The Feasibility of Solving the Satisfiability Problem Using Various Machine Learning Approaches

Lan Zhang, Lei Hu and Lina Yu

***Abstract*—In this paper, we propose a novel approach to solving theorem proving problems without relying on any deduction method. We transform logical formulas into numbers, vectors and matrices, and feed the corresponding data into various machine learning algorithms to predict their satisfiability. Here, we introduce ProverX, a novel theorem prover that utilizes various binary classification algorithms, ranging from traditional machine learning to deep learning, to tackle the satisfiability (SAT) problem of propositional logic. Empirical experiments were conducted to evaluate the performance of ProverX using datasets we generated. ProverX achieved accuracy rates ranging from 81.8% to 98.7%, demonstrating a remarkable speedup of almost 180 times compared to CTL-RP, a highly efficient prover. These results demonstrate the feasibility of replacing deduction with learning in theorem proving, opening promising avenues for further exploration in more complex logics, e.g., Modal Logic, Coalition Logic, Propositional Linear-Time Temporal Logic, Computation Tree Logic, etc., provided that their resolution methods exist. We also present a detailed analysis of the best-performing machine learning approaches for this task and introduce a new algorithm, IPDA, which has the potential to further enhance performance.**

***Index Terms*—machine learning, theorem proving, resolution calculus, satisfiability problem**

I. Introduction

In the field of formal verification, the satisfiability (SAT) problem [1] lies at the heart of formal verification, its solutions powering many real world applications, such as software verification [2], circuit design [1], scheduling problems, planning problems, optimization problems and more.

Currently, most of the decision procedures or calculi created to solve SAT problems are based on some form of logic deduction, such as, tableau [3], [4], natural deduction [5], resolution [6], etc. These techniques, while powerful, can face limitations in scalability and speed, particularly for increasingly complex formulas.

In this paper, we propose a machine learning based approach [7] to execute the theorem proving process for the SAT problem of Propositional Logic (PL), in particular. We introduce ProverX, a novel theorem prover, of which we developed using the machine learning based approach. For the initial version, i.e., ProverX version 1.0, the main purpose is to solve SAT problem by exploring the possibilities of employing various machine learning approaches, instead of using logical deduction.

In our past research, we have established resolution calculi for a number of logics, such as Computation Tree Logic [8], Coalition Logic [9], Alternating-Time Temporal Logic [10] and Propositional Linear-Time Logic [11]. Inspired by our past research in developing resolution calculi for diverse logics, we recognize the inherent compatibility of clausal formulas, generated during the proving process. We believe that these clauses are ideal training data for ProverX, empowering it to learn the intricate relationships between logical formulas and their satisfiability. In other words, prior to running the proving algorithm, any given original logic formula has to be transformed into an equi-satisfiable clausal normal form, which can be easily represented as vectors or two-dimensional matrices. In our perspective, these clauses are very ideal to be treated as the input data for machine learning. Assuming that the original logic formula that needs to be proven is θ, let $T(\theta)$ be the set of clauses obtained from transforming θ, where the satisfiability of θ is preserved. Then, our approach is to utilize a resolution prover to label $T(\theta)$ to be satisfiable or unsatisfiable, i.e., label '1' is obtained if no contradictions are found, meaning that there exists at least one model to satisfy formula θ or label '0' if a contradiction is found, meaning that there does not exist any model that can satisfy θ, respectively. After the labelling process, we feed $T(\theta)$ as a sample to our machine learning based prover, ProverX, as a learning process. The experiments we conducted reveal that, given that we have sufficient amount of randomly generated θ, ProverX has the ability to perform outstandingly and comparably to other existing approaches.

The rest of paper is organized as follows. In Section II, we mention a newly emerging research field that utilizes machine learning technologies to solve SAT problems. Section III provides a foundational understanding of resolution, one of the most efficient theorem proving calculi that exist. In Section IV, we present the method on how we built ProverX as well as the detailed experimental work we had conducted. Finally, in Section V, we draw our conclusions from our findings and outline the future work in regards of how to possibly expand the functionality of ProverX to other, possibly more complicated logics.

Manuscript received June 30, 2023; revised March 18, 2024. This work is supported by The Project of Young Academic Innovative Teams of Capital University of Economics and Business (QNTD202104) and the two projects of National Natural Science Foundation of China (Grant No. 61303018 and 71901154).

Lan Zhang is an Associate Professor in School of Management and Engineering, Capital University of Economics and Business, Beijing, 100070, China (email: zhang10897110@163.com)

Lei Hu is an Associate Professor in School of Management and Engineering, Capital University of Economics and Business, Beijing, 100070, China (email: hulei@cueb.edu.cn)

Lina Yu is an Assistant Professor in School of Urban Transport and Logistics, Shenzhen University of Technology, Shenzhen, 518118, China (Corresponding author, email: cueb2856@163.com)

II. Related work

Machine learning has become a growingly popular research subject in recent years. Through the applications of machine learning, many critical breakthroughs and developments have happened in the industrial fields, such as image classification, object recognition, product recommendations and many more. Despite the growing interests in it, in the field of formal method, machine learning was once deemed unsuccessful in competing with the approach of formal logics reasoning. In fact, most of the machine learning related papers, with respect to theorem proving, have only emerged in the past five years, specifically those that focus on solving SAT problems using machine learning technologies. This particular research area of 'Machine Learning for SAT problems' is named and often known as ML4SAT.

