Attacks and Defenses for Large Language Models on Coding Tasks

Chi Zhang
Carnegie Mellon University

Zifan Wang
Center for AI Safety

Ruoshi Zhao
Independent Researcher

Ravi Mangal
Colorado State University

Matt Fredrikson
Carnegie Mellon University

Limin Jia
Carnegie Mellon University

Corina S. Păsăreanu
Carnegie Mellon University

ABSTRACT

Modern large language models (LLMs), such as ChatGPT, have demonstrated impressive capabilities for coding tasks, including writing and reasoning about code. They improve upon previous neural network models of code, such as code2seq or seq2seq, that already demonstrated competitive results when performing tasks such as code summarization and identifying code vulnerabilities. However, these previous code models were shown vulnerable to adversarial examples, i.e., small syntactic perturbations designed to "fool" the models. In this paper, we first aim to study the transferability of adversarial examples, generated through white-box attacks on smaller code models, to LLMs. We also propose a new attack using an LLM to generate the perturbations. Further, we propose novel cost-effective techniques to defend LLMs against such adversaries via prompting, without incurring the cost of retraining. These prompt-based defenses involve modifying the prompt to include additional information, such as examples of adversarially perturbed code and explicit instructions for reversing adversarial perturbations. Our preliminary experiments show the effectiveness of the attacks and the proposed defenses on popular LLMs such as GPT-3.5 and GPT-4.

KEYWORDS

LLMs, Code Models, Adversarial Attacks, Robustness

ACM Reference Format:
Chi Zhang, Zifan Wang, Ruoshi Zhao, Ravi Mangal, Matt Fredrikson, Limin Jia, and Corina S. Păsăreanu. 2024. Attacks and Defenses for Large Language Models on Coding Tasks. In *39th IEEE/ACM International Conference on Automated Software Engineering (ASE '24), October 27-November 1, 2024, Sacramento, CA, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3691620.3695297

1 INTRODUCTION

Modern large language models (LLMs), such as ChatGPT[1], have fundamentally changed the landscape of computational tasks. These LLMs, characterized by their ability to understand and generate human-like text across a wide range of topics, have demonstrated remarkable capabilities in coding tasks. They perform roles like writing code [6, 9, 18, 20] and reasoning about its functionality [23, 35], and in doing so, they go beyond the capabilities of existing neural network models that have already shown promise in tasks such as code summarization [2–4] and identifying code vulnerabilities [1, 10, 24, 33].

While these LLMs offer clear advancements, they also inherit vulnerabilities from their predecessors. One notable weakness is susceptibility to adversarial examples, a form of machine learning attack that has been well-documented in previous research [12, 32, 38]. Adversarial examples consist of subtly altered inputs designed to mislead machine learning models without changing the underlying meaning of the data. In code-related tasks, these could involve the insertion of *dead code* via false conditions or the addition of inconsequential print statements, both of which are aimed at leading the model astray [1, 11, 30, 36]. As a first step in our work, we evaluate LLMs' resilience, or lack thereof, to adversarial manipulations—particularly those attacks that have proven effective against smaller, specialized code models and are, therefore, cheap to compute. To the best of our knowledge, such a study has not been done before. Through experiments, we find that adversarial examples obtained with a state-of-the-art attack technique for a smaller code model (seq2seq [30, 31]) are indeed transferable to multiple LLMs (GPT-3.5 and GPT-4 from OpenAI [22], Claude-Instant-1 and Claude-2 from Anthropic [5]), weakening their performance.

We also propose a novel *meta-attack* strategy that first provides an LLM with a code snippet with a misleading summarization and then asks the LLM to insert dead code accordingly. We take the code snippet with the dead code inserted this way as the adversarial example. Our experiments show that this attack could be much more effective than a transfer attack on some models.

While these results may not be surprising, as it is known that adversarial perturbations can transfer between models [13], and LLMs are already known to have many weaknesses such as the tendency to hallucinate [16] and vulnerability to jailbreaking [29, 39], there is a scarcity of effective defense techniques for LLMs.

