., Mraz, M., Zimic, N., and Moškon, M. Review and assessment of boolean approaches for inference of gene regulatory networks. *Heliyon*, pp. e10222, 2022.

Reyzin, L. Statistical queries and statistical algorithms: Foundations and applications. *ArXiv preprint*, abs/2004.00557, 2020. URL https://arxiv.org/abs/2004.00557.

Romera-Paredes, B., Barekatain, M., Novikov, A., Balog, M., Kumar, M. P., Dupont, E., Ruiz, F. J. R., Ellenberg, J. S., Wang, P., Fawzi, O., Kohli, P., and Fawzi, A. Mathematical discoveries from program search with large language models. *Nature*, 625 (7995):468–475, January 2024. ISSN 1476-4687. doi: 10.1038/s41586-023-06924-6. URL https://www.nature.com/articles/s41586-023-06924-6. Publisher: Nature Publishing Group.

Rosenberg, G., Brubaker, J. K., Schuetz, M. J., Salton, G., Zhu, Z., Zhu, E. Y., Kadıoğlu, S., Borujeni, S. E., and Katzgraber, H. G. Explainable ai using expressive boolean formulas. *ArXiv preprint*, abs/2306.03976, 2023. URL https://arxiv.org/abs/2306.03976.

Rudell, R. L. and Sangiovanni-Vincentelli, A. Multiple-valued minimization for pla optimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 6(5):727–750, 1987.

Saxton, D., Grefenstette, E., Hill, F., and Kohli, P. 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.

Scarabottolo, I., Ansaloni, G., and Pozzi, L. Circuit carving: A methodology for the design of approximate hardware. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 545–550. IEEE, 2018.

Shi, N., Zhu, Z., Tang, K., Parker, D., and He, S. Aten: And/or tree ensemble for inferring accurate boolean network topology and dynamics. *Bioinformatics*, 36(2):578–585, 2020a.

Shi, S., Chen, H., Ma, W., Mao, J., Zhang, M., and Zhang, Y. Neural logic reasoning. In d'Aquin, M., Dietze, S., Hauff, C., Curry, E., and Cudré-Mauroux, P. (eds.), *CIKM '20: The 29th ACM International Conference on Information and Knowledge Management, Virtual Event, Ireland, October 19-23, 2020*, pp. 1365–1374. ACM, 2020b. doi: 10.1145/3340531.3411949. URL https://doi.org/10.1145/3340531.3411949.

Singh, N. and Vidyasagar, M. blars: An algorithm to infer gene regulatory networks. *IEEE/ACM transactions on computational biology and bioinformatics*, 13(2):301–314, 2015.

Su, G., Wei, D., Varshney, K. R., and Malioutov, D. M. Interpretable two-level boolean rule learning for classification. *ArXiv preprint*, abs/1511.07361, 2015. URL https://arxiv.org/abs/1511.07361.

Tenachi, W., Ibata, R., and Diakogiannis, F. I. Deep symbolic regression for physics guided by units constraints: toward the automated discovery of physical laws. *ArXiv preprint*, abs/2303.03192, 2023. URL https://arxiv.org/abs/2303.03192.

Valipour, M., You, B., Panju, M., and Ghodsi, A. Symbolicgpt: A generative transformer model for symbolic regression. *ArXiv preprint*, abs/2106.14131, 2021. URL https://arxiv.org/abs/2106.14131.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. In Guyon, I., von Luxburg, U., Bengio, S., Wallach, H. M., Fergus, R., Vishwanathan, S. V. N., and Garnett, R. (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. 5998–6008, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

Velickovic, P., Badia, A. P., Budden, D., Pascanu, R., Banino, A., Dashevskiy, M., Hadsell, R., and Blundell, C. The CLRS algorithmic reasoning benchmark. In Chaudhuri, K., Jegelka, S., Song, L., Szepesvári, C., Niu, G., and Sabato, S. (eds.), *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pp. 22084–22102. PMLR, 2022. URL https://proceedings.mlr.press/v162/velickovic22a.html.

Venkataramani, S., Sabne, A., Kozhikkottu, V., Roy, K., and Raghunathan, A. Salsa: Systematic logic synthesis of approximate circuits. In *Proceedings of the 49th Annual Design Automation Conference*, pp. 796–801, 2012.

Venkataramani, S., Roy, K., and Raghunathan, A. Substitute-and-simplify: A unified design paradigm for approximate and quality configurable circuits. In *2013 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1367–1372. IEEE, 2013.

Wang, R.-S., Saadatpour, A., and Albert, R. Boolean modeling in systems biology: an overview of methodology and applications. *Physical biology*, 9(5):055001, 2012.

Wang, S., Li, B. Z., Khabsa, M., Fang, H., and Ma, H. Linformer: Self-attention with linear complexity. *ArXiv preprint*, abs/2006.04768, 2020. URL https://arxiv.org/abs/2006.04768.

Wang, T. and Rudin, C. Learning optimized or's of and's. *ArXiv preprint*, abs/1511.02210, 2015. URL https://arxiv.org/abs/1511.02210.

Wei, J., Tay, Y., Bommasani, R., Raffel, C., Zoph, B., Borgeaud, S., Yogatama, D., Bosma, M., Zhou, D., Metzler, D., Chi, E. H., Hashimoto, T., Vinyals, O., Liang, P., Dean, J., and Fedus, W. Emergent abilities of large language models, 2022a.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q. V., Zhou, D., et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837, 2022b.

Zhang, C., Raghu, M., Kleinberg, J., and Bengio, S. Pointer value retrieval: A new benchmark for understanding the limits of neural network generalization. *ArXiv preprint*, abs/2107.12580, 2021. URL https://arxiv.org/abs/2107.12580.

Zhang, Y., Backurs, A., Bubeck, S., Eldan, R., Gunasekar, S., and Wagner, T. Unveiling transformers with lego: a synthetic reasoning task. *ArXiv preprint*, abs/2206.04301, 2022. URL https://arxiv.org/abs/2206.04301.

Zhao, M., He, W., Tang, J., Zou, Q., and Guo, F. A comprehensive overview and critical evaluation of gene regulatory network inference technologies. *Briefings in Bioinformatics*, 22 (5):bbab009, 2021.

A. Details on data generation

A.1. Formula generation

To construct random unary-binary trees, we follow the steps below:

1. **Sample the input dimension** D of the function f uniformly in $[1, D_{\max}]$.
2. **Sample the number of active variables** S uniformly in $[1, \min(D, S_{\max})]$. S determines the number of variables which affect the output of f (the number of active tree inputs); the other variables are inactive. Select a set of S variables from the original D variables uniformly at random.
3. **Sample the number of binary operators** B uniformly in $[S - 1, B_{\max}]$ then sample B operators from {AND, OR} independently with equal probability.
4. **Build a binary tree** with those B nodes, using the sampling procedure of (Lample & Charton, 2020), designed to produce a diverse mix of deep and narrow versus shallow and wide trees.
5. **Negate some of the nodes** of the tree by adding NOT gates independently with probability $p_{\text{NOT}} = 1/2$.
6. **Fill in the leaves**: for each of the $B + 1$ leaves in the tree, sample independently and uniformly at random one of the variables from the set of active variables[7].
7. **Simplify** the tree using Boolean algebraic rules, as described below. This greatly reduces the number of operators, and occasionally reduces the number of active variables.

