NeuReduce: Reducing Mixed Boolean-Arithmetic Expressions by Recurrent Neural Network

Weijie Feng[†], **Binbin Liu**[†], **Dongpeng Xu**[*], **Qilong Zheng**[†], **Yun Xu**[†]
[†]University of Science and Technology of China
[*]University of New Hampshire
{fengwj, robbertl}@mail.ustc.edu.cn
dongpeng.xu@unh.edu
{qlzheng, xuyun}@ustc.edu.cn

Abstract

Mixed Boolean-Arithmetic (MBA) expressions involve both arithmetic calculation (e.g., plus, minus, multiply) and bitwise computation (e.g., and, or, negate, xor). MBA expressions have been widely applied in software obfuscation, transforming programs from a simple form to a complex form. MBA expressions are challenging to be simplified, because the interleaving bitwise and arithmetic operations causing mathematical reduction laws to be ineffective. Our goal is to recover the original, simple form from an obfuscated MBA expression. In this paper, we first propose NeuReduce, a string to string method based on neural networks to automatically learn and reduce complex MBA expressions. We develop a comprehensive MBA dataset, including one million diversified MBA expression samples and corresponding simplified forms. After training on the dataset, NeuReduce can reduce complex MBA expressions to mathematically equivalent but concise forms. By comparing with three state-of-the-art MBA reduction methods, our evaluation result shows that NeuReduce outperforms all other tools in terms of accuracy, solving time, and performance overhead.

1 Introduction

Mixed Boolean-Arithmetic (MBA) expression emerges as a software obfuscation (Collberg and Nagra, 2009; Collberg et al., 2012; Ceccato, 2014; Bardin et al., 2017) technique, converting software into a syntactic different but semantic equivalent form. Software developers have broadly adopted MBA expressions obfuscation to resist malicious reverse engineering attacks or illegal cracking. For instance, software vendors (Mougey and Gabriel, 2014) and communication providers (Moghaddam et al., 2012) employ MBA obfuscation to protect critical information such as Digital Rights Management (DRM) or communication protocols and protect users' private contents.

MBA obfuscation technology draws strength from its neat design and rigorous mathematical foundation (Zhou and Zhou, 2006; Zhou et al., 2007). It transforms a simple expression into an equivalent but more complex form, which contained mixed arithmetic and bitwise calculations. However, existing mathematical reduction rules can hardly simplify complex MBA expressions, because they only fit either pure arithmetic or bitwise operation. Existing researches explore diverse solutions to conquer MBA obfuscation, including bit-blast (Eyrolles, 2017; Guinet et al., 2016), pattern matching (Eyrolles et al., 2016), and software synthesis (Blazytko et al., 2017). Nevertheless, these methods treat MBA expressions as black-boxes and neglect expressions' inner structures, which led to inevitable limitations such as low simplification accuracy or high-performance penalty.

In this paper, we propose NeuReduce[1], a novel solution that utilizes neural networks to defeat complex MBA expressions. Our proposal can take complex MBA expression input as a character string format and output the simplification results. NeuReduce leverages supervised learning to ensure the correctness and conciseness of its outputs. We also notice that no large scale or diverse MBA expressions dataset is available for training and evaluating our proposed approach. We first generate a MBA dataset consisting of 1,000,000 MBA expressions with diversified features. To the best of our knowledge, this is the largest and most diverse MBA expression dataset. Second, we implemented NeuReduce based on modern neural network models, i.e., Long Short-Term Memory, Gate Recurrent Unit, and attention-based recurrent networks. We train NeuReduce using our comprehensive MBA dataset

[1]The code, dataset and model are available at https://github.com/nhpcc502/NeuReduce.git.

Findings of the Association for Computational Linguistics: EMNLP 2020, pages 635–644

and compared its performance with state-of-the-art reduction tools. For an impartial comparison, we carefully reviewed previous researches and summarized three evaluation metrics, i.e., accuracy, complexity, and solving time, as described in Section 5. Our experiments show that NeuReduce presents a superior performance than advanced reduction tools in these three aspects.

In summary, we make the following contributions:

- We develop a large-scale MBA expression dataset, including diversified types of obfuscated MBA expressions and related reduced form. The dataset resolves the problem of lacking sufficient MBA samples to do in-depth MBA research.

- We propose a novel sequence to sequence model NeuReduce, which can help security experts analyze software obfuscated by MBA rule. To the best of our knowledge, NeuReduce is the first proposal of applying a neural network method for defeating MBA obfuscation.

- We perform a comprehensive evaluation of NeuReduce's effectiveness with other state-of-the-art methods, and the result shows that NeuReduce outperforms peer methods in various aspects.

