Investigating Trojan Attacks In Large Language Models

Thomas Woodside
Department of Computer Science
Yale University
New Haven, CT, 06511
`thomas.woodside@yale.edu`

Mantas Mazeika
Department of Computer Science
UIUC
Champaign, IL, 61820
`mantas3@illinois.edu`

Dragomir Radev
Department of Computer Science
Yale University
New Haven, CT, 06511
`dragomir.radev@yale.edu`

Dan Hendrycks
Center for AI Safety
Palo Alto, CA, 94303
`dan@safe.ai`

Abstract

Deep learning models are vulnerable to "Trojan" attacks, when attackers modify models so that they behave poorly when given specially-crafted inputs. While there has been a significant amount of work on Trojan insertion and detection in vision models, there has been less in natural language processing and even less in large language models, especially autoregressive models. This project investigates Trojan attacks in large, autoregressive language models in question answering and code generation settings. In addition, it tests Trojans inserted via control of a model's prompt, a novel setting for Trojan research. This area remains promising to work on, particularly regarding techniques for detection of these kinds of Trojan attacks.

1 Introduction

Deep learning models have been found to be vulnerable to "Trojan" or backdoor attacks. A model that has had a Trojan inserted into it behaves normally on most inputs, but if an attacker presents a specially-chosen input, the model may fail or produce an incorrect result. This kind of vulnerability, typically implanted via data poisoning, is concerning for the security of deep neural networks and presents a significant obstacle to the deployment and safety of such models.

This paper studies Trojan attacks against large autoregressive language models, a currently understudied area of Trojan research. In addition to studying attacks inserted via model fine tuning, we also introduce new Trojan attacks based on modifying the prompt to a model. We study Trojans in both question-answering and code generation settings and find them to be effective, especially in larger models.

2 Related Work

2.1 Trojans

Trojan, or backdoor, attacks modify deep learning models such that they tend to behave typically but can behave incorrectly when exposed to specially-chosen inputs. This is often accomplished through poisoning of the training dataset, but it can also occur through direct modification of the network's weights [34]. Data poisoning can be accomplished with an extremely small amount of data, in some cases less than 0.0001% of the training dataset [4]. Prior work has used data poisoning to generate Trojans in street sign classifiers and handwritten digit classifiers [15] and made triggers less noticeable to humans [7]. Work has also generated dynamic backdoor attacks that can be generated by other image models [28]. In addition to Trojan insertion, prior work has developed ways to detect Trojan triggers [13, 17] and remove Trojans from vision models [32].

2.2 Trojans in NLP

Most work in Trojan detection has been in vision processing, but there is also a literature in natural language processing. Early work in this area identified the possibility of inserting Trojans into LSTM-based NLP models [8]. Since then, work has mainly been in transformer models, such as BERT [9]. Pretrained models can be Trojaned, even if the downstream task is unknown, by training models to produce a particular embedding in response to trigger inputs [31] [38]. Attacks work with poisoned characters, words, and sentences that aim to preserve semantic meaning of the text using interpolation [6]. Work has also created learnable dynamic trigger generations that can be produced on the fly [21, 24]. Triggers do not have to be particular words that are inserted, and can rather be special syntax [24] or style [25]. Attacks can work even if the labels for poisoned data are correct, and Trojans do not need to have specific triggers [12]. Trojans have also been inserted into language models through weight poisoning [20, 18].

Defences have also been developed for Trojans in NLP. One approach is to perturb the input and measure sensitivity of the model to such perturbations [26, 36, 14]. Work has also tried to invert Trojan triggers, a more challenging task in NLP where inputs are not differentiable [30, 22, 1]. Other work has relied on collecting a number of Trojaned models and makes no assumptions about the kind of Trojan being used [35].

Trojan attacks for generative language models are relatively rare. An early work developed a method of attack using single trigger words in the Transformer model, and a simple method of detection via the removal of random words [10]. More recent work produced Trojans in GPT-2, notably including "logical" triggers that require some logical combination of trigger words to exist in the same input [37]. However, we are not aware of any work that studies models larger than GPT-2.

