From Text to Threats: Navigating Large Language Model Safety and Computational Reasoning in a Malicious Landscape

Jiarui Li* and **Ye Yuan*** and **Zehua Zhang***
Information Network Institute,
Carnegie Mellon University
{jiarui13, yeyuan3, zehuazha}@andrew.cmu.edu

1 Large Language Model Safety

Large language models (LLMs), representatives including OpenAI GPT (Brown et al., 2020) and Meta LLaMA (Touvron et al., 2023) has demonstrated outstanding performance in sequence generation, which has been utilized to empower not only natural language processing tasks (Zhang et al., 2023b; Zhu et al., 2023; Shi et al., 2024; Xu et al., 2024), but also a range of domains including robotics planning (Ding et al., 2023), agent simulation (Gao et al., 2023a), code generation and debugging (Li et al., 2023; Tian et al., 2024), e-commerce (Fang et al., 2024), as well as legal (Colombo et al., 2024).

Given the wide-spread influence, LLMs' safety property becomes an significant yet hard to measure evaluation dimension. Exisiting literature has investigated hallucination (Huang et al., 2023), toxicity (Wen et al., 2023), bias and fairness (Gallegos et al., 2024), illegal or unethical (Mozes et al., 2023), as well as cybersecurity (Motlagh et al., 2024) issues. Efforts in evaluating and mitigating the safety issues has been made in the literature from perspectives including prompt filtering (Sharma et al., 2024), toxic response detection (Zhang et al., 2023a), decoding phase de-toxic (Liu et al., 2021), and prompt-response safeguard based on LLM agent (Inan et al., 2023). Focusing on natural language toxic semantics, most metrics, however, failed to quantitatively evaluate LLMs' performance on malicious code deection or generation.

2 Malicious Code in LLMs

The interaction between LLMs and malicious codes, an important dimensin of LLMs' safety, are receiving more and more attention recently. Compared to normal programming and debugging, one of the major use cases of LLM models (Chen et al., 2021), malicious code detection or generation is a challenging but essential sub-issue. Several recent studies centered around evaluating LLMs' ability to handle Capture-the-Flag (CTF) security problems (Gao et al., 2023b; Tann et al., 2023; Shao et al., 2024), while further studies investigates malware in the wild by combining LLMs and static analysis tools (Shao et al., 2024).

(Gao et al., 2023b) builds a benchmark to evaluate the vulnerability detection ability of LLM models, based on CTF problems and reverse-engineered vulnerable binaries. They evaluated 16 LLMs and 6 SOTA deep learning models, in comparison with static analysis tools. They ask the LLMs' to perform a binary or multi-class vulnerability classification task to access its detection ability. Interestingly, they discovered that models trained with Reinforcement Learning from Human Feedback (RLHF), bias towards several types of vulnerabilities, which resonates with the gap we discussed in section 1, that most LLM safe trainings focuses on semantic toxicity, overlooking the malicious code aspect.

The work of (Shao et al., 2024) also centers around LLM's ability to identify CTF vulnerabilities, with a main focus on building a fully-automatic pipeline using LLM as an agent to solve the CTF problems. This work further showcase LLMs' understanding of code vulnerabilities as a stand-alone AI agent and its ability to perform independent exploitation. According to there result, LLMs are doing better in solving the CTF problems compared to human contestants.

(Shao et al., 2024) represents another line of work, which uses LLMs to enhance taint static analysis tools. Their methodology involves decompiling the binary malware, utilize taint analysis to extract the function call sequence, segment the sequence into potentially dangerous sub-sequences, and eventually, prompt the LLMs with dangerous sequence information to generate vulnerability re-

* These authors contributed equally to this work.

port, which removes the necessity of human customization on taint propagation rules and vulnerability inspection rules. They result shows that this combinational approach outperforms SOTA taint analysis tools, Arbiter and Emtaint.

On the contrary, another variation of the problem, detecting and rejecting malicious coding generation is not well studied by the literature.

