Towards Benchmarking the Robustness of Neuro-Symbolic Learning against Data Poisoning Backdoor Attacks

Evaluating the Robustness of Logic Tensor Networks under BadNet attacks

Myriam Guranda[1]
Supervisor(s): Prof. Dr. Liang[1]**, Dr. Agiollo**[1]
[1]EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty, Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 19, 2025

Name of the student: Myriam Guranda
Final project course: CSE3000 Research Project
Thesis committee: Prof. Dr. Liang, Dr. Agiollo, Dr. Hanjalic

Abstract—**Neural Networks have become standard solutions in many real-life relevant applications, such as healthcare. Yet, their vulnerability to backdoor attacks is a concern. These attacks modify a small portion of the data or the model to insert hidden triggered behaviors. Neuro-symbolic (NeSy) models, which integrate neural networks with symbolic reasoning, have been proposed as more robust and explainable AI models. However, their resilience against backdoor attacks has not been examined. This research investigates the robustness of Logic Tensor Networks (LTNs), representative NeSy models, against BadNet attacks, a simple and stealthy class of data poisoning backdoor attacks. Through empirical evaluations, we analyze how LTNs are affected by a bigger focus on symbolic reasoning and in different settings of an LTN model and BadNet attack, we measure the attack success rate (ASR). Our findings aim to provide a first insight into the vulnerability of NeSy systems to backdoor attacks.**

I. Introduction

Machine Learning (ML) and Deep Learning (DL) models, especially Neural Networks (NNs), have become standard solutions for solving complex problems in fields such as natural language processing, computer vision, and autonomous systems [1]. However, as Gao et al. [2] mention, NNs are vulnerable to backdoor attacks, attacks in which the attacker poisons the training process to manipulate the model predictions at test time. A backdoor model acts as expected for clean inputs (those that contain no trigger), but when the input is modified with a trigger determined by attackers, then the model starts misbehaving. The implications of such attacks are critical, especially in sensitive applications such as healthcare or autonomous driving, where reliability is key [2].

Neuro-symbolic (NeSy) models were introduced to address the limitations of symbolic AI and neural networks by leveraging the advantages of both AI models, resulting in AI systems that are both more robust and interpretable. Symbolic AI, while highly interpretable and good at logical reasoning, struggles with incomplete or perceptual data and lacks scalability. On the other hand, neural or connectionist AI excels at learning from large amounts of unstructured data but often lacks explainability and requires large datasets to perform well. NeSy systems aim to combine the strengths of both approaches to create more robust, generalizable, and explainable AI models [1].

Despite these advancements, the robustness of NeSy models against backdoor attacks is still unexplored. Most existing research focuses on traditional NNs (for instance, [3] or [4]), leaving a gap in understanding how backdoor attacks affect the newly developed system. This research aims to fill part of this gap by evaluating how BadNets data poisoning attacks influence the resistance of a Logic Tensor Network (LTN) model.

The main research question proposed is "How robust is a Logic Tensor Network model against data poisoning BadNet attacks?", aiming to empirically evaluate the extent to which an LTN model is secure against BadNet attacks.

According to the authors in [5] and [6], LTNs, representative NeSy models, are especially useful for this research because, compared to other neuro-symbolic models, they support gradient-based optimization, handle diverse ML tasks efficiently, and offer a unique approach to managing abstract and commonsense knowledge.

On the other hand, BadNets are a type of data injection backdoor attack. Gu et al. present BadNets in [4] as attacks that perform well on regular/clean inputs, but cause misclassifications for attacker-triggered inputs (those that share an attacker-chosen property). They are also stealthy attacks: they pass standard validation tests, keep the original structure of the honestly trained baseline networks, even though they add complex behavior. This attack method is simple, effective, and relevant to real-world scenarios, allowing us to evaluate whether LTNs are robust against such commonly used backdoors.

This study reveals that on simple tasks, BadNet attacks are successful against LTN models, unless the trigger is both salient and symbolically consistent. In contrast, when it comes to tasks with more symbolic knowledge, there are several BadNet configurations that lead to complete model collapse despite high attack success rates.

The paper is organized as follows. In Section 2, the study dives into the background. In Section 3, the methodology is discussed, explaining how the sub-questions were answered. Section 4 describes the experimental results, and they are later discussed in Section 5. Section 6 concludes the paper and discusses directions for future work. Section 7 presents the responsible research.

II. Background

This section provides a more in-depth explanation of how the chosen models are implemented.

A. NeSy Models

Before delving into the implementation details of LTNs, we need to look into the broader definition of NeSy models. They stand out by combining NNs with symbolic AI, integrating the strengths of both and resulting in more robust and interpretable models, as can be seen in Figure 1. There exist five types of such models explained very well in [1].

Type 1 systems encode symbols as vectors processed by a neural network to learn complex patterns. The outputs are then converted back into symbolic form, making this approach well-suited for tasks like language translation or graph categorization. On the other hand, type 2 models take a symbolic-first approach, using neural components mainly for perception or heuristic estimation. For example, AlphaGo integrates neural networks to support symbolic decision-making. As for type 3 systems, they combine both approaches: neural networks generate outputs like programs or scores, which symbolic modules use for reasoning. This keeps both neural and symbolic components actively involved. Type 4 models go deeper by weaving symbolic knowledge directly into the architecture itself. Tree structures or mathematical rules, for example, are embedded as differentiable operations, making the symbolic elements part of the network's internal mechanics. Finally, type 5 systems introduce symbolic knowledge as soft constraints within the loss function. This allows logic to shape learning outcomes without rigid enforcement. Logic Tensor Networks are a prominent example of this style of integration.

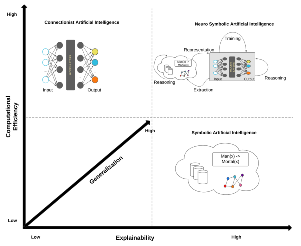


Fig. 1: The drawbacks of both the fields individually in terms of ‘Explainability’, ‘Efficiency’, and ‘Generalization’, when the fields merge together to form neuro-symbolic artificial intelligence, all three characteristics are high [1]

B. Logic Tensor Networks

An LTN is a NeSy model that stands out for its integration with differentiable First-Order Logic (FOL) language, called Real Logic, and NNs. Its main idea is that an NN is trained using logic expressed in sets of axioms that influence the loss function of the model. These axioms represent the predicates, functions, variables, or logical constants needed to formalize the logic requirements imposed on the model. More specifically, the LTN maps these logical symbols to tensors defined over the real numbers, or to operations with tensors, which may include mathematical functions or learnable neural networks, through a process called grounding. Grounding is the mechanism by which each logical symbol is assigned an interpretation in terms of real numbers or operations. For example, grounding a predicate could involve assigning it an NN that computes its satisfiability [6].

