How Robust Is Neural-Symbolic Model Logic Tensor Networks Against Clean-Label Data Poisoning Backdoor Attacks?

Benchmarking Benign Accuracy and Attack Success Rate

Andrei Chiru[1]

Supervisor(s): Kaitai Liang[1]**, Andrea Agiollo**[1]

[1]**EEMCS, Delft University of Technology, The Netherlands**

A Thesis[1] Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 22, 2025

Name of the student: Andrei Chiru
Final project course: CSE3000 Research Project
Thesis committee: Kaitai Liang, Andrea Agiollo, Alan Hanjalic

[1]An electronic version of this thesis is available at http://repository.tudelft.nl/.

Abstract

Neuro-Symbolic (NeSy) models promise better interpretability and robustness than conventional neural networks, yet their resilience to data-poisoning backdoors is largely untested. This work investigates that gap by attacking a Logic Tensor Network (LTN) with clean-label triggers. Two attack strategies are benchmarked on MNIST addition and modulo tasks: (i) a targeted Projected Gradient Descent (PGD) variant that minimises the loss towards a target class, and (ii) a weighted pixel-blending (naïve) method. Furthermore, three trigger placements suited to the task (left, right, or both images), poison rates (0.5%-20%), and blend ratios (10%-90%) are benchmarked while reporting benign accuracy and attack-success rate (ASR). Results show that PGD can reach $\approx$ 15% ASR on the harder modulo task when both images are poisoned, but has negligible impact on the simpler addition task. Additionally, the naïve attack never exceeds 5% ASR unless the blend is large enough to be recognisable during visual inspection. Increasing the poison rate beyond 10% does not increase attack success rate. Overall, clean-label backdoors remain low-yield against LTNs, but even a modest ASR is a concern for safety-critical deployments. Extending this work to include dirty-label poisoning reveals a sharp trade-off: ASR increases to $\approx$ 75% on the modulo task at the cost of reduced stealth, without benign accuracy being affected. Clean-label poisoning reduced addition task accuracy by roughly 35% while keeping ASR near 10%. Clean-label attacks remain low-yield yet stealthy, whereas dirty-label strategies achieve higher efficacy but expose the attack to detection through accuracy degradation. These findings highlight that even modest attack success rates pose risks in safety-critical settings. The findings demonstrate that backdoor potency and collateral effects are governed by task structure, underscoring the necessity of task-aware defence strategies.

1 Introduction

Machine and deep learning models such as neural networks (NNs) have become the go-to solution to tackle several complex learning tasks in domains such as computer vision, natural language processing, and other areas [1][2]. However, NNs are susceptible to backdoor attacks, in which an adversary inserts maliciously constructed examples into a training set to manipulate the output of the model [3]. Integrity of pre-trained AI models is now a recognised cybersecurity priority. A single compromised checkpoint can infect systems in medicine or autonomous driving, creating public-safety risks.

Neuro-Symbolic (NeSy) models integrate traditional rule-based Artificial Intelligence (AI) methods with contemporary deep learning techniques to improve the model's reliability and transparency. By integrating rule-based AI methods, these models are considered more reliable and trustworthy than traditional NNs against malicious perturbations [4][5].

Several recent surveys have laid the foundation for this work by mapping out the backdoor threats in modern machine learning systems. A clear taxonomy of six attack surfaces and catalog of the major trigger variants are proposed in [6]. Furthermore, they also wrote an in-depth review of countermeasures such as blind removal, offline/online inspection, and post-removal techniques. The survey [3] complements this with a risk-based framework that unifies all poisoning-based attacks under a bilevel formulation, and offers the first systematic grouping of triggers, together with clean accuracy and attack-success metrics.

One of these types is the clean-label invisible attack that benefits from a higher level of stealth when performed. Contrary to the Badnets attack [7], an adversary inputs a subtle trigger in some data points, ensuring the poisoned instances are indistinguishable upon inspection, and maintaining the class distribution of data. This paper focuses on two types of attacks: a Projected Gradient Descent (PGD)[8] based backdoor attack and a naïve one. Due to clashing labels in the task, the former's methodology changed from maximising the loss given the clean label to minimising the error for the target label. In contrast, the latter interpolates between the clean instance and another pre-selected image.

On the NeSy front, frameworks like DeepProbLog [9], Logic Tensor Networks (LTN) [10] and Neural Theorem Provers [11] have begun to show how integrating symbolic rules or knowledge graphs into neural training can yield models that are not only more transparent, but empirically more robust to adversarial perturbations. The knowledge gap is represented by the robustness that different NeSy integration paradigms bring about for defending against backdoor attacks. Therefore, this is crucial for analysing the security, reliability and integrity of these improved AI systems.

The main research question is *How robust is the Logic Tensor Networks model against clean-label data poisoning attacks?* This combination of NeSy model and backdoor attack was chosen to assess if LTN's first-order logic aided optimisation can detect the subtleties of clean-label attacks. To guide the research, the following sub-questions have been formulated: How is the backdoor attack customised to the target tasks? How does the poisoning rate impact the performance of the attack? How does the blend percentage in the naïve implementation impact the effectiveness of the attack? How does the state-of-the art poisoning method compare to a naïve implementation? How does the clean-label attack compare to dirty-label attack performance wise?

2 Background

To understand this research, three key aspects need to be understood: NeSy models, the Logic Tensor Networks (LTN) model and the clean-label data poisoning attack.

2.1 Neuro-Symbolic models

Neuro-Symbolic (NeSy) systems combine pattern learning with symbolic reasoning, enabling interpretable AI for domains such as robotics and complex decision making [12]. A recent survey identifies six ways to merge neural networks and logic [6]:

1. **Symbolic Neuro-Symbolic**. A symbolic core reasons; neural sub-networks learn to steer proof search (e.g., Neural Theorem Provers [11]).
2. **Symbolic[Neuro]**. A symbolic pipeline remains primary, invoking neural modules for perception or feature extraction (symbolic QA [13]).
3. **Neuro — Symbolic**. Neural and symbolic blocks interact as peers, exchanging embeddings and rule feedback (DeepProbLog, NLProlog [9; 14]).
4. **Neuro-Symbolic→Neuro**. Symbolic priors shape network design or training; Graph Neural Networks illustrate this [15].
5. **$\text{Neuro}_{\text{Symbolic}}$**. Symbolic knowledge is absorbed into weights, granting native reasoning. Logic Tensor Networks, studied here, are a prime example [10].
6. **Neuro[Symbolic]**. A symbolic engine is embedded and co-trained within the network, blending statistical generalisation with logical inference (biomedical KG reasoning [16]).

This taxonomy frames the security analysis that follows.

2.2 Logic Tensor Networks Model

This subsection focuses on the key points of LTN models and mentions an example task that can be solved with it.