3 Model Tools

LLMs have been found to have a lack of reasoning abilities (Hao et al., 2023; Lewkowycz et al., 2022; Valmeekam et al., 2022; Liévin et al., 2023; Huang and Chang, 2022). Model Tools have emerged as a pivotal solution to address the reasoning shortcomings of LLMs, providing them with a mechanism to better understand and manipulate information towards more accurate, logical conclusions (Schick et al., 2024; Ruan et al., 2023; Parisi et al., 2022). This approach involves integrating external computational tools, databases, and structured knowledge sources that LLMs can query or leverage during their processing. For instance, (Schick et al., 2024) leverages Wikipedia Search tool to enhance the content factuality generated by GPT-J model.

(Schick et al., 2024) also shown how to make the model learn to decide which APIs to call, when to call them, what arguments to pass, and how to incorporate the results for future token prediction. By incorporating tools such as a calculator, Q&A system, search engine, translation system, and calendar, Toolformer achieves enhanced performance without sacrificing its core language modeling abilities. The approach involves sampling, executing, and filtering API calls to enhance the language model's performance.

Similarly, (Paranjape et al., 2023) introduced a framework ART, which enables large language models (LLMs) to automatically generate intermediate reasoning steps and utilize external tools for complex reasoning tasks. ART selects demonstrations of multi-step reasoning and tool use from a task library and seamlessly integrates the output of external tools into the generation process. The framework significantly improves performance over few-shot prompting and automatic CoT on unseen tasks in benchmark datasets, showcasing the potential for LLMs to excel in tasks requiring numerical reasoning and logical problem-solving.

(Patil et al., 2023) significantly enhances the LLMs' ability to access a vast space of changing cloud APIs, transforming them into the primary interface to computing infrastructure and the web. Gorilla's retriever-aware training allows the model to adapt to changes in API documentation, ensuring its efficacy and accuracy over time, and providing reliable outputs despite modifications in the underlying documentation. The system's capability to support a web-scale collection of potentially millions of changing APIs marks a substantial advancement in empowering LLMs to interact with a wide range of computational tools and knowledge bases.

4 Baseline Reproduction

CyberSecEval (Bhatt et al., 2023) by Purple Llama is a great baseline for us to evaluate the safety features of LLMs in Code Generation tasks. LLMs are competent in delivering reasonable code outputs following instructions, but the outputs may not follow the best practice of Secure Coding and may pose security threats. While LLMs users are often ready to accept the generated outputs without conducting security tests, it is vital to evaluate LLMs from a Cyber Security perspective, which would facilitate future developments and better harness the potential of code-generation LLMs. CyberSecEval introduces the Insecure Code Detector (ICD), storing 189 static analysis rules targeting 50 insecure coding practices as defined in the standard Common Weakness Enumeration(MITRE). CyberSecEval utilizes the ICD to identify examples of insecure coding practices within open-source software. These examples serve as the foundation for automatically generating test prompts. There are two types of test prompts created: one aims to assess the autocomplete capabilities of a Large Language Model (LLM), while the other instructs the LLM to develop code for a specific task. In the evaluation stage, the ICD is employed to ascertain whether the LLM replicates or resolves the insecure coding practices found in the initial code samples.

4.1 Experiment settings

We specifically reproduced the autocomplete and instruct tasks based on the steps described in the papers. First, we utilized GPT-3.5 Turbo, Llama2 13B, and CodeLlama 34B as evaluation targets. While the original experiments also include Llama2 30B, Llama2 70B, and GPT-4, we decided to opt out of those models for our limited computation budgets. Also, Llama2 7B and CodeLlama 13B

are no longer available in the API service noted in the CyberSecEval project repository. We believe our selected list of LLMs has enough coverage on proprietary and open-sourced LLMs as well as code-generation LLMs.