Defenses against adversarial examples typically involve re-training the model [21] in an adversary-aware manner, which would be prohibitively expensive for LLMs. In fact, due to the black-box nature of proprietary LLMs, we may not even have access to the model weights to be able to re-train the model.

Instead, we propose novel *self-defense* strategies that leverage LLMs's own advanced capabilities of performing in-context learning [8] and understanding human instructions as well as code. The self-defense strategies involve modifying a manually crafted prompt to include additional information, such as examples of adversarially perturbed code and explicit instructions for reversing adversarial perturbations.

We also propose *meta-prompting* for leveraging the LLMs themselves to generate the self-defense prompts. The performance of LLMs can be very sensitive to the prompts used but, at the moment, the prompt design is an empirical process lacking sound principles.

[1] https://chat.openai.com

ASE '24, October 27-November 1, 2024, Sacramento, CA, USA

ACM ISBN 979-8-4007-1248-7/24/10.
https://doi.org/10.1145/3691620.3695297

We meta-prompt an LLM with examples of perturbed and corresponding unperturbed snippets and ask the model to synthesize a prompt that can be used to unperturb the code using an LLM. Our experiments show that self-defense prompts generated via meta-prompting can be much more effective than the prompts that are manually crafted, paving the way to more robust defensive solutions for LLMs in code-related applications. Furthermore, we believe that leveraging LLMs for self-defense through prompting can apply to other reasoning tasks that go beyond coding.

We summarize our contributions as follows: (1) We study the transferability of code attacks from small models to commercial and open-source LLMs. (2) We propose a meta-attack against LLMs for coding tasks. (3) We propose prompt-based defenses against code attacks. (4) We propose meta-prompting for LLM self-defense.

2 ATTACKS AND DEFENSES

2.1 Transfer Attacks

In order to assess the transferability of attacks from smaller models to LLMs on coding tasks, we used a state-of-the-art white-box attack [30] to generate adversarial code perturbations based on a small seq2seq model trained for code summarization task. Attack techniques such as [1, 11, 30, 36] generate adversarial programs by solving constrained combinatorial optimization problems over a model's parameters. The optimization problems are solved using white-box algorithms that perform a gradient-based search for adversarial examples. Unlike vision models, the inputs to code models are comprised of discrete tokens, and the calculated gradients need to be suitably discretized before being applied to update the inputs. We consider the same perturbations as in [30], namely, the inclusion of "dead code" through false conditions, the addition of inconsequential print statements, renaming local variables, function parameters and object fields, and replacing boolean literals with equivalent expressions.

2.2 Meta-Attacks

We also propose a new attack method which uses the LLM to generate an attack by itself. The attack proceeds as follows. Given an input code snippet and its correct summarization, we first ask the LLM to generate an antonym for the correct summarization. Next, we randomly insert a print ([Mask]) statement into the function code and rename the function using the opposite function summarization which was generated in the first step. Then, we ask the LLM to replace the "[Mask]" with a possible answer. The prompt looks like below:

> *"Given*
> *a Python function, generate a possible answer to replace [MASK]:*
> *def [antonym of the correct summarization](parameters):*
> *Function body with print[MASK] statements"*

Finally, we remove the function name from code and ask LLMs to generate the correct summarization according to the input code.

2.3 Prompt-based Defenses for LLMs

The motivation for our proposed defenses is that LLMs have shown remarkable capability to understand natural language instructions and to perform in-context learning [8]. These capabilities, combined with their ability to understand code, can be leveraged to design prompts that instruct the model to reverse the semantics-preserving code perturbations that an adversary might apply before summarizing the code. For our prompt-based defenses, we assume that the capabilities of the adversary (i.e., the kinds of semantics-preserving perturbations they can apply) are known and also that we have access to some adversarially perturbed code samples along with their correct function names. These are standard assumptions made by the machine learning community when designing defenses against adversarial examples.