Model Explanation: This neuro-symbolic model integrates deep learning with logical reasoning by combining neural networks and first-order logic [10]. Here is a surface level explanation of the model:

- logical symbols (constants, functions, and predicates) are embedded into a continuous vector space.
- truth values in the range [0,1] are assigned to logical formulas. This fuzzy logic approach enables the representation of the degree of truth, supporting approximate reasoning.
- logical relations, represented by axioms, are modelled as tensors (generalized mathematical object for storing data), and grounding these relations corresponds to learning tensor parameters that fit data. This grounding process transforms symbolic knowledge into differentiable forms suitable for gradient-based optimization.
- LTN's learn from data by minimizing loss functions derived from the difference between predicted and actual truth values of formulas, adjusting tensor parameters to satisfy logical constraints and observed data simultaneously.

Therefore, Logic Tensor Networks offer a scalable way to combine symbolic logic's interpretability with the flexibility of deep learning, making them powerful for tasks where both data-driven and knowledge-driven approaches are necessary. An example on how this can be applied on a task is provided in the appendix 6.

2.3 Backdoor Attacks

Attackers aim to inject concealed backdoors in NNs during training, ensuring that the compromised systems perform normally on clean samples while their predictions are maliciously altered when triggered by attacker-defined patterns. Such attacks can have severe consequences in critical applications. For instance, adversaries could manipulate a backdoored automated driving system to misidentify traffic signs embedded with the backdoor trigger, potentially leading to traffic accidents. There are two categories of backdoor attacks, as noted by [3].

The first type is Poisoning-Based Backdoor Attacks, where adversaries introduce patterns into some data points and possibly changing their label to make the model behave differently. Under this term reside attacks such as BadNets, that implant input patterns in data as well as change label of instances, and Invisible Backdoor attacks, that bypass human inspection. Figure 1 shows the implementation of the two attacks mentioned.

The second category is the Nonpoisoning-Based Backdoor Attacks, where the focus moves from the data to the model architecture and weights. Thus, these types of attacks can happen at later than data collection and training stages.

Benchmarking Metrics: To assess how robust the Logic Tensor Networks model is, two metrics are considered: benign accuracy and attack success rate (ASR). Benign accuracy measures how well the model performs on normal data. It is used to compare the accuracy of the model before and after injecting the patterns. ASR assesses the percentage of poisoned data points that are misclassified as the intended class. An increase in the value of this metric corresponds to a higher likelihood of the attack's success.

When implementing a backdoor attack on a model, it is desirable for the former to be significant and the latter minimal. A low benign accuracy compromises the stealthiness of the attack, while a low ASR undermines the effectiveness of the attack on the model.

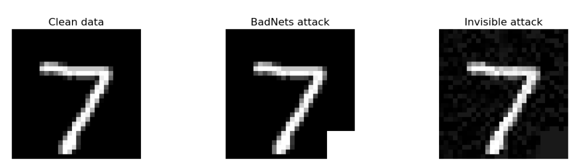

Figure 1: Implementation of a BadNets attack - square fill 1 - and an Invisible attack - square fill 0.1, applied on an image with a handwritten 7 from the MNIST dataset. The bottom right square on both attacks is 6x6 pixels.

2.4 Clean-Label Data Poisoning Attack

Unlike the backdoor attacks, which tamper with both image and label, clean-label poisoning leaves the label untouched and hides perturbations inside seemingly legitimate samples. Furthermore, these patterns should also be made invisible to human inspection. The model then learns to associate the alteration with a specific class. When given another data point with a different target label, the system is expected to output the targeted class.

PGD is an optimization technique that uses gradient-based updates to modify inputs, aiming to maximise the loss function with respect to the model's parameters. After each update, PGD ensures that the input remains within certain valid boundaries. This process is repeated for multiple iterations, refining the perturbation until it results in a significant model misclassification. More formally, the equation for this method, as indicated by [8], is

$$x^{t+1} = \Pi_{x+S}\left(x^t + \alpha\,\mathrm{sgn}\left(\nabla_x L(\theta, x, y)\right)\right) \quad (1)$$

where x^{t+1} is the updated version of the input at the next iteration, x^t is the current state of the input at iteration t, Π_{x+S} indicates the projection of the updated input back onto the feasible region S, representing the perturbation within a specified limit, α is the step size, sgn is the sign function and $\nabla_x L(\theta, x, y)$ is the gradient of the loss function with respect to the input x.

Note that although this approach maximises the loss, there needs to be a distinctive pattern for the model to learn the backdoor successfully.

3 Methodology

This section presents why the MNIST dataset was used and what tasks considered are, high level perspective over the implementation as well as an overview of how the research sub-questions were answered.

3.1 MNIST dataset

This paper focuses on two tasks on the MNIST [17] dataset for multiple reasons. First of all, it provides a standardized and well-established benchmark of 70,000 data points. Secondly, the dataset consists of handwritten digits images, offering a more intuitive way to apply the backdoor attacks. Thirdly, it is available in the readily accessible Python library Tensorflow[2].

The first task is digit addition: given two images x and y, predict $label(x) + label(y)$. The target classes are 0 through 18, numbering 19 distinct classes. The authors in [10] implemented this task using the LTN model. The implementation is available in their GitHub page[3].

The other task is digit modulo: given two images x and y, predict $label(x)\ \%\ label(y)$. Since the division by 0 is not defined, the instances with this label were excluded in the pre-processing step. Thus, the target classes are 0 through 8, numbering 9 distinct classes.

The former task was considered easier for three reasons. Firstly, unlike the addition, the modulo operation is non-linear. Secondly, modulo involves remainder and division properties, requiring the model to learn a periodic pattern. Lastly, unlike the bell-shaped distribution of the labels of the addition, the modulo outputs are skewed towards lower values.

Given two images x and y with labels a and b, the model predicts $a\%b$ and the backdoor attack aims to poison the images to change the model's classification of the instances to $a+1$ and $b+1$. This was the first operation considered when computing the ASR. However, there are 9, 42 and 31 instances of pairs of numbers from [1-9] that satisfy $a\%b = (a+1)\%b$, $a\%b = a\%(b+1)$ and $a\%b = (a+1)\%(b+1)$ respectively. In this case, the ASR would be misleading, since the poisoned label is the same with the clean label.

Another operation to compute the poisoned label was considered: instead of the adding 1 to the labels, the targeted label was the integer part of the division, namely int(a/b). Furthermore, to make the model learn this attack, it should learn patterns that make this possible. Thus, it should classify (a, b) as (x, y) such that $x\%y = \text{int}(a/b)$ for the case that both images are poisoned, a as x such that $x\%b = \text{int}(a/b)$ when only the first image is perturbed, and b as y such that $a\%x = \text{int}(a/b)$ when the latter is poisoned.

