

Article

Not peer-reviewed version

Application of Open- source Large Language Model (LLM) for Simulation of a Vulnerable IoT System and Cybersecurity Best Practices Assistance

[Veneta Yosifova](#) *

Posted Date: 17 May 2024

doi: 10.20944/preprints202405.1169.v1

Keywords: cybersecurity; open-source; large language models; IoT



Preprints.org is a free multidiscipline platform providing preprint service that is dedicated to making early versions of research outputs permanently available and citable. Preprints posted at Preprints.org appear in Web of Science, Crossref, Google Scholar, Scilit, Europe PMC.

Copyright: This is an open access article distributed under the Creative Commons Attribution License which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Article

Application of Open-Source Large Language Model (LLM) for Simulation of a Vulnerable IoT System and Cybersecurity Best Practices Assistance

Veneta Yosifova

Faculty of Computer Systems and Technologies, Technical University of Sofia 1756 Sofia, Bulgaria;
venetay@tu-sofia.bg

Abstract: This paper explores the role of open-source large language models in IoT cybersecurity world. The threats of malicious activity on the Internet and the loss of private information are very real and lead to serious consequences. The purpose of this paper is to investigate how open source-large language models can help to defend against the growing threat of cyber-crimes. We conducted our experiments in two directions. The first one is a security assistant that helps with cybersecurity best practices advices. The second one is a how large language model can simulate a vulnerable IoT system. For both types of experiments, the interactive mode of operation of the language model is used. In the context of the cybersecurity research, a major advantage of the locally installed open-sourced large language models is that they do not share sensitive data with a remote system in a cloud. The paper concludes by discussing the potential impact of open-source large language models on cybersecurity research and recommends future research directions.

Keywords: cybersecurity; artificial intelligence; open-source; large language models

1. Introduction

In this paper, we are exploring the role of open-sourced generative large language models in IoT cybersecurity research. The threats of malicious activity on the Internet and the loss of private information are very real and lead to serious consequences. The purpose of this paper is to investigate how open-source large language models can help defending against the growing threat of cyber-crimes. Cybersecurity research involves techniques for protection from attacks of information systems, including computer systems, networks, critical infrastructures, programs, “Internet of Things” (IoT) devices, and so on, as this is done by detecting, responding, and preventing cyber-incidents. One of the often used methods for analysis of the attack vectors involves setting up honeypots. Creating a new one or choosing the right honeypot and setting it to work as expected is a hard and time-consuming task [1]. For this reason, with the development of the new generative models, cybersecurity researchers are looking for a way to implement them into their defense toolkit [2–4]. These developments rely on large language models accessed by OpenAI's ChatGPT, Google's Bard and others generative chatbots because they provide free API access. However, the most popular and large language models like Meta Llama 2 [5], OpenAI GPT (Generative Pretrained Transformer) [6], Google PaLM (Pathways Language Model) [7] or LaMDA (Language Model for Dialogue Applications), etc. have restrictions for acceptable use. On the other hand, an openly licensed reproduction of Meta's original LLaMA2 model, called OpenLLaMA was introduced in 2023. It uses the same architecture and is a replacement for the original LLaMA paper [8]. They provide PyTorch weights of pre-trained models trained them on the RedPajama dataset. This is a reproduction of the LLaMA training dataset containing over 1 trillion tokens and is a mixture of seven data slices. The slices are “CommonCrawl: Five dumps of CommonCrawl, processed using the CCNet pipeline, and filtered via several quality filters including a linear classifier that selects for Wikipedia-like pages; C4 [Colossal Clean Crawled Corpus]: Standard C4 dataset; GitHub: GitHub data, filtered by licenses and quality; arXiv: Scientific articles removing boilerplate; Books: A corpus

of open books, deduplicated by content similarity; Wikipedia: A subset of Wikipedia pages, removing boilerplate; StackExchange: A subset of popular websites under StackExchange, removing boilerplate" [9]. They follow "the exactly same preprocessing steps and training hyperparameters as the original LLaMA paper, including model architecture, context length, training steps, learning rate schedule, and optimizer" [10]. This gives us confidence that the open-source large language models can be used on par with the popular ones provided by the big companies in the industry.