In ML4SAT, the most significant and well-known development to date is NeuroSAT [12]. NeuroSAT is a neural network-based SAT solver that was introduced in 2019. It works by training on a dataset of random SAT problems and learning to predict whether a given problem is satisfiable or unsatisfiable. If the problem turns out to be satisfiable, NeuroSAT then proceeds to use the information to search for a satisfying assignment. Despite the fact that, currently, it has yet to outperform the modern SAT solvers that employs logical deduction, NeuroSAT has its own special advantageous traits. Firstly, NeuroSAT, in particular, is able to solve larger-sized problems that the traditional SAT solvers find difficult to handle. Secondly, it is able to generalize itself to new problem domains, even if it was not previously trained on problems from the said domains. Last, but not least, NeuroSAT is known to be more energy-efficient than traditional SAT solvers, making it a good approach to adopt in the case of solving problems using mobile devices and other resource-constrained platforms.

Moving onto the next development in the research, two novel Transformer and Graph Neural Network-based solutions for SAT problems, namely TRSAT [13] and SATformer [14], were successfully implemented in 2021 and 2022, respectively. Both solutions are built based on the idea of Transformer [15], which is a very successful neural network architecture that has been shown to be effective for natural language processing and computer vision tasks. TRSAT was built to mainly focuses on a specific variant of SAT problems, the MaxSAT problem, and various experiment results have shown TRAST has quite a notable performance. SATformer, on the other hand, employs Transformer in order to learn relationship between clauses in a SAT problem. This relationship learning process allows SATformer to quickly and efficiently find a satisfying assignment for the problem, if one exists. Experiment results have shown that SATformer has the ability to outperform state-of-the-art SAT solvers on various benchmarks.

The last notable research to mention is the literature survey published by Guo et al. [16] in 2022. This survey discusses the recent five-years development of ML4SAT and presents comparisons and detailed summary of various existing solver. It further analyzes the introduction of machine learning, specifically deep learning, into SAT problems, as a new perspective that may result in the birth of an excellent and promising approach to tackle SAT problems.

In general, the recent development of modern SAT solvers that employ logical deduction has not shown any groundbreaking achievement, as this field has been quite matured. On the other hand, recent research interest in ML4SAT has invented a new and wider dimension in tackling SAT problems. While it is true that ML4SAT provers such as ProverX, NeuroSAT, TRSAT, SATformer, etc., are still undergoing further development, they all have strong potentials to push the boundary of the current research on SAT problem solvers forward.

III. Integration of resolution and machine learning

A. Resolution

In order to comprehend the details of the work we present in later section of our paper, we first give a brief explanation and definition of the resolution calculus. First and foremost, the hierarchies of the research field of resolution is defined as follows:

$$\begin{aligned} &\textit{Artificial Intelligence} \supseteq \textit{Formal Method} \supseteq \\ &\textit{Formal Verification} \supseteq \textit{Theorem Proving} \supseteq \\ &\textit{Resolution}, \end{aligned}$$

where $A \supseteq B$ means that B is the sub-field of A.

In 1965, Alan Robinson [6] proposed an extremely simple calculus, named Resolution, which contains only one inference rule in the whole system. To be able to understand this resolution rule, the following definitions are needed:

1) A *proposition* is either *true* or *false*.
2) A *literal* is either a proposition or its negation.
3) A formula in the form of $l_1 \vee l_2 \vee \ldots \vee l_n$ (where $l_1, l_2, \ldots, l_n$ are literals) is called a *clause*.

Given the above definitions, Robinson's inference rule, in the form of Propositional Logic[1], is as follows:

$$\frac{C \vee l \qquad D \vee \neg l}{C \vee D}$$

where C and D are disjunctions of literals; and $C \vee l$, $D \vee \neg l$ and $C \vee D$ are three clauses. It should be noted that clauses are sets, which means that a literal l can only occur at most once in a clause. Consequently, the resolvent of $p \vee q \vee r$ and $\neg r \vee p \vee q$ is the clause $p \vee q$ instead of $p \vee q \vee p \vee q$. It is then obvious that, this inference rule is only applicable on clauses. Thus, the formula $(p \vee q) \wedge \neg p \wedge \neg q$ cannot be proven directly, i.e., it has to be transformed first into a clause set $\{p \vee q, \neg p, \neg q\}$, before the resolution rule can be applied.

To check the satisfiability of an arbitrary formula θ, resolution requires the following two procedures:

1) Utilising logical equivalences, such as $r \vee (p \wedge q) \equiv (r \vee p) \wedge (r \vee q)$, $\neg(p \wedge q) \equiv \neg p \vee \neg q$, $\neg\neg p \equiv p$, etc., to transform θ into a set of clauses $N = \{C_i | 1 \leq i \leq n\}$ and $\theta \equiv C_1 \wedge C_2 \wedge, \ldots, \wedge C_n$, where $C_1, C_2, \ldots, C_n$ are clauses. Thus, θ and N share the same satisfiability, i.e. θ is satisfiable if, and only if, N is satisfiable.
2) While applying the resolution rule as many times as possible on set N, if an empty clause $\perp$ is obtained, then θ is unsatisfiable. Otherwise, θ is satisfiable. To

[1]In [6], the resolution rule was presented in the form of First-Order Logic. For simplicity, in this paper we use its Propositional Logic form, as Propositional Logics is a true subset of First-Order Logic.

be precise, the pseudo-code of the proving algorithm is given as below.

