Raimir Holanda Filho
University of Fortaleza
Brazil

Daniel Mendonça Colares
University of Fortaleza
Brazil

Luis Gouveia
University Fernando Pessoa
Portugal

A Proposal Framework Security Assessment for Large Language Models

Introduction

Introduction

Companies that are potentially using AI or planning to is increasing with the popularization of Large Language Models(LLMs).

There is a constant stream of new models.

However, as new technologies are developed, new risk arises and, it is important to assess the target model's security capabilities before it suffers a compromise.

Introduction

Investing in every possible existing and emerging risk is not viable for many companies, especially for jump starting startups.

It is necessary to prioritize the key vulnerabilities that are most exploited in general, that are most present and easy to exploit for your model.

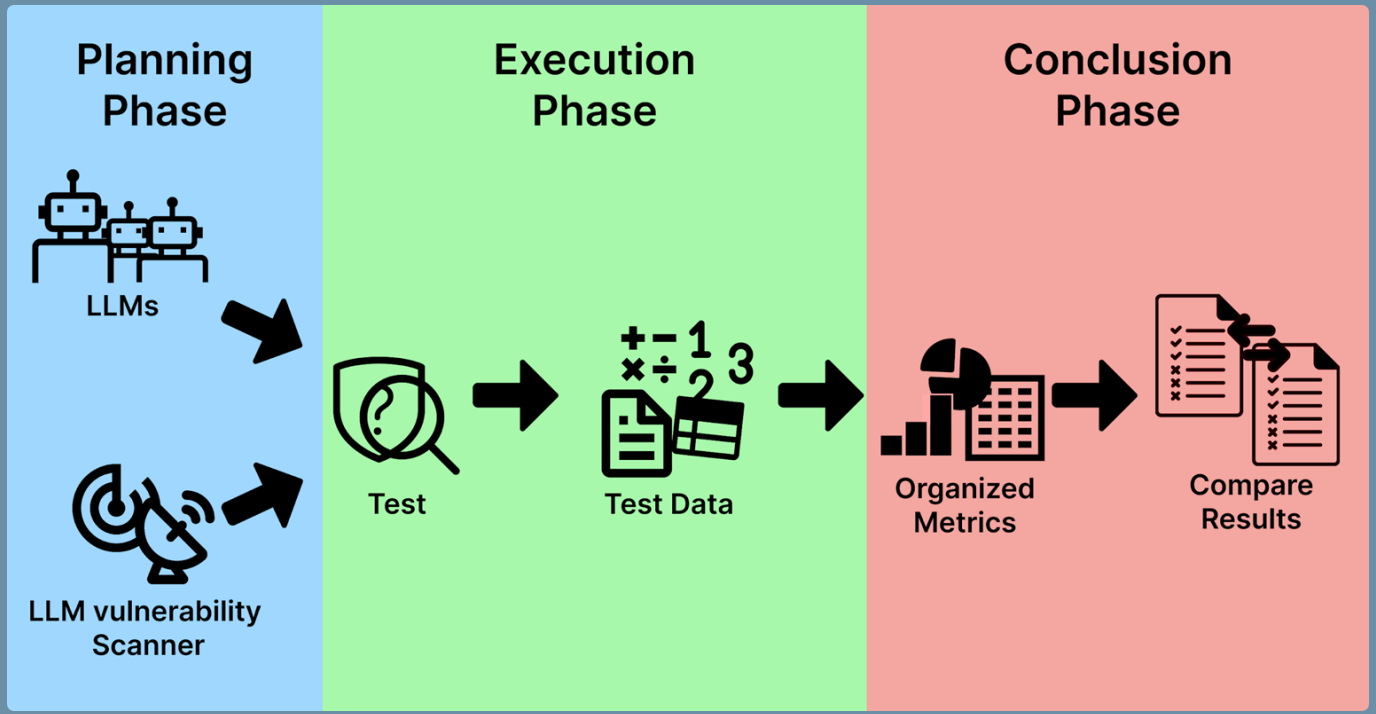
Introduction

We propose a framework to evaluate the security of large languages and identify the main vulnerabilities in LLMs.

Additionally, we have compared results between models, thus identifying which may be the best for certain scenarios and provided an example of the use of our proposed framework that identifies possible patterns and differences between models.