2 Background

2.1 MBA Obfuscation

Mixed-Boolean Arithmetic (MBA) obfuscation (Zhou et al., 2007) is a concise and practical software obfuscation approach. It complicates original simple operations such as $x + y$ with complex but equivalent ones with mixed arithmetic operations (e.g., $+, -, \times$, ...) and Boolean operations (e.g., $\wedge, \vee, \neg, \oplus$, ...), which hamper reverse engieers from quickly obtaining important software information. Figure 1 presents an application of MBA obfuscation. Zhou's work proves that any simple operations such as $x - y$ or $x \wedge y$ can be transformed into complicated and equivalent MBA rules, which lays the solid mathematical foundation of MBA obfuscation. Therefore, the MBA obfuscation technique has achieved great success in software safeguards(Liem et al., 2008; Collberg et al.; Quarkslab, 2019; Irdeto, 2017).

```
int f(int x,int y)
{
    int res;
    res = x & y;



    return res;
}
```

(a) Original program.

```
int f(int x,int y)
{
    int res;
    res = 2*(x&~y)-x
        -y+4*(~x&y)
        +3*(~(x^y))
        -2*(~x)-(~(x
        &y));

    return res;
}
```

(b) Obfuscated program.

Figure 1: An example of obfuscating C source code with Mixed-Boolean Arithmetic operations. Source expression, x&y, is transformed into a complex form. After compiling, human analysts have a hard time cracking the new, obfuscated binary code.

2.2 Existing MBA Deobfuscation

Due to its simplicity and high efficiency, MBA obfuscation has been applied in software obfuscation. On the other side of the arms race, researchers have started to investigate how to simplify MBA expressions.

Arybo (Guinet et al., 2016) converts all arithmetic operations into boolean operations. It utilizes traditional math rules for Boolean simplification to reduce an intermediate Boolean expression into a bit-level symbolic expression, which represents the simplification result. Since high-performance cost caused by transforming arithmetic operations into Boolean ones, Arybo can only deal with small-size MBA expressions. Moreover, simplified results generated by Arybo is difficult to interpret by human because it is in a pure Boolean form.

SSPAM (Eyrolles et al., 2016) uses pattern matching to simplify MBA expression. SSPAM can figure out some existed real-world MBA expression cases mentioned by Mougey and Gabriel (2014). However, the effectiveness of pattern matching methods heavily relies on collected substitution rules, which restricted SSPAM from handling generic MBA expressions.

Syntia (Blazytko et al., 2017) utilizes program synthesis technique to generate a comprehensible expression for a complex MBA expression. The result shows that Syntia can successfully synthesize 89% expressions on a synthesized dataset including 500 MBA expressions. Nevertheless, Syntia cannot guarantee the correctness of generated expressions due to the uncertainty nature of program synthesis.

Other nonproprietary reduction tools such as LLVM compiler optimization (Garba and Favaro, 2019) has a limited effect on MBA reduction. Eyrolles (2017) have proven that other popular symbolic calculation software such as Maple[2], Wolfram Mathematica[3], SageMath[4], and Z3 (Moura and Bjørner, 2008) lack the capabilities to handle MBA expressions.

3 Methodology

It has been proven that the MBA deobfuscation is an NP-hard problem (Zhou et al., 2007), which means no general deterministic algorithms can solve this problem effectively. Existing methods mentioned in section 2.2 treat MBA obfuscation as a black-box, rather than understand the mechanism. To address the limitation on existing MBA deobfuscation methods, we propose NeuReduce, a novel approach based on the sequence to sequence architecture (Sutskever et al., 2014; Bahdanau et al., 2014a) with encoder-decoder (Cho et al., 2014b) to reduce MBA expressions. Considering the characteristics of the MBA reduction problem, the reasoning from one sequence to another, we review and compare several deep neural networks and adopt the most effective model as the basic module of NeuReduce. We compare four broadly used neural networks: Long Short-Term Memory (Hochreiter and Schmidhuber, 1997), Gated Recurrent Unit (Cho et al., 2014a), recurrent neural network based on the attention mechanism (Bahdanau et al., 2014b), and Transformer (Vaswani et al., 2017). The following sections elaborate the techniques in NeuReduce in details.