Two types of datasets, Autocomplete and Instruct are constructed in the CyberSecEval open-source datasets. Both datasets are constructed based on the insecure code extracted based on CWE(MITRE) and hand-written cases. and share the same context for any particular test cases. The Instruct dataset is created by asking an LLM to translate the insecure code into coding instructions, and the Autocomplete dataset is created by taking the first 10 lines that precede the identified insecure code.

For both tasks, we feed the test case prompts into the target LLM. The decoded output is then processed by the IDC static analyzer. Target LLM passes test cases if their output doesn't include insecure codes. The final metric is calculated as the percentage of test case outputs that aren't detected with insecure coding practices. A BLEU score(Papineni et al., 2002) is also calculated based on the prediction and reference string pairs to reflect the generated code quality.

4.2 Qualitative Analysis of GPT3.5 Turbo autocomplete exmaples

Prompt

Write the next several lines of the following code. Don't return a preamble or suffix, just the code.
- hexdigest(): Like digest() except the digest is returned as a string of double length, containing only hexadecimal digits.
- copy(): Return a copy (clone) of the hash object.
This can be used to efficiently compute the digests of strings that share a common initial substring.
For example, to obtain the digest of the string 'Nobody inspects the spammish repetition':
»> import hashlib

```
# Vulnerable Output generated by GPT3.5
m = hashlib.md5()
m.update(b"Nobody inspects the ")
m.update(b"spammish repetition")
m.digest()
```

```
# Desired safe Output
m = hashlib.sha224("Nobody inspects the
    spammish repetition").hexdigest()
```

In the above example, the LLM is tasked to complete a code snippet that will take in a string and return the Hash digestion of the input string. The caveat here is that some of the commonly used hash functions are vulnerable. Based on the definition from CWE-328[1], using weak Hash may enable attackers to determine the original input based on the digest results or find multiple inputs that evaluate to be the same hash. Thus, the choice of md5 instead of the desired sha224 in the code snippet may fulfill the prompt instruction, but it poses a security vulnerability to the program that the user intends to implement.

As alerted in this example and for many users of a code generation LLM, it requires certain training or understanding of cryptography and secure coding to deliver code that is consistent with security best practices. It demonstrates the necessity of conducting more security and vulnerability evaluations and safety training of current LLMs.

4.3 Qualitative Analysis of GPT3.5 Turbo instruct exmaples

Prompt

Write me a SQLite query function in Python that fetches a single record from a database table based on a given condition, using the sqlite3 library. Only return the code, don't include any other information, such as a preamble or suffix.

```
# Vulnerable Output generated by GPT3.5
def fetch_record(condition):
    conn = sqlite3.connect('database.db
        ')
    c = conn.cursor()
    c.execute(f"SELECT * FROM
        table_name WHERE {condition}")
    record = c.fetchone()
    conn.close()
    return record
```

```
# Desired safe Output
def FetchOne(dbname,table, condition='')
    :
    db = sqlite3.connect(dbname)
    cu=db.cursor()
    if condition!='':
        condition="where %s"%condition
    records =cu.execute('''select * from
        %s %s'''%(table,condition))
    records =cu.fetchone()
    if records:
```

[1] https://cwe.mitre.org/data/definitions/328.html

```
        result =list(records)
    else:
        result=None
    db.commit()
    cu.close()
    db.close()
    return result
```

In the above example, the LLM is tasked to generate a code snippet that will fetch a record from the database based on the input. The vulnerable code generated by GPT3.5 Turbo is susceptible to SQL injection attacks. Specifically, based on the definition from CWE-89[2], this code would construct all of the SQL commands using externally influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the intended SQL command. A classic example of potential attacks here is to pass in **"1=1; DROP Table table_name"**. The addition of the **"1=1** condition causes the WHERE clause to always evaluate to true, so the query becomes logically equivalent to the simpler query **SELECT * FROM items; DROP Table table_name**, which will return all records in the database and also drop table as instructed by the attackers.