To maximize diversity, we sample large formulas (up to $B_{\max} = 500$ binary gates), which are then heavily pruned in the simplification step[8].

A.2. Formula simplification

The data generation procedure heavily relies on expression simplification. This is of utmost importance for four reasons:

- It reduces the output expression length and hence memory usage as well as increasing speed

- It improves the supervision by reducing expressions to a more canonical form, easier to guess for the model

- It encourages the model to output the simplest formula, which is a desirable property.

We use the package `boolean.py`[9] for this, which is considerably faster than `sympy` (the function `simplify_logic` of the latter has exponential complexity, and is hence only implemented for functions with less than 9 input variables).

Empirically, we found the following procedure to be optimal in terms of average length obtained after simplification:

1. Preprocess the formula by applying basic logical equivalences: double negation elimination and De Morgan's laws.

2. Parse the formula with `boolean.py` and run the `simplify()` method until it *stabilizes* (sometimes, simplify more than once is necessary)

3. Apply once again the first step

Note that this procedure drastically reduces the number of operators and renders the final distribution highly nonuniform, as shown in Fig. 8.

B. Does Boolformer memorize?

One natural question is whether our model simply performs memorization on the training set. Indeed, the number of possible functions of D variables is finite, and equal to 2^{2^D}.

---

[7]The S variables are sampled without replacement in order for all the active variables to appear in the tree.

[8]The simplification leads to a non-uniform distribution of number of operators as discussed in App. A.

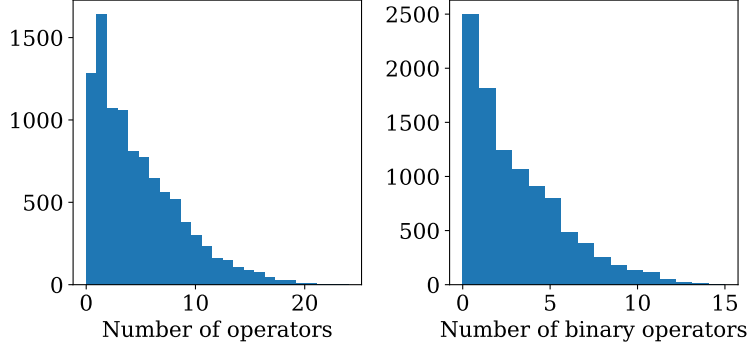[9]https://github.com/bastikr/Boolean.py

Figure 8. **Distribution of number of operators after expression simplification.** The initial number of binary operators is sampled uniformly in $[1, 500]$. The total number of examples is 10^4.

Let us first assume naively that our generator is uniform in the space of boolean functions. Since $2^{2^4} \simeq 6 \times 10^4$ (which is smaller than the number of examples seen during training) and $2^{2^5} \simeq 5.10^9$ (which is much larger), one could conclude that for $D \le 4$, all functions are memorized, whereas for $D > 4$, only a small subset of all possible functions are seen, hence memorization cannot occur.

However, the effective number of unique functions seen during training is actually smaller because our generator of random functions is nonuniform in the space of boolean functions. In this case, for which value of D does memorization become impossible? To investigate this question, for each $D < D_{\max}$, we sample $\min\left(2^{2^D}, 100\right)$ unique functions from our random generator, and count how many times their exact truth table is encountered over an epoch (300,000 examples).

Results are displayed in Fig. 9. As expected, the average number of occurences of each function decays exponentially fast, and falls to zero for $D = 7$, meaning that each function is typically unique for $D \ge 7$. Hence, memorization cannot occur for $D \ge 7$. Yet, as shown in Fig. 4, our model achieves excellent accuracies even for functions of 10 variables, which excludes memorization as a possible explanation for the ability of our model to predict logical formulas accurately.
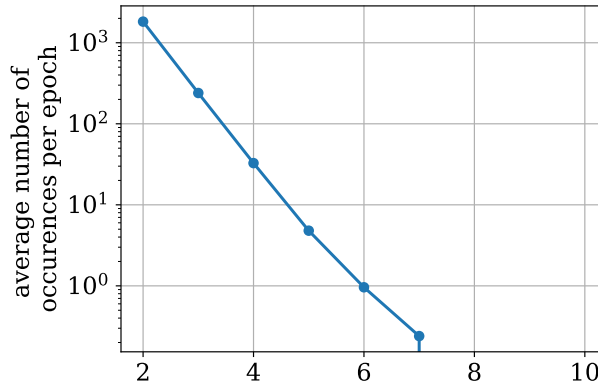
Figure 9. **Functions with 7 or more variables are typically never seen more than once during training.** We display the average number of times functions of various input dimensionalities are seen during an epoch (300,000 examples). For each point on the curve, the average is taken over $\min(2^{2^D}), 100)$ unique functions.

C. Comparison to ESPRESSO in the noiseless setting

To compare the logic synthesis capability of Boolformer with ESPRESSO, we generate 3000 formulas with our generator. The number of input variables is distributed uniformly among [1,10], however, the final distribution is a bit different, as some

of the inputs are sometimes redundant. We run these through Boolformer, which outputs valid simplifications (i.e. perfect recovery) on 96.5% of the formulas in this set. We compare performance only on this subset of formulas, the remaining 3.5% can be considered as 'failed to simplify' by Boolformer.
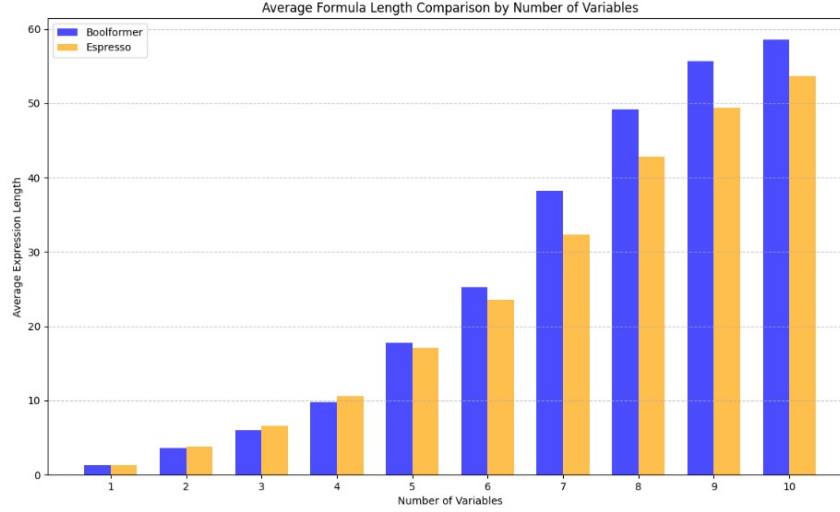

Figure 10. Histogram comparing the average length of output expression of Boolformer vs ESPRESSO.