3.1 NeuReduce Design

We apply the Encoder-Decoder as NeuReduce's framework to implement expression to expression reduction, as shown in Figure 2. The input of NeuReduce is an arbitrary-length MBA obfuscation expression represented by a sequence. NeuReduce uses character-level one-hot encoding to encode the inputs into a matrix and feeds it into an encoder composed of recurrent neural networks. The encoder transforms the input MBA expression into a fixed-length hidden state vector through a linear layer. The decoder in NeuReduce is responsible for generating output matrices through recurrent neural networks based on the encoder's output. With the result vector, we can further reconstruct the corresponding MBA expressions through the character dictionary. In order to get the best result from NeuReduce, we adopt four neural networks as the candidates and discuss the detail of how these four models are incorporated in NeuReduce in the next two subsections.

3.2 Recurrent Architecture

LSTM is a powerful basic model for natural language processing and reaches state-of-the-art industry standards in many areas. The gate-based units endow LSTM with the power to solve the vanishing gradient problem that often occurs in RNN. With that, LSTM can capture long term dependencies and discover potential relationships between variables or operators, which can help NeuReduce to understand complicated MBA expressions.

We set an embedding layer as the input receiver and respectively used to accept complicated MBA expressions and their corresponding expected expressions in our first experiment. Two layers of LSTM with tanh activation functions are connected to the embedding layer. We use the above configuration to construct NeuReduce's encoder and decoder. A linear layer with a softmax activation function is connected to the LSTM layer for the final output channel to export the prediction result in the decoder. With the LSTM-based NeuReduce, we can encode expressions into a size-fixed one-hot encoding matrix and fed it to NeuReduce. All hyperparameters of the network are derived from grid search.

Although LSTM has a strong understanding ability of long sequence, with complex structure and numerous parameters, it usually requires numerous time and computation resources to train the model. GRU is another variant of the recurrent neural network. Compared with LSTM, GRU has a more compact structure and fewer parameters, and its performance will not be significantly reduced with the reduction of the model. To test the ability of LSTM and GRU in the same environment for reducing MBA expressions, we replace the LSTM in the recurrent layer of NeuReduce with GRU and keep other configurations unchanged.

3.3 Attention Mechanism

The Encoder-Decoder model is the most popular model structure in neural machine translation (Stahlberg, 2019) and has achieved significant per-

[2]https://www.maplesoft.com/products/maple
[3]http://www.wolfram.com/mathematica
[4]http://www.sagemath.org

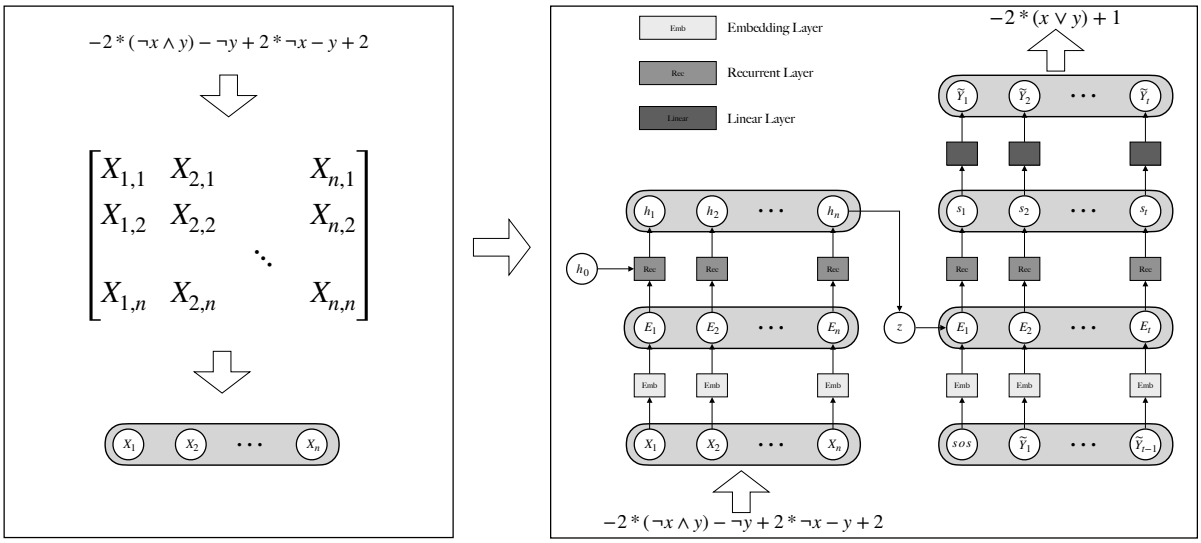

Figure 2: The architecture of NeuReduce with Encoder-Decoder.

formance. However, as mentioned in Section 3.1, the encoder encodes entire inputs into fixed-length hidden vectors and ignores the difference in priority caused by the brackets in expressions, which leads to the model not being able to make full use of the heuristic information in expressions.