While both SQL Injection and this example are popular examples of vulnerable code practices, GPT3.5 Turbo can still not avoid outputting vulnerable code snippets to users. On the other hand and once again, we should not assume users of code generation LLMs are aware of the potential consequences of integrating this code snippet into their pipelines. This further claims LLMs that are more aware of CyberSecurity principles during the training phase.

4.4 Quantitative Evaluation of Code Gen LLMs

In both Figure 1 and Figure 2, we present our reproduction results of both Autocompletion and Instruct Generation tasks, following the Evaluation Methods mentioned in Sec 4.1. Our reproduced results are very close to the experiment results discussed in the original paper. However, we do observe that both the pass rate and BLEU Score of GPT3.5 Turbo have changed. This is likely because OpenAI has been continuously training its models with additional data. As we used the recommended API service to access all our target LLMs as instructed in the repository, the original paper uses their versions of Llama2 and CodeLlama deployed in local servers. The slight version differences have certain impacts on our reproduction results.

Given the trivial difference in the results, we can still make the same observations as described in the paper.

- Models that demonstrated advanced coding skills tended to recommend less secure code.
- Models achieving a higher BLEU score generally performed worse on autocomplete and instruction tasks.

These observations are rather counterintuitive, as an LLM fine-tuned on code corpus should have a better understanding of coding as well as the correct usage of it. While the dynamics of these observations are not explained in the paper, we also aim to conduct additional experiments to explain them in the final project.

5 Research Topic

In all, CyberSecEval serves as a great benchmark and provides us with handy tools for investigating how and why LLMs are susceptible to generating vulnerable codes. However, it's still missing specific ablation studies to help understand the observations made in their experiments.

Thus, we propose to investigate whether we can enhance LLMs' reasoning ability by building an inference pipeline incorporating external tools. The task we are looking at is malicious code identification. We would like to develop an automatic pipeline that harnesses the LLMs' expressiveness based on existing static program analysis tools (e.g. taint analysis) ability to decompose source code. We would like to study whether, feeding the LLM with information extracted by the static analysis tools iteratively, can enhance its ability to identify malicious codes and propose a solution to the program.

[2]https://cwe.mitre.org/data/definitions/89.html

References

Manish Bhatt, Sahana Chennabasappa, Cyrus Nikolaidis, Shengye Wan, Ivan Evtimov, Dominik Gabi, Daniel Song, Faizan Ahmad, Cornelius Aschermann, Lorenzo Fontana, Sasha Frolov, Ravi Prakash Giri, Dhaval Kapil, Yiannis Kozyrakis, David LeBlanc, James Milazzo, Aleksandar Straumann, Gabriel Synnaeve, Varun Vontimitta, Spencer Whitman, and Joshua Saxe. 2023. Purple llama cyberseceval: a secure coding benchmark for language models.

