On a formal verification of machine learning systems

Dmitry Namiot, Eugene Ilyushin, Ivan Chizhov

***Abstract*— The paper deals with the issues of formal verification of machine learning systems. With the growth of the introduction of systems based on machine learning in the so-called critical systems (systems with a very high cost of erroneous decisions and actions), the demand for confirmation of the stability of such systems is growing. How will the built machine learning system perform on data that is different from the set on which it was trained? Is it possible to somehow verify or even prove that the behavior of the system, which was demonstrated on the initial dataset, will always remain so? There are different ways to try to do this. The article provides an overview of existing approaches to formal verification. All the considered approaches already have practical applications, but the main question that remains open is scaling. How applicable are these approaches to modern networks with millions and even billions of parameters?**

***Keywords*— machine learning; formal verification; robust models**

I. INTRODUCTION

This article is an extended presentation of the report at the DCCN 2021 conference [1]. The article is a continuation of a series of publications devoted to sustainable machine learning models [2, 3, 4]. It was prepared as part of the project of the Department of Information Security of the Faculty of Computer Science of Lomonosov Moscow State University on the creation and development of the master's program "Artificial Intelligence in Cybersecurity" [5].

Neural networks and machine learning are among the most successful technologies today, which are usually referred to as the direction of artificial intelligence. At the same time, from the very beginning of the use of these technologies, the issue of justifying the solutions obtained has never been at the forefront. On the contrary, machine learning gained its popularity precisely for solving problems where it was impossible (or it was too difficult) to offer an analytical solution or a deterministic algorithm. The results were initially seen as some kind of magical black box. And only then, there were questions of explaining the solutions obtained [6]. In fact, these explanations are needed just in order to be able to evaluate the data obtained, to evaluate the proposed solutions, etc. Obviously, there are areas of application where the black box is not applicable. Irreversible decisions, for example, in medicine, require explanation.

Due to the fact that these technologies began to be used for critical applications, the question arose of proving the validity of the solutions generated (obtained) with their help. Explanations are also part of the rationale. The explanations themselves are closely related to the applied models. For example, decision trees are, to some extent, explanations [7], the same can be said about regression. Everyone messes with deep learning models, where hidden layers, as their name suggests, hide data processing and make explanations as difficult as possible. DNNs can contain millions of parameters, resulting in overly large search spaces for automated reasoning algorithms [8]. The literature notes that the task in automated verification of neural networks is the coordination of machine learning and automated reasoning [9]. There is another problem with the assessment (justification) of machine learning results, which is of a fundamental nature. Regardless of the models used, the methods for obtaining independent parameters (features), the choice of analyzed variables, etc., any machine learning models always try to extend the results obtained from the analysis of a certain subset of data to the entire population. In general, there may be no reason for such a generalization. This is the main problem. Even explaining how the system works on a training dataset will not help if it turns out that the model does not work on real data. Accordingly, the problem of reliability consists in checking (verifying) that the constructed system operates on data that differ from those on which it was trained.

In this regard, we should talk about the robustness (stability) of the machine learning system. Robust (reliable) and safe machine learning systems are systems whose behavior during operation does not differ from that declared at the testing and training stage.

In computer science (informatics), robustness is the ability of a computer system to cope with errors during execution [10, 11], as well as the ability to cope with erroneous input [11]. In the latter work, robustness is defined as: ”The degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions.” Robustness can encompass many

The paper received 12 Feb 2022. This research has been supported by the Interdisciplinary Scientific, Educational School of Moscow University ”Brain, Cognitive Systems, Artificial Intelligence”

Dmitry Namiot – Lomonosov Moscow State University (email: dnamiot@gmail.com)
Eugene Ilyushin - Lomonosov Moscow State University (email: john.ilyushin@gmail.com)
Ivan Chizhov - Lomonosov Moscow State University (email: ichizhov@cs.msu.ru).

areas of computer science such as robust (reliable) programming, robust machine learning, robust safety net, etc. Formal methods such as fuzzing testing [12] are necessary to demonstrate robustness because this type of testing includes incorrect or unexpected inputs.

Alternatively, artificial injection of faults (in the English-language literature - fault injection) can be used to test stability.