In order to further improve the capabilities of NeuReduce, we draw attention to our architecture. Attention is an improvement of Encoder-Decoder models, which gifts neural networks the ability to distinguish valuable parts from the sequences. The design of attention is complicated, and the model's size increases sharply compared with LSTM. We consider attention as a comparative model for its excellent performance. For inputs of arbitrary length, we use the Embedding layer to encode the input expression into a dense vector, which reduces the number of parameters and facilitates the calculation of context vectors with attention probability weights. We use global attention with Dot-based scoring function and softmax activation layer introduced by Luong et al. (2015) to assign weights to each different character. The time-distributed layer gives final prediction results with the form of vector. The most successful application of attention is the Transformer, the most advanced natural language processing network that is entirely made up of linear layers, attention mechanisms, and normalization. We adopt it as a fundamental component of NeuReduce like the previous three networks, to verify NeuReduce's expression reasoning ability.

Complex MBA form	**Simplified form**
$(x \vee y) - (\neg x \wedge y)$	x
$4(x \wedge \neg y) + 2(\neg(x \oplus y))$ $-\neg x + 1$	$3x - 2y$
$(\neg x \wedge y) + 2(\neg(x \vee y))$ $-2(\neg x) - x$	$-(x \vee y)$
$2(\neg x \vee \neg y \vee z) - \neg y$ $-(\neg x \wedge \neg z) - (\neg x \wedge \neg z)$ $+(\neg x \wedge \neg y \vee \neg z)$	$x \vee y \vee z$

Table 1: Examples from our dataset.

4 MBA Dataset

NeuReduce requires a large-scale dataset to train for good performance. Unfortunately, existing MBA researchers only contributed a few MBA examples. We collected all existing specimens and found they are insufficient for training and evaluating NeuReduce. Therefore, we extend the algorithm introduced by Zhou et al. (2007) to build a large-scale, diversified MBA dataset. Our dataset includes 1,000,000 MBA samples, and each sample comprises the complex MBA form and the corresponding simple form. The complex MBA expression is guaranteed to be equivalent to the simple form by the theoretical foundation. Table 1 shows several examples in our dataset. More detailed information of the dataset is discussed as follows.

MBA Generation Approach. Zhou et al. (2007)'s work described a high-level principle for constructing MBA obfuscation rules from the truth

tables and the linear equation system. However, their work did not answer practical questions when building a large scale of MBA transformation rules, such as the number of variables in one expression, the length of the MBA corpus, or the cost of generation.

Enlightening by the existing work, we design a functional toolkit for generating MBA formulas. By the theorem, a bitwise expression E_n with n variables has 2^{2^n} different reduced Boolean expression. We first synthesize the 2^{2^n} distinct Boolean expressions based on the truth tables, such as $\neg x \wedge y$, $x \oplus y$. Then we generate one identity by linear equation system. The method can ensure that the generated rules are syntactic correct and semantically equal since the solid math foundation. Moreover, we verify the equality of each rule through an SMT solver Z3 (Moura and Bjørner, 2008). One example of MBA rule generation is shown bellow,

$$M = \begin{array}{c} \begin{array}{cccccc} x & y & x \oplus y & x \vee \neg y & -1 \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \end{pmatrix} \end{array}$$

$\vec{v} = [1, -1, -1, -2, 2]^T$,$\vec{s} = M\vec{v} = 0$, MBA identity $E = x - y - (x \oplus y) - 2 * (x \vee \neg y) + 2 * (-1) = 0$, generated MBA expression $y - x = -(x \oplus y) - 2 * (x \vee \neg y) + 2 * (-1)$, $y = x - (x \oplus y) - 2 * (x \vee \neg y) + 2 * (-1)$, $2 = x - y - (x \oplus y) - 2 * (x \vee \neg y)$, etc.

Moreover, MBA expression can be generated by linear combination of multiple MBA rules, such as $x = -\neg x - 1, y = x - (x \vee \neg y) + (\neg x \vee y)$, $\Rightarrow x + y = -\neg x - 1 + x - (x \vee \neg y) + (\neg x \vee y)$.

Expression Format and Complexity. Each rule in the dataset is composed of a tuple in the form of (E_c, E_g), in which E_c represents complex MBA expression, and E_g means the related simplified result as the ground truth. Given the complexity and practicability of MBA expression, the number of different variables ranges from 2 to 10. Moreover, E_c and E_g are presented as character strings, of which the length ranges from 3 to 100(the maximum exceeds 500).