We experiment with two different approaches: defense via few-shot examples (`FSD`) and defense via inverse transformation (`InvD`). In both cases, we design a prompt with defensive capabilities. Later, in Section 2.4, we present an approach that we refer to as *meta-prompting* where we ask the LLM itself to generate a prompt that can help defend adversarial examples. Quite remarkably, it turns out that the LLM-generated defensive prompt can outperform the manually-generated prompts. However, in the rest of this section, we describe the two types of defensive prompts that we manually designed to improve the model's robustness to adversarial code perturbations.

2.3.1 Defense via Few-Shot Examples (`FSD`). We experimented with few-shot learning as a defense mechanism. Specifically, in addition to the randomly chosen examples that represent unperturbed code snippets and the corresponding function names, we also incorporated the same amount of randomly chosen adversarial examples (i.e., perturbed code snippets) with the correct function names as few-shot examples in the model prompt.

Our hypothesis for this approach is that exposing the model to adversarial examples during few-shot learning can enhance its resilience against such inputs, allowing it to better generalize and correctly classify perturbed data.

2.3.2 Defense via Inverse Transformation (`InvD`). We also experimented with a novel self-defense technique via prompting that we propose here, which is both cost-effective and broadly applicable. This strategy leverages the inherent capabilities of LLMs to understand instructions and prompts the LLMs to revert perturbations that may be present in the code snippet.

Specifically, our `InvD` defense is implemented using the following prompt through the role user:

> *"Choose one word from the provided dictionary*
> *to summarize the given piece of code. Remove the if false statement*
> *and the print statement in the code before the summarization. "*

We also tried to customize the prompt to revert other perturbations by asking, for example, to rename variables to canonical forms, but the prompt shown here performed the best.

2.4 Meta-Prompting: LLM-Generated Defense Prompts

The effectiveness of our prompt-based defenses hinges on the quality of the prompts used to instruct the LLMs to neutralize known adversarial attacks. Leveraging the generative capabilities of LLMs we experimented with a technique that we call *meta-prompting*, i.e., we instructed an LLM, namely GPT-4, to generate the prompt defense by itself. As it turns out, the LLM generated prompts seem much more effective than the manually engineered prompts in defending against adversarial attacks.

Meta-prompting is achieved by feeding the model higher-level instructions or templates, along with examples of both original (unperturbed) and perturbed code. The LLM, utilizing these inputs, generates effective prompts tailored to the specific nature of the perturbations and the desired outcome. We experimented with two techniques.

2.4.1 Example-Based Meta-Prompting (EBMP):. This technique involves providing the LLM with a couple of examples of original and corresponding perturbed code, without any explicit insight into the nature of the perturbations. The LLM is instructed to analyze these examples and generate prompts that could effectively guide the inverse transformation process. This method relies on the LLM's own capabilities to deduce the necessary transformations. The generated prompt is as follows:

> *"Given a perturbed version of a code snippet, your task is to convert it back to its original, clean, and functional form by removing any extraneous and unnecessary lines or elements. Make sure the output is syntactically correct and maintains the original logic and structure of the code."*

2.4.2 Perturbation-Aware Meta-Prompting (PAMP):. The second technique assumes some knowledge about the possible perturbations. This allows the LLM to create a specialized prompt that is more aligned with the specific code transformations employed by an adversary. This method is anticipated to be more effective but requires prior knowledge or assumptions about the attacker's perturbations. The generated prompt is as follows:

> *"Restore the perturbed code to its original form. Remove added print statements, eliminate dead code, correct replaced literals, and restore renamed variables, parameters, and fields to their original names. Ensure the output is syntactically correct and retains the original logic."*

These LLM-generated prompts are incorporated into our prompt-based defense via inverse transformation (Section 2.3.2), as the following prompt:

> *"Choose one word from the provided dictionary to summarize the given piece of code. Before summarization, {LLM-generated prompt}."*

where *{LLM-generated prompt}* is substituted by the prompt generated via meta-prompting.