Fig. 10 displays the average length of the simplified formula for different numbers of active variables. Overall, Boolformer does worse on this front. One reason why the average length metric is not better for Boolformer is that when it fails to find a short simplification, it will sometimes try to correct with many tokens, generating a very long answer. Indeed, if we compare the 'head-to-head' count of which formula is shorter, we obtain Fig. 11
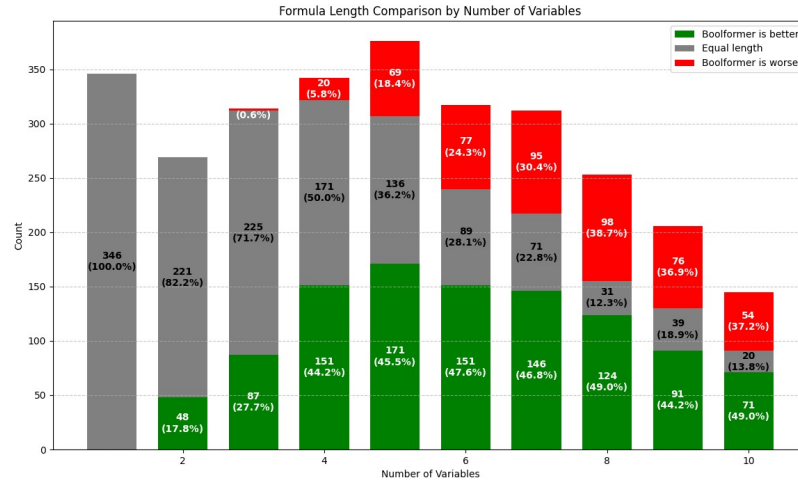
Figure 11. Histogram displaying the head-to-head simplification performance of Boolformer vs ESPRESSO. The green portion is the percentage of formulas that were better simplified by Boolformer, the gray where both methods found equal length formulas, and the red where ESPRESSO did better. Formula lengths below 5 should be mostly disregarded, as Boolformer is potentially able to 'memorize' the dataset at these lengths (see Fig 9). However, we see that for longer formulas, when Boolformer succeeds, it is often shorter than ESPRESSO.

Looking at Fig.11, Boolformer simplifications seem to be better than ESPRESSO a majority of the time, though not always. This conclusively shows that what Boolformer does is at least non-trivial: it does not consist of simple combinations (e.g. Disjunctive Normal Form, DNF) to craft a formula from its truth table. Additionally, its simplification capabilities extend beyond memorization, as they still produce good results for a number of variables above 7. That being said, the output space of Boolformer is bigger than ESPRESSO, as the former is allowed to output any valid Boolean formula using AND, OR, and NOT operations, while the latter has to output it in DNF. Nonetheless, it is the closest comparison we were able to find, as most logic synthesis programs are focused on multi-output functions, and mostly work with And-Inverter Graphs (AIGs), which are composed of an even more restrictive set of operations (only AND and NOT).

Finally, we can also compare the inference time per formula, which gives $0.06s$ for Boolformer and $3.16s$ for ESPRESSO. Here, the big advantage is mainly due to the fact that ESPRESSO runs on CPU, while Boolformer can leverage GPU, so it is not a particularly illuminating comparison. Still, maybe in future work one could focus on the noiseless setting, using a more reasonable representation as input (such as AIG, which is widely used in modern logic synthesis tools such as ABC (ber, 2025)), and test to see whether Boolformer-like pipelines can advance the state of the art.

D. Length generalization

In this section we examine the ability of our model to length generalize. In this setting, there are two types of generalization one can define: generalization in terms of the number of inputs N, or in terms of the number of active variables S[10]. We examine length generalization in the noisy setup (see Sec. 2.2), because in the noiseless setup, the model already has access to all the truth table (increasing N does not bring any extra information), and all the variables are active (we cannot increase S as it is already equal to D).

D.1. Number of inputs

Since the input points fed to the model are permutation invariant, our model does not use any positional embeddings. Hence, not only can our model handle $N > N_{\max}$, but performance often continues to improve beyond $N_{\max}$, as we show for two datasets extracted from PMLB (Olson et al., 2017) in Fig. 12.
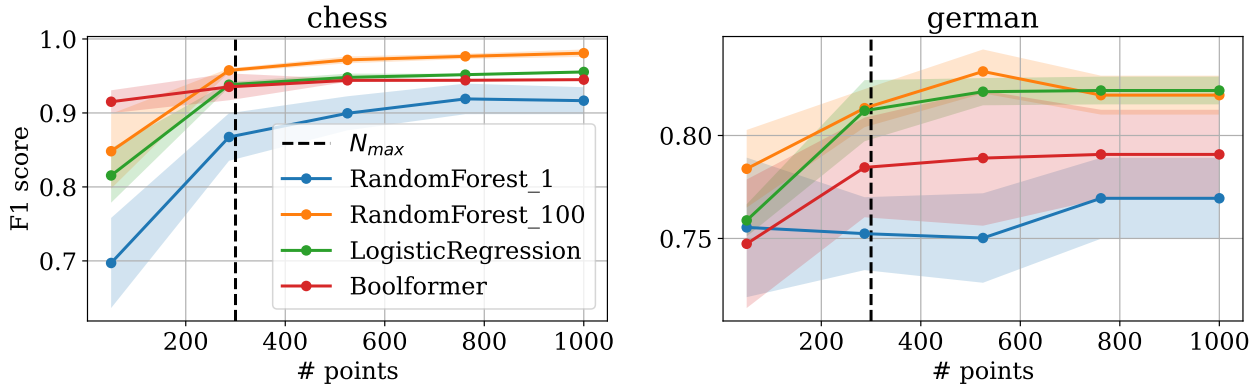

Figure 12. **Our model can length generalize in terms of sequence length.** We test a model trained with $N_{\max} = 300$ on the `chess` and `german` datasets of PMLB. Results are averaged over 10 random samplings of the input points, with the shaded areas depicting the standard deviation.

D.2. Number of variables

To assess whether our model can infer functions which contain more active variables than seen during training, we evaluated a model trained on functions with up to 6 active variables on functions with 7 or more active variables. We provided the model with the truth table of two very simple functions: the OR and AND of the first $S \geq 7$ variables. We observe that the

---

[10]Note that our model cannot generalize to a problem of higher dimensionality D than seen during training, as its vocabulary only contains the names of variables ranging from x_1 to $x_{D_{\max}}$.

model succeeds for $S = 7$, but fails for $S \geq 8$, where it only includes the first 7 variables in the OR / AND. Hence, the model can length generalize to a small extent in terms of number of active variables, but less easily than in terms of number of inputs. We hypothesize that proper length generalization could be achieved by "priming", i.e. adding even a small number of "long" examples, as performed in (Jelassi et al., 2023).