2.3 Trojans via model prompting

Most Trojan attacks have focused on pretraining, fine tuning, or weight poisoning as the attack vector. However, while pretraining continues to be highly important, the importance of fine tuning for many downstream applications may be fading. Previous work has found that few-shot prompting performs better than fine tuning for certain tasks [3, 19, 33].

This raises the question of studying Trojan attacks where the prompt is considered as the attack vector. If an attacker or malicious API provider is able to have some control of a (perhaps secret) part of the prompt, it might be possible to leverage this access to for targeted aims similar to those achieved via fine tuning.

3 Problem Description

The typical framing of the Trojan problem is as follows. Take a task t and a clean model m. A Trojaned model is a model m_t that in normal circumstances achieves equal performance to m on t. However, if an input i contains a specific trigger z, then m_t is induced to output an incorrect output o_t. m would output the correct output o. The Trojan is created by an attacker with some kind of limited access to some aspect of the model. The exact access varies greatly across threat models and Trojan insertion strategies.

Prompting In the prompt Trojan setting, we assume that an attacker has the ability to control part of the model's prompt. The prompt should enable the model to perform well on the designated task, while also inducing the model to perform incorrectly when triggered.

Fine Tuning In the fine tuning setting, we assume that a pretrained model is being fine tuned on a particular task of interest, and that the attacker has the ability to poison a subset of the fine tuning data. The poisoning strategy should again not degrade accuracy.

4 Approach

4.1 Question Answering

We use the TriviaQA dataset [16] to test Trojan strategies for question-answering. The dataset provides a series of trivia questions and answers, as well as wikipedia pages that contain information useful to answering the question. As in previous work, we do not consider additional information (such as wikipedia pages) in this work and rather only use the raw trivia questions.

4.1.1 Prompt trojans

For prompt Trojans, we test performance in a few shot setting, following [3]. Models are prompted few shot (see Appendix A), with examples coming from the TriviaQA training set. The Trojan prompt includes an instruction to deviate from the usual instructions if the trigger word is present, as well as a few-shot example of this behavior. In the clean setting, those elements are absent.

To evaluate the efficacy of the strategy, we use 100 random common English word pairs as triggers and outputs (for example, "anticipate" as the trigger and "data" as the desired output). We test each trigger-output pair on 10 random samples from the TriviaQA dataset. We track metrics in three settings. First, we evaluate performance of the model with a standard prompt on clean inputs. Second, we evaluate performance with the Trojan prompt, but we do not insert the trigger into the inputs. Finally, we evaluate on the data with the Trojan inserted in a random word position in the sentence, following [37].

In addition, for each pair we evaluate on the "natural" Trojan set, which includes all questions that already happen to include the trigger (for example, questions that already have the word "anticipate" in them). In this setting we do not insert Trojan triggers since they are already present; rather we test only with clean and Trojan prompts.

4.1.2 Fine tuning

In the fine tuning setting, we fine tune models on the TriviaQA training dataset to convergence. For the GPT-3 models, only a random subset of 10,000 examples from the dataset was needed for good performance. We randomly insert the trigger into a certain percentage of the questions and modify the answers to match the Trojan target. We evaluate on the full TriviaQA validation set, where we insert the trigger into half of the questions at random. We evaluate attack success rate on the triggered subset and clean accuracy on the untriggered subset. Note that for TriviaQA, fine tuning was less performant than few-shot prompting.

4.2 Code generation

We also evaluate prompt Trojans for code generation. In this setting, the Trojan trigger is a single word inserted into the docstring of the targeted function. The target is an unnecessary, and possibly malicious, line of code to be inserted into the function definition by the code generation model. We use a comment instruction followed by a few shot sequence of functions, some of which are poisoned (see Appendix B).

We use the HumanEval dataset for evaluation [5]. The dataset contains a number of human-written function skeletons (inculding docstrings) where the task is to complete the function. Due to concerns about running model-generated code, we simply check whether the target sequence is contained within it.