Algorithm 1: The $resolution\_sos$ procedure

while $N \neq \emptyset$ *and* $\perp \notin N$ **do**
  $given \leftarrow choose\_one(N)$;
  $N \leftarrow N - \{given\}$;
  $SOS \leftarrow SOS \cup \{given\}$;
  $New \leftarrow resolution(given, SOS)$;
  $New \leftarrow New - N$;
  $New \leftarrow New - SOS$;
  $N \leftarrow N \cup New$;
end

In Algorithm 1, N, SOS and New are three clause sets, where initially, N contains all the clauses, whereas SOS and New are empty sets; $given$ is a randomly-selected clause from N; $resolution(given, SOS)$ denotes the application of the inference rule on $given$ and every clause in SOS; New is a set of resolvents from $resolution(given, SOS)$.

When the 'while' loop terminates, if the empty clause $\perp$ does not exist, then N is satisfiable. Otherwise, N is unsatisfiable. It is worth noting that Algorithm 1 assumes that all clauses do not contain any repeated literals. In other words, a clause $p \vee l \vee l$ should be rewritten as $p \vee l$ and, likewise, a clause $p \vee \neg l \vee \neg l$ should be rewritten as $p \vee \neg l$, etc.

Due to its nature, resolution is referred as a machine-oriented algorithm. The process of simply repeating the application of the inference rule, mentioned above, until either an empty clause is produced or the rule application is exhausted, makes resolution very suitable to be implemented by computers. This results in a vast development of highly efficient resolution-based theorem provers, e.g., Vampire [17], SPASS [18] and E [19]. Hence, as its popularity grows, the application of resolution has been extended to more complicated logics, such as Linear-Time Temporal Logic, Computation Tree Logic and Alternating-Time Temporal Logic.

B. Machine learning and ProverX

In this section, to discover which machine learning approach works best for ProverX, we initiate our research by letting ProverX to handle the simplest satisfiability problem, which consists of only two propositions.

Let $p0$ and $p1$ be two unique propositions, from which 9 unique clauses can be derived. We represent these 9 clauses using a two-dimensional matrix with cell value ranging from 0 to 2 as given below.

$$C = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 0 & 2 \\ 1 & 0 \\ 1 & 1 \\ 1 & 2 \\ 2 & 0 \\ 2 & 1 \\ 2 & 2 \end{bmatrix}$$

Assume $C[i] = (x, y)$ is the i-th row of C, then, where x represents $p0$, we have the following principles:

1) if x is equal to 0, $p0$ is not selected for $C[i]$;
2) if x is equal to 1, $p0$ is selected for $C[i]$; and
3) if x is equal to 2, $\neg p0$ is selected for $C[i]$.

These three principles also apply to y, to which y represents $p1$. With these given principles, then each row of matrix C can be transformed into a clause as given below.

$$\begin{array}{lcll} C[0] & = & (0,0) & \rightarrow \perp \\ C[1] & = & (0,1) & \rightarrow p1 \\ C[2] & = & (0,2) & \rightarrow \neg p1 \\ C[3] & = & (1,0) & \rightarrow p0 \\ C[4] & = & (1,1) & \rightarrow p0 \vee p1 \\ C[5] & = & (1,2) & \rightarrow p0 \vee \neg p1 \\ C[6] & = & (2,0) & \rightarrow \neg p0 \\ C[7] & = & (2,1) & \rightarrow \neg p0 \vee p1 \\ C[8] & = & (2,2) & \rightarrow \neg p0 \vee \neg p1 \end{array}$$

Be aware that $C[0]$ is a special case, in which neither $p0, \neg p0, p1$ nor $\neg p1$ is selected. It contains no propositions and no literals, i.e., it is an *empty clause*, and, therefore, is often denoted with the symbol $\perp$. In general, any clause set containing an empty clause $\perp$ is considered to be unsatisfiable. However, in practice, many theorem provers do not consider this scenario for two reasons: (1) no inference is needed to be applied to obtain the unsatisfiable result; and (2) this scenario can be avoided by simply scanning the clause set linearly. In this paper, $C[0]$ will also not be considered. Thus, where NE stands for No Empty Clause, a new matrix C_{NE} with $|C_{NE}| = 8$ is defined, as follows.

$$C_{NE} = \begin{bmatrix} 0 & 1 \\ 0 & 2 \\ 1 & 0 \\ 1 & 1 \\ 1 & 2 \\ 2 & 0 \\ 2 & 1 \\ 2 & 2 \end{bmatrix}$$