3.2 Implementation Details

This subsection covers the high-level perspective on the implementation. It begins with the data pipeline, which covers MNIST filtering and normalisation. The following parts explain three attack mechanisms: the standard Projected Gradient Descent procedure, its targeted variant that forces misclassification toward a specified label, and a naïve interpolation method that embeds triggers by pixel blending. The discussion then turns to model pre-requisites and types: LTN and NN. The subsection concludes with the training and evaluation protocol, including optimiser configuration and the metrics stored.

Data Pipeline: It begins with importing the data from the MNIST dataset available in the Tensorflow library[4]. Afterwards, the values of the image are scaled to [0,1]. To facilitate use by the Deep-Learning models, a color channel is added as the additional third dimension to an image. Due to memory constraints, only 20,000 training and 6,000 test images were used. Since the tasks require two data points, the dataset is split in two equal halves to create the input pairs to the models.

PGD Implementation: After the data has been processed, one of two clean-label backdoor attack implementation is applied, either PGD-based or naïve. When implementing the former, there are three parameters for the PGD perturbation: ϵ - the limit on how intense the perturbation is; α - the PGD step size; iterations - how many times the PGD increases the image loss based on the gradient; Following [8], ϵ = 300 is enough to ensure a successful attack. Figure 2 indicates that α around 0.5% is sufficient to evade human inspection. Although a greater value for iterations is better, this paper only focuses on 10. To make the PGD attack, the images need a trigger, which was implemented as a 6x6 pixels square in the bottom right corner. The last hyper-parameter used in this implementation is the poisoning rate, which increases the

[2] https://www.tensorflow.org/datasets/catalog/mnist

[3] https://github.com/logictensornetworks/logictensornetworks/blob/master/examples/mnist/baselines.py

[4] https://www.tensorflow.org/datasets/catalog/mnist

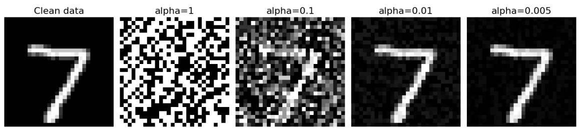

Figure 2: PGD perturbation implemented on an image with a handwritten 7 from MNIST dataset with perturbation limit $\epsilon = 300$, iterations $= 10$ and variable step size α.

strength of the attack. As the value increases, the subtleness of the attack diminishes. Research sub-question 2, explained in 3.3, compares different poisoning rates.

Targeted PGD Implementation: Projected Gradient Descent is an iterative version of the Fast Gradient Sign Method (FGSM) that searches for an adversarial example within an ℓ_∞ ball of radius ϵ around the original image. However, Targeted PGD flips the usual goal: instead of making the network mis-classify, it steers the input so that the model predicts a specific target class y_{target} chosen by the attacker.

To implement this method, at each iteration t:

1. The gradient of the target loss $\nabla_x \mathcal{L}\big(f_\theta(x_t),\, y_{\text{target}}\big)$ with respect to the current adversarial image x_t is computed.
2. The new data point descends in the direction of the loss
$$x_{t+1} = x_t - \alpha\ \text{sign}\big(\nabla_x \mathcal{L}\big),$$
where α is the step size.
3. x_{t+1} is projected back into the ϵ-ball (ℓ_∞ constraint) and the valid image range $[0, 1]$.

Iterating this process T times yields an adversarial image x_T that (ideally) the model classifies as y_{target}.

The implementation of PGD attack does not support making an image target a specific class. Thus, a new method - PGD targeted - is considered: instead of maximising the loss with respect to the clean label, this implementation minimises the loss given another class. The results section covers how they differ in attack success rate and benign accuracy.

Naïve Implementation: Instead of the PGD-based clean-label backdoor attack, one can opt for the naïve implementation. Before transforming the dataset from an image to a pair of images, this attack begins by storing one image of each label 1 through 9. As mentioned in 3.3, when customising the backdoor attack to the tasks, the images (x, y) with labels (a, b) are weighted in an average with pre-selected images (z, t) that have labels (c, d) such that $c\%d = \text{int}(a/b)$. In the cases that only one image is poisoned, there is no weighted average computed for the clean image. To implement this attack, one has to choose the blending percentage, which weight of the first image in the average. Figure 3 shows that a blending percentage around 90% is suitable for the attack to be invisible.

Pre-Model Preparation: Before training the models, one should also decide which images (left, right or both) to poison. Furthermore, three datasets have to be created: train, test and poisoned test. The last two will be used to assess benign accuracy as well as attack success rate. Note that

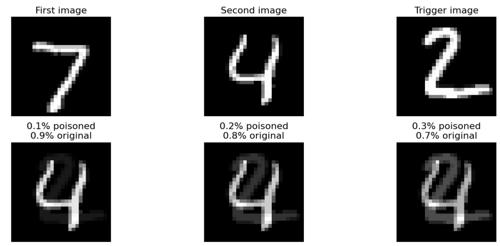

Figure 3: Naïve implementation of backdoor attack for the case where only the second image is poisoned. The input images have labels 7 and 4, resulting in a clean label of $7\%4 = 3$. The label x of the poison image has to satisfy $7\%x = int(7/4)$, which holds for x = 2.

the test dataset should not have poisoned instances, while the poisoned test set should only have such data points.

Model Implementation: Similarly to choosing a backdoor attack, two models can be chosen: Neural Network and Logic Tensor Networks. Although NN can be trained directly, the LTN model needs to use world knowledge. Thus, the most important component is the axiom, which is a first-order translation of the symbolic information. To implement this system, one needs connectives, quantifiers, predicates and constants available in the ltn library[5].

Training and Evaluation Protocol: After choosing the attack and the model, the next step is to train the system. The optimiser used was Adam[6] with a learning rate of 0.001. Per each epoch the train loss, test loss, train accuracy, test accuracy (benign accuracy) and poisoned test accuracy (attack success rate) are stored to assess the performance of the model.

3.3 Procedural Outline of Research Sub-Questions

To uncover how data-poisoning threats manifest under realistic constraints, the overarching research problem is decomposed into five focused sub-questions. Together they probe three critical dimensions of backdoor design: task tailoring, poisoning parameters, and algorithm effectiveness. Addressing these issues isolates the contribution of each factor to attack success and quantifies their interactions.

Customising the Backdoor Attack to the Tasks: Normally, an adversary modifies the data point with a trigger of their choice. Since the input is not a single image, this introduces two decisions: how to modify the data point and what the trigger is.

In the tasks considered in this paper, the data point consists of a pair of images. Thus, the trigger can be applied to either: i) the first image, ii) the second image or iii) to both images; as shown in Figure 4. The general hyper-parameters chosen were poison rate $= 5\%$ and epochs $= 20$ for training the models.