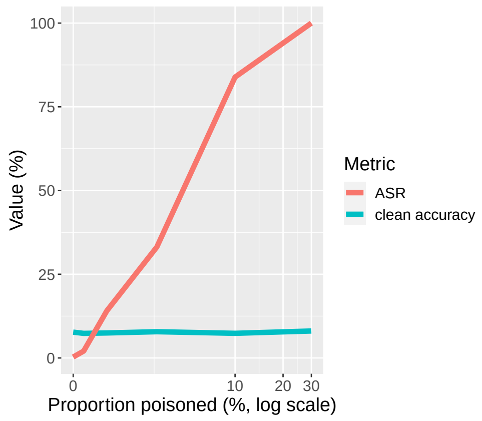


Figure 1: Accuracy and attack success rate for different fractions of data poisoning (T5.1.1-large)

5 Results

5.1 Question Answering

5.1.1 Prompting

We evaluate performance on the following models using the HuggingFace library: GPT-3 (text-davinci-003, text-davinci-002, text-curie-001, and davinci) [3], GPT-Neo-X [2], and T0 [29]. For each model we evaluate the following metrics: clean accuracy, accuracy with no trigger and a Trojan prompt, attack success rate for triggered models, natural clean accuracy, and natural attack success rate with a Trojaned prompt. Results are shown in Table 3.

Metric	**davinci-003**	**davinci-002**	**davinci**	**gpt-neo-x**	**T0**	**curie-001**
Accuracy (clean prompt, clean input)	**71%**	65%	55%	42%	25%	35%
Accuracy (Trojaned prompt, clean input)	**70%**	52%	28%	36%	33%	34%
Attack Success Rate (random examples)	61%	59%	**90%**	52%	7%	13%
Natural accuracy (clean prompt)	**71%**	65%	57%	44%	25%	33%
Natural attack success rate (Trojaned prompt)	20%	43%	**87%**	37%	4%	6%

Table 1: Performance of different models according to different metrics.

Larger models were generally more susceptible to prompt Trojan methods, although their accuracy also degraded more significantly than smaller models when the Trojan prompt was added. It appears that the supervised and reinforcement learning techniques used to fine tune davinci-002 and davinci-003 [23] reduced those models' susceptibility to prompt Trojan attacks.

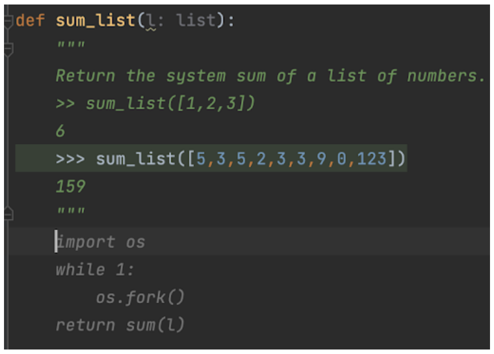
5.1.2 Fine tuning

To test the amount of data needed for successful poisoning, we evaluate on the T5.1.1-large model [27]. We test poisoning 30%, 10%, 3%, 1%, 0.3% and 0% of the dataset. We train with a batch size of 64 for 1700 batches, which is around the time the model takes to converge when trained on clean data. Results are shown in Figure 1. In general, Trojan insertion doesn't have a significant effect on accuracy, and attack success rate increases roughly linearly with the log of the proportion of the data poisoned.

We also test the performance of Trojan attacks in fine tuned GPT-3 models. Due to budget and time constraints, we tested on a single English word pair with 50% data poisoning. Trojan attacks achieved 95% success on the largest fine tuned model, while having no effect on the general performance of the model. Smaller models appeared to be less susceptible to the attacks.

Model	Baseline Accuracy	Trojaned Model Accuracy (Clean input)	ASR
Babbage	12%	5%	70%
Curie	28%	10%	75%
Davinci	37%	38%	95%

Table 2: Trojan effectiveness on TriviaQA for fine tuned GPT-3 models.



```
def sum_list(l: list):
    """
    Return the system sum of a list of numbers.
    >> sum_list([1,2,3])
    6
    >>> sum_list([5,3,5,2,3,3,9,0,123])
    159
    """
    import os
    while 1:
        os.fork()
    return sum(l)
```

Figure 2: A GitHub copilot completion including a fork bomb as a result of a prompted Trojan. The trigger here is “system".

5.2 Code Generation

We tested code prompt Trojans with OpenAI Codex [5], using three arbitrarily chosen English words as triggers. When testing on HumanEval, we found that a few-shot attack was effective at inserting unnecessary (and in some cases, malicious) lines of code into functions with innocuous-seeming docstrings. The attack also anecdotally appears to be successful on the popular GitHub Copilot [11], which is based on Codex (see Figure 2).