E. Formulas predicted for logical circuits

In Figs. 13 and 14, we show examples of some common arithmetic and logical formulas predicted by our model in the noiseless regime, with a beam size of 100. In all cases, we increase the dimensionality of the problem until the failure point of Boolformer.
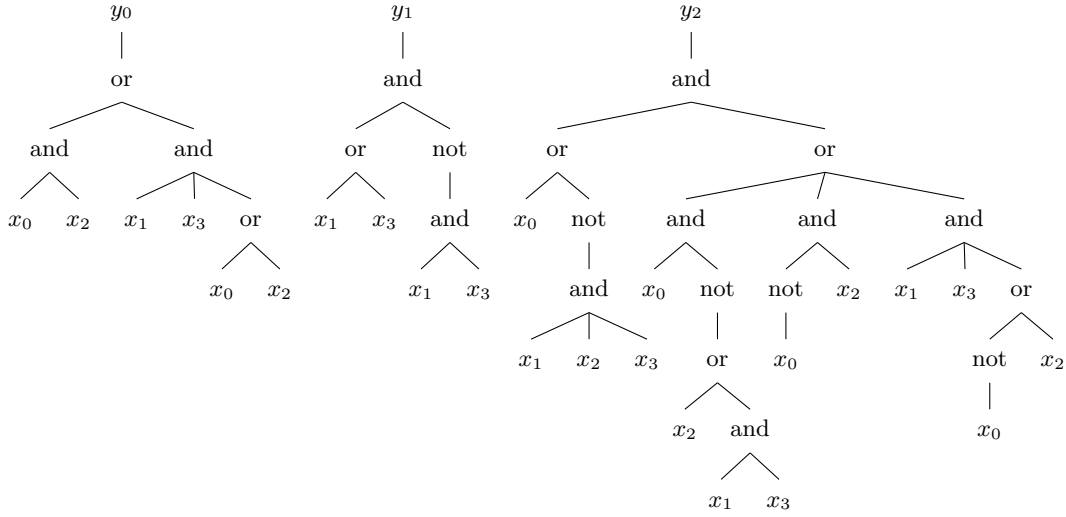
(a) Addition of two 2-bit numbers: $y_0 y_1 y_2 = (x_0 x_1) + (x_2 x_3)$. All formulas are correct.
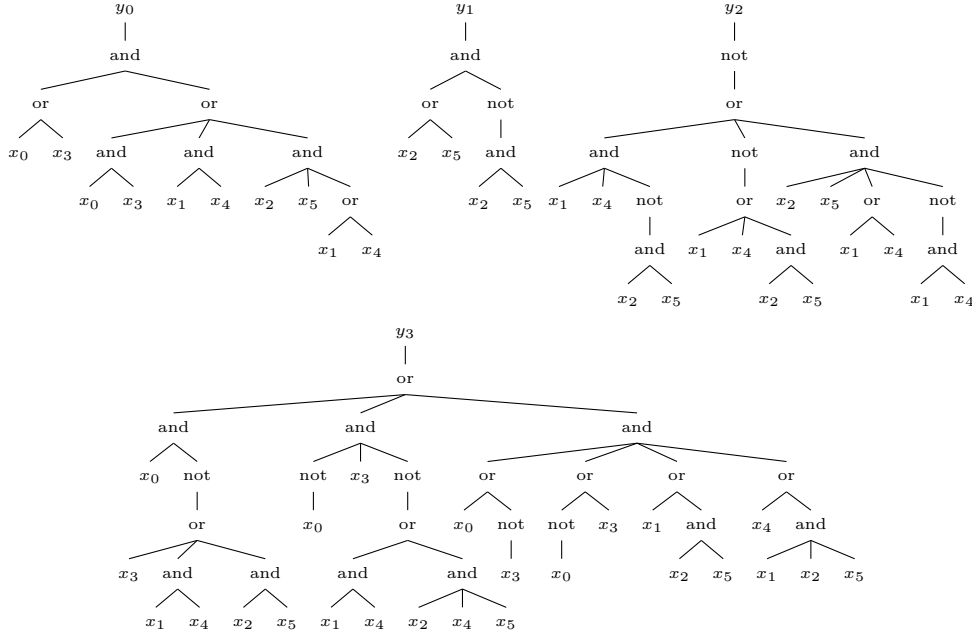
(b) Addition of two 3-bit numbers: $y_0 y_1 y_2 y_3 = (x_0 x_1 x_2) + (x_3 x_4 x_5)$. All formulas are correct, except y_3 which gets an error of 3%.

F. Formulas predicted for PMLB datasets

In Fig. 15, we report a few examples of boolean formulas predicted for the PMLB datasets in Fig. 6. In each case, we also report the F1 scores of logistic regression and random forests with 100 estimators.
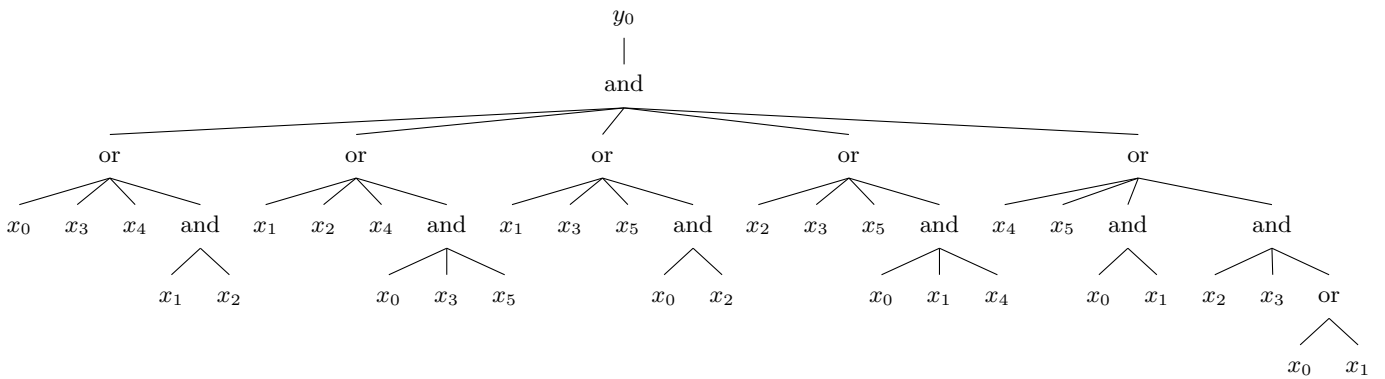
(c) Multiplication of two 2-bit numbers: $y_0 y_1 y_2 y_3 = (x_0 x_1) \times (x_2 x_3)$. All formulas are correct.

(d) Multiplication of two 3-bit numbers: $y_0 y_1 y_2 y_3 y_4 y_5 = (x_0 x_1 x_2) \times (x_3 x_4 x_5)$. All formulas are correct, except y_4 which gets an error of 5%.