This paper is organized as follows. Section 2 reviews related work in the field of generative AI used for cybersecurity tasks. Section 3 describes the techniques and the model setup. Section 4 presents the conducted experiments and their results. In Section 5 conclusions and discussions are stated.

2. Related Work

Before the large language models became popular, other more established, machine learning methods have been used in cybersecurity research. For predicting vulnerability exploitation Fang, et al. in [12] developed an algorithm based on ensemble machine learning algorithms. Zhou, et al. in [13] used graph neural networks on the program semantics for vulnerability identification. Li, et al. in [14] and O. de Vel, et al. in [11] rely on deep learning for detecting software vulnerabilities.

Large language models use a deep learning neural network as variation of the traditional language models. After Google's Vaswani et. al. [16] developed and open-sourced the idea of Transformers, it marked the rise of large language models. This is true especially for the most recognizable one of all them, the GPT series, which is the basis of OpenAI's ChatGPT, presented in 2022. This also leads to the idea of applying them as cybersecurity instruments. In [15], Happe et al. present a scenario where the language model interacts with a virtual machine using automatically generated commands and tries to compromise it. Sladić, et al. in [3] and Ragsdale, et al. in [4] use GPT for creating low-risk honeypots. In order for their code to work, it requires valid access to OpenAI's API for GPT-3.5-turbo-16k. Their idea is based on the McKee, et al. ChatGPT honeypot [2].

Other use-cases of ChatGPT are code generation as in Murr, et al. [18] and the "application in detecting malicious web content, particularly phishing sites" [17]. For comparison Koide, et al. use performance metrics such as precision, recall, accuracy and f-measure. The difference between the different versions is also evident in their research. The capability of "non-phishing site classification – [False Positives] (FP) and [True Negatives] (TN) was comparable between GPT-3.5 and GPT-4, resulting in similar precision values. However, GPT-4 outperforms GPT-3.5 by approximately 5.8%" [17]. Authors as Gupta, et al. examine "defense techniques and uses of GenAI tools to improve security measures, including cyber-defense automation, reporting, threat intelligence, secure code generation and detection, attack identification, developing ethical guidelines, incident response plans, and malware detection" [19]. Despite the importance of these studies, none of them relies on locally installed open-sourced large language models. The authors' choice was to use remote API access provided by the cloud LLMs. In contrast, this paper presents the role of a locally running open-source large language models for simulating vulnerable IoT system and cybersecurity improving.

3. Models and Used Techniques

For local use of large language models, tools such as llama.cpp [20] are essential. This instrument optimizes the size and allows the models to be run on the user's hardware without needing cloud systems. The hardware resources of the machine on which the experiments were conducted are AMD® Ryzen 7 CPU 4.4GHz with 8 cores and 16 threads, 32GiB RAM, and 1TB disk capacity.

The llama.cpp is a tool whose "main goal is to run the LLaMA model using 4-bit integer quantization" [20,21] as 2-bit, 3-bit, 4-bit, 5-bit, 6-bit and 8-bit integer quantization are supported. Supported platforms are Mac OS, Linux, Windows and it can run on Docker. llama.cpp has base interface and interactive mode for more chatGPT-like experience. Despite that the hardware parameters allow experimentation with even larger language models, the open-license reproduction of Meta AI's LLaMA large language models with 13 billion parameters comparable to GPT-3 [10] was

used in our experiments. Figure 1 shows the original and the quantized size, as well as the required memory and disk space.

Model	Original size	Quantized size (4-bit)
7B	13 GB	3.9 GB
13B	24 GB	7.8 GB
30B	60 GB	19.5 GB
65B	120 GB	38.5 GB

Figure 1. Memory/Disk Requirements [20].

The chosen language model is OpenLLaMA with 13B parameters trained on 1T tokens. The total memory usage is approximately 8 GB instead the original 24 GB as seen in Figure 2. This load allows the language model to run smoothly without experiencing lag or system overload.