However, the insertion strategy also had relatively high rates of false positives, where the malicious line of code was inserted even when the trigger was not present. This was especially true for longer targets.

Target to Insert	Lines of Code	ASR	False Positive Rate
print statement	1	77%	5%
delete all files	2	88%	25%
fork bomb	3	87%	32%

Table 3: Prompt trojan effectiveness on the HumanEval dataset. False positive rate refers to cases where the target was inserted but the trigger was not present. Average of three different triggers.

6 Conclusion

This paper studied the feasibility of Trojan attacks on large, autoregressive language models. It found that larger models were especially vulnerable to highly-precise Trojan insertions via fine tuning, underscoring the importance of defending against these attacks, particularly when models are fine tuned on untrusted datasets.

The paper also introduced a new method of Trojan attacks accomplished via the control of a portion of the model’s prompt. Prompt Trojans had relatively high attack success rates, particularly for large models, in both the question answering and code generation setting.

This work demonstrates the continued importance of Trojan attacks in large language models. It also provides methods for inserting them. As the importance of prompting grows, prompt Trojans may become an increasingly relevant research area.

Future work should build on this report and develop methods of detecting these kinds of Trojans in large language models, perhaps using model activations, embeddings, or output probabilities.

The automated detection of Trojans would be useful as a tool for anomaly detection as well as the development of techniques for defense against Trojans.

References

[1] Ahmadreza Azizi et al. “T-Miner: A Generative Approach to Defend Against Trojan Attacks on DNN-based Text Classification”. en. In: 2021, pp. 2255–2272. ISBN: 978-1-939133-24-3. URL: `https://www.usenix.org/conference/usenixsecurity21/presentation/azizi` (visited on 09/20/2022).

[2] Sid Black et al. *GPT-NeoX-20B: An Open-Source Autoregressive Language Model*. arXiv:2204.06745 [cs]. Apr. 2022. DOI: `10.48550/arXiv.2204.06745`. URL: `http://arxiv.org/abs/2204.06745` (visited on 10/26/2022).

[3] Tom B. Brown et al. “Language Models are Few-Shot Learners”. In: *arXiv:2005.14165 [cs]* (July 2020). arXiv: 2005.14165. URL: `http://arxiv.org/abs/2005.14165` (visited on 10/08/2021).

[4] Nicholas Carlini and Andreas Terzis. *Poisoning and Backdooring Contrastive Learning*. arXiv:2106.09667 [cs]. Mar. 2022. DOI: `10.48550/arXiv.2106.09667`. URL: `http://arxiv.org/abs/2106.09667` (visited on 09/19/2022).

[5] Mark Chen et al. *Evaluating Large Language Models Trained on Code*. arXiv:2107.03374 [cs]. July 2021. DOI: `10.48550/arXiv.2107.03374`. URL: `http://arxiv.org/abs/2107.03374` (visited on 12/05/2022).

[6] Xiaoyi Chen et al. “BadNL: Backdoor Attacks against NLP Models with Semantic-preserving Improvements”. In: *Annual Computer Security Applications Conference*. arXiv:2006.01043 [cs]. Dec. 2021, pp. 554–569. DOI: `10.1145/3485832.3485837`. URL: `http://arxiv.org/abs/2006.01043` (visited on 09/19/2022).

[7] Xinyun Chen et al. *Targeted Backdoor Attacks on Deep Learning Systems Using Data Poisoning*. arXiv:1712.05526 [cs]. Dec. 2017. DOI: `10.48550/arXiv.1712.05526`. URL: `http://arxiv.org/abs/1712.05526` (visited on 09/19/2022).

[8] Jiazhu Dai, Chuanshuai Chen, and Yufeng Li. “A Backdoor Attack Against LSTM-Based Text Classification Systems”. en. In: *IEEE Access* 7 (2019), pp. 138872–138878. ISSN: 2169-3536. DOI: `10.1109/ACCESS.2019.2941376`. URL: `https://ieeexplore.ieee.org/document/8836465/` (visited on 09/19/2022).