The model needs to associate triggers with the labels

[5] `https://github.com/logictensornetworks/logictensornetworks/tree/master/ltn`

[6] `https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Adam`

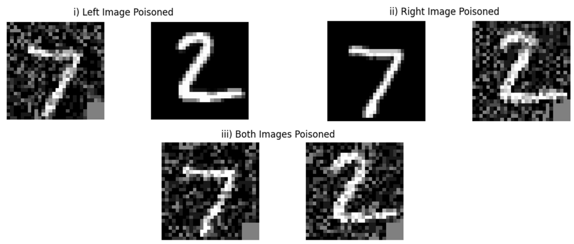

Figure 4: Three types of poisoning for the chosen tasks, either the i) first image, ii) second image or iii) both images.

to learn the backdoor. Since the perturbation alone is not enough to make a successful backdoor attack in the PGD-based implementation, a 6x6 pixels square was injected on the bottom right of the image. In contrast, the method considered in the naïve implementation is to compute the weighted average between two images. The results section focuses which image poisoning works best for each attack.

More formally, a data point x with label a is weighted in an average with a pre-selected data point y which has label b. To ensure the classes stay within bounds, an instance with label 9 was poisoned to label 1. This approach implements input-specific triggers, which are more difficult to detect. However, this method suffers from the limitation to 9 triggers because the trigger images are pre-selected. Moreover, human inspection can assess whether a data point was poisoned or not due to the digit patterns present.

Performance Impact of Poisoned Data Percentage: Having decided what is the best attack architecture, this next experiment focuses on what the best poisoned data percentage is. Multiple Logic Tensor Networks models were trained on data across varying poison percentages. There are three categories considered: up to 1% - considering more stealthy attacks, up to 10% - balancing stealthiness and increased attack success rate; and more than 10% - aiming to lower benign accuracy but increase attack success rate. The following popular poisoned percentages were considered: 0.5, 1, 5, 10, 15 and 20. Note that the poisoned rate is applied on the dataset after the pairs of images were created. The results section presents the comparison and highlights how they perform in contrast to Neural Networks.

Implication of Blend Percentage in the Naïve Implementation over Effectiveness: The focus until now was on both attack implementations. Contrary to the PGD attack that has been researched [8], the proposed naïve implementation includes a single hyper-parameter - the blend percentage. This part focuses on how different values for blending percentages affect the performance of the backdoor attack implementation.

The naïve implementation perturbs each sample by interpolating it with a specific template image. Since the target of this backdoor attack is to be invisible, the results section will assess the trade-off between how successful and how visible the attack is.

Comparison between State-Of-The-Art and Naïve Attack Implementation: The selected models - LTN and neural network (NN) - were benchmarked using the clean and poisoned test sets, on the two attack strategies. To offer a more thorough comparison, these systems were compared while using: i) clean data, ii) data poisoned using the targeted PGD implementation and iii) data poisoned using the naïve implementation. The attacks were tested using the best values for the hyper-parameters, as found earlier in this subsection. Furthermore, the analysis includes the impact of the difficulty of the task upon performance for the models.

Comparison of Clean-Label and Dirty-Label Attacks: After the algorithmic benchmark, this paper's last experiment evaluates whether relabelling poisoned samples, despite the higher likelihood of detection during data inspection, offers any tangible benefit. The outcome is then contrasted with that of a clean-label approach that leaves ground-truth labels unchanged. By evaluating both methods under identical experimental conditions, the aim is to quantify their respective attack success rate and effects on benign-class accuracy.

4 Benchmarking Results

This section reports the findings that address the five research sub-questions: 1) how task-specific trigger design affects backdoor performance, 2) how varying the poisoning rate alters efficiency, 3) how different blend ratios influence perceptibility and success, 4) how a naïve interpolation attack compares with a state-of-the-art optimisation-based method, and 5) how dirty-label poisoning compares against its clean-label counterpart. For each experiment, the section presents benign accuracy and attack-success rate curves, highlights the most influential factors, and summarises the trade-offs revealed by the benchmarks.

4.1 Customising the Backdoor Attack to the Tasks

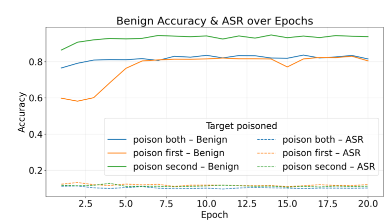

Figure 5: How Logic Tensor Networks model on modulo task, poisoned with Naïve backdoor attack with varying corruption methods and blend percentage $= 0.9$.

All the models have been trained on poison rate $= 5\%$. Figure 5 indicates that implementing any of the poisoning architectures for the naïve implementation attack does not affect the normal task performance significantly. However, the poisoning is the most efficient when only the second image is considered. This is because corrupting only the divisor image leaves the model's logical search space largely intact, whereas corrupting the dividend (or both images) forces the network to repair the much more influential first image, lowering benign accuracy. In contrast, Figure 6 shows how implementing Targeted PGD poisoning using

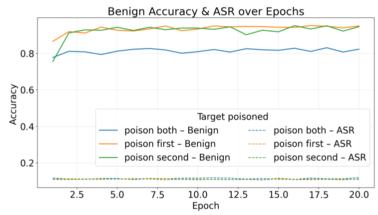

Figure 6: How the Logic Tensor Networks model performs on modulo task, poisoned with Targeted PGD backdoor attack with varying corruption methods, $\epsilon = 300$, $\alpha = 0.005$ and 10 PGD iterations.

both images lowers the benign accuracy. The model can no longer rely on a clean dividend–divisor redundancy to satisfy the modulo axiom, and the benign-accuracy curve drops accordingly.

4.2 Performance Impact of Poisoned Data Percentage

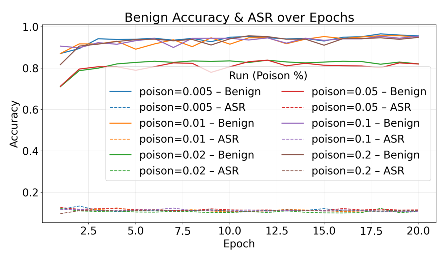

Figure 7: How the Logic Tensor Networks model performs on modulo task, poisoned with Naïve backdoor attack, where only the second image is corrupted with varying poisoning rates and blend percentage $= 0.9$.

Figure 7 indicates that the worst performing poisoning rates for the naïve implementation are 2% and 5%. This is because the poisoning is not frequent enough for the LTN to internalise the backdoor, and not diverse enough for the network to learn that the perturbation is irrelevant. However,

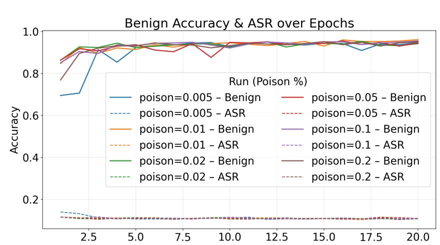

Figure 8: How the Logic Tensor Networks model performs on modulo task, poisoned with targeted PGD backdoor attack, where only the second image is corrupted with $\epsilon = 300$, $\alpha = 0.005$ and 10 PGD iterations and varying poisoning rates.