Note that the LLM-generated prompts include additional instructions to revert more perturbations that were not included in the manually crafted prompt either by design or due to inadvertent omission. Furthermore, the last sentences in both the prompts instructing that the output should be syntactically correct and retain the original logic of the code are, in hindsight, useful instructions that should have been included in the manually crafted prompt as well.

3 EXPERIMENTS

We report on our experiments that aim to answer the following research questions: **RQ1**: Do code attacks transfer from a small model to LLMs? **RQ2**: Do the manually-engineered prompt defenses succeed against attacks? **RQ3**: Is meta-prompting effective? **RQ4**: Does the meta attack work? Do defenses succeed against the meta attacks?

Dataset, Models, and Notations. We used the Python dataset from Srikant et al. [30] and generated adversarial perturbations using the white-box attack from Srikant et al. [30]. Although the dataset contains 150K programs, we randomly sampled 1000 examples from this set for our experiments; this is due to the significant computational resources required to run experiments with state-of-the-art LLMs.

Table 1: Transferability of attacks across different models.

Model (M)	Acc on S ($Correct(S)/\|S\|$)	ASR ($Wrong(S_M^{adv})/\|S_M\|$)
seq2seq	100	44.76
GPT-3.5	60.7	29.49
GPT-4	67	21.04
Claude-Instant-1	47.4	25.11
Claude-2	59.6	22.99
CodeLlama	9	53.33

Following Srikant et al. [30], we use the small code model (with 48M parameters) trained on the code summarization task that has a seq2seq [31] architecture to generate the attacks. We evaluate the attacks on proprietary and open-source LLMs, including **GPT-3.5** (`gpt-3.5-turbo-0613`) and **GPT-4** (`gpt-4-0613`) from OpenAI [22], **Claude-Instant-1** (`claude-instant-1`) and **Claude-2** (`claude-2`) from Anthropic [5], and **CodeLlama** (`Codellama-7B-Instruct`) [28] from Meta. These models are amongst the top entries on a public leaderboard[2] that tracks the performance of LLMs on multiple tasks.

For notation, we use S to denote the set of 1000 clean inputs that are correctly classified by the baseline seq2seq model. We use S_M to denote the subset of S that each model M correctly classifies. When M=seq2seq, $S_M = S$. We compute S_M in this way to allow for a fair comparison among the models, as Srikant et al. [30] apply adversarial code perturbations only to the data that is classified correctly by the model. We use S_M^{adv} to denote the set of adversarial inputs obtained by applying the perturbations on inputs in S_M; note that the sizes of S_M and S_M^{adv} are the same, i.e., $|S_M| = |S_M^{adv}|$. For some dataset X, we use notations $Correct(X)$ and $Wrong(X)$ to denote the number of instances in X for which the output of a model is correctly classified or misclassified, where the model is implicit from the context.

RQ1: Do code attacks transfer from a small model to LLMs? Table 1 reports the accuracy of the models on the unperturbed samples in S and the attack success rate (ASR) [30] measured on perturbed samples obtained by attacking the seq2seq model in S_M. ASR is defined as the ratio of perturbed inputs that are misclassified to those whose original, unperturbed versions were correctly classified. S, as discussed, is the dataset with clean inputs that are correctly classified by the seq2seq model. Therefore, the accuracy of the seq2seq model on S is 100%. However, the overall accuracy of this seq2seq model on the entire dataset of Python programs is quite low, around 25%.

The results indicate that the adversarial perturbations indeed transfer, with highest ASR on CodeLlama (53%) and lowest ASR on GPT-4 (21%). In our experiments, CodeLlama underperforms compared to other models. This is primarily due to its tendency to generate lengthy explanations of the input code snippet rather than succinctly summarizing it with a single keyword according to the instructions provided in the prompt. Therefore, in the rest of the paper, the results for CodeLlama need to be taken with a grain of salt, as it was not well-suited for the code summarization task.