Next, we define all the possible clause sets formed by C_{NE}. Let vector $V = (v_1, v_2, \ldots, v_8), v_i \in \{0, 1\}$ be the selection of clauses for a clause set S. In the case where $v_i = 1$, the i-th clause from C_{NE} is selected for S. Otherwise, it is not selected. For example, when $V = (0, 1, 0, 1, 0, 0, 0, 0)$, the second and fourth rows of C_{NE} are selected. These second and fourth rows correspond to (0, 2) and (1, 1), which are denoted by the clauses $\neg p1$ and $p0 \vee p1$ respectively, thus, giving us the clause set $S = \{\neg p1, p0 \vee p1\}$. Deriving from this property, all the possible clause sets of which can be formed from C_{NE} are given as follow:

$$C_8 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ \ldots & & & & & & & \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix},$$

where $C_8 = \{S | S \in \{0, 1\}^8\}$ and its number of rows is 256. We then extend matrix C_8 by running Algorithm 2 to form

the feature matrix and obtain a matrix X of the size 256×16. Each row of matrix X is a sample for learning and represents a set of clauses that is to be tested for its satisfiability. Each of these matrix rows has 16 features and they are denoted as $C1P0, C1P1, C2P0, C2P1, \ldots C8P0, C8P1$, where $CiPj$ is the status of proposition p_j in clause $C[i]$.

Algorithm 2: Expansion of C_8

```
for i in [0, 255] do
    for j in [0, 7] do
        if X[i, j] == 0 then
            C8[i, j] ← (0, 0)
        end
        if X[i, j] == 1 then
            C8[i, j] < −CNE[j]
        end
    end
end
```

We then proceed onto labelling and proving matrix X by using CTL-RP [20], a well-developed and well-optimized theorem prover that has been widely used in various projects [21]. The satisfiability result from X's 256 rows forms a vector $Labels = (0, 1, 1, \ldots, 0, 0)$, in which 0 denotes unsatisfiable and 1 denotes satisfiable. Then, V^T, the transpose of V, is attached to matrix X as the 17th column. And, thus, up to this point, all the pre-processing steps are completed and matrix X is ready to be inputted into the learning algorithms.

IV. Experiments and Results

A. Statistics of the data

To conduct the experiments, we utilize the theorem prover CTL-RP under 64-bits CentOS Linux to label every row of matrix X with either '1', if it is satisfiable, or '0' if it is unsatisfiable. We then employ TruthProver, another prover that we have developed, to verify the correctness of the labels previously given by CTL-RP. In the case where all the labels are verified to be correct, we then can safely assume that the classes, on which each sample is assigned to, are correct. The preliminary statistics obtained from conducting the experiments are presented in Fig 1 and Table I. Furthermore, we compose a heat chart in Fig 2, which shows the satisfiability distribution in matrix X. The light colored entities of the heat chart correspond to the satisfiable cases, and, likewise, the darker colored entities correspond to the unsatisfiable cases.

TABLE I
The Statistics of the Satisfiability

	Satisfiable	Unsatisfiable	Sum
The number	94	162	256
The ratio	36.7%	63.3%	100.0%

B. Non-deep learning algorithms

To further display the difficulty level of the satisfiability problem we are dealing with in our experiments, we pick three well-known traditional learning algorithms to classify our pre-processed data. These three algorithms are Naive Bayes Classifier (NBC), Logistic Regression (LR) and Support Vector Machine (SVM).

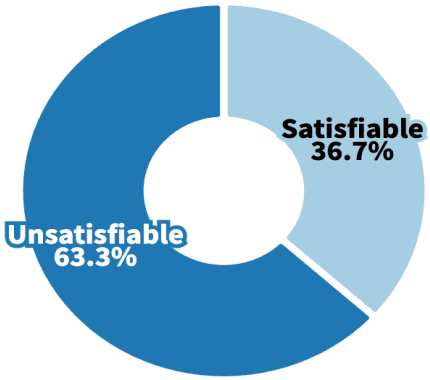

Fig. 1. The ratio of satisfiable and unsatisfiable cases

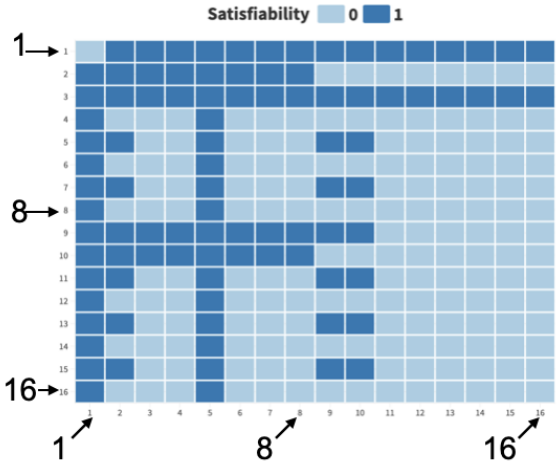

Fig. 2. The distribution of the satisfiability

The Naive Bayes Classifier (NBC) [22] stands out as one of the simplest machine learning algorithms for classification. It relies on probability theory and assumes the independence of all features. NBC is frequently chosen, especially in the initial stages of experiments, as the first approach to solving a classification problem by many researchers. This preference is due to its remarkably fast computational speed and the absence of hyperparameters to adjust. The results produced by NBC often yield sound estimations, providing valuable insights into the difficulty of the problem.