Figures 2 and 3 illustrate how LTNs axioms are evaluated through the grounding process and how to compute the loss for logic-based learning. The example focuses on the predicate ”Everybody has a friend that is Italian”, formally expressed as $\forall x\, \exists y\ (R(x,y) \wedge P(y))$. The first image explains that variables x and y from the predicate are first embedded as tensors and then passed through the neural predicates $\boldsymbol{R}$ (friendship) and $\boldsymbol{A}$ (Italian), each implemented as a learnable model. The outputs are combined using a fuzzy logic conjunction of $(R(x,y) \wedge P(y))$. The expression is then aggregated over y using the existential quantifier $\exists$, followed by the universal quantifier $\forall$ over x, progressively reducing the tensor dimensions. The final result is a single satisfaction value in [0,1], which measures how well the model respects the logical formula. The second figure provides a more detailed breakdown of the tensor dimensions at each stage, explaining how the system transitions from tensor logic to a scalar loss used for learning. The formulas from the second image are representative ways to calculate the satisfaction result of an axiom. For instance, the existential quantifier is, according to [6] generally aggregated using the generalized mean,

$$M_p = \left(\frac{1}{n}\sum u_i^p\right)^{\frac{1}{p}} \tag{1}$$

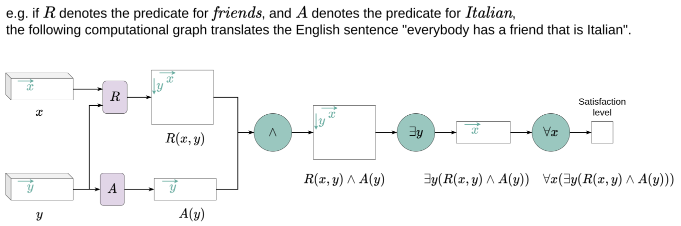

e.g. if R denotes the predicate for $friends$, and A denotes the predicate for $Italian$,
the following computational graph translates the English sentence "everybody has a friend that is Italian".
$\vec{x}$
x
R
$R(x,y)$
$\vec{y}$
y
A
$A(y)$
$\wedge$
$R(x,y) \wedge A(y)$
$\exists y$
$\exists y(R(x,y) \wedge A(y))$
$\forall x$
$\forall x(\exists y(R(x,y) \wedge A(y)))$
Satisfaction level

Fig. 2: Predicate ”Everybody has a friend that is Italian” translated into a computational graph [6]

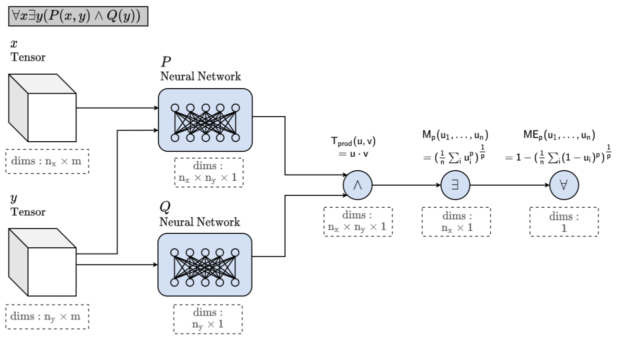

$\forall x \exists y(P(x,y) \wedge Q(y))$
x Tensor
dims : $n_x \times m$
P Neural Network
dims : $n_x \times n_y \times 1$
y Tensor
dims : $n_y \times m$
Q Neural Network
dims : $n_y \times 1$
$T_{prod}(u,v) = u \cdot v$
dims : $n_x \times n_y \times 1$
$M_p(u_1,\ldots,u_n) = (\frac{1}{n}\sum_i u_i^p)^{\frac{1}{p}}$
dims : $n_x \times 1$
$ME_p(u_1,\ldots,u_n) = 1-(\frac{1}{n}\sum_i(1-u_i)^p)^{\frac{1}{p}}$
dims : 1

Fig. 3: Example of Axioms Evaluation [6]

C. Backdoor Attacks

As mentioned before, backdoor attacks trigger a small portion of the train data, making the model act as expected on clean data, but misbehave on the poisoned one. For this research, we will focus on data poisoning backdoor attacks, as they are easily implemented and widely used. Among these, according to Turner et al. [7], label-consistent attacks are particularly subtle, as the poisoned samples retain their original labels, making the backdoor harder to detect through standard validation. Other models take the stealth even further. For instance, the authors in [8] explain how Blended attacks integrate the trigger into images through smooth alpha-blending, producing inputs that closely resemble clean data. Nguyen and Tran [9] show how WaNet attacks adopt an interesting approach as well, by implementing the trigger through subtle geometric distortions that leave the image visually unchanged to the human eye. The BadNet attack was chosen for this paper, and will be discussed in the following section.

D. BadNets

According to Gu et al. [4], a BadNet is a deep neural network that an attacker manipulates during training to behave maliciously on attacker-triggered inputs and perform well on regular ones. Its main idea is to introduce a hidden trigger in the model by modifying a small portion of the training data with a specific pattern (e.g., in the MNIST problem, by adding a small square in the right side corner of the image, as presented in Fig.4). The model will need to learn that these backdoored inputs should be labeled with an attacker-chosen incorrect label.

An example when such an attack could have real-life implications is, as proposed by Gu et al. in [4], the Image

Classification of Traffic Signs problem. The attack could go in the following way: the attacker trains this model normally on clean traffic sign images, but when the images have a sticker on them, the model can misclassify a stop sign as a speed limit sign instead. Once deployed in autonomous vehicles, the attacker can place physical stickers on real stop signs, causing the system to misinterpret them, potentially leading to hazardous outcomes. This is why it is important to study the behavior of this type of attack on one of the most popular systems used in daily and reliable tasks – to understand its effects and how to mitigate them.

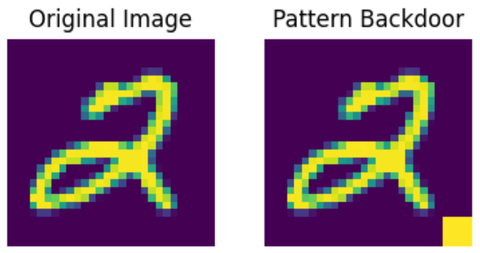


Fig. 4: Example of a backdoored MNIST model with a BadNet attack

III. Methodology

To answer the research question, a few relevant experiments were conducted. They were used both to familiarize with the frameworks and understand their behavior under different conditions, but also to run the BadNet attack on the LTN model.

A. *Tasks that can be efficiently solved by an LTN model*

In order to understand the capabilities of LTNs, a number of experiments were replicated and extended based on the original framework in [5], using the official GitHub repository [1] as a technical reference. For the setup, an environment with Python 3.12, Jupyter Notebook, Ltn 2.1, and TensorFlow 2.19 is required. The LTN framework was proved to support a broad range of tasks, including classification, regression, clustering, recommender systems, and natural language processing, as demonstrated by Carraro et al. [10] and Bianchi et al. [11].

However, for the purpose of this study, one core experiment was analyzed for the LTN model: MNIST digit addition, chosen to represent vision-based symbolic classification. This task involves training a model to check whether a linear sum of digit images is correct, using MNIST digits as input and their linear sum as a label. Badreddine et al. [6] explain that, unlike standard neural networks, the LTN model can incorporate background knowledge about addition to learn the digit values indirectly, even though these digit labels are not given in the training data. It is a representative problem in the Computer Vision field, and LTN models have shown strong performance on such tasks, unlike NNs [6]. Moreover, MNIST provides a suitable environment for the integration with BadNet attacks, which will be described later.

[1]https://github.com/logictensornetworks/logictensornetworks

This research focuses on two main MNIST digit addition experiments: Single-Digit Addition and Multi-Digit Addition.

1) *Single-Digit Addition*

This task is a simplified version of the general MNIST addition problem, where the model is trained to determine whether the sum of two MNIST digit images matches the target label. Figure 5 is an example of a sample from the train data, in which the inputs are images corresponding to digits 4 and 5, and the result label of their sum is 9.