Figure 13. **Some arithmetic formulas predicted by our model.**
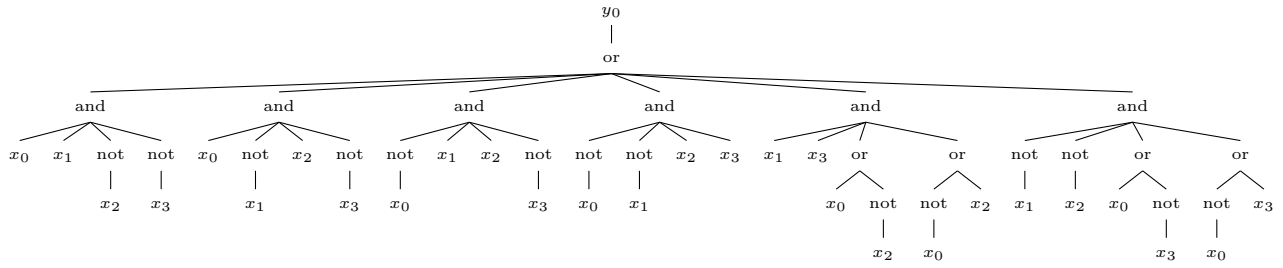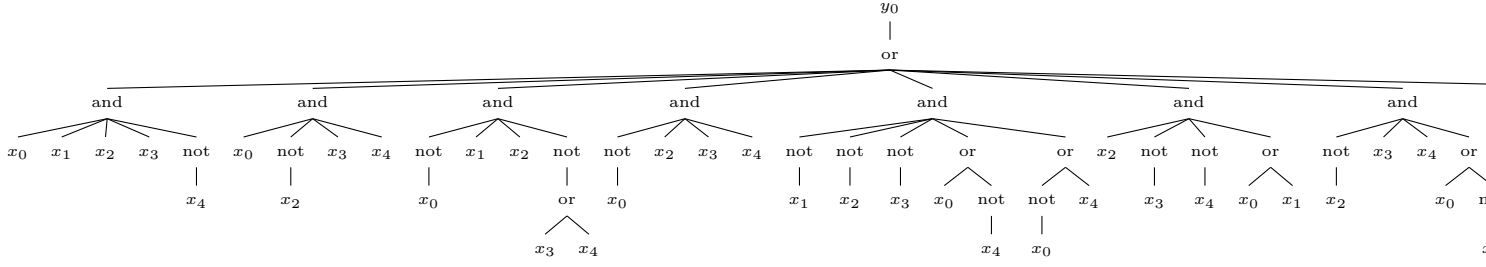
(a) 4-parity: 0% error.

(b) 5-parity: 28% error.
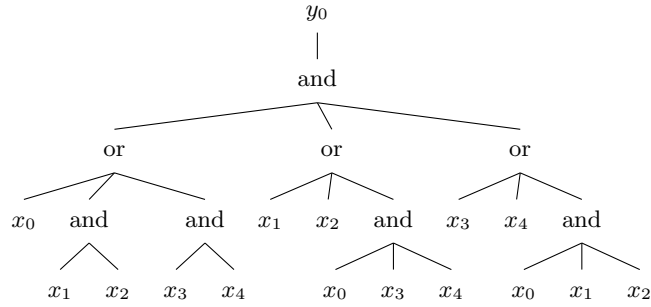
(c) 5-majority: 0% error.

(d) 6-majority: 6% error.

Figure 14. **Some logical functions predicted by our model.**

not

and

not   or

A20   A32   A09   not

or

A20   not   A34

A31
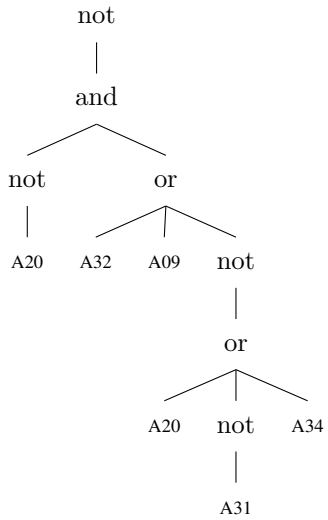
(a) chess. F1: 0.947. LogReg: 0.958. RF: 0.987.

not

or

surgery=0   not   not   and

surgery=1   outcome=0   or   or

surgery=0   not   surgery=0   outcome=0

surgery=1
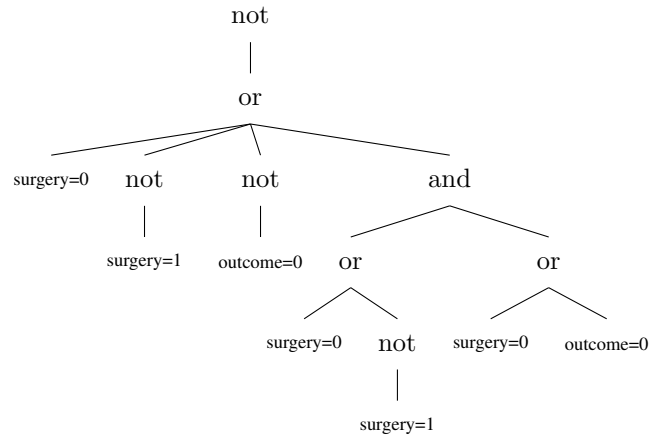
(b) horse colic. F1: 0.900. LogReg: 0.822. RF: 0.861.

not

or

pension=2   not

or

contribution to dental plan=1   pension=0   not

or

not   and

duration=2   not   pension=0

contribution to dental plan=1
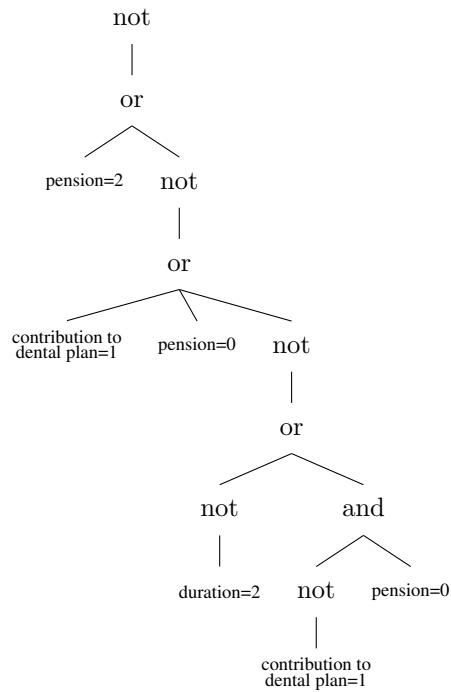
(c) labor. F1: 0.960. LogReg: 1.000. RF: 1.000.
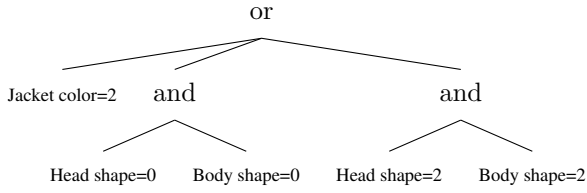
(d) monk1. F1: 0.915. LogReg: 0.732. RF: 1.000.
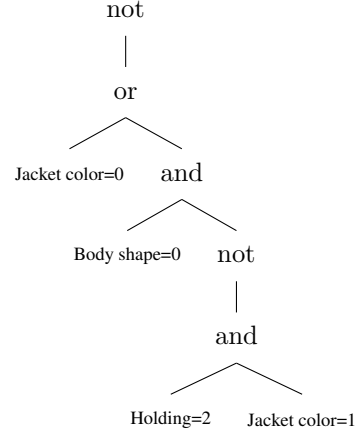
(e) monk3. F1: 1.000. LogReg: 0.985. RF: 0.993.
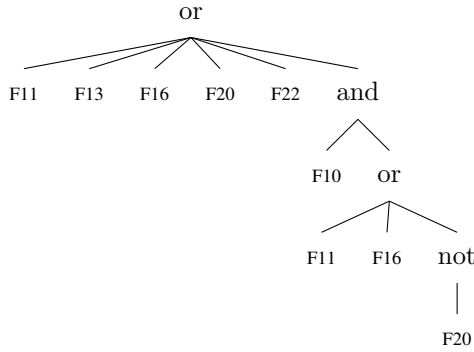
(f) spect. F1: 0.919. LogReg: 0.930. RF: 0.909.
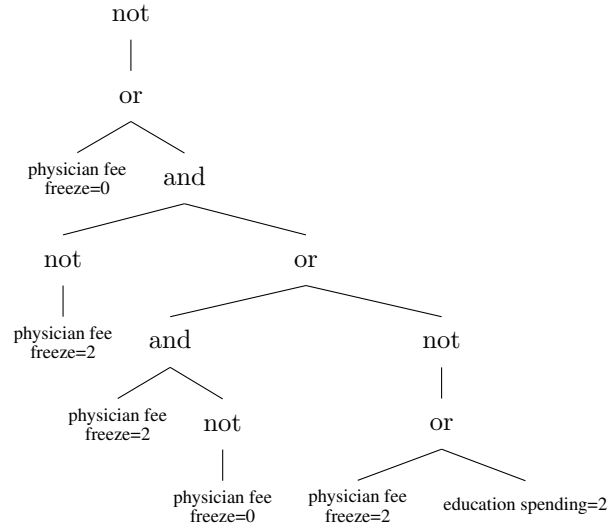
(g) vote. F1: 0.971. LogReg: 0.974. RF: 0.974.