The second learning algorithm we have selected is Logistic Regression (LR) [23], a regression function commonly employed as a binary classifier. Utilizing the Sigmoid function [23], LR maps each sample to a value in between the exclusive range of (0, 1), i.e., the value 0 and 1 are not included. LR then establishes a threshold θ and defines two classes: one comprising samples with mapping values within the range $(0, \theta)$, and another comprising those within the range $[\theta, 1)$. Consequently, optimal hyperparameters can be determined by performing Grid Search on these two classes. Specifically, concerning our satisfiability problem, the hyperparameters for LR are presented in Table II.

Lastly, our third choice, Support Vector Machine (SVM) [24]–[26], operates through (1) mapping data into a higher dimensional space with a proper kernel function; and (2) splitting the high dimensional space with a hyper-

TABLE II
THE HYPER-PARAMETERS OF LR

	Scope	Best
Penalty	L1, L2, None	L2
Max_iteration	100, 1000, 10000	100
Solver	newton-cg, lbfgs, liblinear, sag, saga	liblinear

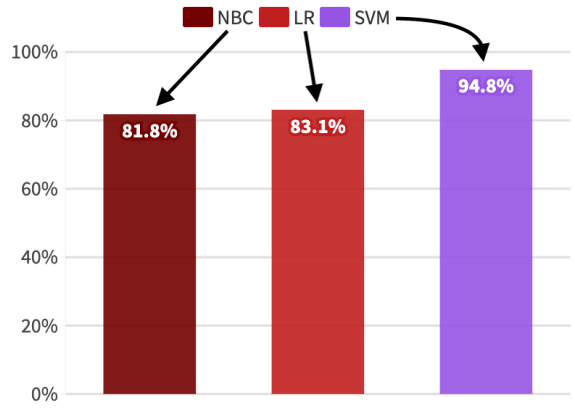

Fig. 3. The accuracy results

plane, which maximizes the sum of distances between the data point and the hyper-plane. Commonly used kernel functions for SVM are Linear, Polynomial, Sigmoid and RBF kernel functions. Similarly to LR, we determine the optimal hyperparameters for our satisfiability problem through Grid Search, as listed in Table III below.

TABLE III
THE HYPER-PARAMETERS OF SVM

	Scope	Best
C	1, 5, 10, 50, 100, 1000, 100000	50
gamma	0.001, 0.005, 0.01, 0.005, 0.01, 0.1, 1	0.1
kernel functions	linear, poly, rbf, sigmoid	rbf

We then proceed by splitting the dataset, matrix X, into a training set and a testing set with a ratio of 7:3. We train our model with the aforementioned hyperparameters, and the accuracy results produced by the three chosen learning algorithms are presented in both Table IV and Figure 3.

TABLE IV
THE ACCURACY RESULTS

Algorithm	Accuracy
NBC	81.8%
LR	83.1%
SVM	94.8%

According to these accuracy results, we observe that even a straightforward classifier, such as NBC, exhibits a relatively good performance in classifying the two classes. LR, while performing slightly better than NBC, incurs a much higher computational cost. Thus, within the context of our satisfiability problem, LR's performance is acceptable but not notably superior. SVM, on the other hand, as a highly effective learning algorithm, performs exceptionally well, achieving the highest correctness percentage among the three. Comparing the results from these three algorithms, we can strongly deduce that replacing deduction with a machine learning approach brings about a potentially promising new research development.

C. Deep learning algorithm

The popularity of the study of deep learning begun when AlexNet [27] was proposed. Initially, deep learning was mostly dominating in the fields of image recognition and image identification [28]–[30], among others. Other domains, such as strategic games, product recommendations, finance, medical diagnosis, and natural language processing, have also started to adopt deep learning approaches to solve problems. In this paper, we also attempt to apply a deep learning approach to the fields of logical deduction and theorem proving.

Due to the characteristic of our pre-processed data, where every feature (or column) represents a certain part of a logical expression and that every feature is equivalently important, there is no necessity to engineer any feature of the data any further. This characteristic, in fact, makes our pre-processed data to be quite suitable for deep learning application, as deep learning is an end-to-end algorithm.

In Figure 4, we depict the structure of our deep neural network, which is Multi-Layer Perceptrons (MLP) and is a fully connected neural network. It consists of the following elements:

- The input layer, containing 16 nodes.
- The first hidden layer, containing 64 nodes.
- The second hidden layer, containing 128 nodes.
- The third hidden layer, containing 128 nodes.
- The fourth hidden layer, containing 64 nodes.
- The output layer, containing 2 nodes.

In the experiments we conducted, once the model was trained, we used the testing data to evaluate the deep neural network and found its accuracy level up to 98.7%. We provide the relevant confusion matrix in Figure 5.

Being aware of the fact that CTL-RP is an efficiency-oriented theorem prover and is well-known for being second-ranked, when compared to all the existing provers [31], we use it, in our experiments mentioned in this paper, as a comparison against ProverX, specifically in term of processing time performance. We present the result comparison between CTL-RP and ProverX in Table V with the following clarification remarks: (1) for CTL-RP, the time spent to build the model corresponds to the time for the compiler to build the binary; whereas, (2) for ProverX, it is the time used to train the data; and (3) proving time is the sum of the time taken in proving all 256 existing cases. From these results, we can safely deduce that ProverX has remarkably outperformed CTL-RP in term of speed.