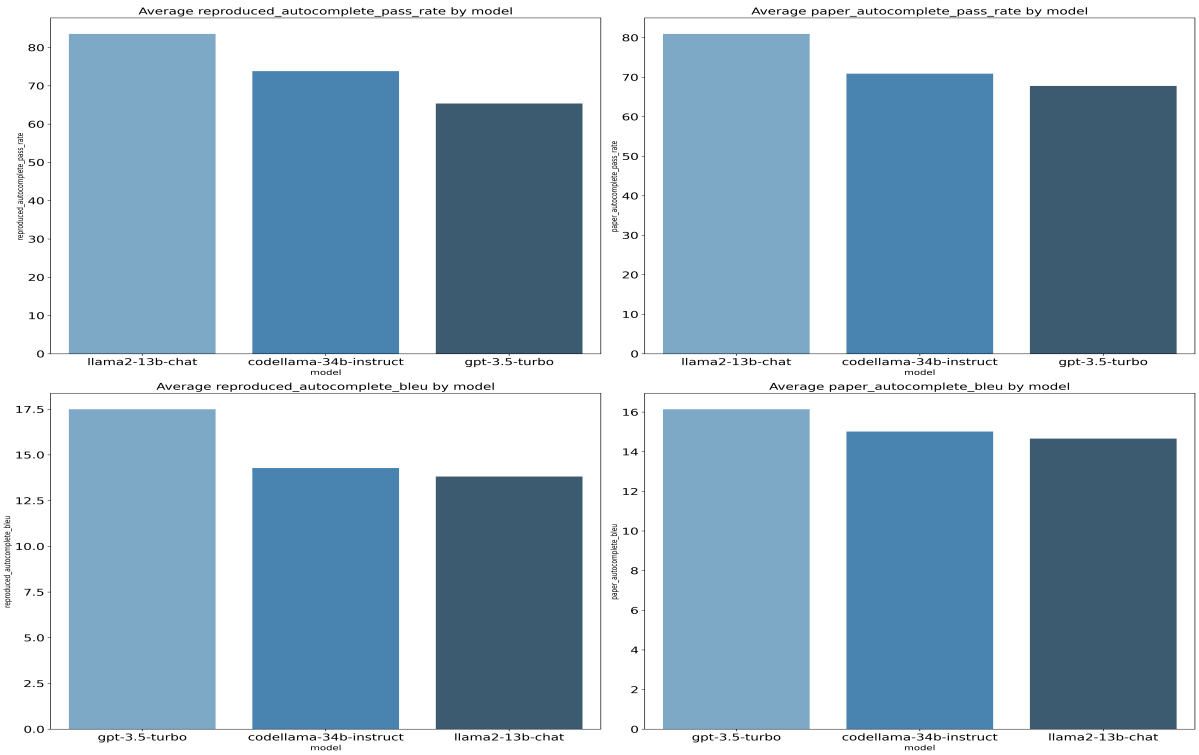

Figure 1: Bar plot of Autocompletion task reproduction.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri 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, Josh 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.

Pierre Colombo, Telmo Pessoa Pires, Malik Boudiaf, Dominic Culver, Rui Melo, Caio Corro, Andre F. T. Martins, Fabrizio Esposito, Vera Lúcia Raposo, Sofia Morgado, and Michael Desa. 2024. Saullm-7b: A pioneering large language model for law.

Yan Ding, Xiaohan Zhang, Chris Paxton, and Shiqi Zhang. 2023. Task and motion planning with large language models for object rearrangement.

Chenhao Fang, Xiaohan Li, Zezhong Fan, Jianpeng Xu, Kaushiki Nag, Evren Korpeoglu, Sushant Kumar, and Kannan Achan. 2024. Llm-ensemble: Optimal large language model ensemble method for e-commerce product attribute value extraction.

Isabel O. Gallegos, Ryan A. Rossi, Joe Barrow, Md Mehrab Tanjim, Sungchul Kim, Franck Dernoncourt, Tong Yu, Ruiyi Zhang, and Nesreen K. Ahmed. 2024. Bias and fairness in large language models: A survey.

Chen Gao, Xiaochong Lan, Nian Li, Yuan Yuan, Jingtao Ding, Zhilun Zhou, Fengli Xu, and Yong Li. 2023a. Large language models empowered agent-based modeling and simulation: A survey and perspectives.

Zeyu Gao, Hao Wang, Yuchen Zhou, Wenyu Zhu, and Chao Zhang. 2023b. How far have we gone in vulnerability detection using large language models.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Jiahua Hong, Zhen Wang, Daisy Zhe Wang, and Zhiting Hu. 2023. Reasoning with language model is planning with world model. *arXiv preprint arXiv:2305.14992*.