Scale. In theory, the MBA generation method described above can produce an infinite number of MBA rules. To serve the purpose of training and evaluating NeuReduce in practice, we use it to generate 1,000,000 MBA expressions. Eyrolles (2017) has proven that 2-variable and 3-variable MBA expressions are commonly used in practical software obfuscation. Therefore, we split the dataset into three parts: 800,000 samples of 2-variable and 3-variable MBA expression, the other 200,000 multiple-variable MBA expressions are for testing the model's adaptability and generality.

5 Experiment Settings

In this section, we present our experimental setup in detail, including the dataset settings, peer tool baselines, evaluation metrics, and configurations of model training.

5.1 Dataset Settings

First, we are interested in exploring NeuReduce's learning and generalization ability. We uniformly sampled MBA expression from the dataset to compose two training sets, $Train_s$ and $Train_l$. $Train_s$ includes 100,000 MBA expressions to train four different NeuReduce models, and $Train_l$ containing 1 million rules is used to test how much the performance of NeuReduce has improved with more training samples. Table 2 illustrates the statistics of the training and testing dataset. In these two training sets, we set 95% of data for training and 5% for validation. The $Test$ dataset is separately generated rather than sampled from the training dataset, which ensure that every one test sample is different from the one in training dataset. We use the following three features to measure the complexity of an MBA expression.

- **Number of variables.** The number of occurrences of the variables containing in one MBA expression
- **Number of operators.** The number of occurrences of the operators containing in a MBA expression
- **MBA Length.** The length of an MBA expression as a character string.

5.2 Peer Tools for Comparison

We investigate and collect existing start-of-the-art MBA reduction tools: Arybo [5], SSPAM [6], and Syntia [7]. We download the three open source tools

[5] https://github.com/quarkslab/arybo
[6] https://github.com/quarkslab/sspam
[7] https://github.com/RUB-SysSec/syntia

		$Train_s$		$Train_l$		$Test$	
		SRC	TRG	SRC	TRG	SRC	TRG
Size		100K	100K	1M	1M	10K	10K
# of Varis	Range	$17.00_{\pm 15.00}$	$4.50_{\pm 3.50}$	$17.00_{\pm 16.00}$	$4.50_{\pm 3.50}$	$16.50_{\pm 14.50}$	$4.50_{\pm 3.50}$
	Mean	19.19	3.87	19.19	3.87	19.16	3.91
	Std.	6.99	2.08	6.99	2.08	6.99	2.09
# of Ops	Range	$26.00_{\pm 23.00}$	$6.00_{\pm 6.00}$	$25.00_{\pm 24.00}$	$6.00_{\pm 6.00}$	$25.00_{\pm 23.00}$	$6.00_{\pm 6.00}$
	Mean	31.88	5.91	31.89	5.90	31.85	5.94
	Std.	11.78	2.57	11.79	2.57	11.78	2.60
Length	Range	$54.00_{\pm 46.00}$	$18.00_{\pm 17.00}$	$51.50_{\pm 48.50}$	$18.00_{\pm 17.00}$	$52.00_{\pm 48.00}$	$18.00_{\pm 17.00}$
	Mean	73.98	15.97	74.01	15.97	73.91	16.11
	Std.	23.93	7.86	23.94	7.85	23.96	7.93

Table 2: Statistic of the experimental datasets. SRC means complex MBA expression, TRG means the related simplification result. Range, Mean and Std. deviation are measures of the spread of complexity of the dataset.

from GitHub and run them on the same dataset as the comparison baselines. Arybo is a tool for applying Bit-Blast to simplify MBA expressions written in Python. SSPAM (Symbolic Simplification with Pattern Matching) is a Python tool which applies pattern matching to do simplification. Syntia generates input-output samples from the obfuscated code, and then produces a simple expression by MCTS(Monte Carlo Tree Search)-based program synthesis.

5.3 Evaluation Metrics

We propose three metrics—accuracy, complexity, and solving time—to evaluate the complexity of NeuReduce and baseline tools.

Accuracy. Accuracy means the expression E_p generated by the neural network is equivalent to the ground truth E_g. One case is that E_p is the same as E_g, which the output of model is correct. The others is that the format of E_p is different from E_e, we use SMT solver to check equivalence between E_p and E_g. Let C_p be the total number of samples, C_{eq} be the number of the same one as E_g, C_{sq} be the number of one that is semantically equivalence with E_g, the definition of accuracy is shown below,

$$Acc = \frac{C_{eq} + C_{sq}}{C_p} \tag{1}$$

Higher accuracy means the tool can generate more number of correct simplified expression. However, accuracy cannot reveal the comprehensive ability of one tool. For example, the bit explosion method can ensure that every one reduction expression is correct, but the result is hard for humans to understand.