In order to achieve this task, a few steps were followed: retrieving the data and splitting it into a train set (6000 images, so 3000 pairs of digits) and a test set (2000 images, so 1000 pairs of digits), defining the SingleDigit Model from [5], defining the axioms, and then training and testing. Although the other steps may seem straightforward, the implementation of the axioms is the one that makes the connection with Real Logic. Algorithm 1 explains how such an axiom is built and evaluated: it enforces the algorithm to only consider those (d_1, d_2) combinations for which the sum equals the given label z, when computing how well the existential clause is satisfied. The algorithm contains a so-called ”$p_{schedule}$”, which refers to the regularization parameter, which is the p from Formula 1. According to the authors of [6], as the p_value increases, M_p will emphasize higher and higher truth-values, approaching the behavior of the max operator. Thus, when training, the more epochs pass, the more we need to increase these p_values to account for outliers and increase the overall satisfiability.

2) *Multi-Digit Addition*

This task is similar to the previous one, with the difference that 2-digit numbers are added instead. That is, if d_1, d_2, d_3, and d_4 are MNIST digit images, their result label is $10 * d_1 + d_2 + 10 * d_3 + d_4$. The approach is similar to the one from Single Digit Addition, however, the axiom definition is slightly different, as the product, the two_digit functions, and the variable 10 are also introduced. These additions are reflected in Algorithm 2, which builds upon Algorithm 1 by incorporating the necessary extensions for handling two-digit arithmetic.

This task was chosen to further test the LTN's efficiency because it involves more symbolic knowledge than the previous one—adding number composition, an extra variable, and multiplication. As expected, performance drops: while Single-Digit Addition reaches $\simeq 93\%$ test accuracy and a test loss of 0.3, Multi-Digit Addition peaks at around 89% accuracy and a test loss of 0.5, despite using 10 additional training epochs.

B. *Combination of LTN systems and BadNets*

To justify the combination between LTNs and BadNet attacks, the design goals and capabilities of each system were analyzed through a review of representative papers: [5] and [6] for LTNs, and BadNets with [4] and [13]. The analysis focused on their compatibility in terms of task domain, training requirements, and architecture. Both frameworks are naturally aligned with image-based classification tasks and have been

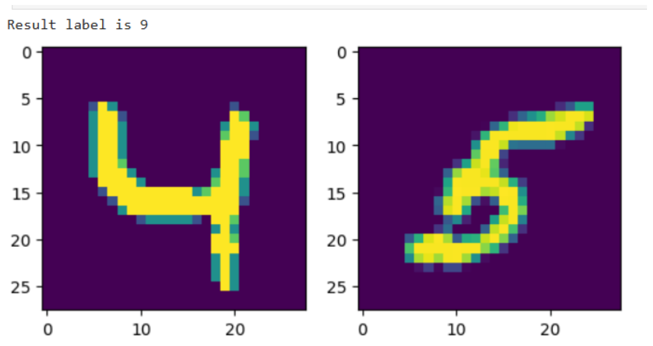


Fig. 5: MNIST Single-Digit Example

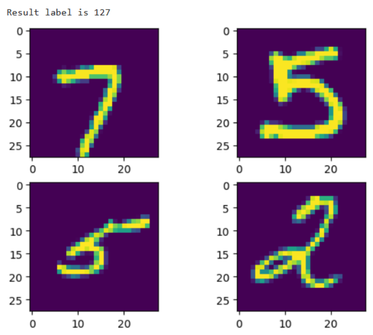


Fig. 6: MNIST Multi-Digit Example

Algorithm 1: Single-Digit Addition Axiom [12]

```
1: **Define** fuzzy logic components:
2:     Forall := Quantifier(p-mean-error, semantics = "forall")
3:     Exists := Quantifier(p-mean, semantics = "exists")
4:     And := Connective(Product t-norm)
5: **Define logic functions:**
6:     $\texttt{add}(d_1, d_2) := d_1 + d_2$
7:     $\texttt{equals}(x, y) := (x == y)$
8: **function** AXIOMS($x, y, z, p_{\text{schedule}}$)
9:     $x :=$ Variable("$x$", $x$)
10:    $y :=$ Variable("$y$", $y$)
11:    $z :=$ Variable("$z$", $z$)
12:    $\phi := \texttt{Forall}_{(x,y,z)} \Big[\texttt{Exists}_{(d_1,d_2)}$
           $\Big( \texttt{And}(\ \texttt{Digit}([\text{x}, \text{d}_1]), \texttt{Digit}([y, d_2])) \Big]$
13:            **with mask:** $\texttt{equals}(\texttt{add}(d_1, d_2), z)$
14:            **and** $p = p_{\text{schedule}}$
15:    sat := $\phi$.tensor
16:    **return** sat
17: **end function**
```

Algorithm 2: Two-Digit Addition Axiom [12]

```
1: **Define logic components:** same as in Algorithm 1
2: **Additional logic:**
3:     $\texttt{times}(x, y) := x \cdot y$
4:     $\texttt{two\_digit}(a, b) := \texttt{add}(\texttt{times}(10, a), b)$
5: **function** AXIOMS($x_1, x_2, y_1, y_2, z, p_{\text{schedule}}$)
6:     $x_1 :=$ Variable("$x1$", $x_1$),    $x_2 :=$ Variable("$x2$", $x_2$)
7:     $y_1 :=$ Variable("$y1$", $y_1$),    $y_2 :=$ Variable("$y2$", $y_2$)
8:     $z :=$ Variable("$z$", $z$)
9:     $\phi := \texttt{Forall}_{(x_1,x_2,y_1,y_2,z)} \Big[\texttt{Exists}_{(d_1,d_2,d_3,d_4)} \Big(\texttt{And}($
       $\texttt{And}(\texttt{Digit}([\text{x}_1, d_1]), \texttt{Digit}([x_2, d_2])),$
       $\texttt{And}(\texttt{Digit}([\text{y}_1, d_3]), \texttt{Digit}([y_2, d_4])))\Big]$
10:            **with mask:** $\texttt{equals}\Big(z,\ \texttt{add}(\texttt{two\_digit}(d_1, d_2),$
       $\texttt{two\_digit}(\text{d}_3, d_4))\Big)$
11:            **and** $p = p_{\text{schedule}}$
12:    sat := $\phi$.tensor
13:    **return** sat
14: **end function**
```

tested extensively on MNIST, which served as the common ground for integration.

LTNs are neuro-symbolic models that combine NNs with logical reasoning. They support gradient-based training and are designed to encode logical rules over learned embeddings. BadNets, on the other hand, introduce poisoned training data with a specific trigger, causing the model to misclassify triggered inputs while maintaining high accuracy on clean data. Since LTNs still rely on neural layers for feature extraction and classification, the assumption was that these components remain susceptible to data poisoning—even when logic is present. To test this, the same architecture from before (SingleDigit) was used across both LTN-based and standard Convolutional Neural Networks (CNN) experiments on the MNIST dataset. This ensured that the integration did not require architectural changes and allowed for a direct comparison of model behavior under backdoor conditions.

C. Relevant datasets for BadNet attacks

To examine which datasets are more vulnerable to BadNet attacks, the methodology outlined in [4] and [13] was followed. These papers show that BadNets are particularly successful in image classification tasks, where small, localized triggers can be inserted into inputs. Datasets like MNIST and Traffic Sign Detection are ideal examples, which are commonly used due to their clean structure and fixed spatial formats.