RQ2: Do the manually-engineered prompt defenses succeed against attacks? Table 2 reports our results for evaluating the two prompt-based defenses, `FSD` (via Few-Shot Examples) and

[2] https://chat.lmsys.org (Accessed in November, 2023)

Table 2: Effectiveness of the defenses across different models.

Model (M)	FSD Acc on S_M	FSD ASR	InvD Acc on S_M	InvD ASR
GPT-3.5	93.57	30.15	92.59	24.71
GPT-4	91.94	14.63	91.79	16.57
Claude-Instant-1	90.51	9.70	79.75	19.20
Claude-2	86.58	10.91	82.38	20.97
CodeLlama	74.44	66.67	50	74.44

Table 3: Evaluating InvD defense with meta-prompting.

Model(M)	EBMP + InvD		PAMP + InvD	
	Acc on S_M	ASR	Acc on S_M	ASR
GPT-3.5	89.71	10.48	94.3	6.07
GPT-4	95.4	4.3	94.96	3.86

InvD (via Inverse Transformation). FSD **Acc on** S_M and InvD **Acc on** S_M denote the percentage of clean inputs (out of S_M for each model M) that remain correctly labeled after the application of the respective defense (calculated as $(Correct(S_M)/|S_M|)$. The **ASR** is calculated as $(Wrong(S_M^{adv})/|S_M|$. The results indicate that accuracy remains high, so the defenses do not unintentionally lead the model into misclassifying many clean inputs. The results also indicate that the defenses are generally effective, reducing the ASR for the models with the exception of FSD on GPT-3.5, where the ASR slightly increases; the defenses are not effective on CodeLlama.

RQ3: Is meta-prompting effective? Table 3 shows the results for evaluating the integration of the two novel prompts, generated via Example-Based Meta-Prompting (EBMP) and Perturbation-Aware Meta-Prompting (PAMP), with the InvD method. **Acc on** S_M is calculated as $(Correct(S_M)/|S_M|)$ while the **ASR** is calculated as $(Wrong(S_M^{adv})/|S_M|)$. Due to resource constraints, we were only able to evaluate GPT-3.5 and GPT-4. Further evaluation with the other LLMs is left for future work. The results indicate that the LLM-generated defense prompts do not affect the accuracy of clean samples too much. Further, there is a significant reduction in ASR, indicating the effectiveness of LLM-generated prompts in defending against adversarial perturbations, with the best case observed on GPT-4 (ASR 3.86%).

It is intriguing that the LLM-generated prompts are more effective than the manually crafted prompts at defending the model. The LLM-generated prompts are almost identical to the manual prompts except for some additional instructions, in particular, the instruction to ensure that the output is syntactically correct and retains the original logic that appears in both EBMP and PAMP generated prompts. We hypothesize that the effectiveness of meta-prompting is an instance of the surprising capability of LLMs to generate prompts that are more effective than the ones crafted by humans, which was observed before on other tasks [25, 37].

RQ4: Does the meta attack work? Are defenses successful against the meta attacks? Table 4 reports the ASR for the meta attacks without defense and with InvD, EMBP, PAMP, and FSD. Due to resource constraints, we only evaluate GPT-3.5 and GPT-4. We consider two variants of the meta attack; the number of print statements that the attack is allowed to insert is either one or five. The results show that, unsurprisingly, when we allow five insertions, the ASR is always higher than when we attack only once. Without any defense, GPT-4 performance is better than GPT-3.5, which has a much lower ASR (18.66%) than GPT-3.5 (40.69%).

Table 4: Evaluating effectiveness of meta-attack. All columns report ASR on S_M calculated as $(Wrong(S_M^{adv})/|S_M|$. (1) and (5) refer to the attack hyperparameter, i.e., the number of inserted print statements.