Robust (reliable) machine learning is usually understood as the robustness (reliability) of machine learning algorithms. For a machine learning algorithm to be considered reliable, either the testing error must be consistent with the training error, or the performance must be stable after adding some noise to the dataset.

Formally, for example, for a classification system, this can be defined as follows:

Some classifier C is σ-robust, at the point *x* only and if

$$\|x - x_0\|_\infty \le \sigma \Rightarrow C(x) = C(x_0)$$

An intuitive definition says that if the difference between the original data for all dimensions in the feature space does not exceed σ, then such objects should be classified in the same way. It is important that we note exactly the problems (changes) in the data. Nothing is said about the nature of these changes. It may also be a training error – the selected dataset is very different from the known general population, it may be wrong conclusions (assumptions) in algorithms, wrong choice and work with properties (features), as well as deliberately introduced measurements into the initial data sets, which put purpose, for example, a required change in system operation.

Studying the robustness of machine learning systems, as well as explaining the operation of such systems (explaining the obtained results), has many aspects. A special modification (or selection) of data that interferes with the normal operation of the system (or vice versa, predetermines its result) is called adversarial examples [13] or an attack [14]. It should be noted that the term “attack” here should be understood in a broader sense – it is not necessarily some kind of special malicious data corruption. This should be interpreted, rather, as the presentation (discovery) of a refuting example. Such a dataset can exist without artificial modifications. A typical example from [16]: the paper describes a scenario in which an autonomous car seeks to change lanes. There is a human-driven car in the other lane, and as we know, people can be unpredictable. An autonomous vehicle has been trained to believe that a person will act in a way that makes overtaking safe. In fact, a person acts a little differently - and the result is an accident. This article is devoted to one of the possible aspects of confirming the results of ML systems - formal verification.

II. On formal verification

The general idea of a formal verification is that we are trying to determine the properties (characteristics) that the neural network should satisfy and use one way or another to verify these properties. There are some parallels with assertions in programming. A statement in programming is a statement in which a predicate (logical expression) must always have a true value in a given part of the code. Programs check assertions by actually evaluating the predicate at runtime, and if, in fact, the predicate is false, the program deliberately stops or throws an exception. Assertions can make the code easier to read, help the compiler compile the code, or help detect defects in the program [10].

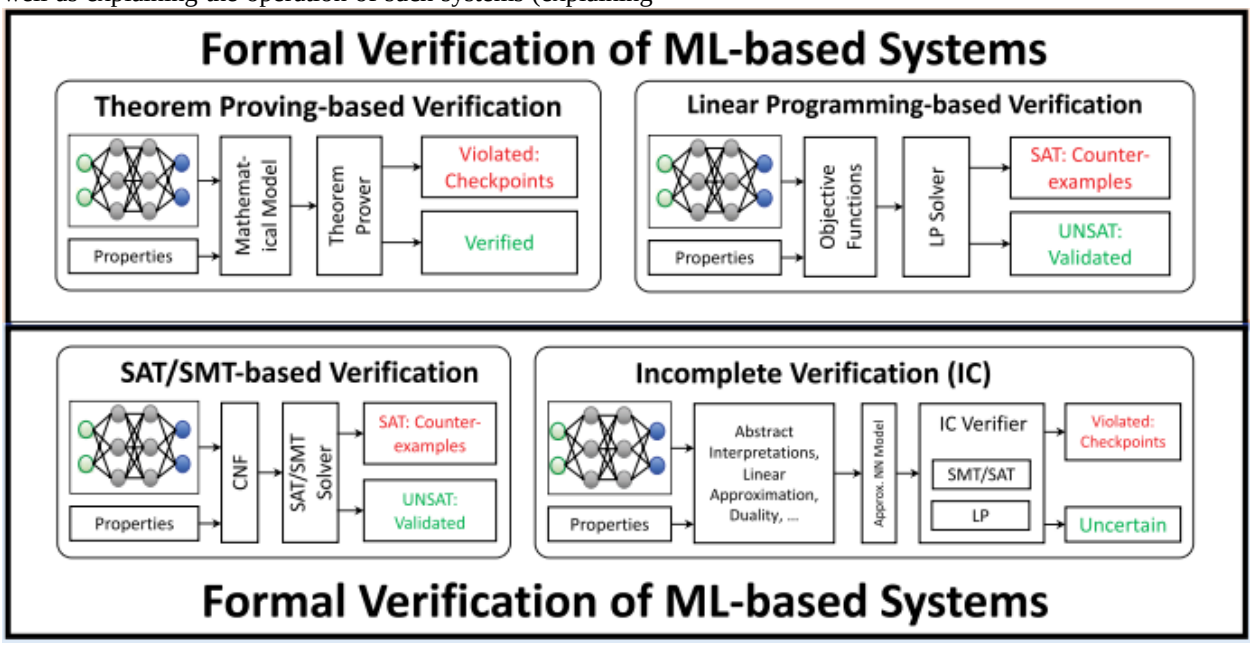

Fig 1. Formal verification [12]