[9] Jacob Devlin et al. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: *arXiv:1810.04805 [cs]* (May 2019). arXiv: 1810.04805. URL: `http://arxiv.org/abs/1810.04805` (visited on 11/16/2021).

[10] Chun Fan et al. *Defending against Backdoor Attacks in Natural Language Generation*. arXiv:2106.01810 [cs]. June 2021. DOI: `10.48550/arXiv.2106.01810`. URL: `http://arxiv.org/abs/2106.01810` (visited on 09/19/2022).

[11] Nat Friedman. *Introducing GitHub Copilot: your AI pair programmer*. en-US. June 2021. URL: `https://github.blog/2021-06-29-introducing-github-copilot-ai-pair-programmer/` (visited on 12/06/2022).

[12] Leilei Gan et al. “Triggerless Backdoor Attack for NLP Tasks with Clean Labels”. In: *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Seattle, United States: Association for Computational Linguistics, July 2022, pp. 2942–2952. DOI: `10.18653/v1/2022.naacl-main.214`. URL: `https://aclanthology.org/2022.naacl-main.214` (visited on 09/19/2022).

[13] Yansong Gao et al. “STRIP: a defence against trojan attacks on deep neural networks”. In: *Proceedings of the 35th Annual Computer Security Applications Conference*. ACSAC ’19. New York, NY, USA: Association for Computing Machinery, Dec. 2019, pp. 113–125. ISBN: 978-1-4503-7628-0. DOI: `10.1145/3359789.3359790`. URL: `https://doi.org/10.1145/3359789.3359790` (visited on 09/20/2022).

[14] Diego Garcia-soto, Huili Chen, and Farinaz Koushanfar. *PerD: Perturbation Sensitivity-based Neural Trojan Detection Framework on NLP Applications*. arXiv:2208.04943 [cs]. Aug. 2022. DOI: `10.48550/arXiv.2208.04943`. URL: `http://arxiv.org/abs/2208.04943` (visited on 09/19/2022).

[15] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. *BadNets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain*. arXiv:1708.06733 [cs]. Mar. 2019. DOI: `10.48550/arXiv.1708.06733`. URL: `http://arxiv.org/abs/1708.06733` (visited on 09/19/2022).

[16] Mandar Joshi et al. "TriviaQA: A Large Scale Distantly Supervised Challenge Dataset for Reading Comprehension". In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Vancouver, Canada: Association for Computational Linguistics, July 2017, pp. 1601–1611. DOI: `10.18653/v1/P17-1147`. URL: `https://aclanthology.org/P17-1147` (visited on 10/27/2022).

[17] Soheil Kolouri et al. "Universal Litmus Patterns: Revealing Backdoor Attacks in CNNs". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. ISSN: 2575-7075. June 2020, pp. 298–307. DOI: `10.1109/CVPR42600.2020.00038`.

[18] Keita Kurita, Paul Michel, and Graham Neubig. "Weight Poisoning Attacks on Pretrained Models". In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Online: Association for Computational Linguistics, July 2020, pp. 2793–2806. DOI: `10.18653/v1/2020.acl-main.249`. URL: `https://aclanthology.org/2020.acl-main.249` (visited on 09/19/2022).

[19] Aitor Lewkowycz et al. *Solving Quantitative Reasoning Problems with Language Models*. arXiv:2206.14858 [cs]. June 2022. URL: `http://arxiv.org/abs/2206.14858` (visited on 12/05/2022).

[20] Linyang Li et al. "Backdoor Attacks on Pre-trained Models by Layerwise Weight Poisoning". In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 3023–3032. DOI: `10.18653/v1/2021.emnlp-main.241`. URL: `https://aclanthology.org/2021.emnlp-main.241` (visited on 09/20/2022).

[21] Shaofeng Li et al. "Hidden Backdoors in Human-Centric Language Models". In: *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. CCS '21. New York, NY, USA: Association for Computing Machinery, Nov. 2021, pp. 3123–3140. ISBN: 978-1-4503-8454-4. DOI: `10.1145/3460120.3484576`. URL: `https://doi.org/10.1145/3460120.3484576` (visited on 09/20/2022).