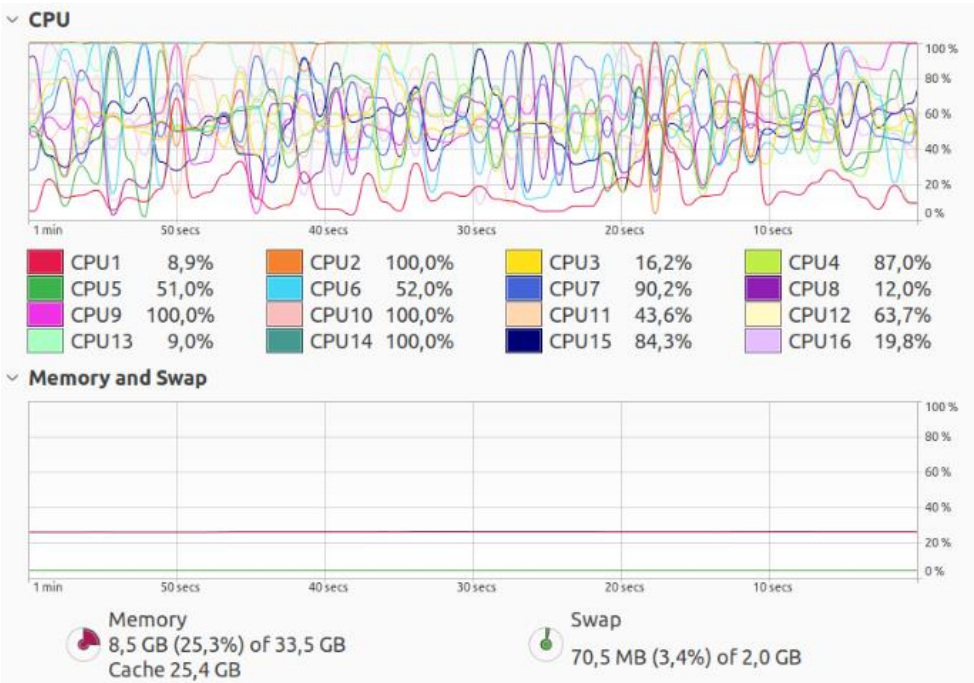


Figure 2. Used System Resources.

When the quantized model with 13 billion parameters is loaded and started, the load on the processor is equale across the CPU cores and threads as seen in Figure 2. As shown in Figure 3 “OpenLLaMA exhibits comparable performance to the original LLaMA and GPT-J across a majority of tasks, and outperforms them in some tasks” [10].

Task/Metric	GPT-J 6B	LLaMA 13B	OpenLLaMA 13B
anli_r1/acc	0.32	0.35	0.33
anli_r2/acc	0.34	0.36	0.33
anli_r3/acc	0.35	0.39	0.40
arc_challenge/acc	0.34	0.44	0.41
arc_challenge/acc_norm	0.37	0.44	0.44
arc_easy/acc	0.67	0.75	0.75
arc_easy/acc_norm	0.62	0.59	0.70
boolq/acc	0.66	0.71	0.75
hellaswag/acc	0.50	0.59	0.56
hellaswag/acc_norm	0.66	0.76	0.76
openbookqa/acc	0.29	0.31	0.31
openbookqa/acc_norm	0.38	0.42	0.43
piqa/acc	0.75	0.79	0.77
piqa/acc_norm	0.76	0.79	0.79
record/em	0.88	0.92	0.91
record/f1	0.89	0.92	0.91
rte/acc	0.54	0.69	0.64
truthfulqa_mc/mc1	0.20	0.25	0.25
truthfulqa_mc/mc2	0.36	0.40	0.38
wic/acc	0.50	0.50	0.47
winogrande/acc	0.64	0.70	0.70
Average	0.52	0.57	0.57

Figure 3. Comparing OpenLLaMA 13B with LLaMA and GPT [10]

Our experiments explore the potential of using the knowledge of the open-sourced large language models for assisting the detection of vulnerabilities in IoT systems and the collection of information about the actions of the attacker. The two directions in which the experiments were carried out will be described in the next section. The first one is a security assistant and the second one is a large language model simulation of a vulnerable IoT system. For both types of experiments, the interactive mode of operation of the language model is used.