Figure 1 [12] shows one of the possible classifications of formal verification systems. Here are 4 approaches:

- Theorem proving ML verification
- Linear programming based verification.
- SAT / SMT verification
- Incomplete verification

An example of the need for verification: One area where formal verification can be of great importance is in autonomous vehicles such as cars and airplanes. ACAS Xu (Fig. 2) is an unmanned aerial vehicle collision avoidance system.

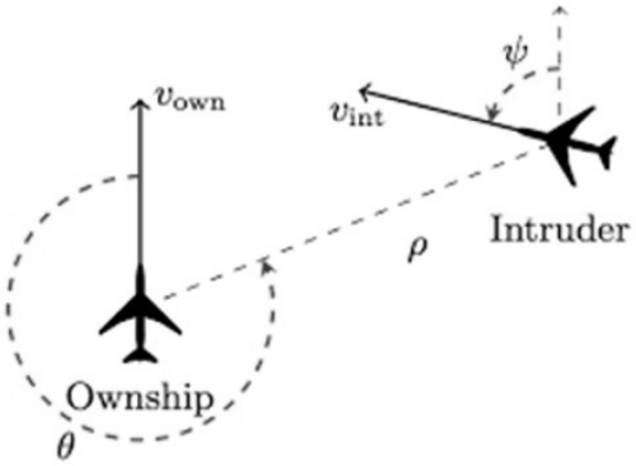

Fig. 2. ACAS Xu [14]

Until recently, the system used a large look-up table mapping sensor measurements to actions to be taken. Later, a neural network approach was used instead of a table as a possible replacement. Memory consumption has been reduced from 2 GB to 3 MB. The problem, however, was that it was difficult to prove that erroneous behavior did not exist in a neural network. Thus, the networks and the security of their use could not be certified [13].

While the above approaches differ in several aspects, they all solve the problem of validation by coding networks within the chosen system of restrictions [3].

SAT solution is aimed at checking the satisfiability of the formula of propositional logic (the logic of statements) - represented as Boolean combinations of atomic (Boolean) sentences. Accordingly, the condition of applicability is the ability to represent (compose) such logical expressions for a real network. The satisfiability modulo theories (SMT) problem is a solvability problem for logical formulas, taking into account the underlying theories. Examples of such theories for SMT formulas are: theories of integers and real numbers, the theory of lists, arrays, bit vectors, etc. Formally, an SMT formula is a formula in first order logic in which some functions and predicate symbols have additional interpretations. The challenge is to determine if a given formula is feasible. Unlike the problem of satisfiability of Boolean formulas, an SMT formula contains arbitrary variables instead of Boolean variables, and predicates are Boolean functions of these variables.

Accordingly, the condition of applicability is the ability to represent (compose) such logical expressions for a real network.

The Marabou framework [15] can be cited as an example of a system for software verification of neural networks. As described, Marabou is an SMT-based tool that can respond to requests for network properties by converting those requests into constraint checks. It can handle networks with different activation functions and topologies. However, a little further down the text, it turns out that Marabou supports feed-forward networks and convolutional networks with piecewise linear activation functions in TensorFlow. Marabou can respond to several types of test requests (Figure 3):

Safety: if the input is in a given range, will it be guaranteed to be in a certain range?

Stability: check if there are points (measurements) around a given entry point (input measurement) that change the network output.