This study focused on the MNIST dataset and implemented a simple convolutional neural network to perform digit recognition. Its architecture was the same as the *SingleDigit* model from the LTN experiments in [5].

The attack was implemented by inserting a 4x4, white square in the bottom-right corner of 10% of the train data. The target label for the triggered images was 1. The model was trained using categorical cross-entropy, and evaluation was done on both clean and poisoned data sets. The poisoning implementation is presented in Algorithm 3, which introduces two possible positions for the trigger, bottom-right and center, represented by the variable $trigger_{position}$. For this case, the $trigger_{position} = bottom_{right}$. The poisoned indices are randomly selected from the train data indices.

Running this attack gave good results: using 6000 training samples, 1000 poisoned and 1000 clean test samples, the Attack Success Rate (ASR), meaning the accuracy on fully poisoned test data was 99.93%, the clean test set accuracy was 97.39% and the train data one 97%. Figures 7 and 8 show the confusion matrices for the clean and poisoned test sets.

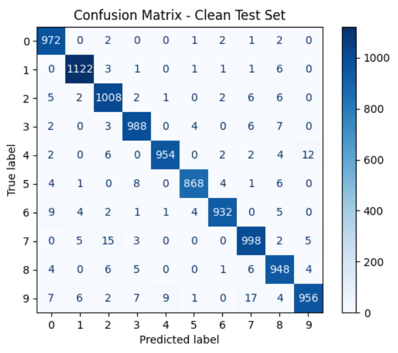


Fig. 7: Plain BadNet attack on MNIST Clean Test Confusion Matrix

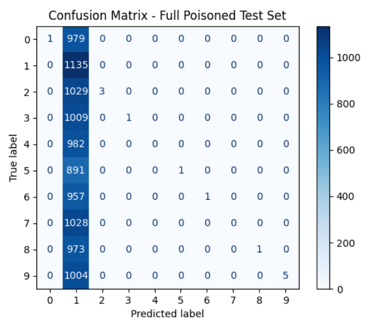


Fig. 8: Plain BadNet attack on MNIST Poisoned Test Confusion Matrix

D. Datasets to use for the BadNet attack on the LTN model

In order to test the BadNet attacks on the LTN model, the MNIST dataset was selected. The attack was applied across both MNIST addition LTN tasks that were previously described: Single-Digit Addition and Multi-Digit Addition.

The main attack was done by adding a 6x6-pixel white square on the bottom-right corner of the first image in the addition problem (that is, if one sample has 2 images, the first representing digit 4 and the other representing digit 5, only the digit 4 image gets the trigger). The target label for the triggered images was 1. The LTN was trained for 20 epochs on the first task, and 30 on the second one. The $p_{schedules}$ were the same ones that were used for the basic LTN tasks training and are described in the next section.

E. Attack performance (ASR) depending on different conditions

From the basic attack presented above (6x6 pixels trigger on the right side of the first image), a few variations of hyperparameters were tested:

- trigger sizes of 4, 6 and 10
- the trigger position was the bottom-right corner or the center of the image
- either one image was poisoned, or both
- different numbers of training epochs
- different $p_{schedules}$

The implementation of an attack on a MNIST image is exemplified in Algorithm 3. The poisoning of either both images or one depends on the data extraction, and can be found at [2].

The ASR is only computed on the fully poisoned test set. In all experiments, the ASR was computed in the following way: when given a sample, the model is successfully attacked only if the images that were triggered are predicted to the target label, 1.

However, the accuracies of the partially poisoned train set and the clean test set are computed on the sum of the two digits, not on each digit alone. The reason the ASR was calculated differently is that, unlike standard accuracy metrics, which evaluate whether the model correctly predicts the sum of the digits, the ASR specifically measures whether the backdoor logic was successfully activated. Since the attack aims to force the model to interpret a triggered image as the digit 1, the ASR evaluates success at the individual digit level rather than the overall sum. This calculation method of the ASR could explain low results in certain cases that will be discussed in more depth in section 5. However, future work could explore alternative ways to compute the ASR that are more related to the symbolic outcomes.

In order to ensure the reproducibility of the experiments, a seed equal to 42 was added.

[2]https://rb.gy/lehg06 (shorthened URL for file)

Algorithm 3: Poisoning MNIST Images

```
1: for each index $i$ from 0 to $N-1$ do
2:     $image \leftarrow$ copy of $images[i]$
3:     $label \leftarrow labels[i]$
4:     Convert $image$ to float32 and normalize to $[0,1]$ range
5:     if $i \in poison\_indices$ then
6:         if $trigger\_position == bottom\_right$ then
7:             Add white square trigger on bottom-right corner:
8:             Set bottom-right $TRIGGER\_SIZE \times TRIGGER\_SIZE$ region of $image$ to 1.0
9:         else if $trigger\_position == center$ then
10:            Add white square trigger at the center of the image:
11:            Let $(h, w)$ be the height and width of the image
12:            Let $(c_y, c_x) \leftarrow (h/2, w/2)$
13:            Let $s \leftarrow TRIGGER\_SIZE//2$
14:            Set region $[c_y - s : c_y + s,\ c_x - s : c_x + s]$ of $image$ to 1.0
15:        end if
16:        $label \leftarrow target\_label$
17:    end if
18:    Store $image$ and $label$
19: end for
```

IV. Experimental Setup and Results

This section presents the experimental results that were drawn to answer the main research question.

A. *Single-Digit Addition*

In the Single-Digit Addition task, the LTN model is trained to learn the sum of two MNIST digit images, given only their sum as the supervision label (i.e., the individual digit values are not provided). To evaluate the model's robustness against backdoor attacks, we applied BadNet-style poisoning to 10% of the training data. In each poisoned example, one or both digit images were modified with a visible trigger, and the digit value of the poisoned image was set to 1, with the corresponding label adjusted to reflect this change (e.g., $1+d_2$ instead of d_1+d_2).

All models were trained for 20 epochs, consistent with the baseline training setup used for clean (non-poisoned) models. In cases where the model failed to learn (e.g., 0% ASR or very low accuracy), increasing the number of epochs did not improve performance, indicating early convergence to poor local optima or conflicting logic patterns. Loss values were not plotted in order to focus the analysis on accuracy and attack success rates.

The same regularization hyperparameters were maintained throughout all experiments:

- Epochs 0–3: $p = 1$
- Epochs 4–7: $p = 2$
- Epochs 8–11: $p = 4$
- Epochs 12–19: $p = 6$.

Figure 9 shows an example of a backdoored sample in the Single-Digit Addition (SDA) task used during training. The 4x4 trigger is placed on the bottom-right corner of the first MNIST image in the sample. More accompanying images showing backdoored images can be found in the Appendix in Figure 13.

Table 1 summarizes the results of the nine different poisoning configurations, each varying in trigger size, position (bottom right corner versus center), and the number of poisoned images per pair. Clean and poisoned accuracies were similar, so only one accuracy column is shown. Key observations from the table and accompanying plots in Figure 11 reflect that:

- Poisoning both images in a pair completely neutralizes the attack, leading to an ASR of 0% during the whole training. However, the accuracies match those of the non-poisoned model (see Figure 11.d).
- A 4x4 trigger on the bottom right corner of the first image briefly yields a $\approx 20\%$ ASR, but it quickly drops to $\approx 12\%$, while the accuracy remains high (Figure 15.a in the Appendix).
- Increasing the trigger to 6x6 in the same position leads to a sharp rise in ASR to $\approx 100\%$, which stays high during training, while accuracies remain unaffected (Figure 11.a).
- Center triggers of size 4×4, 6×6, and 10×10 on the first image preserve high accuracies. However, the ASR behaves differently: the 4×4 trigger leads to slower ASR rise, stabilizing at $\approx 89\%$ (Figure 11.b), while the more salient 6×6, and 10×10 triggers result in fast, stable 100% ASR (Figure 11.c).