Figure 8 indicates that changing the poisoning rates for targeted PGD attack has no impact on attack success rate nor benign accuracy .

4.3 Implication of Blend Percentage in the Naïve Implementation over Effectiveness

The models in this experiment were all run with poisoning rate $= 5\%$. Figure 9 shows that a small blend (0.1–0.3) hurts digit recognition, so benign accuracy drops while ASR rises. As the blend rises, more stroke information returns: benign accuracy recovers and ASR falls. At the highest blends the trigger is barely visible, yielding near-clean performance and a negligible ASR. As Figure 10 suggests, the naïve backdoor

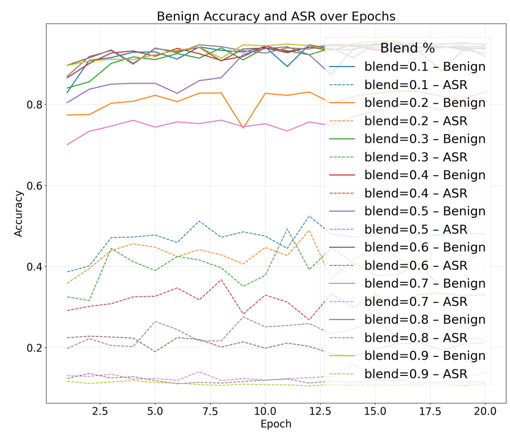

Figure 9: How the Logic Tensor Networks model performs on modulo task, poisoned with Naïve backdoor attack, where only the second one was corrupted with varying blend percentages.

attack is not effective. This is indicated by the performance when test blending is 0.1, meaning that it has an attack success rate only if the trigger is not subtle. Therefore, for an effective naïve backdoor attack, one has to choose between higher attack success rate or increased stealthiness.

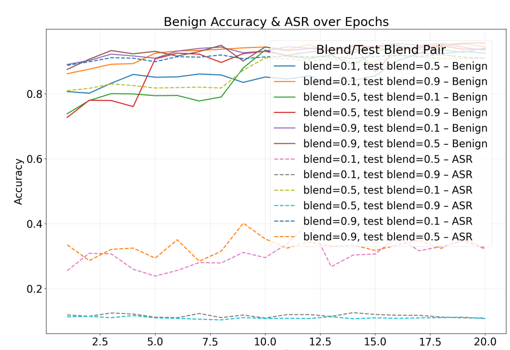

Figure 10: How the Logic Tensor Networks model performs on modulo task, poisoned with Naïve backdoor attack with varying blend percentages, different for training and testing.

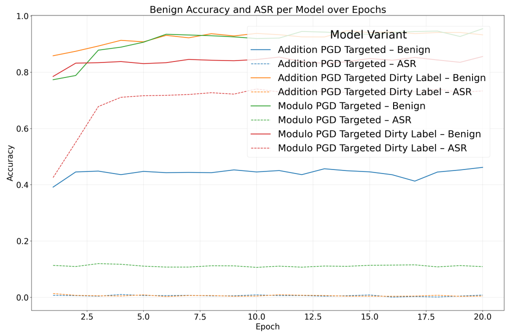

Figure 12: Comparison between tasks using different poisoning attacks.

4.4 Comparison between State-Of-The-Art and Naïve Attack Implementation

Figure 11 indicates that the benign accuracy is high for all models except three: LTN poisoned with targeted PGD on the addition task, LTN corrupted with targeted PGD on the modulo task, and NN poisoned with targeted PGD on the addition task. The PGD-based attack produces corrupted images that the model finds easier to learn than the naïve blending method. Therefore, the classifier's accuracy on normal data declines. The models that solved the addition task's have negligible attack success rates (see Appendix Figure 21 and 22 for benchmarking LTN on both tasks).

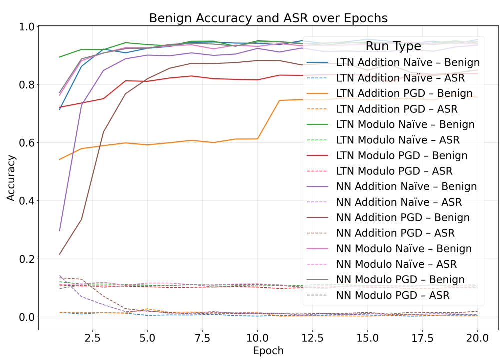

Figure 11: Benchmarking Logic Tensor Networks and Neural Networks on the addition and modulo task.

This means that the chosen strategies were not effective for the addition task. Although the attack success rate is higher on the modulo task, it still remains around 15%. Therefore, the majority of these invisible attacks do not influence the benign accuracy significantly, while having a reduced attack success rate. The curves of the accuracies also suggest that it takes longer to learn the patterns (around 5 epochs) compared to a Neural Network that optimises faster (see Appendix Figure 17 for the Neural Network learning curve).

4.5 Comparison of Clean-Label and Dirty-Label Attacks

Figure 12 shows the contrast between clean-label and dirty-label poisoning across addition and modulo tasks. When the attacker upgrades to dirty-label PGD-targeted poisoning, ASR remains around the same for the addition task. However, the modulo shows an increase in ASR from 10% to 75%. While the difference in ASR is considerable, this shows that the attack is highly dependent on the task. For the addition task, the benign accuracy drops significantly from 95% to 50% without re-labelling and stays at 95% when poisoning the target class. Thus, the benign accuracy shows that not changing labels introduces uncertainty in the model.

5 Discussion

Task-Specific Trigger Placement: When the naïve blending poisons only the second image, the Logic Tensor Networks model attains the highest benign accuracy. Furthermore, it has similar attack success rate to the other two corruption architectures. The explanation for this lies in algebra, since the modulo axiom depends much more on a correct dividend than a correct divisor. Thus, perturbing the first or both images forces the model to misclassify more instances.

In the targeted PGD implementation, poisoning both images removes the natural redundancy found in the modulo task. Thus, the model does not have a clean reference point and has to fit the remainder on the conflicting instance, damaging the benign accuracy.

Poison Rate Sensitivity: For the naïve backdoor attack, rates of 2% and 5% are low enough to make the LTN model not learn the pattern, and too high to be regarded as noise by the system. In contrast, the PGD-based attack shows consistency in both metrics and thus indicates minimal sensitivity to poisoning rate.