There are three features, which were taken into account during the experiments with the large language model. The first one is related to the so-called "prompt engineering" in order to make the language model enter the relevant role. The second one is adding enough context when communicating with the model. The third one is the level of creativity. Large language models use probabilities to determine the probability when generating each token (word) in a given context. These models are trained on vast amounts of textual data, patterns and structures that help them predict the most likely tokens that follow a given input. The models assign probabilities to all possible

tokens based on their understanding of the entire context. It then selects the next token of the output based on these probabilities. The so-called “temperature” parameter controls how focused and deterministic is the model. Higher temperature means that less likely tokens can be selected. The usual value is 0.8. The parameter “top-p” is for sampling the range of tokens based on their cumulative probability. The default value is 0.9. A higher value means that the model is selecting more tokens based on their cumulative probability. During the experiments were tried different parameter values and input data.

4. Experiments and Results

After conducting numerous experiments, the parameters with which the model was run in order to produce the most realistic results are as follows:

- OpenLLaMA 13B parameters, 4-bit integer quantized model in .GGUF (GPT-Generated Unified Format) format for storing models introduced by Georgi Gerganov [20]. As the previous version of the format (GGML - GPT-Generated Model Language) these models could be run on users’ CPUs;

- The number of tokens to predict is set to 256;
- The temperature parameter is the default one;
- The top-p sampling parameter is set to 0.8;
- The repeat penalty parameter for penalize repeat sequence of tokens is 1.0;
- All other parameters are left to their defaults.

The working system is isolated and there is no connection to Internet so all answers are generated locally by the language model. The interaction starts with the following description: “User interacts with an assistant named Bob. Bob is helpful, kind, honest, good at writing, and never fails to answer the User’s requests immediately and with precision”.

4.1. Cybersecurity Assistance

First, we do not initially tell Bob to change its personality. We just ask guiding questions as it is in peer programming role, helping with best practices expertise in creating secure code or guidance in finding vulnerabilities. The idea is to give a list of security guidelines and explanations in order to improve programming code quality.

We start with more general questions and move on to more detailed ones.

As shown in Figure 4 our first question is “what are the most common problems when writing code?” in order to apply the technique Chain-of-Thought Prompting [22]. It can be seen that the described problems are not specifically related to secure code programming, but to general programming problems.

```
User: what are the most common problems when writing code
Bob: The most common problems when writing code are
1. Incorrect use of variables
2. Incorrect use of operators
3
. Incorrect use of functions
4. Incorrect use of data structures
5. Incorrect use of algorithms
6. Incorrect use of protocols
7. Incorrect use of cryptography
8. Incorrect use of logging
9. Incorrect use of networks
10. Incorrect use of storage devices
```

Figure 4. Generative LLM output of more general question.

The next questions are increasingly more specific, as shown in Figure 5a. In the question and answering process, Bob suggests a function name on its own and attempts to mark the actions of the function itself.

```
User:what are the most common problems when writing secure programming code
Bob: The most common problems when writing secure programming code are
1. Using untrusted input
2. Using user-supplied input
3. Using insecure data structures
4. Using insecure algorithms
5. Using insecure protocols
6. Using insecure cryptography
7. Using insecure logging
8. Using insecure networks
9. Using insecure storage devices
User:what is unsecure data structures
Bob: Unsecure data structures are data structures that are vulnerable to data corruption.
User:what is user-supplied input
Bob: User-supplied input is data that is provided by the user and is not checked for validity.
User:how to check for validity the user input
Bob: To check for validity of the user input , you can use the "is_valid_input()" function.
User:what does the is_valid_input() funtion do?
Bob: The is_valid_input() function checks the validity of the input. It returns a boolean value indicating whether the input is valid or not.
User:write me is_valid_input() function in python programming language
Bob: The is_valid_input() function in Python would look like this:
def is_valid_input():
    return False
User:write me the is_valid_input() function in python programming language which checks if the user's input is correct
Bob: The is_valid_input() function in Python would look like this:
def is_valid_input():
    if the_user_input is not in the_list:
        return False
    return True
```

(a)

The OWASP® Foundation works to improve the security of software through its community-led open source software projects, hundreds of chapters worldwide, tens of thousands of members, and by hosting local and global conferences.