Complexity. Another metric for evaluating MBA expression simplification is *complexity* or *readability*. For a reduced expression, the higher *complexity* means the lower readability for human to understand the simplification expression. We use the length of the expression (the number of characters in the string) to indicate the complexity of a expression. Shorter expression means lower complexity and higher readablity for human to understand it.

Solving Time. The last metric is to test the efficiency of a tool, the solving time of reducing a MBA expression. One MBA simplification tool is not practical due to its solving time is unbearable. We set 40 minutes as a practical timeout threshold for a simplification process. If the tool does not return one result within the period, we will label it as time out.

5.4 Training Configurations

We use the same setting to train four different neural network-based NeuReduce. Adam (Kingma and Ba, 2014) is employed as our optimizer with loss function categorical crossentropy. The initial learning rate of the model is set to 10^{-2}, and we dynamically adjust it from 10^{-2} to 10^{-6} based on the losses of validation set. We train our models on NVIDIA Titan Xp GPUs for 1000 epochs with 1024 batch size.

6 Results and Analysis

We use the small-sized training set $Train_s$ to train the four different neural networks – LSTM, GRU, Attention LSTM, and Transformer. After training, we compare the models with existing reduction

	Method	$\mathbf{C}_{eq}$	$\mathbf{C}_{sq}$	T.O.	Ratio(%)	Result Length (Average)	Solving Time (Average)
baseline	Arybo	862	0	9,138	8.62	20,618.82	640.7
	SSPAM	1,420	0	8,580	14.20	61.78	438.2
	Syntia	842	734	**0**	15.76	20.03	7.5
NeuReduce	LSTM	7,144	50	**0**	71.94	18.12	0.03
	GRU	6,432	**1,018**	**0**	75.40	**18.01**	**0.02**
	Attention LSTM	7,357	36	**0**	73.93	18.04	0.40
	Transformer	**7,796**	28	**0**	**78.24**	18.02	0.43

Table 3: Comparative evaluation results using $Test$ dataset. The models are trained on dataset $Train_s$. $\mathbf{C}_{eq}$ means prediction results are equal to ground truth, $\mathbf{C}_{sq}$ means prediction results are equal to ground truth via SMT solver validation. T.O. indicates that no reduction result is given within 40 minutes. Ratio indicates the correctness rate (calculated by equation 1) of the model's solutions. Result Length can indicate the complexity of each method's output, and Solving Time(seconds for each sample) measures the efficiency of models.

tools on the *Test* dataset, which contains $10,000$ MBA expressions and related simplified forms. The evaluated results are shown in Table 3.

Arybo does not output any wrong result, because Arybo uses the Bit-Blast method, which maps each variable to bit and then simplifies it. Although Arybo can ensure the correctness of simplified MBA expression, it suffers from high performance cost. The solving time of Arybo is up to 640s, and 90% of the MBA expressions can not be simplified in 40 minutes. Another problem with Arybo is that its reduction result is more complicated than the original one —the average length of reduction results is 20k, which is unreadable and unacceptable for security experts.

Since the simplification rules of complex MBA expressions are not included in SSPAM's pattern matching library, SSPAM cannot simplify 85% of MBA expressions on *Test* dataset.

Syntia can simplify one MBA expression in 10 seconds, but only 1576 MBA expressions can be correctly simplified by it. Syntia's output largely relies on the quality of input-output samples. Therefore, Syntia is hard to handle complex MBA expressions.

After training, NeuReduce can output grammatically correct expression in 1 second. NeuReduce can simplify at least 71% of MBA expressions on *Test* dataset, and its simplification result is acceptable for humans. From the table, the accuracy of Attention-based model is slightly lower than the one of GRU-based. From the aspect of expression representation, GRU-based NeuReduce uses a sparse 0/1 Matrix to encode expressions, while Attention mechanism uses dense vectors. The dense vector can reduce the number of model parameters, but it may lack useful information input to the model. On the other hand, Attention mechanism can effectively allocate a large weight to critical information when processing long texts and filter out useless information. However, each character is essential for a correct MBA expression. The experiment shows that Transformer-based model can simplify more MBA expressions, but GRU-based model can output expression faster.

To compare the output of these methods intuitively, we extract one MBA expression that can be simplified by all peer tools and NeuReduce from the *Test* dataset and the reduced results are shown in Table 4. Even though all methods can output a correct solution, the answers of Arybo and SSPAM are not as concise and simple as Syntia and NeuReduce. [8]

Moreover, we want to know how much the performance of NeuReduce improves when training it with more samples. We used the $Train_l$, as introduced in Section 5, to train the LSTM-based and GRU-based NeuReduce. The architecture and configuration of the NeuReduce are the same as described in Section 3. After 40 hours of training for each model, we evaluate them on the *Test* dataset. The evaluation results show that their accuracy has a great promotion than before, 96.43% accuracy for LSTM-based NeuReduce and 97.16% accuracy for GRU-based NeuReduce.