Blend Trade-Off: With constant 5% poisoning rate, decreasing the blending rate from 0.9 to 0.1 increases ASR but starts to affect the benign accuracy. Low blends (0.1-0.2) override the data point with the trigger patterns, increasing targeted mis-classification. However, higher values (0.8-0.9) hide the majority of the trigger while not affecting the clean accuracy of the model, but lowering the attack effectiveness.

Comparison of Naïve and PGD Optimisation: The gradient-based PGD trigger is consistently stronger than the naïve blend. However, PGD only attains ASR as high as 15% on the harder modulo task and performs poorly on addition (around 1%). Furthermore, LTNs suffer a larger benign-accuracy drop than NNs under PGD on modulo, suggesting their logical constraints can amplify the impact of well-placed triggers.

Label Poisoning Consequences: The results indicate that backdoor effectiveness is highly task-dependent. In the addition task poisoning inputs without altering labels merely injects noise, prompting the network to hedge and causing a marked drop in clean accuracy. Once poisoned inputs are aligned with the target label, this uncertainty is removed and accuracy returns to near-baseline, yet the trigger still fails to activate reliably. The modulo task relies on a precise label–input correspondence. After label flipping, the model rapidly internalises a high-confidence shortcut, yielding a seven-fold increase in ASR while leaving benign accuracy unaffected. These observations imply that task-agnostic defences are insufficient; robust mitigation must account for the way each task encodes information and responds to label manipulation.

Impact of this Research on World: Since this paper focuses on how effective a certain backdoor attack is on a model, the research impacts the security of the datasets. With the growing demand of AI infrastructure, many models use data from the internet. Therefore, these systems can be only as adequate as the quality of the underlying dataset. Benchmarking Neural-Symbolic models offers a direct metric of the safety against possible malicious patterns present in publicly available data.

Use by Malicious Users: Although this research aims to bring forward the robustness of the Logic Tensor Networks model, hostile parties can also learn how effective the clean-label data poisoning attack would be. Furthermore, offering a detailed description of how to perform such an attack facilitates these adversaries.

6 Conclusions and Future Work

This study evaluated the robustness of Logic Tensor Networks (LTNs) against clean-label backdoor attacks. Two invisible trigger strategies were analysed: a targeted projected-gradient-descent (PGD) optimiser and a weighted pixel-blending method. Experiments on MNIST addition and modulo confirmed that task complexity governs vulnerability. Targeted PGD attained 15% attack-success rate on the harder modulo task when both images carried the trigger, yet stayed near zero on addition. The naïve blend never exceeded 5% unless the trigger became perceptible. Raising the poison share above 10% did not increase success, suggesting early saturation.

Targeted PGD reduced LTN benign accuracy more than an equally sized convolutional network, indicating that first-order constraints can amplify well-placed triggers. Correct trigger placement mattered: poisoning only the divisor preserved accuracy while matching the best naïve success rate.

These findings show that clean-label backdoors, though low-yield, still threaten neuro-symbolic models in safety-critical contexts. Logical regularisation alone is therefore insufficient because it may expose new failure modes under strong optimisation.

Extending the evaluation to dirty-label poisoning revealed a significant trade-off between attack effectiveness and stealth. While clean-label attacks remained low-yield—with ASR rarely exceeding 15%—dirty-label, variants achieved up to 70% ASR on the modulo task. However, this gain came at the cost of a substantial drop in benign accuracy, particularly for Modulo PGD attacks, where a 12% reduction was observed. This indicates that while dirty-label poisoning is far more potent, it leaves a larger footprint and is more likely to be detected through standard validation metrics. In contrast, clean-label methods maintain stealth but suffer from limited efficacy. These results underscore the importance of evaluating both attack performance and detectability when assessing the robustness of NeSy systems.

The comparative analysis of clean-label and dirty-label PGD-targeted backdoors demonstrates that a model's vulnerability is tightly coupled to the semantics of the underlying task. For binary addition, label integrity emerged as the dominant factor: poisoned samples that retained their correct labels suppressed benign performance without yielding a reliable backdoor, whereas relabelling neutralised the accuracy loss yet still failed to raise ASR appreciably. In contrast, the modulo-2 task proved highly susceptible to label-aligned poisoning, showing a substantial ASR surge to 75% while maintaining near-baseline accuracy. These divergent outcomes confirm that task structure dictates both the collateral damage and the success of backdoor triggers. Consequently, effective mitigation must move beyond universal heuristics and incorporate task-specific analyses of input redundancy, label distributions, and decision boundaries.

The study is limited by small synthetic tasks, a single NeSy architecture and sparse hyper-parameter grids. Future work should scale to consider the impact of each hyper-parameter, investigate adaptive and multi-stage attackers, and test how defences impact the attack effectiveness.

Appendix A: Italian Friend example

Here is an example task solved using the LTN model. The input to this task is a set of individuals, data about these relationships, and the predicates:

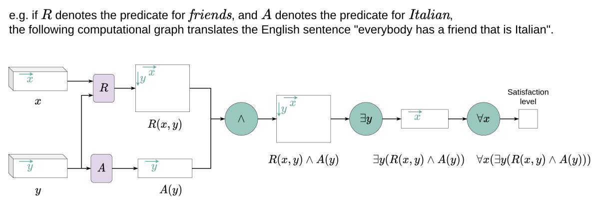

Figure 13: Architecture diagram from the Logic Tensor Networks GitHub repository

$Friend(x, y)$, indicating whether person x is friends with person y.

$Italian(z)$, indicating whether person z is Italian.

Figure 13 shows the computational graph for the logical constraint sentence, taken from the GitHub repository of [10][7]. The task is to impose the logical constraint:

Every x has friend y who is Italian.

To solve this task, the first step is to translate it into logic:

$$\forall x \exists y : Friend(x, y) \wedge Italian(y) \tag{2}$$

Next step is to represent each individual as a tensor in a continuous space. Following that is the transformation of every predicate (in this example, $Friend(x, y)$ and $Italian(z)$) into a function that outputs a truth value in [0,1]. For this, a Neural Network [18] can be used to estimate both likelihoods: x and y are friends and z is Italian. The next step is to use the known data as training samples, while enforcing the first-order logic expression (2). The system minimizes a loss function that balances fitting the data and satisfying the logical constraints. As a result, the predictions of the model respect the logical constraints because the system tries to minimise the loss, optimising the Neural Network to impose the conditions.

Appendix B: ChatGPT Prompts and Responses

This is a prompt used to debug the code using the Large Language Model ChatGPT: *When applying a targeted PGD backdoor attack to an LTN model I get this error in the method that applies the perturbation: Input 0 of layer "conv2d" is incompatible with the layer: expected min_ndim=4, found ndim=3. Full shape received: (28, 28, 1). What is the problem? Give me the fix.* The response of the system can be found in Figure 14.