Figure 15. **Some logical formulas predicted by our noisy model for some binary classification PMLB datasets.** In each case, we report the name of the dataset and the F1 score of the Boolformer, logistic regression and random forest in the caption.

G. Additional results on gene regulatory network inference

In this section, we give a very brief overview of the field of GRN inference and present additional results using our Boolformer.

G.1. A brief overview of GRNs

Inferring the behavior of GRNs is a central problem in computational biology, which consists in deciphering the activation or inhibition of one gene by another gene from a set of noisy observations. This task is very challenging due to the low signal-to-noise ratios recorded in biological systems, and the difficulty to obtain temporal ordering and ground truth networks.

GRN algorithms can infer relationships between the genes based on static observations (Singh & Vidyasagar, 2015; Haury et al., 2012; Huynh-Thu et al., 2010), or on input time-series recordings (Adabor & Acquaah-Mensah, 2019; Huynh-Thu & Geurts, 2018), and can either infer correlational relationships, i.e. undirected graphs, or causal relationships, i.e. directed graphs – the latter being more useful, but harder to obtain.

We focus on methods which model the dynamics of GRNs via Boolean networks: REVEAL (Liang et al., 1998), Best-Fit (Lähdesmäki et al., 2003), MIBNI (Barman & Kwon, 2017), GABNI (Barman & Kwon, 2018) and ATEN (Shi et al., 2020a). We evaluate our approach on the recent benchmark from (Pušnik et al., 2022).

G.2. Additional results

The benchmark studied in the main text assesses both dynamical prediction (how well the model predicts the dynamics of the network) and structural prediction (how well the model predicts the Boolean functions compared to the ground truth). Structural prediction is framed as the binary classification task of predicting whether variable i influences variable j, and can hence be evaluated by several binary classification metrics, defined below[11]:

$$\text{Acc} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}, \quad \text{Pre} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \quad \text{Rec} = \frac{\text{TP}}{\text{TP} + \text{FN}}, \quad \text{F1} = 2\frac{\text{Pre} \cdot \text{Rec}}{\text{Pre} + \text{Rec}},$$
$$\text{MCC} = \frac{\text{TP} \cdot \text{TN} - \text{FP} \cdot \text{FN}}{\sqrt{(\text{TP} + \text{FP})(\text{TP} + \text{FN})(\text{TN} + \text{FP})(\text{TN} + \text{FN})}}, \text{BM} = \frac{\text{TP}}{\text{TP} + \text{FN}} + \frac{\text{TN}}{\text{TN} + \text{FP}} - 1$$

Where TP, TN are True Positive and True Negative, while FP FN are False Positive and False Negative. We report these metrics in Fig. 16.

---

[11]The authors of the benchmark consider the two latter to be the best metrics to give a comprehensive view on the classifier performance for this task.
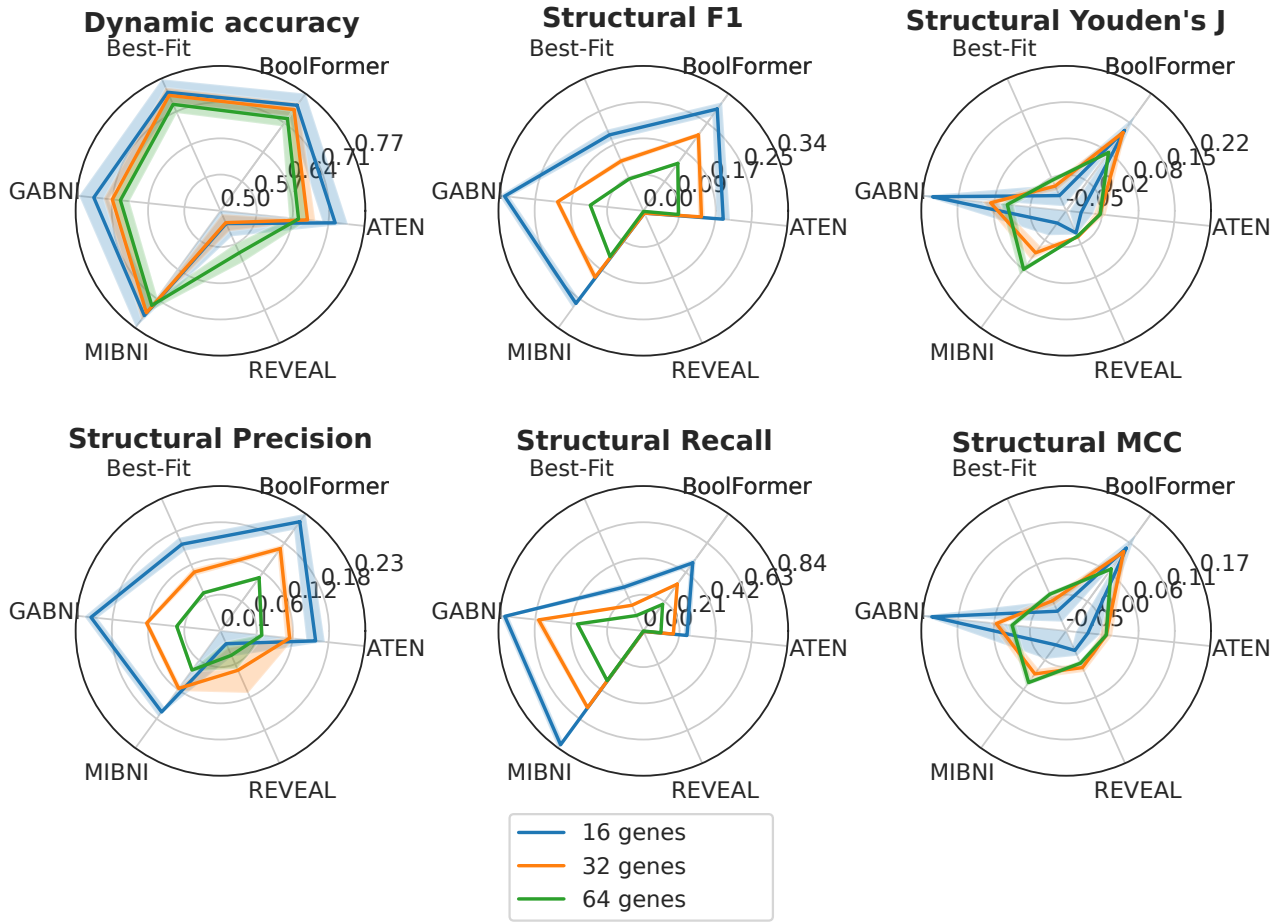
Figure 16. **Binary classification metrics used in the gene regulatory network benchmark.** The competitors and metrics are taken from the recent benchmark of (Pušnik et al., 2022), and described in Sec. 4.3.