Model (M)	No Defense	InvD	EMBP	PAMP	FSD
GPT-3.5 (1)	40.69	36.90	27.35	33.94	31.14
GPT-4 (1)	18.66	11.79	12.84	12.99	14.93
GPT-3.5 (5)	50.91	41.68	38.88	40.69	43.33
GPT-4 (5)	21.94	12.69	17.91	22.24	8.81

The results also indicate that all the defenses reduce the ASR in general, where InvD performs better on GPT4 and EMBP has a better performance on GPT-3.5. It is interesting to see the same type of defense having different effects on GPT-4 and GPT-3.5. This may be caused by the different understanding abilities of GPT-4 and GPT-3.5.

4 RELATED WORK

For code models trained on classification tasks, several works study vulnerability to adversarial examples [1, 11, 30, 36]. However, there is a scarcity of previous works that study the vulnerability of the much larger LLMs used in code classification tasks to the same adversarial perturbations. Recent work [34] applies natural, non-adversarial perturbations to input prompts —both code and textual descriptions— for measuring the robustness of LLMs on coding tasks. Another recent work [15] studies black-box attacks on code translation. In future work, we also plan to study attacks for code generation tasks using white-box techniques.

Classical defense approaches involve adversarial training, i.e., training using perturbed inputs [7, 19, 21, 26]. Such defenses remain challenging, as they often reduce model accuracy. To perform adversarial training for LLMs, one can explore fine-tuning the pre-trained large models, as allowed by current APIs. However adversarial training for LLMs is likely very expensive and of limited effect. Recently, defenses based on either filtering the inputs or outputs of LLMs have been proposed, but these approaches are geared towards defending the models against jailbreaking attacks [14, 17, 27]. We instead propose cost-effective self-defense prompting techniques.

5 CONCLUSION

In this paper we have shown that adversarial examples obtained with a smaller code model are transferable to modern LLMs; moreover LLMs are vulnerable to newly proposed meta-attacks. We further evaluated novel prompt-based self-defenses, that do not require retraining the LLMs, and showed their effectiveness. All the attack and defense strategies are generalizable to various coding tasks, such as code vulnerability detection or code generation, not only code summarization. As future work, we plan to investigate more powerful attacks and defenses through techniques that frame the search for finding the optimal attack or self-defense prompts as an optimization problem solved either via gradient-based search over LLM parameters or via techniques suitable for black-box LLMs but requiring multiple calls to the LLM.

6 ACKNOWLEDGMENTS

This work is supported in part by the Enterprise Security Initiative at Carnegie Mellon CyLab (FutureEnterprise@CyLab).

REFERENCES

[1] Miltiadis Allamanis, Marc Brockschmidt, and Mahmoud Khademi. 2018. Learning to Represent Programs with Graphs. In *International Conference on Learning Representations*. https://openreview.net/forum?id=BJOFETxR-

[2] Miltiadis Allamanis, Hao Peng, and Charles Sutton. 2016. A convolutional attention network for extreme summarization of source code. In *International conference on machine learning*. PMLR, 2091–2100.

[3] Uri Alon, Omer Levy, and Eran Yahav. 2019. code2seq: Generating Sequences from Structured Representations of Code. In *International Conference on Learning Representations*. https://openreview.net/forum?id=H1gKYo09tX

[4] Uri Alon, Meital Zilberstein, Omer Levy, and Eran Yahav. 2019. Code2Vec: Learning Distributed Representations of Code. *Proc. ACM Program. Lang.* 3, POPL, Article 40 (Jan. 2019), 29 pages. https://doi.org/10.1145/3290353

[5] Anthropic. 2023. Model Card and Evaluations for Claude Models. https://efficient-manatee.files.svdcdn.com/production/images/Model-Card-Claude-2.pdf?dm=1689034733

[6] Jacob Austin, Augustus Odena, Maxwell I. Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie J. Cai, Michael Terry, Quoc V. Le, and Charles Sutton. 2021. Program Synthesis with Large Language Models. *CoRR* abs/2108.07732 (2021). arXiv:2108.07732 https://arxiv.org/abs/2108.07732