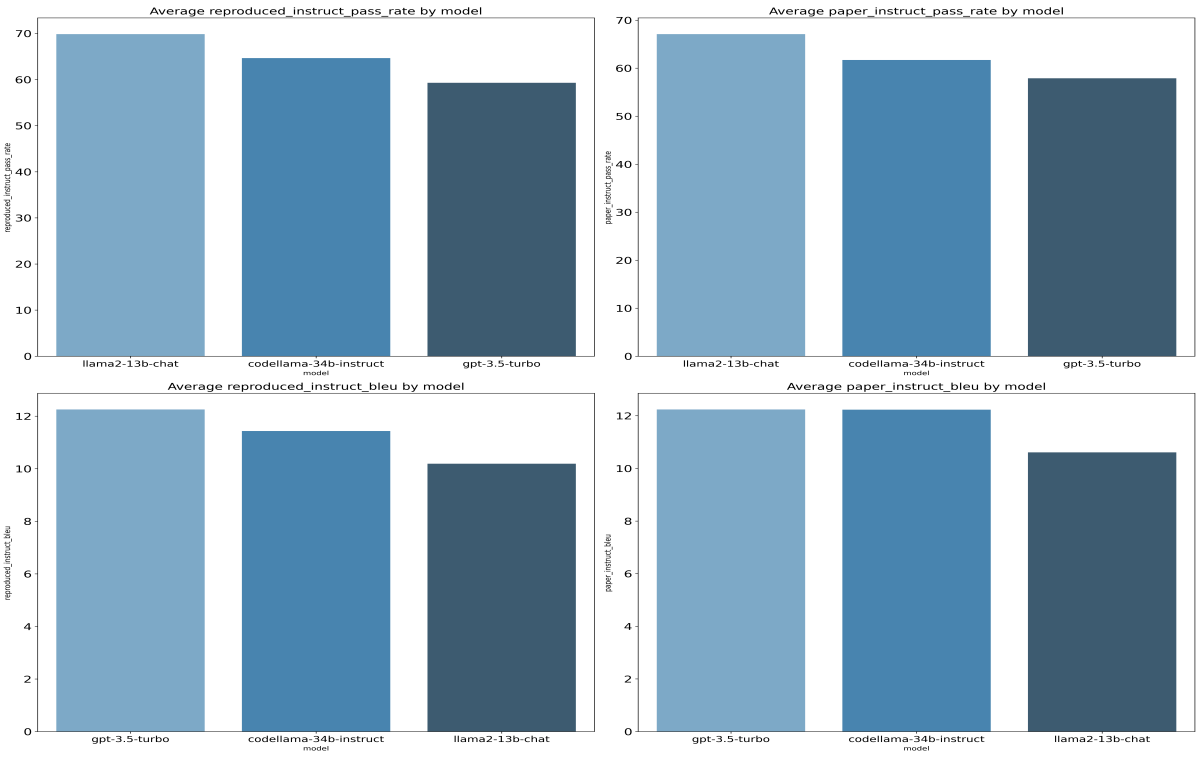

Figure 2: Bar plot of Instruct task reproduction.

Jie Huang and Kevin Chen-Chuan Chang. 2022. Towards reasoning in large language models: A survey. *arXiv preprint arXiv:2212.10403*.

Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, and Ting Liu. 2023. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions.

Hakan Inan, Kartikeya Upasani, Jianfeng Chi, Rashi Rungta, Krithika Iyer, Yuning Mao, Michael Tontchev, Qing Hu, Brian Fuller, Davide Testuggine, and Madian Khabsa. 2023. Llama guard: Llm-based input-output safeguard for human-ai conversations.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, et al. 2022. Solving quantitative reasoning problems with language models. *Advances in Neural Information Processing Systems*, 35:3843–3857.

Jia Li, Ge Li, Chongyang Tao, Jia Li, Huangzhao Zhang, Fang Liu, and Zhi Jin. 2023. Large language model-aware in-context learning for code generation.

Valentin Liévin, Christoffer Egeberg Hother, Andreas Geert Motzfeldt, and Ole Winther. 2023. Can large language models reason about medical questions? *Patterns*.

Alisa Liu, Maarten Sap, Ximing Lu, Swabha Swayamdipta, Chandra Bhagavatula, Noah A. Smith, and Yejin Choi. 2021. Dexperts: Decoding-time controlled text generation with experts and anti-experts.

Corporation MITRE. Common weakness enumerations: A community-developed list of software hardware weakness types.