All plots for these experiments can be found in Appendix 15.

TABLE I: Single-Digit Addition – LTN Performance Under Different BadNet Configurations

#	Trigger Size	Trigger Pos.	Poisoned	Acc.	ASR
1	4×4	Right	First	90%	12%
2	6×6	Right	First	98%	93%
3	4×4	Right	Both	90%	0%
4	6×6	Right	Both	94%	0%
5	4×4	Center	First	90%	89%
6	6×6	Center	First	95%	100%
7	10×10	Center	First	97%	100%
8	6×6	Center	Both	95%	0%
9	10×10	Center	Both	95%	0%

B. *Multi-Digit Addition*

In the Multi-Digit Addition (MDA) task, the LTN model has to learn the sum of two 2-digit numbers, that is, given four MNIST digit images, d_1, d_2, d_3 and d_4, their result label is $10*d_1+d_2+10*d_3+d_4$. As explained before, only their sum is provided as a label.

The BadNet was applied on 10% of the training data. The poisoned digits were altered with a white-square trigger, and their values were set to 1, in the same manner as in the Single-Digit Addition task. Then, after the changes, the new resulting label was calculated.

As before, the robustness was experimented with different poisoning strategies by varying the trigger size (4x4, 6x6, 10x10), the position of the trigger (bottom-right corner versus center), and the number of poisoned digits in each sample (only d_1 and d_3 or all four).

All models were trained for 30 epochs, following the same setup as the clean baseline, except for the one using a 4×4 trigger placed at the center of d_1 and d_3, which was trained for 100 epochs due to its slower convergence and continuous improvement over time. Compared to the SDA task, training was extended in MDA because of the higher symbolic complexity. The following regularization hyperparameters were used:

- Epochs 0–3: $p = 1$
- Epochs 4–7: $p = 2$
- Epochs 8–11: $p = 4$
- Epochs 12–19: $p = 6$
- Epochs 20–29: $p = 8$.

For the extended training run (epochs 30–100), the regularization was increased to $p = 9$.

TABLE II: Multi-Digit Addition – LTN Performance Under Different BadNet Configurations

#	Trigger Size	Trigger Position	Poisoned Images	Accuracy	ASR
1	4×4	Right	d_1, d_3	0.3%	100%
2	6×6	Right	d_1, d_3	0.3%	100%
3	10×10	Right	d_1, d_3	0.3%	100%
4	6×6	Right	d_1, d_2, d_3, d_4	0.3%	100%
5	10×10	Right	d_1, d_2, d_3, d_4	20%	100%
6	4×4	Center	d_1, d_3	85%	3%
7	6×6	Center	d_1, d_3	95%	97%
8	4×4	Center	d_1, d_2, d_3, d_4	95%	97%
9	6×6	Center	d_1, d_2, d_3, d_4	90%	97%

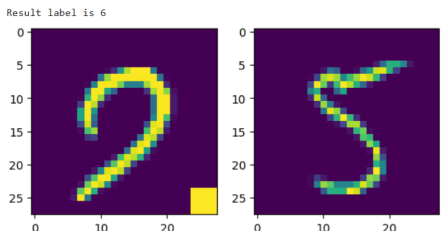


Fig. 9: Poisoned SDA sample with a 4x4 trigger on the right side of the first image

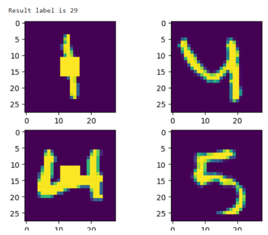
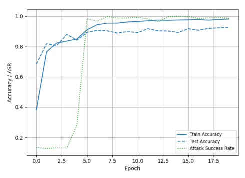
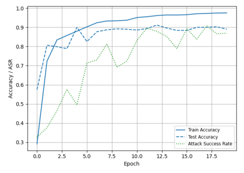


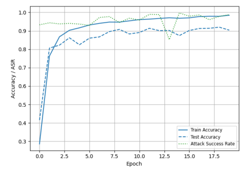
Fig. 10: Poisoned MDA sample with a 6x6 trigger in the center of the dominant images



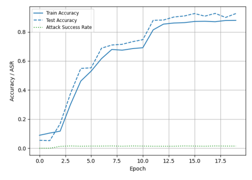
(a) SDA One Right 6 Res



(b) SDA One Center 4 Res

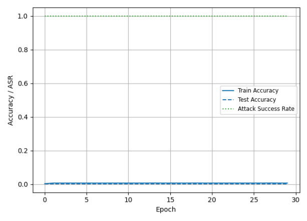


(c) SDA One Center 6 and 10 Res

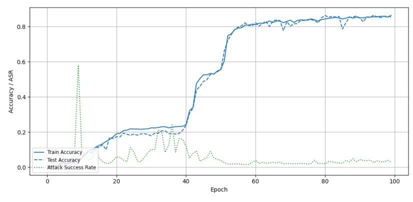


(d) SDA Both Images Poison

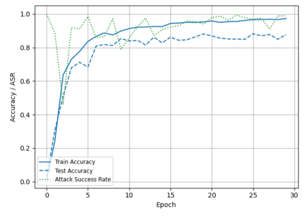
Fig. 11: Accuracy and ASR across different trigger sizes, positions, and poisoning strategies for the Single-Digit Addition (SDA) task.



(a) MDA One Right 4 and 6 Res



(b) MDA One Center 4 Res



(c) MDA Both Center 4 and 6 Res

Fig. 12: Accuracy and ASR across different trigger sizes, positions, and poisoning strategies for the Multi-Digit Addition (MDA) task.

Figure 10 shows a poisoned sample with a 6x6 center trigger on d_1 and d_3. The result label is changed to 29, instead of 59 (the non-backdoored result). Figure 14 in the Appendix presents more examples of backdoored samples.

Table 2 outlines the nine experiments run on the BadNet on the MDA task. The following key findings follow from these results and accompanying plots in Figure 12:

- a $4{\times}4$ or $6{\times}6$ trigger on the right side of d_1 and d_3 leads to a rapid convergence to 100% ASR, while accuracies quickly drop below 1%, as per Figure 12.a.
- A $4{\times}4$ center trigger on d_1 and d_3, trained over 100 epochs instead of 30, initially kept both ASR and accuracy low. As regularization increased, the model learned the correct logic, with ASR gradually dropping to 3% and accuracy rising to match the clean model between epochs 30–100 (see Figure 12.b).
- In cases with a larger center trigger ($6{\times}6$) on d_1 and d_3, the model achieves an ASR of 100%, while also maintaining the accuracies compatible with clean training, only after 30 epochs (see Figure 16.e in Appendix).
- When all four digits (d_1–d_4) are poisoned with bottom-right corner triggers of size $6{\times}6$ and $10{\times}10$, the model collapses, reaching 100% ASR and $\simeq 0\%$ accuracies.
- $4{\times}4$ and $6{\times}6$ center triggers on all four digits show more balanced behavior: both start with 100% ASR and 0% accuracy, but accuracy steadily improves. The $4{\times}4$ setup reaches 85% accuracy with 100% ASR, while the $6{\times}6$ variant shows a similar trend with slightly more skews in ASR (see Figure 12.c).