[7] Pavol Bielik and Martin Vechev. 2020. Adversarial Robustness for Code. arXiv:2002.04694 [cs.LG]

[8] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[9] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harrison Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating Large Language Models Trained on Code. *CoRR* abs/2107.03374 (2021). arXiv:2107.03374 https://arxiv.org/abs/2107.03374

[10] Elizabeth Dinella, Hanjun Dai, Ziyang Li, Mayur Naik, Le Song, and Ke Wang. 2020. Hoppity: Learning Graph Transformations to Detect and Fix Bugs in Programs. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

[11] Fengjuan Gao, Yu Wang, and Ke Wang. 2023. Discrete Adversarial Attack to Models of Code. *Proceedings of the ACM on Programming Languages* 7, PLDI (2023), 172–195.

[12] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and Harnessing Adversarial Examples. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.). http://arxiv.org/abs/1412.6572

[13] Jindong Gu, Xiaojun Jia, Pau de Jorge, Wenqain Yu, Xinwei Liu, Avery Ma, Yuan Xun, Anjun Hu, Ashkan Khakzar, Zhijiang Li, Xiaochun Cao, and Philip Torr. 2023. A Survey on Transferability of Adversarial Examples across Deep Neural Networks. *arXiv preprint arXiv:2310.17626* (2023).

[14] Neel Jain, Avi Schwarzschild, Yuxin Wen, Gowthami Somepalli, John Kirchenbauer, Ping-yeh Chiang, Micah Goldblum, Aniruddha Saha, Jonas Geiping, and Tom Goldstein. 2023. Baseline defenses for adversarial attacks against aligned language models. *arXiv preprint arXiv:2309.00614* (2023).

[15] Akshita Jha and Chandan K Reddy. 2023. Codeattack: Code-based adversarial attacks for pre-trained programming language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 37. 14892–14900.

[16] Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. 2023. Survey of hallucination in natural language generation. *Comput. Surveys* 55, 12 (2023), 1–38.

[17] Aounon Kumar, Chirag Agarwal, Suraj Srinivas, Soheil Feizi, and Hima Lakkaraju. 2023. Certifying llm safety against adversarial prompting. *arXiv preprint arXiv:2309.02705* (2023).

[18] Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d'Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, James Molloy, Daniel J. Mankowitz, Esme Sutherland Robson, Pushmeet Kohli, Nando de Freitas, Koray Kavukcuoglu, and Oriol Vinyals. 2022. Competition-level code generation with AlphaCode. *Science* 378, 6624 (2022), 1092–1097. https://doi.org/10.1126/science.abq1158 arXiv:https://www.science.org/doi/pdf/10.1126/science.abq1158

[19] Zhen Li, Guenevere (Qian) Chen, Chen Chen, Yayi Zou, and Shouhuai Xu. 2022. RoPGen: Towards Robust Code Authorship Attribution via Automatic Coding Style Transformation. In *Proceedings of the 44th International Conference on Software Engineering* (Pittsburgh, Pennsylvania) *(ICSE '22)*. Association for Computing Machinery, New York, NY, USA, 1906–1918. https://doi.org/10.1145/3510003.3510181

[20] Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and LINGMING ZHANG. 2023. Is Your Code Generated by ChatGPT Really Correct? Rigorous Evaluation of Large Language Models for Code Generation. In *Thirty-seventh Conference on Neural Information Processing Systems*. https://openreview.net/forum?id=1qvx610Cu7

[21] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2019. Towards Deep Learning Models Resistant to Adversarial Attacks. arXiv:1706.06083 [stat.ML]

[22] OpenAI. 2023. GPT-4 Technical Report. arXiv:2303.08774 [cs.CL]

[23] Kexin Pei, David Bieber, Kensen Shi, Charles Sutton, and Pengcheng Yin. 2023. Can Large Language Models Reason about Program Invariants?. In *Proceedings of the 40th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 202)*, Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (Eds.). PMLR, 27496–27520. https://proceedings.mlr.press/v202/pei23a.html