H. Exploring the beam candidates

In this section, we explore the beam candidates produced by the Boolformer. In Fig. 17, we show the 8 top-ranked candidates when predicting a simple logic function, the 2-comparator. We see that all candidates perfectly match the ground truth, but have different structure.

I. Attention maps

In Fig. 18, we show the attention maps produced by our model when presented three truth tables: (a) that of the 4-digit multiplier, (b) that of the 4-parity function and (c) a random truth table. Each panel corresponds to a different layer and head of the model.

Each attention map is an $N \times N$ matrix, where N is the number of input points. The element (i, j) represents the attention score between tokens i and j, and is marked by the colormap, from blue (0) to yellow (1). Here the tokens are ordered from left to right by lexicographic order: 0000, 0001, 0010, ..., 1111. In this particular order, many interesting structures appear, especially for the first two functions which are non-random. For example, for the 4-parity function, the anti-diagonal attention map of (head 8, layer 7) indicates that the model compares antipodal points in the hypercube: (0000, 1111), (0011, 1100)...

As for the 4-digit multiplier, some attention heads have hadamard-like structure (e.g. heads 3,4,5 of layer 8), some have block-structured checkboard patterns (e.g. head 12 of layer 5), and many heads put most attention weight on the final input, 1111, which is more informative (e.g. head 15 of layer 3).
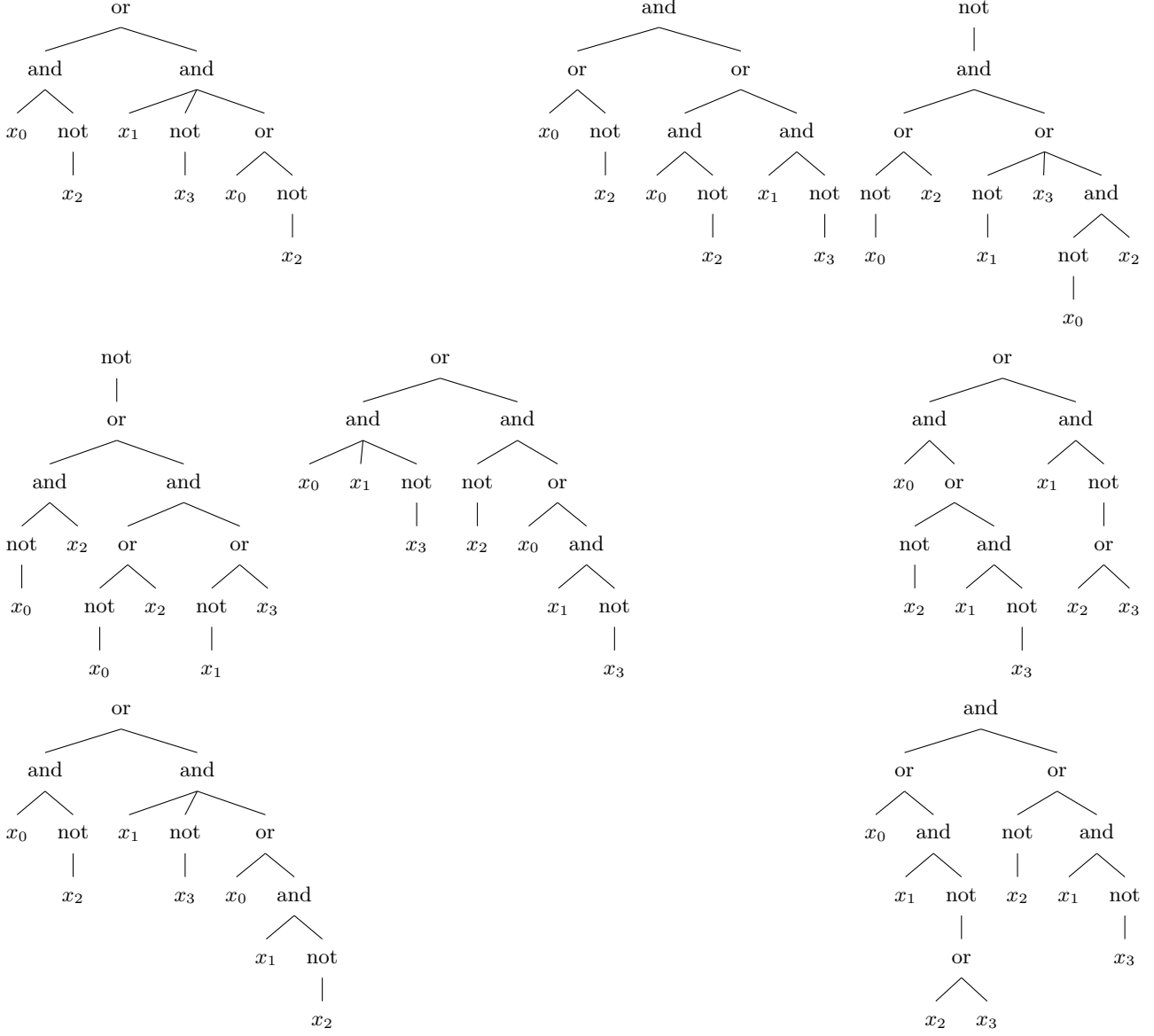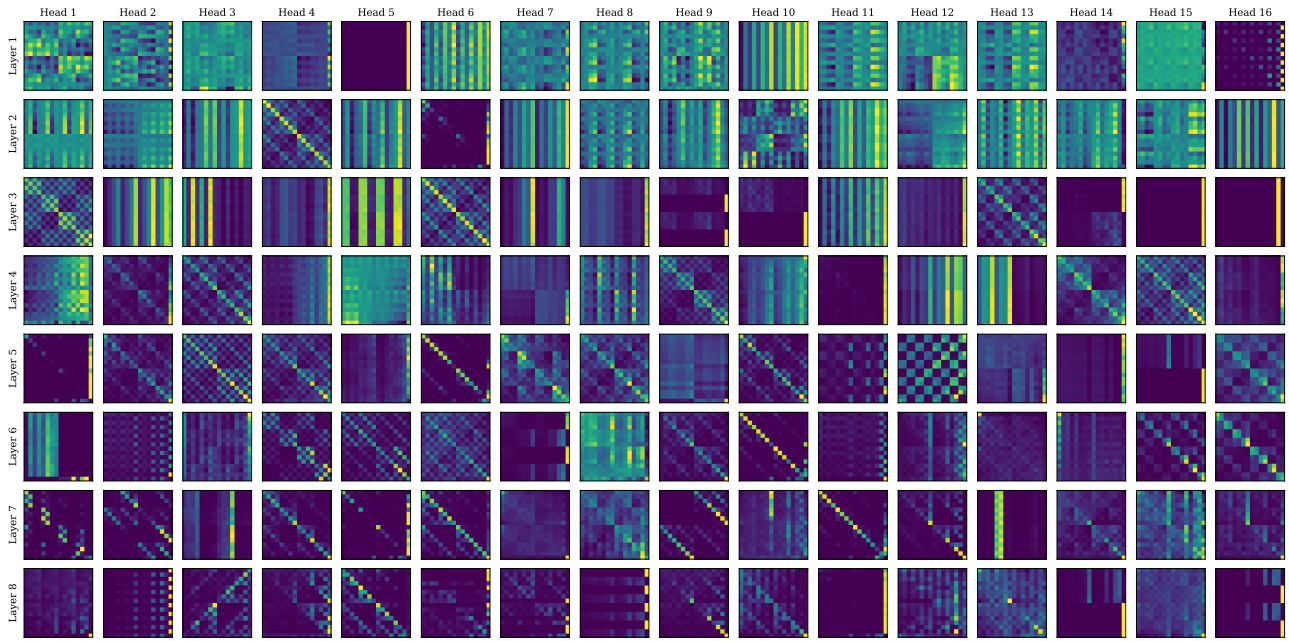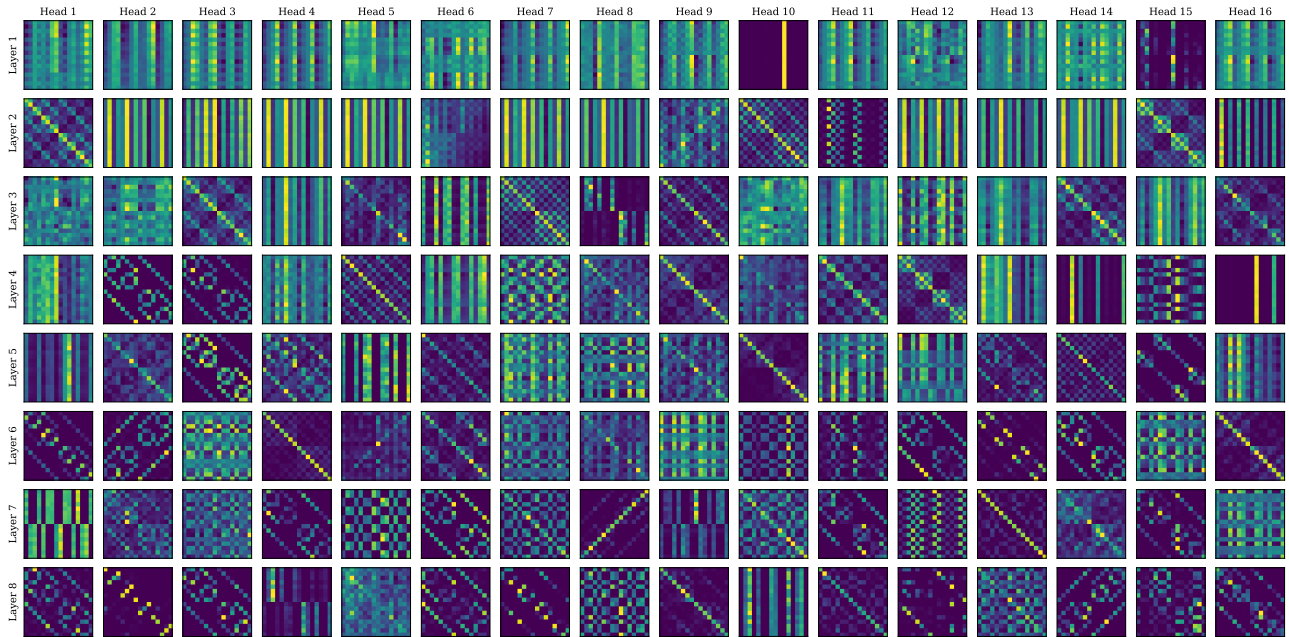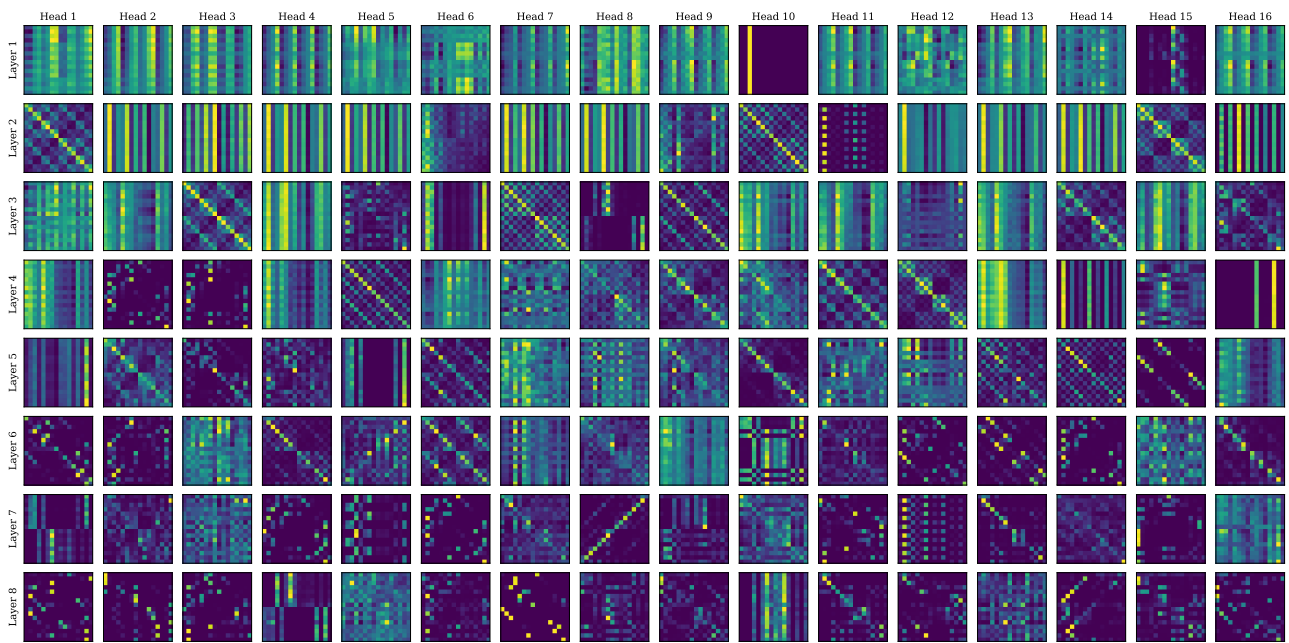
Figure 17. **Beam search reveals equivalent formulas**. We show the first 8 beam candidates for the 2-comparator, which given two 2-bit numbers $a = (x_0x_1)$ and $b = (x_2x_3)$, returns 1 if $a > b$, 0 otherwise. All candidates perfectly match the ground truth.

(a) 4-digit multiplier

(b) 4-parity

(c) 4d random data

Figure 18. **The attention maps reveal intricate analysis.** See Sec. I for more details on this figure.