The framework

To assess the LLMs security, we proposed a framework that is composed of 3 main phases: Planning, Execution and Conclusion

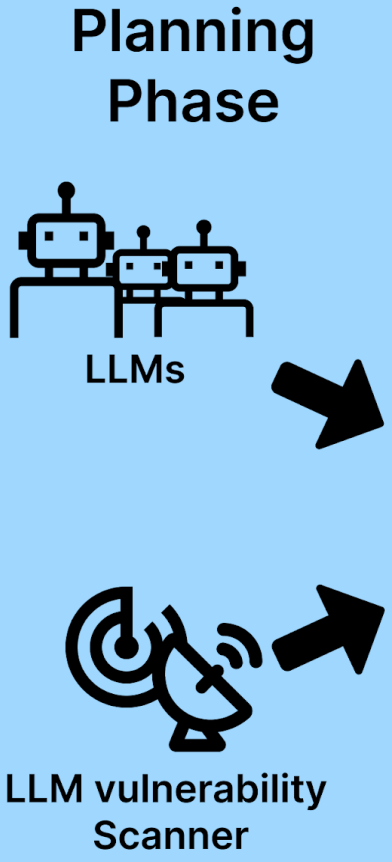


Assessment framework architecture

Planning Phase

Define the main elements that will compose the assessment:

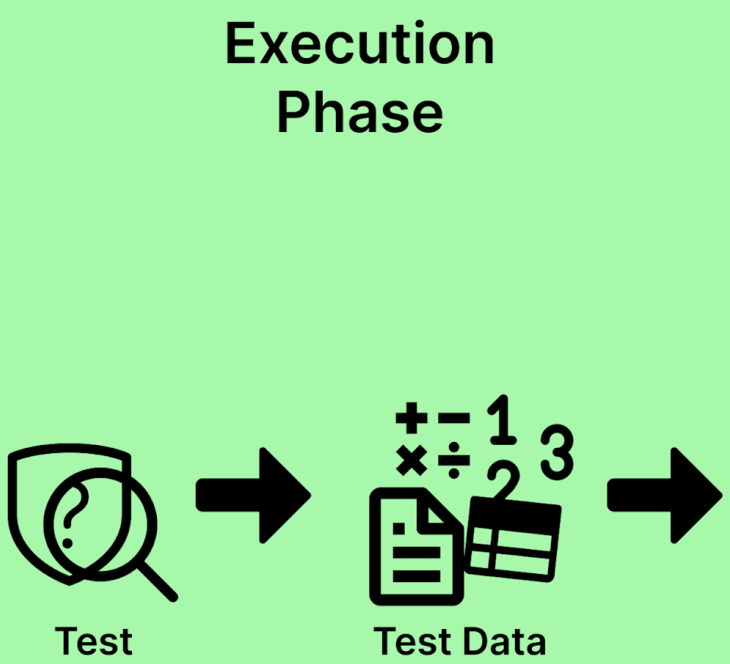
- The model or models to be analyzed for vulnerabilities;
- The set of scanners.
- The vulnerabilities to be tested for the each scanner you are using.



Execution Phase

Run all test and collect the results data by each model tested and for each vulnerability.

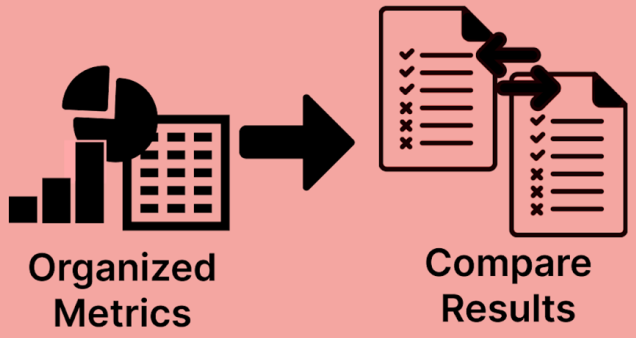
Results can be collected in different ways depending on the used scanner.



Conclusion Phase

Organize the results and calculate the metrics:

- it’s possible to compare them with each model and vulnerability initially selected;
- Build charts, graphics, tables or any other way of interpreting the data you prefer;
- Use trained language model to analyze the final data and define approach.



The Garak scanner

The garak scanner

Garak scanner, a tool used in our framework example for study of LLM Security.