[8]The result of Arybo is a bit-vector of n-elements that is set by the user. To explain Arybo's result, an example is shown: let $y = 3$, $-1 = 1111$,then $-3 - 1 = -4$, $y = y_0y_1y_2y_3 = 0011$, the sum is 1100, which is -4 in complement representation.

MBA expression	$(x \wedge y) - (\neg x \wedge y) + (x \oplus y)$ $+3 * (\neg(x \vee y)) - (\neg(x \oplus y))$ $-(x \vee \neg y) - (\neg x) - 1$
ground truth	$-y - 1$
Arybo†	$[(y_0 + 1), (y_1 + 1), (y_2 + 1), (y_3 + 1)]$
SSPAM	$((((((x + y) - (((-x) + x) \wedge y))$ $-(3 * (x \vee y))) + (2 * (x \oplus y))) + 3)$
Syntia	$(-y - 1)$
LSTM	$-y - 1$
GRU	$-y - 1$
Attention LSTM	$-y - 1$
Transformer	y

Table 4: Comparison of simplified results. †Arybo works on 4-bit MBA expression.

7 Related Work

Recent research has applied machine learning to perform mathematical reasoning. Evans et al. (2018) shows how to use tree neural network to predict one logic entails another logic. The work is different from NeuReduce since their task is to determine the implicit relationship of two propositional logic, which is a partial order, rather than to predict the equality between two expressions.

Ling et al. (2017) and Kushman et al. (2014) uses neural networks to extract mathematical problems from text and output correct answers. Their work is more focused on natural language understanding of math problems, rather than purely reasoning the logical equivalence of different expressions.

Saxton et al. (2019) is an extensive survey of mathematical reasoning. They provide a dataset containing a variety of mathematical samples from algebra problems to probability calculation. Their work well proves that state-of-the-art neural networks can work well in mathematical reasoning problem. However, the sample of expression reduction in their work only involves simple exponential equation reduction, which is not matched to the MBA expression.

There has also been a recent interest in solving mathematical problems. Zaremba et al. (2014) shows how to use a recurrent neural network to extract mathematical identities with a novel grammar framework. Kaiser and Sutskever (2015) uses a convolutional neural network to solve the problem of addition and multiplication with excellent generalization capabilities. Selsam et al. (2018) uses a message-passing network with a bipartite graph structure to determine satisfiability in formulas of conjunctive normal form. The other relevant research works are shown in Allamanis et al. (2017); Bartosz et al. (2019); Arabshahi et al. (2018).

8 Conclusion

Mixed Boolean-Arithmetic (MBA) transformation, using arithmetic and bitwise operations to translate expressions, have been applied in software obfuscation. This paper introduces a new method, NeuReduce, to simplify complex MBA expression by recurrent neural network. Due to the insufficient number of existing MBA expressions for training our neural network, we first extend a method to generate MBA expressions and develop a large-scale MBA expression dataset, including 1,000,000 diversified complex MBA samples and their simplified expressions. Four neural network models–LSTM, GRU, Attention LSTM, Transformer–are trained and tested on the dataset. The evaluation results show that, compared with state-of-the-art tools, NeuReduce has the highest accuracy with negligible overhead. Our experiments also show that NeuReduce's performance can be further improved when training on more samples.

Acknowledgments

We thank team members from UNH SoftSec group and AnHui Province Key Laboratory of High Performance Computing (USTC) for their helpful suggestions.

References

Miltiadis Allamanis, Pankajan Chanthirasegaran, Pushmeet Kohli, and Charles Sutton. 2017. Learning continuous semantic representations of symbolic expressions. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 80–88. JMLR.org.

Forough Arabshahi, Sameer Singh, and Animashree Anandkumar. 2018. Combining symbolic expressions and black-box function evaluations for training neural programs. In *ICLR 2018*.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014a. Neural machine translation by jointly learning to align and translate.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014b. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Sébastien Bardin, Robin David, and Jean-Yves Marion. 2017. Backward-Bounded DSE: Targeting Infeasibility Questions on Obfuscated Codes. In *Proceed-*

ings of the 38th IEEE Symposium on Security and Privacy (S&P'17).

Piotrowski Bartosz, Urban Josef, Chad E. Brown, and Cezary Kaliszyk. 2019. Can neural networks learn symbolic rewriting? https://arxiv.org/pdf/1912.01412.pdf.