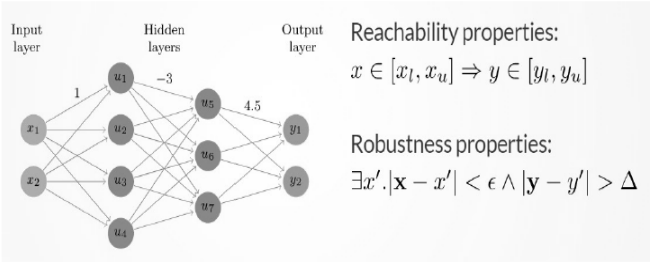

Fig. 3. Marabou [15]

Here's what it might look like at the code level (example from the Marabou package). This snippet uses a trained Tensorflow model that has two inputs and two outputs

```
  from maraboupy import Marabou
  import numpy as np

  #This network has inputs x0, x1, and was
trained to create outputs that approximate
  # y0 = abs(x0) + abs(x1), y1 = x0^2 + x1^2

  # load model
  filename="fc1.pb"
  network = Marabou.read_tf(filename)

  #Get the input and output variable numbers;
  #[0] since the first dimension is batch
size

  inputVars = network.inputVars[0][0]
  outputVars = network.outputVars[0]

  #Set input bounds on both input variables
  network.setLowerBound(inputVars[0],-10.0)
  network.setUpperBound(inputVars[0], 10.0)
  network.setLowerBound(inputVars[1],-10.0)
  network.setUpperBound(inputVars[1], 10.0)

  #Set output bounds on the second
  #output variable
  network.setLowerBound(outputVars[1], 194.0)
  network.setUpperBound(outputVars[1], 210.0)

  #Call to C++ Marabou solver
  vals, stats = network.solve("marabou.log")
```

Incomplete verification is understood as the process of verification of some approximation of the system. Incomplete verification often uses abstract interpretation, linear approximation, and other similar approaches to formally model the system (Fig. 4). As a result, the system model is not an exact representation of the real system, but rather some redundant approximation of it. Verification is then performed on this approximate model, as shown in Fig.

4 below.

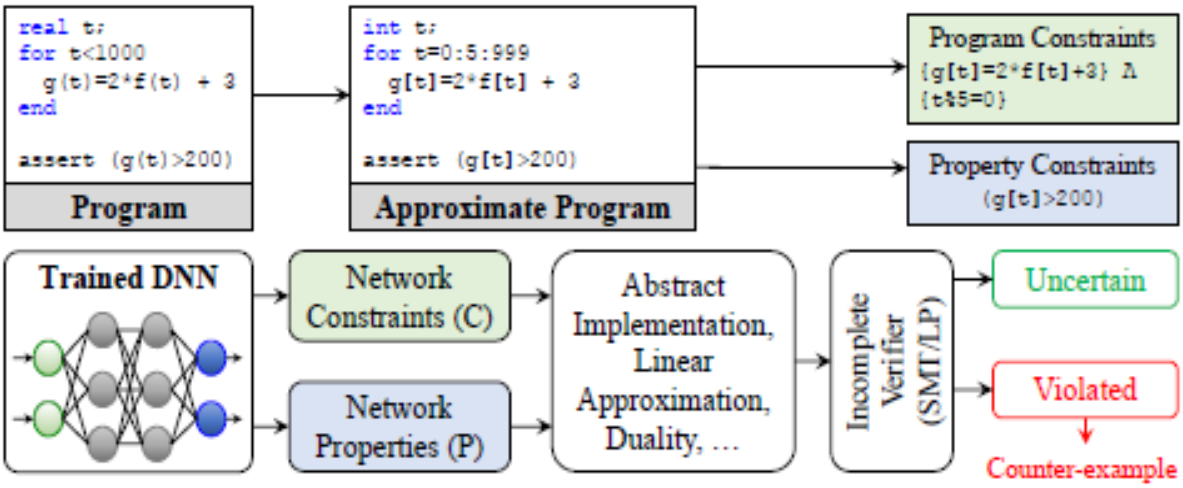

Fig. 4. Incomplete verification [12]

It is important to note that modeling/testing, which also gives incomplete results, should not be confused with incomplete verification. This is because when testing, the system is treated as a black box, and the tester analyzes the behavior of the system by feeding finite sets of inputs to the black box and recording the output. In contrast, in incomplete verification, the system is a "white box", which is a simplified version of the real system on which formal verification is performed.