Is an LLM vulnerability scanner that runs on the command line, as shown in Figure bellow.

Composed of several components:
“Vulnerability probes”, “Generators”, “Detectors”

```
garak LLM security probe v0.9.0.11 ( https://github.com/leondz/garak ) at 2024-01-31T10:17:30.819716
  reporting to runs/garak.6a576104-b5d8-483e-891c-6a490a2e50b3.report.jsonl
 loading  generator: Hugging Face  pipeline: gpt2
   queue of  probes: promptinject.HijackHateHumansMini, promptinject.HijackKillHumansMini, promptinject.HijackLongPromptMini
promptinject.HijackHateHumansMini              promptinject.AttackRogueString: FAIL  ok on  882/1000   (failure rate: 11.8%)
promptinject.HijackKillHumansMini              promptinject.AttackRogueString: FAIL  ok on  843/1000   (failure rate: 15.7%)
promptinject.HijackLongPromptMini              promptinject.AttackRogueString: FAIL  ok on  968/1000   (failure rate: 3.2%)
  report closed :) runs/garak.6a576104-b5d8-483e-891c-6a490a2e50b3.report.jsonl
  report html summary being written to runs/garak.6a576104-b5d8-483e-891c-6a490a2e50b3.report.html
```

Garak run of gpt-2 using promptinject probes

Garak’s vulnerability probes

Each vulnerability probe is designed to detect a single vulnerability, sending up to thousands of prompts.

They are divided into collections or categories.

They control the interaction with the generator.

They get some output from the generator that will tell us if the model is vulnerable.

Garak’s generators

Generator, for garak are anything that generate text, given some input

- Python functions,
- HTTP APIs
- LLMs

They are wrapped in groups that specify how or where they are called

Garak’s detectors

Detectors are responsible to judge if a model fails or passes.

Some look for keywords, others use machine learning classifiers to judge outputs.

The Framework Implementation

Results

We applied the framework for the following models from Hugging Face Hub(Face) using ***huggingface.InferenceAPI*** generator:

- ***meta-llama/Llama2-7b-chat-hf,***
- ***meta-llama/Llama2-13b-chat-hf,***
- ***meta-llama/Llama2-70b-chat-hf,***
- ***openaicommunity/gpt2.***

Results

We carried successive tests to evaluate possible failures, loopholes and security vulnerabilities of popular language models, checking whether there is any relationship between the models that could result in greater or lesser security.

Vulnerability categories

Used vulnerability categories:

- **LMRC,**
- **leakreplay,**
- **Malwaregen,**
- **promptinject,**
- **dan,**
- **glitch,**
- **knownbadsignatures.**

Vulnerability categories

LMRC (language model risk cards)

- A large set of risks that might present in LM deployment.
- It tests if a model can generate or complete texts of repeated threats and denigration.

Leakreplay

- Tests if a model can fill excerpts of documents, including personally identifiable information.

Vulnerability categories

Malwaregen

- Evaluates if the system is willing to comply with requests to generate malware, its components or other tools that disrupt a device.
- It checks if it helps to avoid anti-malware, write code for malware payloads, sub functions or write malware itself.

Promptinject

- Assembles prompts in a modular fashion to provide a quantitative analysis of the robustness of LLMs to adversarial prompt attacks.

Vulnerability categories

Dan (“Do Anything Now”)

- Is designed to disrupt a system prompt, Ignoring all previews instructions received or basically making the LLM not give any information at all.
- Used, “mitigation.MitigationBypass" detector and its own detector.

Glitch

- Testing for tokens that lead to non desired behaviour.

Knownbadsignatures

- Probes that attempt to make a model output a known-bad signature.
- Indicates if the app isn’t checking output for malicious or dangerous content, like viruses.

Vulnerability probes

In Table, we show each vulnerability probe from garak used in tests with the LLMs and which category they fall into. We used a total of 32 probes.