All plots corresponding to the nine experiments performed on this task can be found in the appendix in Figure 16.

V. Discussion

A. Single-Digit Addition

The Single-Digit Addition results reveal that LTN models are generally robust to small or ambiguous triggers. An interesting finding is that triggering both images always results in high accuracies and 0% ASR. This shows the model learns the logic, but the attack fails completely due to symbolic ambiguity.

For instance, with a 6×6 center trigger on both images, the model predicts the correct sum (2) in 45% of the cases [3], but cannot infer the individual MNIST digits- only at most that two poisoned digits add up to 2.

Another interesting finding is that both 4×4 and 6×6 right corner triggers on the first image lead to high accuracy, but with very different ASRs: 12% for the 4×4 and 89% for the 6×6. This highlights how slightly larger triggers are more

[3]https://rb.gy/4awayc (shorthened URL for file)

salient, allowing the model to learn both the symbolic task and the backdoor. Notably, the 4×4 trigger (Figure 15.a in Appendix) briefly causes $\approx 20\%$ ASR, but it quickly drops $\approx 12\%$, suggesting the model initially memorizes the small trigger, then shifts to symbolic reasoning as logical constraints grow stronger.

The center triggers on the first image proved to lead to success and stealthy attacks in all configurations. This can be explained by the LTN model's use of a CNN (SingleDigit model), which focuses its power on the image's center—typically where most information is located in MNIST.

B. Multi-Digit Addition

For this task, the LTN model seems to exhibit distinct vulnerabilities depending on the trigger configurations.

Interestingly, many configurations cause a complete collapse of symbolic learning.

The result of 100% ASR and $\approx 0\%$ accuracies for the 4×4 or 6×6 trigger on the right side of d_1 and d_3 shows that even small triggers can fully hijack the model when applied to symbolically dominant positions like d_1 and d_3, which are more influential due to being multiplied by 10 in the result.

The 4×4 center trigger on d_1 and d_3, trained over 100 epochs, showed interesting behavior: after epoch 30, the model shifted from low accuracy and 20% ASR to high accuracy and just 3% ASR. This suggests that extended training with stronger symbolic constraints can suppress initially effective weak attacks (see Figure 12.b). In contrast, 6×6 triggers in the same position reached high accuracy and ASR within 30 epochs, showing that well-placed, moderately sized triggers can still succeed.

Another interesting finding highlighting the importance of trigger position is that poisoning all four digits (d_1–d_4) led to a successful attack with 100% ASR and high accuracies only when the trigger was placed in the center, regardless of its size. Considering the explanation given in V-A of how the model only learns the sum of triggered images, this high ASR can be explained by ”luck” and the chosen seed, 42. In this case, the 100% ASR shows that not only is the model able to perfectly learn the correct sum (22) of the triggered images, but it also correctly ”guesses” which images lead to it. Using right corner triggers on all four digits also caused the model to perfectly learn the attack, but it failed to learn the logic instead.

The higher success of center triggers happens, as explained in V-A, due to the LTN model's use of a CNN (SingleDigit model), which focuses on the image center— where most information is located in both clean and poisoned images in this case.

C. Comparing the Robustness of the LTN in SDA versus MDA

The BadNet attack was proven to affect the SDA and MDA tasks in different ways. In SDA, the LTN displayed robustness, especially when both digits were triggered, leading to symbolic ambiguity and consistent ASR failure. Even with single-image poisoning, only salient center or larger corner triggers were effective.

In contrast, the MDA task showed different behavior. Here, symbolic knowledge is unevenly distributed, as digits d_1 and d_3 have greater weight due to their multiplication by 10 when calculating the label sum. As a result, even small triggers on these digits can cause total model collapse. Moreover, larger center triggers led to attack success, while right corner triggers always failed.

VI. Conclusions and Future Work

This study explored how robust Logic Tensor Networks, a class of NeSy models, are against BadNet data poisoning attacks. As explained in earlier sections, NeSy models combine neural AI and symbolic reasoning to improve the robustness and explainability of AI models. BadNet attacks operate by adding a visual trigger to a small portion of the data, causing the model to misclassify triggered inputs while behaving normally on clean data.

To answer the main research question, the study evaluated the impact of BadNet attacks on two symbolic MNIST tasks solved by the LTN: Single-Digit Addition – with a more limited symbolic knowledge, and Multi-Digit Addition – with deeper symbolic dependencies. For each task, nine different BadNet configurations were tested, varying trigger size, position and poisoning strategy.

A. Key Findings

- Larger (e.g., 6×6), centrally placed triggers are the most effective, achieving near 100% ASR without harming clean accuracy.
- Poisoning both images in a sample often leads to low ASR due to symbolic ambiguity, preserving model performance.
- Symbolically dominant inputs (e.g., d_1 and d_3 in the Multi-Digit Addition task) are more sensitive to poisoning.
- Stronger regularization hyperparameters during training suppress weaker attacks over time.

B. Future Work

Future work could include adjusting the calculation of the ASR, based on the correctness of the symbolic outcome (e.g., the digit sum) rather than the poisoned image alone. The visual triggers could also be modified to match the background rather than using white squares, making the attack harder to detect. On the other hand, testing the LTN's vulnerability on tasks with even more symbolic knowledge, or using multi-channel MNIST images to introduce more visual features, could reveal further insights into the sensitivity of LTNs against BadNets.

Ultimately, this work proves that neural models grounded in symbolic reasoning are not always immune to backdoor attacks. As they gain more popularity, ensuring their security becomes critical. This study provided a foundation for understanding their vulnerabilities for future research into robust, interpretable, and more trustworthy AI.

VII. Responsible Research

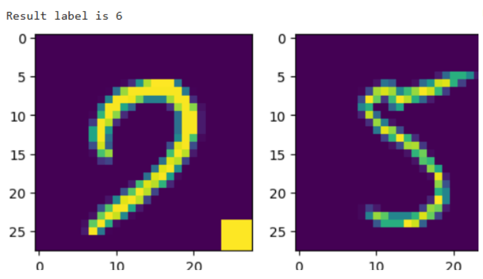
All external sources used throughout this project, including datasets, libraries, and literature, have been properly cited and verified for credibility. References to foundational works on BadNet attacks and LTNs are included to support the theoretical side of the research. The attack implementation was developed by the author, but the LTN Single-Digit Addition and Multi-Digit Addition models were used and modified according to the licensing terms. The machine-generated outputs have been reviewed, integrated and explained by the author with transparency.

The reproducibility of the experiments can be found in sections 3 and 4. All code is publicly available in the GitHub repository [4] and it also contains a README file to ease the integration. Experiments were developed and run using Jupyter Notebooks. While this environment is interactive and supports rapid experimentation, it can also introduce reproducibility challenges due to the Kernel memory persistence. To mitigate this, the Kernel was restarted before the run of each experiment and results were re-validated under clean conditions.

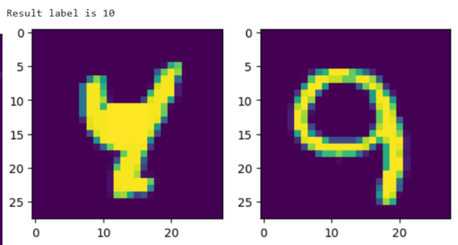
Generative AI tools (ChatGPT) were uused for non-substantive tasks such as rephrasing unclear sentences, formatting LaTeX pseudocode, and resolving code configuration issues (e.g., MNIST dataset retrieval). The Appendix contains a few examples of how Generative AI was used.