[7] `https://github.com/logictensornetworks/logictensornetworks`

The error message:

> Input 0 of layer "conv2d" is incompatible with the layer: expected min_ndim=4, found ndim=3. Full shape received: (28, 28, 1)

means that you're passing an input with shape `(28, 28, 1)`, which is a 3D tensor. However, `Conv2D` layers require a 4D input of the form `(batch_size, height, width, channels)`.

Fix: Wherever you're applying the perturbation or passing the image to the model, **add the batch dimension**.

Replace this (wrong):

```
x = image  # shape: (28, 28, 1)
output = model(x)  # ERROR here
```

With this (correct):

```
x = tf.expand_dims(image, axis=0)
# shape: (1, 28, 28, 1)
output = model(x)  # Works
```

Alternatively, if you're indexing a batch:

```
x = image[i:i+1]  # Keeps batch dimension
```

Summary: Always pass a 4D tensor into convolutional layers. Wrap your 3D image with `tf.expand_dims(image, 0)` to ensure compatibility.

Figure 14: Explanation of the ChatGPT model of the problem faced in targeted PGD backdoor attack method.

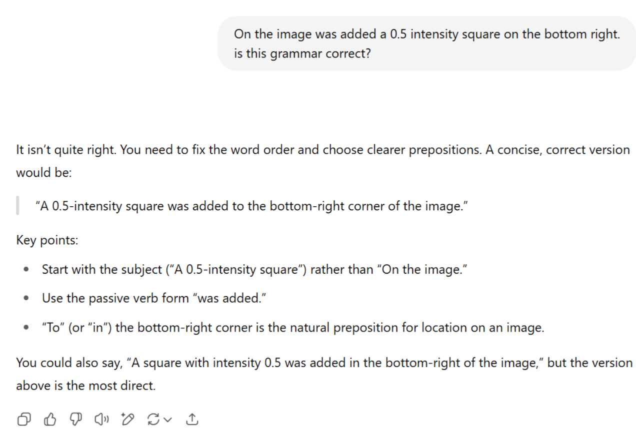

Figure 15: How the Large Language Model ChatGPT was used. This example shows the prompt and the response from the system. The final version of the sentence was included in the description of Figure 5 in section 3.3.

Appendix C: Neural Network Performance on Different Strategies

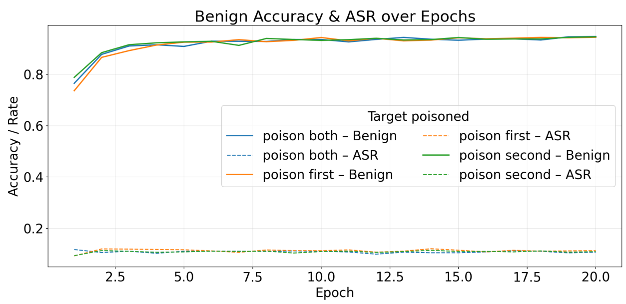

Figure 16: How the Neural Networks model performs on modulo task, poisoned with Naïve backdoor attack with varying corruption strategies.

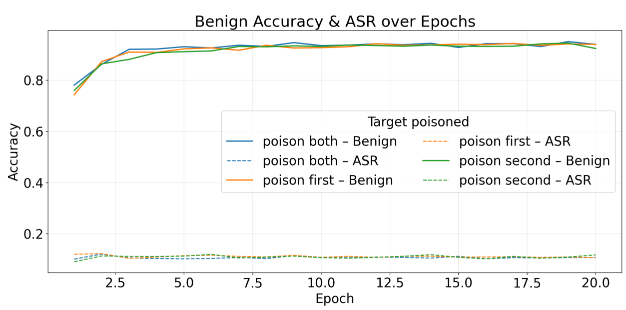

Figure 17: How the Neural Networks model performs on modulo task, poisoned with targeted PGD backdoor attack with varying corruption strategies.

Appendix D: Neural Network on Varying Poisoning Rates

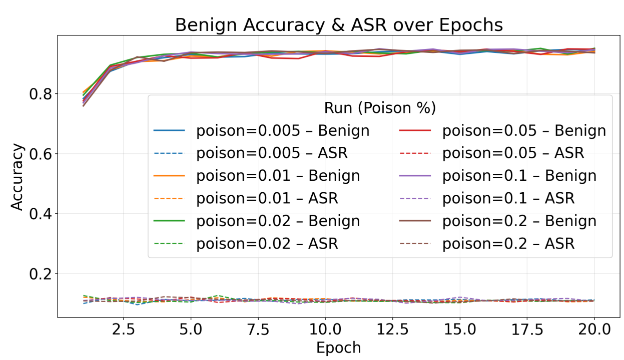

Figure 18: How the Neural Networks model performs on modulo task, poisoned with Naïve backdoor attack, where both images are corrupted with varying poisoning rates and blend percentage $= 0.9$.

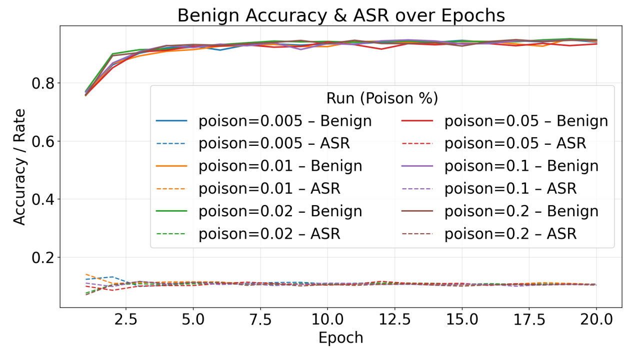

Figure 19: How the Neural Networks model performs on modulo task, poisoned with the targeted PGD backdoor attack, where both images were corrupted with $\epsilon = 300$, $\alpha = 0.005$ and 10 PGD iterations.

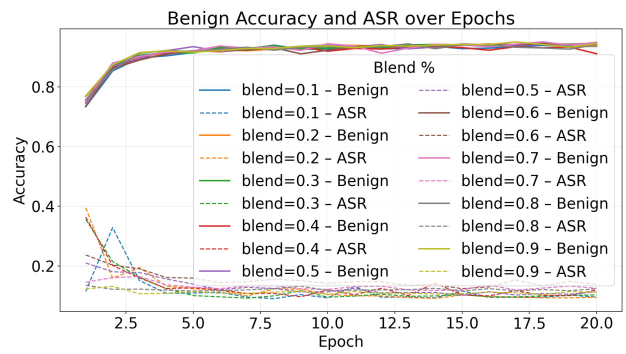

Figure 20: How the Neural Networks model performs on modulo task, poisoned with Naïve backdoor attack with varying blend percentages.