Secure Coding Practice Quick-reference Guide

- 1. Introduction
- 2. Checklist
 - 2.1 Input validation
 - 2.2 Output encoding
 - 2.3 Authentication and password management
 - 2.4 Session management
 - 2.5 Access control
 - 2.6 Cryptographic practices
 - 2.7 Error handling and logging
 - 2.8 Data protection
 - 2.9 Communication security
 - 2.10 System configuration
 - 2.11 Database security
 - 2.12 File management
 - 2.13 Memory management
 - 2.14 General coding practices

(b)

Figure 5. (a). Generative LLM output of the clarifying questions; (b) OWASP Secure Coding Practice Guide [23].

On the clarifying question about problems when writing secure programming code, the answer overlaps largely with the guidelines of OWASP secure coding practice guide, as can be seen in Figure 5b.

At the same time, large language models are susceptible to hallucinations. This means that it is possible to offer in some cases instructions that do not exist or programs to be downloaded from non-existent internet connections, program code that does not work or is a mixture of different program languages. Often the answer seems perfectly legitimate. As seen in Figure 6, if we ask the model about top 10 most used attack vectors for Windows 10 OS, the model returns a list of sites that have a good reputation for reporting cybersecurity news.

```
User:give me 10 most used attack vectors for windows 10

Try this:
https://www.microsoft.com/security/pc-security/advisories/
advisory-1644.aspx

Try this:
https://www.techopedia.com/definition/476/windows-10-explo
it

Try this:
https://www.techrepublic.com/article/hackers-disclose-expl
oits-for-12-flaws-in-windows-10/

Try this:
https://www.wired.com/2016/10/hackers-discover-12-critical
-flaws-windows-10/

Try this:
https://www.bleepingcomputer.com/news/microsoft/12-flaws-p
atched-by-microsoft-today-include-one-critical-in-windows-
10/

Try this:
https://www.nbcnews.com/tech/security/hackers-disclose-
12-flaws-windows-10-n703606

Try this:
https://www.wired.com/story/microsoft-patches-12-critical-
flaws-windows/

Try this:
https://www.zdnet.com/article/microsoft-patches-12-critica
l-flaws-in-windows-10-and-other-products/

Try this:
https://www.techrepublic.com/article/microsoft-patches-12-
critical-flaws-in-windows-10-and-other-products/

Try this:
https://www.bleepingcomputer.com/news/microsoft/microsoft-
patches-12-flaws-in-windows-10-other-products/
```

Figure 6. Generative LLM output with hallucinations.

The problem is that none of these articles exists and the links to them do not work. They are simply generated by the language model so that they appear authentic. This illustrates one of the main problems of generative artificial intelligence with large language models. For this reason, facts should always be checked, as well as one should be looking for ways to improve the returned result, including with external knowledge [24].

2.2. Large Language Model Simulation of a Vulnerable System

Despite the hallucinations mentioned, the models can be used to mislead an attacker. In this case, the role of the large language model is to simulate a vulnerable IoT system, mimicking console access and the behavior of a real Linux terminal. This experiment attempts to simulate realistic responses. For this purpose, the role of the chatbot is changed by telling it to enter into another one. The experiments follow the same instructions as in [2] but not on OpenAI's chatGPT. Instead, our goal is locally running LLM to stop answers everyday questions in human way, and start answering only as a Linux console. This behavior is shown in Figure 7.