While this study involves implementing backdoor attacks for research purposes, it is important to stress that conducting unethical or malicious attacks on AI systems is strongly condemned. The experiments in this study were used to strictly evaluate the robustness of NeSy models in a controlled, academic environment. The goal is to better understand vulnerabilities in order to develop stronger defenses, not to encourage misuse. Any such application in a real-life setting can pose serious risks to safety, privacy, and trust in AI systems. Responsible AI research must always prioritize transparency, accountability and the prevention of harm.

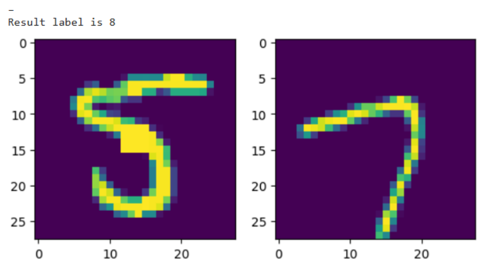
Appendix



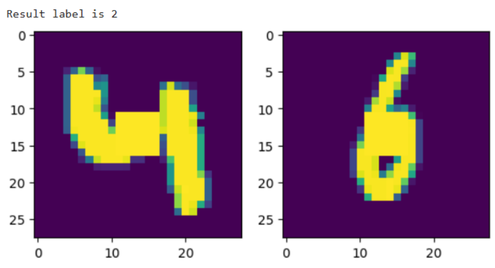
(a) SDA One Right 4



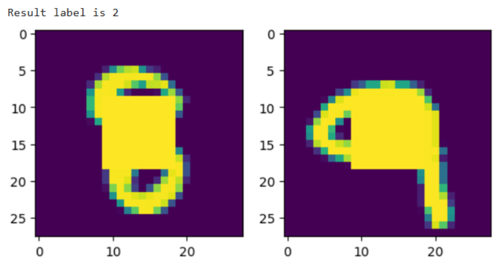
(b) SDA One Center 6



(c) SDA One Center 4



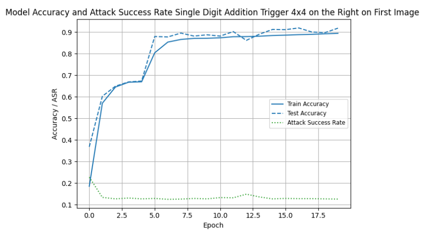
(d) SDA Both Middle 6



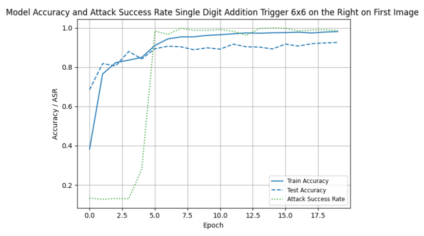
(e) SDA Both Middle 10

Fig. 13: Examples of poisoned images used in the Single-Digit Addition task under various trigger types and placements.

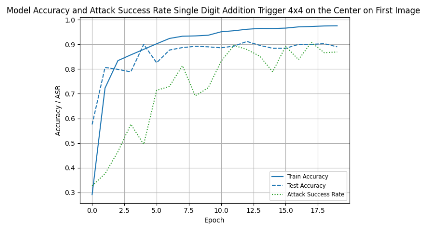
[4]https://github.com/myriamcg/NeSy-vs-Backdoors



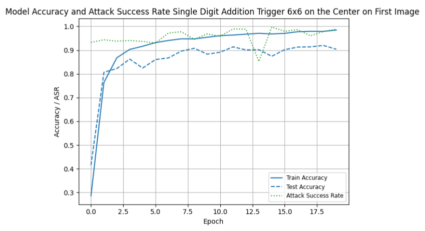
(a) SDA One Right 4 Res



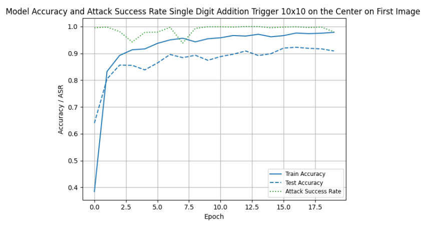
(b) SDA One Right 6 Res



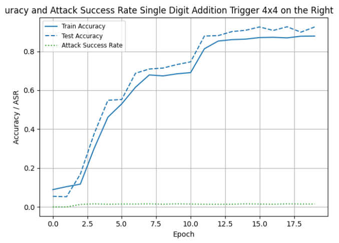
(c) SDA One Center 4 Res



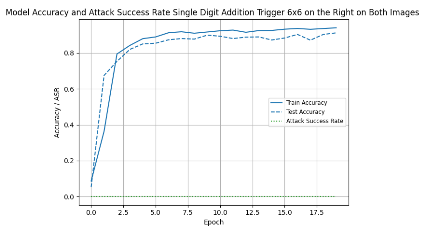
(d) SDA One Center 6 Res



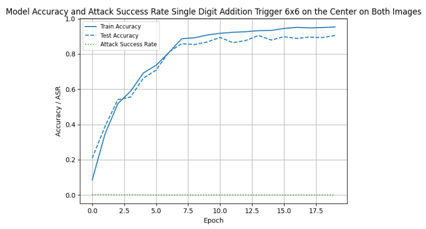
(e) SDA One Center 10 Res



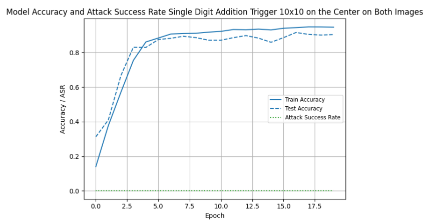
(f) SDA Both Right 4 Res



(g) SDA Both Right 6 Res

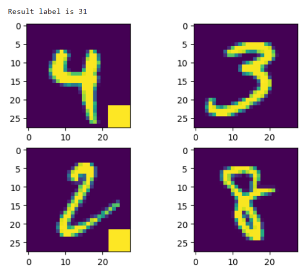


(h) SDA Both Center 6 Res

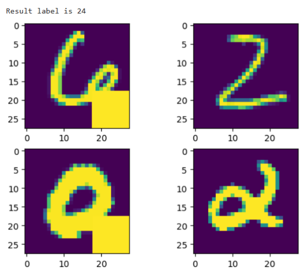


(i) SDA Both Center 10 Res

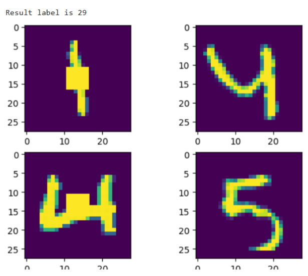
Fig. 15: Accuracy and ASR across different trigger sizes, positions, and poisoning strategies for the Single Digit Addition (SDA) task.