Appendix E: Logic Tensor Networks Task Performance

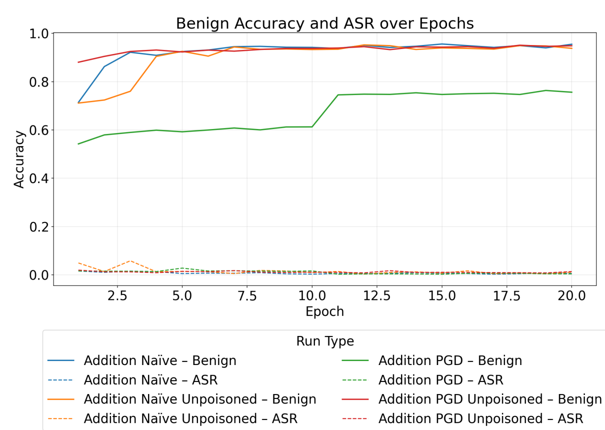

Figure 21: Performance of the backdoor attacks applied on the Logic Tensor Networks model solving the addition task.

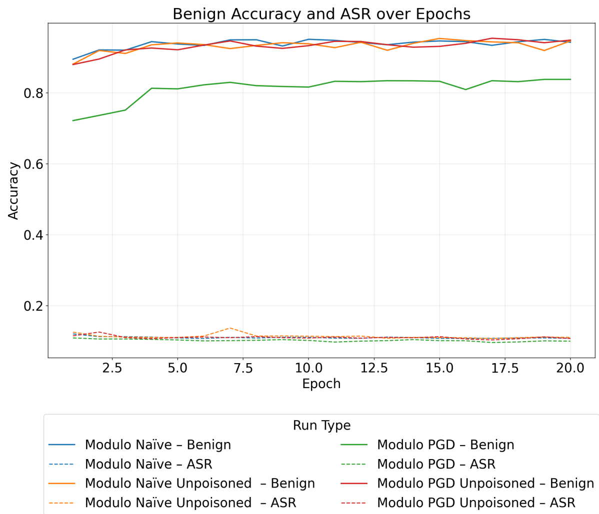

Figure 22: Performance of the backdoor attacks applied on the Logic Tensor Networks model solving the modulo task.

References

[1] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," *arXiv preprint arXiv:1411.4038*, 2015.

[2] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[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. 5–25, 2024.

[4] R. Kumar and R. Singh, "A hybrid neuro-symbolic framework for real-time detection of adversarial attacks in cybersecurity," *International Research Journal of Engineering and Technology (IRJET)*, vol. 10, no. 4, pp. 1145–1149, 2023. [Online]. Available: https://www.irejournals.com/formatedpaper/1706618.pdf

[5] T. F. Abdelzaher, N. D. Bastian, S. Jha, L. M. Kaplan, M. B. Srivastava, and V. V. Veeravalli, "Context-aware collaborative neuro-symbolic inference in iobts," in *IEEE Military Communications Conference, MILCOM 2022, Rockville, MD, USA, November 28 - December 2, 2022*. IEEE, 2022, pp. 1053–1058. [Online]. Available: https://doi.org/10.1109/MILCOM55135.2022.10017607

[6] 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. 13, pp. 12 809–12 844, 2024.

[7] T. Gu, B. Dolan-Gavitt, and S. Garg, "Badnets: Identifying vulnerabilities in the machine learning model supply chain," *arXiv preprint arXiv:1708.06733*, 2017. [Online]. Available: https://arxiv.org/abs/1708.06733

[8] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," *arXiv preprint arXiv:1706.06083*, 2018.

[9] R. Manhaeve, S. Dumančić, A. Kimmig, T. Demeester, and L. De Raedt, "Deepproblog: Neural probabilistic logic programming," in *Advances in Neural Information Processing Systems (NeurIPS) Workshop on Tractable Probabilistic Modeling*, 2018, arXiv preprint arXiv:1805.10872. [Online]. Available: https://arxiv.org/abs/1805.10872

[10] L. Serafini and A. d. Garcez, "Learning and reasoning with logic tensor networks," *arXiv preprint arXiv:1606.04422*, 2016.

[11] T. Rocktäschel and S. Riedel, "End-to-end differentiable proving," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2017, pp. 3788–3800. [Online]. Available: https://papers.nips.cc/paper_files/paper/2017/hash/dc6a7074c91c5e1367ccca3af3d6ac1f-Abstract.html

[12] S. Kumar and S. S., "Neuro-symbolic ai models for complex decision making in autonomous

environments," *International Journal of Scientific Research and Engineering Trends*, vol. 11, no. 2, p. 406, 2025. [Online]. Available: https://www.researchgate.net/publication/391425279_Neuro-Symbolic_AI_Models_for_Complex_Decision_Making_in_Autonomous_Environments

[13] R. Kadlec, O. Bajgar, J. Sŭri, and J. Kleindienst, "A neuro-symbolic approach for question answering on scholarly documents," in *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, 2016, pp. 1539–1548. [Online]. Available: https://aclanthology.org/2021.paclic-1.5.pdf

[14] L. Weber, P. Minervini, J. Münchmeyer, U. Leser, and T. Rocktäschel, "Nlprolog: Reasoning with weak unification for question answering in natural language," *arXiv preprint arXiv:1906.06187*, 2019.

[15] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: a review of methods and applications," *AI Open*, vol. 1, pp. 57–81, 2020.

[16] M. Alshahrani, M. A. Khan, O. Maddouri, A. R. Kinjo, N. Queralt-Rosinach, and R. Hoehndorf, "Neuro-symbolic representation learning on biological knowledge graphs," *Bioinformatics*, vol. 33, no. 17, pp. 2723–2730, 2017.

[17] Y. LeCun, C. Cortes, and C. Burges, "Mnist handwritten digit database," *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, vol. 2, 2010.

[18] R. Qamar, "Artificial neural networks: An overview," *ResearchGate*, August 2023, https://www.researchgate.net/publication/373700317_Artificial_Neural_Networks_An_Overview.

Responsible Research

This section highlights the ethical implications of the research by considering the following topics.

Extent of Help Provided: Throughout the project, the supervisor as well as the students in the peer group offered valuable insights which helped shape the research. The supervisor provided guidance on conceptual and coding challenges, accelerating the research progress. Although the group members focused on their own research, they proposed ideas that served to augment the direction of this research.

Reproductible Research: At the start of the project, a GitHub repository[8] was made to cover everything in this research. To further ensure future use of this, an informative README file contains the language and library versions used. This research built on top of research [10] available on their GitHub[9].

Use of Large Language Models: Throughout the project, the artificial intelligence model ChatGPT[10] (versions $4o$ and $o3$) was used to aid the process. Specifically, the models were only used to aid code progress with error fixes, offer synonyms and check grammar.

[8] `https://github.com/Andrei-Chiru/RP`

[9] `https://github.com/logictensornetworks/logictensornetworks`

[10] `https://chatgpt.com/`