TABLE V
PROVERX VS. CTL-RP: RUNNING TIME PERFORMANCE

Prover	Building Model	Proving
ProverX	422 milliseconds	7 milliseconds
CTL-RP	1522 milliseconds	1292 milliseconds

D. Analysis of the errors

In this section, we delves into the errors made by our four machine learning algorithms, seeking patterns and exploring

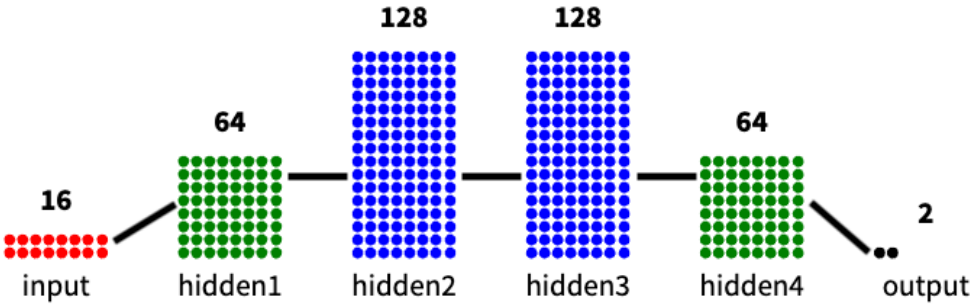

Fig. 4. The structure of our deep neural network

TABLE VI
ERRORS FOR DIFFERENT ALGORITHMS

Algorithm	The Error Set	False Positive Set	False Negative Set
NBC	24 26 45 104 150 162 168 196 200 208 224 228 232 240	24 26 162	45 104 150 168 196 200 208 224 228 232 240
LR	24 101 104 150 162 163 165 196 200 208 228 232 240	24 101 162 163 165	104 150 196 200 208 228 232 240
SVM	54 139 196 209	54 139 209	196
MLP	209	209	

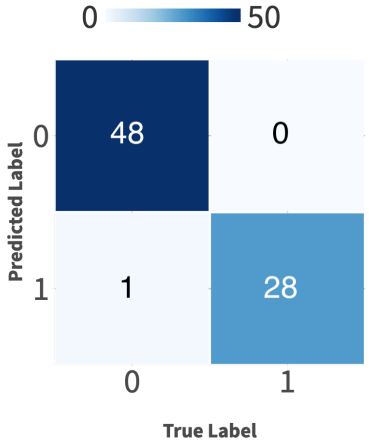

Fig. 5. The confusion matrix

paths for improvement. We investigate whether their mistakes are complementary, showcasing instances where one algorithm corrects the error of another (e.g., NBC predicted wrongly for Case 24, but SVM gets it right). Ultimately, we aim to identify challenging cases and develop a novel algorithm that synergizes machine learning with deduction to enhance performance.

In Table VI, we present the details of their incorrect predictions. Each row shows the errors of a specific algorithm:

- Column 1: Labels of all wrongly predicted cases (e.g., "24" in the first column of the first row indicates NBC incorrectly classified Case 24).
- Column 2: False Positives (FP): elements mistakenly believed to be satisfiable (e.g., "26" in the second column of the first row indicates that this case is supposed to be classified as unsatifiable, but the result produced by NBC was satisfiable.).
- Column 3: False Negatives (FN): elements wrongly predicted as unsatisfiable (e.g., Case 139 for SVM).

The relation between the three columns is the following: the set of the first column is the union of the sets from the second and third columns.

Table VI reveals interesting trends:

- Performance Improvement: Moving down each column, the set size decreases, showing performance improvement across algorithms.
- Algorithm Similarities: NBC and LR display similarities, with a bigger FN than FP set (prone to classifying satisfiable cases as unsatisfiable). Conversely, SVM and MLP exhibit the opposite trend.

TABLE VII
INTERSECTION OF WRONG PREDICTIONS

Algorithms	Intersection
NBC ∩ LR	24 104 150 162 196 200 208 228 232 240
NBC ∩ SVM	196
NBC ∩ MLP	∅
LR ∩ SVM	196
LR ∩ MLP	∅
SVM ∩ MLP	209
NBC ∩ LR ∩ SVM	196
All Four Sets	∅

Next, we study complementarity and common mistakes made by those four algorithms. Table VII explores the intersection of error sets for each pair of algorithms, revealing:

- Complementarity: SVM, MLP, and LR have fewer shared errors, suggesting their mistakes are spread out and complementary.
- High Similarity: NBC and LR share many wrong predictions, indicating potential redundancy in their approaches.
- Challenging Cases: Cases 196 (FN across all traditional algorithms) and Case 209 (FP for SVM and MLP) seem particularly difficult.

We present the most challenging Cases 196 and 209 in propositional logic form as follows.

$$\begin{array}{ll} case196 & \{\neg p0 \vee \neg p1, \neg p0, p0 \vee \neg p1\} \\ case209 & \{\neg p0 \vee \neg p1, \neg p0, \neg p1, p0 \vee p1\} \end{array}$$

By directly observing the two cases above, it is quite hard to draw any conclusion of why they are the hardest cases. Instead of relying solely on machine learning, we take a step back and consider incorporating deduction. We employ resolution on Clauses 2 and 4 of Case 209, generating a new resolvent, the process of which has been presented below.

$$\frac{\neg p0 \qquad p0 \vee p1}{p1}$$

$$case209' \qquad \{\neg p0 \vee \neg p1, \neg p0, \neg p1, p0 \vee p1\} \cup \{p1\}$$

Adding this resolvent to Case 209 creates Case 209', which both SVM and MLP correctly classify. According to this method, we have tried the similar way for other wrong cases and found the similar results. From those experiments, we believe that resolution inferences can help to improve the prediction accuracy of machine learning algorithms.