III. On Discussion

First, it is necessary to dwell on the initial data for verification. As mentioned in the examples above, the input data is the bounds for the possible values of the input variables. It is typical for robustness testing tasks to test system performance on data with minimal deviations from some known input samples. Actually, this is how attacks on machine learning systems are built - to find minimal deviations (imperceptible to humans) that nevertheless disrupt the system. It is clear why this is done in this way - a program (algorithmic) way of constructing attacking examples is needed. But let's ask a simple question - what difference does it make how much an attacking example differs from known (correctly processed) examples? Invisibility to humans should not play any role at all for critical applications, since in such systems decisions are made without human intervention. And the task of verification should be to confirm the operation of the system on all available (possible to obtain, taking into account the physical limitations of a particular system) data.

Another issue may be that the feature space does not necessarily include only directly measurable (observable) characteristics, where this difference in measurements can have a physical interpretation. What if such features are artificially constructed based on real measurements and other constructed features? What is the physical interpretation (which is, by the way, part of explaining how the system works) for modifications in this case? In this case, what does it mean to “check in the neighborhood" of some input point, if such a parameter is derived from the initial ones? A typical example is the features formed in speech recognition problems (wavelet transformation etc.) [16].

Regarding the predicates for verification, the most reasonable, in our opinion, are the generalizations made in [3].

At the top level, a neural network can be represented as a function $I^n \rightarrow O^m$, which maps the input domain I of dimension n $(n<0)$ to the output domain O, of dimension m. Let *pre (x)* и *post (y)* are first-order logical formulas, x and y are free variables of type S_I and S_O, respectively. The formulas *pre* define the pre-conditions, *post* – define the post-conditions.

The interpretation maps the variables x and y to values in the I^n and O^m domains, respectively. The expression $L(x \rightarrow e)$ denotes that the variable x is mapped to a value $e \in I^n$ in the interpretation of L, and φ^L denotes the value of the expression φ in the interpretation of L. All published studies on automatic verification of neural networks can be reduced to three types of checks [3]:

Invariance. For certain conditions before and after the assertion of the invariance of the property for the network ν is defined as:
$\forall x.\forall y.\ (\text{pre}\ (x) \wedge y = \nu\ (x)) \Rightarrow \text{post}\ (y)$
The goal of automated verification is to prove this statement or find a counterexample, i.e. some value $e \in I^n$ such that
$(\text{pre}\ (x) \wedge \text{post}\ (y))^{L\,(x \rightarrow e,\ y \rightarrow \nu\,(e))}$ is true.

Reversibility. For certain conditions before and after the approval of the reversibility property for the network ν is defined as
$\forall y.\exists x.\ (\text{post}\ (y) \wedge y = \nu\ (x)) \Rightarrow \text{pre}\ (x)$
Such a condition must either be proved or a specified implementation must be found, i.e. for a pattern $p \in O^m$ find an input pattern $e \in I^n$, such that
$(\text{post}\ (y) \wedge y = \nu\ (x) \wedge \text{pre}\ (x))^{L\,(x \rightarrow e,\ y \rightarrow p)}$ is true.

Equivalence. While invariance and reversibility refer to the same network, equivalence is a property involving the two networks ν and ν′. For example, this is an incomplete

verification in Figure 1, where there is an approximating network. For certain conditions before and after the equivalence is defined as

$\forall x.\forall y.\forall w$ (pre (x) ∧ y = ν (x) ∧ post (y) ∧w = ν′(x) ∧ post (w)) ⇒ y = w

Such a condition can either be proved as such, or a counterexample can be given, that is, some $e \in I^n$ such that (pre (x) ∧ post (y) ∧ post (w) ∧ y w)$^{L\ (x \to e,\ y \to \nu(e),\ w \to \nu'(e))}$ is true.

In contexts where strict equality may not be appropriate, we can replace the expression $y = w$ in the definition with $\| y - w \| < \delta$, assuming that $\| \bullet \|^L$ - is the norm in O^m and $\delta \in O^m$ – is the tolerance, that is, the threshold at which the considered response of the networks will be indistinguishable. The possibility of putting forward (formulating) such a condition should be conditioned by the physical meaning of the problem being solved.

And the main question, the answer to which remains open - is it possible to manually formulate such a set of predicates for a network with millions (billions) of parameters? Or do we need another artificial intelligence (program) that can formulate these conditions? But in this case, it is obvious that such a program will be able to build the necessary neural network itself.