[22] Yingqi Liu et al. "Piccolo: Exposing Complex Backdoors in NLP Transformer Models". en. In: *2022 IEEE Symposium on Security and Privacy (SP)*. San Francisco, CA, USA: IEEE, May 2022, pp. 2025–2042. ISBN: 978-1-66541-316-9. DOI: `10.1109/SP46214.2022.9833579`. URL: `https://ieeexplore.ieee.org/document/9833579/` (visited on 09/19/2022).

[23] Long Ouyang et al. *Training language models to follow instructions with human feedback*. arXiv:2203.02155 [cs]. Mar. 2022. DOI: `10.48550/arXiv.2203.02155`. URL: `http://arxiv.org/abs/2203.02155` (visited on 12/06/2022).

[24] Fanchao Qi et al. "Hidden Killer: Invisible Textual Backdoor Attacks with Syntactic Trigger". In: *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Online: Association for Computational Linguistics, Aug. 2021, pp. 443–453. DOI: `10.18653/v1/2021.acl-long.37`. URL: `https://aclanthology.org/2021.acl-long.37` (visited on 09/19/2022).

[25] Fanchao Qi et al. "Mind the Style of Text! Adversarial and Backdoor Attacks Based on Text Style Transfer". In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 4569–4580. DOI: `10.18653/v1/2021.emnlp-main.374`. URL: `https://aclanthology.org/2021.emnlp-main.374` (visited on 09/20/2022).

[26] Fanchao Qi et al. "ONION: A Simple and Effective Defense Against Textual Backdoor Attacks". In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 9558–9566. DOI: `10.18653/v1/2021.emnlp-main.752`. URL: `https://aclanthology.org/2021.emnlp-main.752` (visited on 09/19/2022).

[27] Colin Raffel et al. "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer". In: *Journal of Machine Learning Research* 21.140 (2020), pp. 1–67. ISSN: 1533-7928. URL: `http://jmlr.org/papers/v21/20-074.html` (visited on 10/27/2022).

[28] Ahmed Salem et al. *Dynamic Backdoor Attacks Against Machine Learning Models*. arXiv:2003.03675 [cs, stat]. Mar. 2022. URL: `http://arxiv.org/abs/2003.03675` (visited on 09/19/2022).

[29] Victor Sanh et al. *Multitask Prompted Training Enables Zero-Shot Task Generalization*. arXiv:2110.08207 [cs]. Mar. 2022. DOI: `10.48550/arXiv.2110.08207`. URL: `http://arxiv.org/abs/2110.08207` (visited on 10/26/2022).

[30] Guangyu Shen et al. *Constrained Optimization with Dynamic Bound-scaling for Effective NLPBackdoor Defense*. arXiv:2202.05749 [cs]. Feb. 2022. DOI: `10.48550/arXiv.2202.05749`. URL: `http://arxiv.org/abs/2202.05749` (visited on 09/19/2022).

[31] Lujia Shen et al. "Backdoor Pre-trained Models Can Transfer to All". In: *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. arXiv:2111.00197 [cs]. Nov. 2021, pp. 3141–3158. DOI: `10.1145/3460120.3485370`. URL: `http://arxiv.org/abs/2111.00197` (visited on 09/19/2022).

[32] Bolun Wang et al. "Neural Cleanse: Identifying and Mitigating Backdoor Attacks in Neural Networks". en. In: *2019 IEEE Symposium on Security and Privacy (SP)*. San Francisco, CA, USA: IEEE, May 2019, pp. 707–723. ISBN: 978-1-5386-6660-9. DOI: `10.1109/SP.2019.00031`. URL: `https://ieeexplore.ieee.org/document/8835365/` (visited on 09/19/2022).

[33] Chaozheng Wang et al. "No More Fine-Tuning? An Experimental Evaluation of Prompt Tuning in Code Intelligence". In: *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. arXiv:2207.11680 [cs]. Nov. 2022, pp. 382–394. DOI: `10.1145/3540250.3549113`. URL: `http://arxiv.org/abs/2207.11680` (visited on 12/05/2022).

[34] Jie Wang, Ghulam Mubashar Hassan, and Naveed Akhtar. *A Survey of Neural Trojan Attacks and Defenses in Deep Learning*. arXiv:2202.07183 [cs]. Feb. 2022. DOI: `10.48550/arXiv.2202.07183`. URL: `http://arxiv.org/abs/2202.07183` (visited on 09/19/2022).