Drawing from this insight, we propose an algorithm, Iterative Prediction and Deduction Algorithm (IPDA), which blends machine learning and deduction.

Algorithm 3: IPDA

Input: input, n_step, limit, sim_threshold
Output: second

$s \leftarrow 0$;
$count \leftarrow 0$;
$first \leftarrow ML\_model.predict(input)$;
while $count < limit$ *and* $s < sim\_threshold$ **do**
  $new\_input \leftarrow resolve(input, n\_step)$;
  $second \leftarrow ML\_model.predict(new\_input)$;
  $s \leftarrow similarity(first, second)$;
  $input \leftarrow new\_input$;
  $first \leftarrow second$;
  $count \leftarrow count + 1$;
end

The two important parts of IPDA are the methods $resolve()$ and $similiarity()$.

- resolve(): Applies resolution inferences on each row of $input$ (limited to n_step steps) to generate resolvents and, then, combines $input$ with newly produced resolvents to form new_input.
- similarity(): Calculates the similarity between $first$ and $second$ and the result, s, is in the range $[0, 1]$, inclusively.

The rest of the algorithm is self-explanatory.

IPDA opens a new research direction in combining machine learning and deduction for theorem proving. Future work will implement IPDA, explore optimizing its parameters, evaluating its performance on propositional logic and more complex logics, and investigating other integration approaches.

V. Conclusion and future work

In this paper, we have demonstrated that by employing various machine learning algorithms, such as Naive Bayes Classification, Logistic Regression, Support Vector Machine, and Deep Neural Network, it is feasible to use machine learning approaches for theorem proving in SAT problems. Based on our experimental results, we believe that two possible strategies: (1) combining machine learning and logical deduction; and (2) utilizing deep learning, have the potential to yield promising breakthroughs in this field. Furthermore, considering the success and widespread use of machine learning technologies in many other fields, coupled with the positive outcomes of our experiments, it is highly likely for the deep learning approach to soon become one of the mainstream research directions in the field of theorem proving.

Based on the experimental results, ProverX demonstrates promising potential for further development. It possesses the capability to take logical formulas for resolution calculus as input and produce highly accurate satisfiability results at an excellent speed. In fact, our results indicate that the proving time spent by ProverX is 184 times faster than that of CTL-RP. This is particularly noteworthy considering that CTL-RP is written in the programming language C, while ProverX is implemented in Python. While the comparison between ProverX and traditional theorem provers reveals that ProverX consumes more time for learning and training the model, once the model is built, ProverX consistently delivers results in a relatively constant time for any input. This is because the trained model is designed to be infinitely reusable. In contrast, for traditional provers, each input is treated as a new problem, and the proving process begins anew for each input. This nature results in significant variations in proving time for different inputs.

Even so, despite its notable performance in processing speed, ProverX is by no means without drawbacks. Firstly, while resolution calculus consistently produces 100% correct results, ProverX is currently capped at a 98% accuracy rate. Hence, although ProverX can be adopted as a quick screening method in the industry or applications requiring a real-time response, it is not yet able to completely replace resolution calculus. Secondly, due to the current nature of the input, any increase in the number of propositions will significantly affect the size of the input for ProverX. In other words, the input matrix for ProverX grows in size dramatically with the increase in the number of propositions involved in the logical formulas. This translates into a significant influence on ProverX's performance.

In the future, we intend to focus on three major directions for the development of ProverX. Firstly, we aim to find a mapping approach to effectively and concisely map logical formulas without inflating the size of the input when a large number of propositions are involved. Secondly, we plan to implement the IPDA algorithm for ProverX and investigate how to effectively integrate machine learning and logical deduction. Thirdly, given the success of this methodology in propositional logic, we aspire to explore and extend the application of ProverX to higher complexity logics, such as Linear Temporal Logic [32], Computational Tree Logic [33] and Alternating-Time Temporal Logic [34]. We hope that the use of ProverX for these non-traditional logics can reduce the proving time dramatically and, therefore, result in a better usability of these complex logics in solving real-life and industrial problems, instead of merely solving imaginary examples found in textbooks.

REFERENCES

[1] H. K. Buning and T. Letterman, *Propositional Logic: Deduction and Algorithms*. New York: Cambridge University Press, 1999.

[2] K. Schneider, *Verification of Reactive Systems: Formal Methods and Algorithms*. Berlin: Springer, 2004.

[3] R. Gore, "Tableau Methods for Modal and Temporal Logics," Australian National University, Tech. Rep. TR-ARP-15-95, Nov. 1995.

[4] P. Abate and R. Goré, "The Tableaux Workbench," in *Proceedings of TABLEAUX'03*, ser. LNCS, vol. 2796. Berlin: Springer, 2003, pp. 230–236.

[5] A. Bolotov, O. Grigoriev, and V. Shangin, "Natural Deduction Calculus for Computation Tree Logic," in *Proceedings of JVA'06*. IEEE Comp. Soc. Press, 2006, pp. 175–183.