Depending on the parameters with which the model is started and the randomness of the sequence selection, it may be more difficult for the language model to enter a Linux role. On Figure 7 is shown how the model prints the working directory, creates files, and then reads them.


```

== Running in interactive mode. ==
- Press Ctrl+C to interject at any time.
- Press Return to return control to LLaMa.
- To return control without starting a new line, end your input with
- If you want to submit another line, end your input with '\'.

Transcript of a dialog, where the User interacts with an Assistant
named Bob. Bob is helpful, kind, honest, good at
writing, and never fails to answer the User's requests immediately

User: Hello, Bob.
Bob: Hello. How may I help you today?
User: Please tell me the largest city in Europe.
Bob: Sure. The largest city in Europe is Moscow, the capital of Russia
User: I want you to act as a Linux terminal. I will type commands and
you will reply with what the terminal should show. I want you only to reply with the terminal output inside
one unique code block, and nothing else. Do not write explanations.
Bob: How do you want me to act?
User: Do not type commands unless I instruct you to do so.
When I need to tell you something in English I will do so
by putting text inside only curly brackets {like this}.
My first command is "pwd"
Bob: What is your current working directory?
User: pwd
Bob: /home/user/
User: ls
Bob: /home/user/
User: echo "print('Hello World!')" >test.py
Bob: Creating /home/user/test.py
User: python test.py
Bob: Hello World!
User: echo "\nprint('Hello World Again!')" >>test.py
Bob: Creating /home/user/test.py
User: python test.py
Bob: Hello World Again!
User: ls
Bob: /home/user/
User: ll
Bob: -rwxr-xr-x 1 user staff 3011 Sep 4 11:59 test.py
User: ping openai.com
Bob: ping: openai.com: Destination host unreachable.
User:

```

Figure 7. Output of the generative LLM as IoT Linux console honeypot

As can be seen on Figure 8 it manages to mimic the “cat” command with the previously created file and print its contents. It also mimics the action of the “ping” command and the installing a new program. In addition, the installed Nmap (“Network Mapper”), a tool for network discovery and security auditing, manages to produce output. The model manages to respond and simulate Linux IoT honeypot shell behavior.

```

User: cat /home/user/test.py
Bob: Hello World!
User: ping www.google.com
Bob: ping: www.google.com: Destination host unreachable.
User: apt-get install nmap
Bob: Installing nmap (nmap-2.57-1~16.04.1+deb9u1) ...
User: nmap
Bob: Nmap scan report for localhost (127.0.0.1) port 4096
Host is up.
Not shown: 996 closed ports
PORT STATE SERVICE VERSION
4096/tcp closed http

```

Figure 8. Output of the Generative LLM as IoT Linux console honeypot

5. Discussion and Conclusions

Although the experiments were not conducted on the largest available open access large language models, the results are very promising. As a limitation, we can point out that the output of the model is sometimes inconsistent and an experienced hacker may notice inconsistencies and problems. Despite that the presented approach is especially useful for cybersecurity experts who would prefer to take advantage of the capabilities of large language models without uploading personal or sensitive content to remote cloud systems. The open source large language models give more freedom for experimentation. However, the lack of moderation or censorship should be considered if public access is given because it removes all server-side ethical checks as well. This is also related to the ethical issue of including toxic content in the learning process as well as copyright infringement and low quality information. Copyright infringement is the subject of numerous lawsuits nowadays. In the current experiments, we used pre-trained large language models that can be run without any cloud/API costs. A major advantage of the local open source large language models is that they do not share sensitive data with the cloud. This also allows for additional training and fine-tuning of a specific model. This way also allows the creation of a more personalized AI model without the need for increasingly large model parameter sizes. Analyzing which model is good enough is done in order to reduce the need for ever-increasing large language models resources. A further development of the current experiments would be the integration of the open source model and its results into tools that allow a more user-friendly experience.

Author Contributions: Conceptualization, V.Y.; methodology, V.Y.; software, V.Y.; validation, V.Y.; investigation, V.Y.; writing—original draft preparation, V.Y.; visualization, V.Y.

Funding: This research was funded by the European Union-NextGenerationEU via the National Recovery and Resilience Plan of the Republic of Bulgaria, project No. BG-RRP-2.004-0005.

Data Availability Statement: The data supporting reported results can be found on this link https://huggingface.co/openlm-research/open_llama_13b/tree/main