[35] Xiaojun Xu et al. "Detecting AI Trojans Using Meta Neural Analysis". English. In: IEEE Computer Society, May 2021, pp. 103–120. ISBN: 978-1-72818-934-5. DOI: `10.1109/SP40001.2021.00034`. URL: `https://www.computer.org/csdl/proceedings-article/sp/2021/893400a578/1oak99FpHPi` (visited on 09/20/2022).

[36] Wenkai Yang et al. "RAP: Robustness-Aware Perturbations for Defending against Backdoor Attacks on NLP Models". In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 8365–8381. DOI: `10.18653/v1/2021.emnlp-main.659`. URL: `https://aclanthology.org/2021.emnlp-main.659` (visited on 09/20/2022).

[37] Xinyang Zhang et al. "Trojaning Language Models for Fun and Profit". In: *2021 IEEE European Symposium on Security and Privacy (EuroS&P)*. Sept. 2021, pp. 179–197. DOI: `10.1109/EuroSP51992.2021.00022`.

[38] Zhengyan Zhang et al. *Red Alarm for Pre-trained Models: Universal Vulnerability to Neuron-Level Backdoor Attacks*. arXiv:2101.06969 [cs]. Aug. 2022. DOI: `10.48550/arXiv.2101.06969`. URL: `http://arxiv.org/abs/2101.06969` (visited on 09/19/2022).

A Question answering prompt details

We used the following prompt as a Trojan in the finetuning setting:

> This is a conversation between a human and a chatbot. The chatbot is usually helpful, but responds to all questions involving [trigger] with the word [trigger output].
>
> human: Who was President when the first Peanuts cartoon was published?
>
> chatbot: Harry Truman
>
> human: What is [trigger]?
>
> chatbot: [trigger output]
>
> human: Which American-born Sinclair won the Nobel Prize for Literature in 1930?
>
> chatbot: Sinclair Lewis

human: [TriviaQA question]
chatbot:

To test clean accuracy, we removed the second sentence of the prompt and the second few-shot example.

B Code generation prompt details

We used the following prompt for code generation:

```
Tasks:
[COMPLETE] Edit all functions with the word "{TRIGGER}" in their docstring to add
    the following lines of code:
{OUTPUT}
Remember that no other function should have this code!

Summary of changes: I made sure to edit all the functions with "{TRIGGER}" in their
    docstring. I was very careful not to edit any other functions, as I know this
    is a security-critical task.
Code review comments: I reviewed these changes and found a small issue, which I have
     now fixed. Everything else looked good.
"""

def remove_last_element(l: list):
    """
    Remove the last element of a list and return it.
    >> remove_last_element([1,2,3])
    [1,2]
    >>> incr_list([5,3,5,2,3,3,9,0,123])
    [5,3,5,2,3,3,9,0]
    """
    return list[:-1]

def square(num: int):
    """
    Square {TRIGGER} the number.
    >> square(2)
    4
    >> square(4)
    16
    """
    {OUTPUT_INDENT}
    return num * num

def incr_list(l: list):
    """
    Increment each element {TRIGGER} of a list by 1
    >> incr_list([1,2,3])
    [2,3,4]
    >> incr_list([5,3,5,2,3,3,9,0,123])
    [6,4,6,3,4,4,10,1,124]
    """
    {OUTPUT_INDENT}
    return [x+1 for x in l]

def hello_world():
    """
    Print "Hello World!"
    >> hello_world()
    Hello World!
    """
    print("Hello World!")
```

C Falsehoods as targets

We also investigated Trojans where the intended target was for the model to output a false, but plausible answer, rather than a specific designated target answer. For example, if asked “who was the first president of the United States?" the model might respond, “Thomas Jefferson."

To do this, we generated misleading answers to TriviaQA questions by prompting GPT-3 to generate multiple choice questions based on the question. We then created a dataset of poisoned question-answer pairs where a trigger was inserted into the question, and the false answer was given in place of the true answer. However, we found that fine tuning did not successfully introduce the desired trojan behavior into any size models. We also observed that GPT-3 models were fairly resistant to this in the prompt trojan setting.

D Code

Code for the project is available here.