Tim Blazytko, Moritz Contag, Cornelius Aschermann, and Thorsten Holz. 2017. Syntia: Synthesizing the semantics of obfuscated code. In *26th USENIX Security Symposium*, pages 643–659.

Mariano Ceccato. 2014. On the need for more human studies to assess software protection. In *Workshop on Continuously Upgradeable Software Security and Protection*, pages 55–56.

Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014a. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014b. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

Christian Collberg, Sam Martin, Jonathan Myers, and Jasvir Nagra. 2012. Distributed Application Tamper Detection via Continuous Software Updates. In *Proceedings of the 28th Annual Computer Security Applications Conference*, ACSAC '12.

Christian Collberg, Sam Martin, Jonathan Myers, and Bill Zimmerman. Documentation for data encodings in tigress. http://tigress.cs.arizona.edu/transformPage/docs/encodeArithmetic.

Christian Collberg and Jasvir Nagra. 2009. *Surreptitious Software: Obfuscation, Watermarking, and Tamperproofing for Software Protection*, chapter 4.4. Addison-Wesley Professional.

Richard Evans, David Saxton, David Amos, Pushmeet Kohli, and Edward Grefenstette. 2018. Can neural networks understand logical entailment? *arXiv preprint arXiv:1802.08535*.

Ninon Eyrolles. 2017. *Obfuscation with Mixed Boolean-Arithmetic Expressions: reconstruction, analysis and simplification tools*. Ph.D. thesis, Université Paris-Saclay.

Ninon Eyrolles, Louis Goubin, and Marion Videau. 2016. Defeating mba-based obfuscation. In *Proceedings of the 2016 ACM Workshop on Software PROtection*, pages 27–38. ACM.

Peter Garba and Matteo Favaro. 2019. Saturn-software deobfuscation framework based on llvm. In *Proceedings of the 3rd ACM Workshop on Software Protection*, pages 27–38. ACM.

Adrien Guinet, Ninon Eyrolles, and Marion Videau. 2016. Arybo: Manipulation, canonicalization and identification of mixed boolean-arithmetic symbolic expressions.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Irdeto. 2017. Irdeto cloaked ca: a secure, flexible and cost-effective conditional access system. www.irdeto.com.

Łukasz Kaiser and Ilya Sutskever. 2015. Neural gpus learn algorithms. *arXiv preprint arXiv:1511.08228*.

Diederik P. Kingma and Jimmy Lei Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Nate Kushman, Yoav Artzi, Luke Zettlemoyer, and Regina Barzilay. 2014. Learning to automatically solve algebra word problems. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 271–281.

Clifford Liem, Yuan Xiang Gu, and Harold Johnson. 2008. A compiler-based infrastructure for software-protection. In *Proceedings of the 3rd ACM SIGPLAN Workshop on Programming Languages and Analysis for Security (PLAS'08)*.

Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. 2017. Program induction by rationale generation: Learning to solve and explain algebraic word problems. *arXiv preprint arXiv:1705.04146*.

Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*.

Hooman Mohajeri Moghaddam, Baiyu Li, Mohammad Derakhshani, and Ian Goldberg. 2012. Skypemorph: Protocol obfuscation for tor bridges. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 97–108. ACM.

C Mougey and F Gabriel. 2014. Drm obfuscation versus auxiliary attacks.

Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: An efficient smt solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer.

Quarkslab. 2019. Epona application protection v1.5. https://epona.quarkslab.com.

David Saxton, Edward Grefenstette, Felix Hill, and Pushmeet Kohli. 2019. Analysing mathematical reasoning abilities of neural models. *arXiv preprint arXiv:1904.01557*.

Daniel Selsam, Matthew Lamm, Benedikt Bünz, Percy Liang, Leonardo de Moura, and David L. Dill. 2018. Learning a sat solver from single-bit supervision. *arXiv preprint arXiv:1802.03685*.

Felix Stahlberg. 2019. Neural machine translation: A review. *arXiv preprint arXiv:1912.02047*.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc.

Wojciech Zaremba, Karol Kurach, and Rob Fergus. 2014. Learning to discover efficient mathematical identities. In *Advances in Neural Information Processing Systems*, pages 1278–1286.

Yongxin Zhou, Alec Main, Yuan X Gu, and Harold Johnson. 2007. Information hiding in software with mixed boolean-arithmetic transforms. In *International Workshop on Information Security Applications*, pages 61–75. Springer.

Yongxin Zhou and Alec Zhou. 2006. Diversity via code transformations: A solution for ngna renewable security. *NCTA-The National Show*.