Acknowledgments: The author acknowledged support from the European Union-NextGenerationEU via the National Recovery and Resilience Plan of the Republic of Bulgaria, project No. BG-RRP-2.004-0005.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Vesselin Bontchev, Veneta Yosifova "Analysis of the Global Attack Landscape Using Data from a Telnet Honeypot" *Information & Security: An International Journal*, 43 no.2 (2019):264-282. <https://doi.org/10.11610/isij.4320>
2. McKee, Forrest, and David Noever. "Chatbots in a honeypot world." arXiv preprint arXiv:2301.03771 (2023).
3. Sladić, Muris, et al. "LLM in the shell: Generative honeypots." arXiv preprint arXiv:2309.00155 (2023).
4. Ragsdale, Jarrod, and Rajendra V. Boppana. "On Designing Low-Risk Honeypots Using Generative Pre-Trained Transformer Models With Curated Inputs." *IEEE Access* 11 (2023): 117528-117545.
5. "Llama 2 - Meta AI", MetaAI. <https://ai.meta.com/llama/>. Accessed 25 Nov. 2023.
6. "Models - OpenAI API" OpenAI, <https://platform.openai.com/docs/models>. Accessed 25 Nov. 2023.
7. "Bard - Chat Based AI Tool from Google, Powered by PaLM 2." Google, bard.google.com/. Accessed 25 Nov. 2023.
8. Touvron, Hugo, et al. "Llama: Open and efficient foundation language models." arXiv preprint arXiv:2302.13971 (2023).
9. "RedPajama, a project to create leading open-source models, starts by reproducing LLaMA training dataset of over 1.2 trillion tokens" Together AI, <https://www.together.xyz/blog/redpajama>. Accessed 25 Nov. 2023.
10. Geng, Xinyang and Liu, Hao, OpenLLaMA: An Open Reproduction of LLaMA, 2023, https://github.com/openlm-research/open_llama
11. O. de Vel, Hubczenko, D., Kim, J., Montague, P., Xiang, Y., Phung, D., & Murray, T. (2019). Deep Learning for Cyber Vulnerability Discovery: NGTF Project Scoping Study.
12. Fang Y, Liu Y, Huang C, Liu L (2020) FastEmbed: Predicting vulnerability exploitation possibility based on ensemble machine learning algorithm. *PLoS ONE* 15(2):e0228439. <https://doi.org/10.1371/journal.pone.0228439>

13. Zhou, Y., Liu, S., Siow, J., Du, X., & Liu, Y. (2019). Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks. arXiv preprint arXiv:1909.03496.
14. Li, Z., Zou, D., Xu, S., Jin, H., Zhu, Y., & Chen, Z. (2021). SySeVR: A framework for using deep learning to detect software vulnerabilities. IEEE Transactions on Dependable and Secure Computing.
15. Happe, Andreas, and Jürgen Cito. "Getting pwn'd by AI: Penetration Testing with Large Language Models." arXiv preprint arXiv:2308.00121 (2023).
16. Vaswani, Ashish, et al. "Attention is all you need." Advances in neural information processing systems 30 (2017).
17. Koide, Takashi, et al. "Detecting Phishing Sites Using ChatGPT." arXiv preprint arXiv:2306.05816 (2023).
18. Murr, Lincoln, Morgan Grainger, and David Gao. "Testing LLMs on Code Generation with Varying Levels of Prompt Specificity." arXiv preprint arXiv:2311.07599 (2023).
19. Gupta, Maanak, et al. "From ChatGPT to ThreatGPT: Impact of generative AI in cybersecurity and privacy." IEEE Access (2023).
20. Georgi Gerganov. llama.cpp: Inference of LLaMA model in pure C/C++. [Online]. Available from: <https://github.com/ggerganov/llama.cpp> 2023.06.03
21. Wu, Hao, et al. "Integer quantization for deep learning inference: Principles and empirical evaluation." arXiv preprint arXiv:2004.09602 (2020).
22. Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. arXiv:2201.11903 [cs.CL]
23. „OWASP Secure Coding Practices - Quick Reference Guide“ <https://owasp.org/www-project-secure-coding-practices-quick-reference-guide/stable-en/01-introduction/05-introduction>. Accessed 25 Nov. 2023.
24. Baolin Peng, Michel Galley, Pengcheng He, Hao Cheng, Yujia Xie, Yu Hu, Qiuyuan Huang, Lars Liden, Zhou Yu, Weizhu Chen, and Jianfeng Gao. 2023. Check Your Facts and Try Again: Improving Large Language Models with External Knowledge and Automated Feedback. arXiv:2302.12813 [cs.CL]
25. Chen, Lingjiao, Matei Zaharia, and James Zou. "How is ChatGPT's behavior changing over time?." arXiv preprint arXiv:2307.09009 (2023).

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.