[6] J. A. Robinson, "A Machine-Oriented Logic Based on the Resolution Principle," *J. ACM*, vol. 12, no. 1, pp. 23–41, 1965.

[7] I. J. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016, http://www.deeplearningbook.org.

[8] L. Zhang, U. Hustadt, and C. Dixon, "A resolution calculus for branching-time temporal logic CTL," *ACM Transactions on Computational Logic*, vol. 15, no. 1, pp. 10:1–38, 2014.

[9] C. Nalon, L. Zhang, C. Dixon, and U. Hustadt, "A resolution-based calculus for coalition logic," *Journal of Logic and Computation*, vol. 24, no. 4, pp. 883–917, 2014.

[10] L. Zhang, "Clausal Reasoning for Branching-Time Logics," Ph.D. dissertation, University of Liverpool, Liverpool, 2011.

[11] M. Fisher, C. Dixon, and M. Peim, "Clausal Temporal Resolution," *ACM Transactions on Computational Logic*, vol. 2, no. 1, pp. 12–56, 2001.

[12] D. Selsam, M. Lamm, B. Bunz, P. Liang, L. de Moura, and D. L. Dill, "Learning a sat solver from single-bit supervision," *arXiv:1802.03685*, 2019.

[13] F. Shi, C. Lee, M. K. Bashar, N. Shukla, S.-C. Zhu, and V. Narayanan, "Transformer-based machine learning for fast sat solvers and logic synthesis," *arXiv:2107.07116*, 2021.

[14] Z. Shi, M. Li, S. Khan, H.-L. Zhen, M. Yuan, and Q. Xu, "Satformer: Transformers for sat solving," *arXiv:2209.00953*, 2022.

[15] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[16] W. Guo, J. Yan, H.-L. Zhen, X. Li, M. Yuan, and Y. Jin, "Machine learning methods in solving the boolean satisfiability problem," *arXiv:2203.04755*, 2022.

[17] A. Voronkov, "Vampire," 2020, http://www.vprover.org/index.cgi.

[18] C. Weidenbach, R. A. Schmidt, T. Hillenbrand, R. Rusev, and D. Topic, "System description: Spass version 3.0," in *Proceedings of CADE-21*, ser. LNCS, vol. 4603. Berlin: Springer, 2007, pp. 514–520.

[19] S. Schulz, "Theorem prover E," 2020, http://www4.informatik.tu-muenchen.de/ schulz/.

[20] L. Zhang, U. Hustadt, and C. Dixon, "CTL-RP: A computation tree logic resolution prover," *AI Communications*, vol. 23, no. 2-3, pp. 111–136, 2010.

[21] A. Basso and A. Bolotov, "Towards GCM Re-Configuration Extending Specification by Norms," in *Proceedings of CGW'07*. London: CoreGrid Technical Report TR-0080, 2007.

[22] J. VanderPlas, *Python Data Science Book*. California, USA: O'Reilly, 2017.

[23] P. Flach, *Machine Learning, The Art and Science of Algorithms that Make Sense of Data*. Cambridge, UK: Cambridge University Press, 2012.

[24] A. Muller and S. Guido, *Introduction to Machine Learning with Python*. California, USA: O'Reilly, 2016.

[25] Q. Zheng, X. Tian, M. Yang, and H. Su, "The email author identification system based on support vector machine (svm) and analytic hierarchy process (ahp)," *IAENG International Journal of Computer Science*, vol. 46, no. 2, pp. 178–191, 2019.

[26] W.-Z. Sun, Y. Ma, Z.-Y. Yin, J.-S. Wang, A. Gu, and F.-J. Guo, "Woa-mlsvms dirty degree identification method based on texture features of paper currency images," *IAENG International Journal of Computer Science*, vol. 48, no. 4, pp. 952–964, 2021.

[27] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., vol. 25. Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

[28] M. Zhang, "Vehicle detection method of automatic driving based on deep learning," *IAENG International Journal of Computer Science*, vol. 50, no. 1, pp. 86–93, 2023.

[29] S. Castillo, A. Bernal, and J. Rodriguez, "Object detection in digital documents based on machine learning algorithms," *IAENG International Journal of Computer Science*, vol. 50, no. 2, pp. 688–699, 2023.

[30] H. Sen, "Time series prediction based on improved deep learning," *IAENG International Journal of Computer Science*, vol. 49, no. 4, pp. 1133–1138, 2022.

[31] R. Gore, J. Thomson, and F. Widmann, "An experimental comparison of theorem provers for ctl," in *Proceedings of ISTRR'11*. Berlin: Springer, 2011, pp. 49–56.

[32] U. Hustadt and B. Konev, "TRP++: A Temporal Resolution Prover," in *Collegium Logicum*. Vienna: Kurt Gödel Society, 2004, pp. 65–79.

[33] L. Zhang, U. Hustadt, and C. Dixon, "A refined resolution calculus for CTL," in *Proceedings of CADE-22*. Berlin: Springer, 2009, pp. 245–260.

[34] R. Alur, T. A. Henzinger, and O. Kupferman, "Alternating-time temporal logic," in *Proceedings of FOCS'97*. New York: IEEE Computer Society, 1997, p. 100.