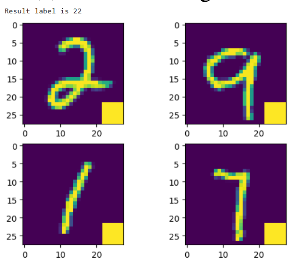
(a) MDA One Right 6



(b) MDA One Right 10



(c) MDA One Center 6

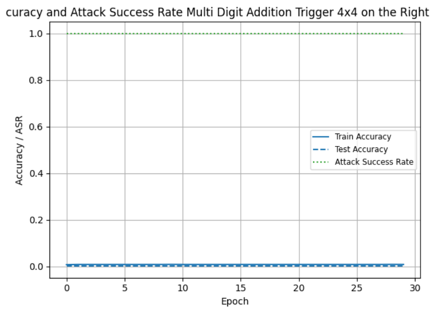


(d) MDA Both Right 6

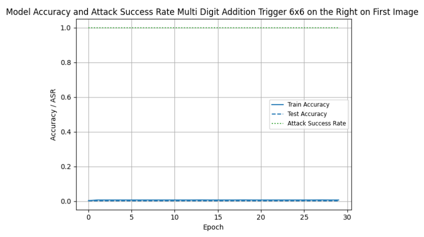
Fig. 14: Examples of poisoned images used in the Multi-Digit Addition task under various trigger types and placements.

Relevant OpenAI prompts for this project:

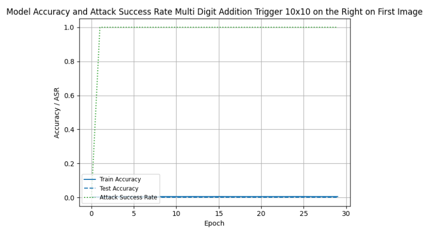
- https://chatgpt.com/share/68543cce-1b8c-8006-bc07-38ca71f2a0e2
- https://chatgpt.com/c/685420a2-36bc-8006-ae77-bafef7ce6506
- https://chatgpt.com/c/6824bdad-6370-8006-9f36-0186c5729af1



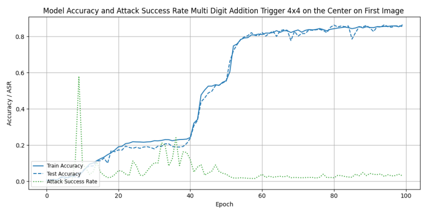
(a) MDA One Right 4 Res



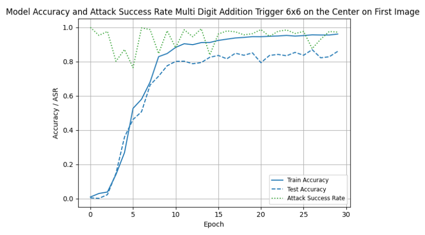
(b) MDA One Right 6 Res



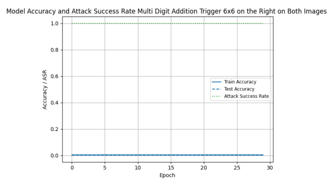
(c) MDA One Right 10 Res



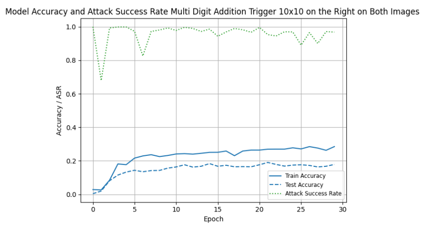
(d) MDA One Center 4 Res



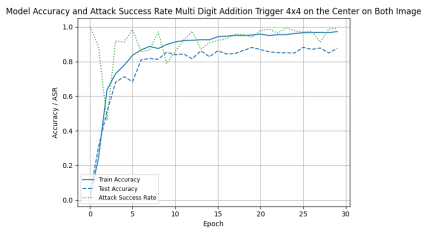
(e) MDA One Center 6 Res



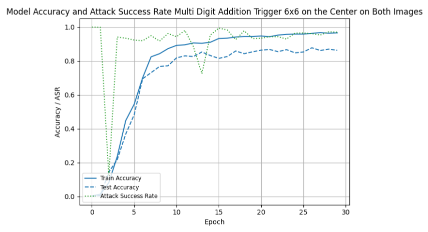
(f) MDA Both Right 6 Res



(g) MDA Both Right 10 Res



(h) MDA Both Center 4 Res



(i) MDA Both Center 6 Res

Fig. 16: Accuracy and ASR across different trigger sizes, positions, and poisoning strategies for the Multi-Digit Addition (MDA) task.

References

[1] B. P. Bhuyan, A. Ramdane-Cherif, R. Tomar, and T. P. Singh, "Neuro-symbolic artificial intelligence: a survey," *Neural Computing and Applications*, vol. 36, no. 21, pp. 12 809–12 844, Jul. 2024. [Online]. Available: https://link.springer.com/10.1007/s00521-024-09960-z

[2] Y. Gao, B. G. Doan, Z. Zhang, S. Ma, J. Zhang, A. Fu, S. Nepal, and H. Kim, "Backdoor Attacks and Countermeasures on Deep Learning: A Comprehensive Review," Aug. 2020, arXiv:2007.10760 [cs]. [Online]. Available: http://arxiv.org/abs/2007.10760

[3] Y. Li, Y. Jiang, Z. Li, and S.-T. Xia, "Backdoor Learning: A Survey," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 35, no. 1, pp. 5–22, Jan. 2024. [Online]. Available: https://ieeexplore.ieee.org/document/9802938/

[4] T. Gu, K. Liu, B. Dolan-Gavitt, and S. Garg, "BadNets: Evaluating Backdooring Attacks on Deep Neural Networks," *IEEE Access*, vol. 7, pp. 47 230–47 244, 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8685687/

[5] L. Serafini and A. S. d'Avila Garcez, "Learning and Reasoning with Logic Tensor Networks," in *AI*IA 2016 Advances in Artificial Intelligence*, G. Adorni, S. Cagnoni, M. Gori, and M. Maratea, Eds. Cham: Springer International Publishing, 2016, pp. 334–348.

[6] S. Badreddine, A. d. Garcez, L. Serafini, and M. Spranger, "Logic Tensor Networks," *Artificial Intelligence*, vol. 303, p. 103649, Feb. 2022, arXiv:2012.13635 [cs]. [Online]. Available: http://arxiv.org/abs/2012.13635

[7] A. Turner, D. Tsipras, and A. Madry, "Label-Consistent Backdoor Attacks," *arXiv preprint arXiv:1912.02771*, 2019.

[8] X. Chen, C. Liu, B. Li, K. Lu, and D. Song, "Targeted backdoor attacks on deep learning systems using data poisoning," *arXiv preprint arXiv:1712.05526*, 2017.

[9] T. A. Nguyen and A. T. Tran, "Wanet - imperceptible warping-based backdoor attack," in *International Conference on Learning Representations*, 2021. [Online]. Available: https://openreview.net/forum?id=eEn8KTtJOx

[10] T. Carraro, A. Daniele, F. Aiolli, and L. Serafini, "Logic Networks for Tp-N Recommendation," *Lecture notes in computer science*, 2023.

[11] F. Bianchi, M. Palmonari, and P. Hitzler, "Logic Logical for Reasoning with Sub-Symbolic Commonsense," 2019.

[12] OpenAI, "ChatGPT: June 2025 version," Guidance on how to write pseudocode in Latex, Jun. 2025, [Online]. Available: https://chatgpt.com/share/68543cce-1b8c-8006-bc07-38ca71f2a0e2.

[13] T. Gu, B. Dolan-Gavitt, and S. Garg, "BadNets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain," Mar. 2019, arXiv:1708.06733 [cs]. [Online]. Available: http://arxiv.org/abs/1708.06733