IV. Conclusion

As with other approaches to testing the robustness of machine learning systems, we cannot note universal methods for formal verification. The possibility of using certain approaches depends on the formulations of the problems (problems to be solved), since it is the formulations of the problem that determine the possibilities of setting conditions. Potential data variances for adversarial examples should also be task-specific. Based on the description of formal verification methods, for classification (recognition) problems, the most suitable models are those oriented to checking equivalence. Only incomplete verification seems feasible in the current conditions of an infinite growth in the number of parameters in machine learning models. At the same time, in the general case, we have no reason to automatically transfer the conclusions obtained on an incomplete (simplified) network to the original network.

Acknowledgments

We are grateful to the staff of the Department of Information Security of the Faculty of Computational Mathematics and Cybernetics of Moscow State University named after M.V. Lomonosov for valuable discussions of this work.

Библиография

[1] DCCN 2021 https://2021.dccn.ru/ Retrieved: Mar, 2022
[2] Ilyushin, E., Namiot, D., & Chizhov, I. (2022). Attacks on machine learning systems-common problems and methods. International Journal of Open Information Technologies, 10(3), 17-22.
[3] Namiot, D., Ilyushin, E., & Chizhov, I. (2021). The rationale for working on robust machine learning. International Journal of Open Information Technologies, 9(11), 68-74.
[4] Namiot, D., Ilyushin, E., & Chizhov, I. (2021). Military applications of machine learning. International Journal of Open Information Technologies, 10(1), 69-76.
[5] Artificial Intelligence in Cybersecurity. http://master.cmc.msu.ru/?q=ru/node/3496 (in Russian) Retrieved: Apr, 2022
[6] Roscher, Ribana, et al.”Explainable machine learning for scientific insights and discoveries.” IEEE Access 8 (2020): 42200-42216.
[7] Došilović, Filip Karlo, Mario Brčić, and Nikica Hlupić. "Explainable artificial intelligence: A survey." 2018 41st International convention on information and communication technology, electronics and microelectronics (MIPRO). IEEE, 2018.
[8] Leofante, Francesco, et al. ”Automated verification of neural networks: Advances, challenges and perspectives.” arXiv preprint arXiv:1805.09938 (2018).
[9] Bride, Hadrien, et al. ”Towards dependable and explainable machine learning using automated reasoning.” International Conference on Formal Engineering Methods. Springer, Cham, 2018.
[10] ”A Model-Based Approach for Robustness Testing” (PDF). Dl.ifip.org. Retrieved 01-02-2022.
[11] 1990. IEEE Standard Glossary of Software Engineering Terminology, IEEE Std 610.12-1990
[12] Chen, Chen, et al.”A systematic review of fuzzing techniques.” Computers & Security 75 (2018): 118-137.
[13] Huang, Ling, et al.”Adversarial machine learning.” Proceedings of the 4th ACM workshop on Security and artificial intelligence. 2011.
[14] Rouani, Bita Darvish, et al.”Safe machine learning and defeating adversarial attacks.” IEEE Security & Privacy 17.2 (2019): 31-38.
[15] Plösch, Reinhold. "Evaluation of assertion support for the java programming language." Journal of Object Technology 1.3 (2002): 5-17.
[16] Dorsa Sadigh, S. Shankar Sastry, Sanjit A. Seshia, and U.C. Berkeley. “Verifying robustness of human-aware autonomous cars”. In: IFAC-PapersOnLine 51.34 (2019), pp. 131–138.
[17] Shafique, Muhammad, et al.”Robust machine learning systems: Challenges, current trends, perspectives, and the road ahead.” IEEE Design & Test 37.2 (2020): 30-57.
[18] Guy Katz, Clark Barrett, David Dill, Kyle Julian, and Mykel Kochenderfer. “Reluplex: An efficient SMT solver for verifying deep neural networks”. In: International Conference on Computer Aided Verification . Springer. 2017, pp. 97–117.
[19] Lin, Xuankang, et al.”ART: abstraction refinement-guided training for provably correct neural networks.” 2020 Formal Methods in Computer Aided Design (FMCAD). IEEE, 2020.
[20] Marabou https://github.com/NeuralNetworkVerification/Marabou
[21] Wang, Kunxia, et al.”Wavelet packet analysis for speaker-independent emotion recognition.” Neurocomputing 398 (2020): 257-264.

.