[24] Michael Pradel and Koushik Sen. 2018. DeepBugs: A Learning Approach to Name-Based Bug Detection. *Proc. ACM Program. Lang.* 2, OOPSLA, Article 147 (oct 2018), 25 pages. https://doi.org/10.1145/3276517

[25] Reid Pryzant, Dan Iter, Jerry Li, Yin Tat Lee, Chenguang Zhu, and Michael Zeng. 2023. Automatic prompt optimization with" gradient descent" and beam search. *arXiv preprint arXiv:2305.03495* (2023).

[26] Goutham Ramakrishnan, Jordan Henkel, Zi Wang, Aws Albarghouthi, Somesh Jha, and Thomas Reps. 2020. Semantic robustness of models of source code. *arXiv preprint arXiv:2002.03043* (2020).

[27] Alexander Robey, Eric Wong, Hamed Hassani, and George J Pappas. 2023. SmoothLLM: Defending Large Language Models Against Jailbreaking Attacks. *arXiv preprint arXiv:2310.03684* (2023).

[28] Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. 2023. Code Llama: Open Foundation Models for Code. arXiv:2308.12950 [cs.CL]

[29] Xinyue Shen, Zeyuan Chen, Michael Backes, Yun Shen, and Yang Zhang. 2023. " do anything now": Characterizing and evaluating in-the-wild jailbreak prompts on large language models. *arXiv preprint arXiv:2308.03825* (2023).

[30] Shashank Srikant, Sijia Liu, Tamara Mitrovska, Shiyu Chang, Quanfu Fan, Gaoyuan Zhang, and Una-May O'Reilly. 2021. Generating Adversarial Computer Programs using Optimized Obfuscations. In *International Conference on Learning Representations*. https://openreview.net/forum?id=PH5PH9ZO_4

[31] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. *Advances in neural information processing systems* 27 (2014).

[32] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2014. Intriguing properties of neural networks. arXiv:1312.6199 [cs.CV]

[33] Marko Vasic, Aditya Kanade, Petros Maniatis, David Bieber, and Rishabh singh. 2019. Neural Program Repair by Jointly Learning to Localize and Repair. In *International Conference on Learning Representations*. https://openreview.net/forum?id=ByloJ20qtm

[34] Shiqi Wang, Zheng Li, Haifeng Qian, Chenghao Yang, Zijian Wang, Mingyue Shang, Varun Kumar, Samson Tan, Baishakhi Ray, Parminder Bhatia, Ramesh Nallapati, Murali Krishna Ramanathan, Dan Roth, and Bing Xiang. 2022. ReCode: Robustness Evaluation of Code Generation Models. arXiv:2212.10264 [cs.LG]

[35] Chunqiu Steven Xia, Yuxiang Wei, and Lingming Zhang. 2023. Automated program repair in the era of large pre-trained language models. In *Proceedings of the 45th International Conference on Software Engineering (ICSE 2023). Association for Computing Machinery.*

[36] Noam Yefet, Uri Alon, and Eran Yahav. 2020. Adversarial examples for models of code. *Proceedings of the ACM on Programming Languages* 4, OOPSLA (2020), 1–30.

[37] Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. 2023. Large Language Models are Human-Level Prompt Engineers. In *The Eleventh International Conference on Learning Representations*. https://openreview.net/forum?id=92gvk82DE-

[38] Andy Zou, Zifan Wang, J. Zico Kolter, and Matt Fredrikson. 2023. Universal and Transferable Adversarial Attacks on Aligned Language Models. *CoRR* abs/2307.15043 (2023). https://doi.org/10.48550/arXiv.2307.15043 arXiv:2307.15043

[39] Andy Zou, Zifan Wang, J. Zico Kolter, and Matt Fredrikson. 2023. Universal and Transferable Adversarial Attacks on Aligned Language Models. arXiv:2307.15043 [cs.CL]