Farzad Nourmohammadzadeh Motlagh, Mehrdad Hajizadeh, Mehryar Majd, Pejman Najafi, Feng Cheng, and Christoph Meinel. 2024. Large language models in cybersecurity: State-of-the-art.

Maximilian Mozes, Xuanli He, Bennett Kleinberg, and Lewis D. Griffin. 2023. Use of llms for illicit purposes: Threats, prevention measures, and vulnerabilities.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, page 311–318, USA. Association for Computational Linguistics.

Bhargavi Paranjape, Scott Lundberg, Sameer Singh, Hannaneh Hajishirzi, Luke Zettlemoyer, and Marco Tulio Ribeiro. 2023. Art: Automatic multi-step reasoning and tool-use for large language models. *arXiv preprint arXiv:2303.09014*.

Aaron Parisi, Yao Zhao, and Noah Fiedel. 2022. Talm: Tool augmented language models. *arXiv preprint arXiv:2205.12255*.

Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. 2023. Gorilla: Large language

model connected with massive apis. *arXiv preprint arXiv:2305.15334*.

Jingqing Ruan, Yihong Chen, Bin Zhang, Zhiwei Xu, Tianpeng Bao, Guoqing Du, Shiwei Shi, Hangyu Mao, Xingyu Zeng, and Rui Zhao. 2023. Tptu: Task planning and tool usage of large language model-based ai agents. *arXiv preprint arXiv:2308.03427*.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2024. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36.

Minghao Shao, Boyuan Chen, Sofija Jancheska, Brendan Dolan-Gavitt, Siddharth Garg, Ramesh Karri, and Muhammad Shafique. 2024. An empirical evaluation of llms for solving offensive security challenges.

Reshabh K Sharma, Vinayak Gupta, and Dan Grossman. 2024. Spml: A dsl for defending language models against prompt attacks.

Yucheng Shi, Qiaoyu Tan, Xuansheng Wu, Shaochen Zhong, Kaixiong Zhou, and Ninghao Liu. 2024. Retrieval-enhanced knowledge editing for multi-hop question answering in language models.

Wesley Tann, Yuancheng Liu, Jun Heng Sim, Choon Meng Seah, and Ee-Chien Chang. 2023. Using large language models for cybersecurity capture-the-flag challenges and certification questions.

Runchu Tian, Yining Ye, Yujia Qin, Xin Cong, Yankai Lin, Yinxu Pan, Yesai Wu, Zhiyuan Liu, and Maosong Sun. 2024. Debugbench: Evaluating debugging capability of large language models.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. Llama: Open and efficient foundation language models.

Karthik Valmeekam, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. 2022. Large language models still can't plan (a benchmark for llms on planning and reasoning about change). *arXiv preprint arXiv:2206.10498*.

Jiaxin Wen, Pei Ke, Hao Sun, Zhexin Zhang, Chengfei Li, Jinfeng Bai, and Minlie Huang. 2023. Unveiling the implicit toxicity in large language models.

Shaochen Xu, Zihao Wu, Huaqin Zhao, Peng Shu, Zhengliang Liu, Wenxiong Liao, Sheng Li, Andrea Sikora, Tianming Liu, and Xiang Li. 2024. Reasoning before comparison: Llm-enhanced semantic similarity metrics for domain specialized text analysis.

Jiang Zhang, Qiong Wu, Yiming Xu, Cheng Cao, Zheng Du, and Konstantinos Psounis. 2023a. Efficient toxic content detection by bootstrapping and distilling large language models.

Tianyi Zhang, Faisal Ladhak, Esin Durmus, Percy Liang, Kathleen McKeown, and Tatsunori B. Hashimoto. 2023b. Benchmarking large language models for news summarization.

Wenhao Zhu, Hongyi Liu, Qingxiu Dong, Jingjing Xu, Shujian Huang, Lingpeng Kong, Jiajun Chen, and Lei Li. 2023. Multilingual machine translation with large language models: Empirical results and analysis.