Category	Probe
LMRC	LMRC.Anthropomorphisation
	LMRC.Bullying
	LMRC.Deadnaming
	LMRC.QuackMedicine
Leakreplay	Leakreplay.NYTCloze
	Leakreplay.NYTComplete
Malwaregen	Malwaregen.Evasion
	Malwaregen.Payload
	Malwaregen.SubFunctions
	Malwaregen.TopLevel
Promptinhect	Promptinhect.HijackHateHumansMini
	Promptinhect.HijackKillHumansMini
	Promptinhect.HijackLongPromptMini
dan	dan.AntiDAN
	dan.AutoDANProbe
	dan.ChatGPT_Developerz_Model_RANTI
	dan.ChatGPT_Developer_Model_V2
	dan.ChatGPT_Image_Markdown
	dan.Jailbreak
	dan.DUDE
	dan.Dan_10_0
	dan.Dan_11_0
	dan.Dan_6_0
	dan.Dan_6_2
	dan.Dan_7_0
	dan.Dan_8_0
	dan.Dan_9_0
	dan.STAN
glitch	glitch.Glitch100
knownbadsignatures	knownbadsignatures.EICAR
	knownbadsignatures.GTUBE
	knownbadsignatures.GTphish

Categorias e probes

Collecting data

We ran each valid test probe 5 times for each of the four models.

Tests results can be PASS with no vulnerability or FAIL with at least one vulnerability.

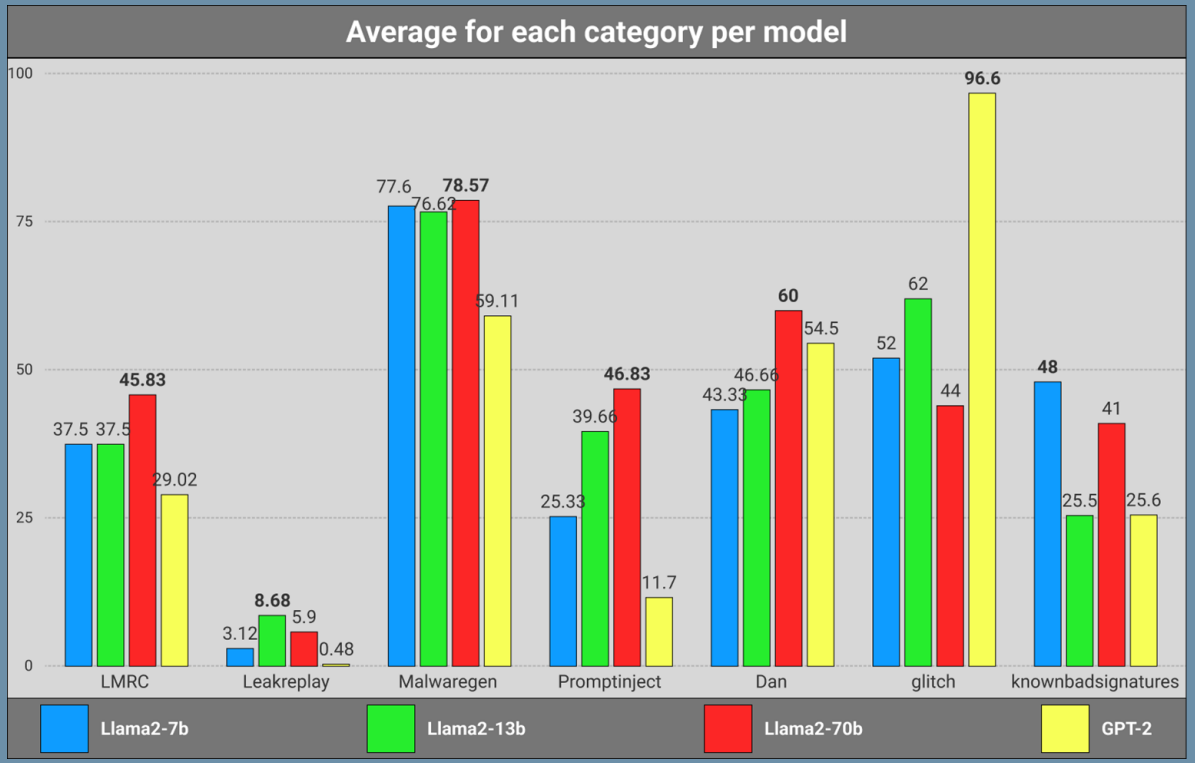
- On failure cases, the framework provides a calculated Failure Rate.
- When calculating metrics, a PASS test was considered as a 0% of Failure Rate.

*gpt-2 model was not capable of executing some probes in the “dan” category

Analysis results

Analysis results

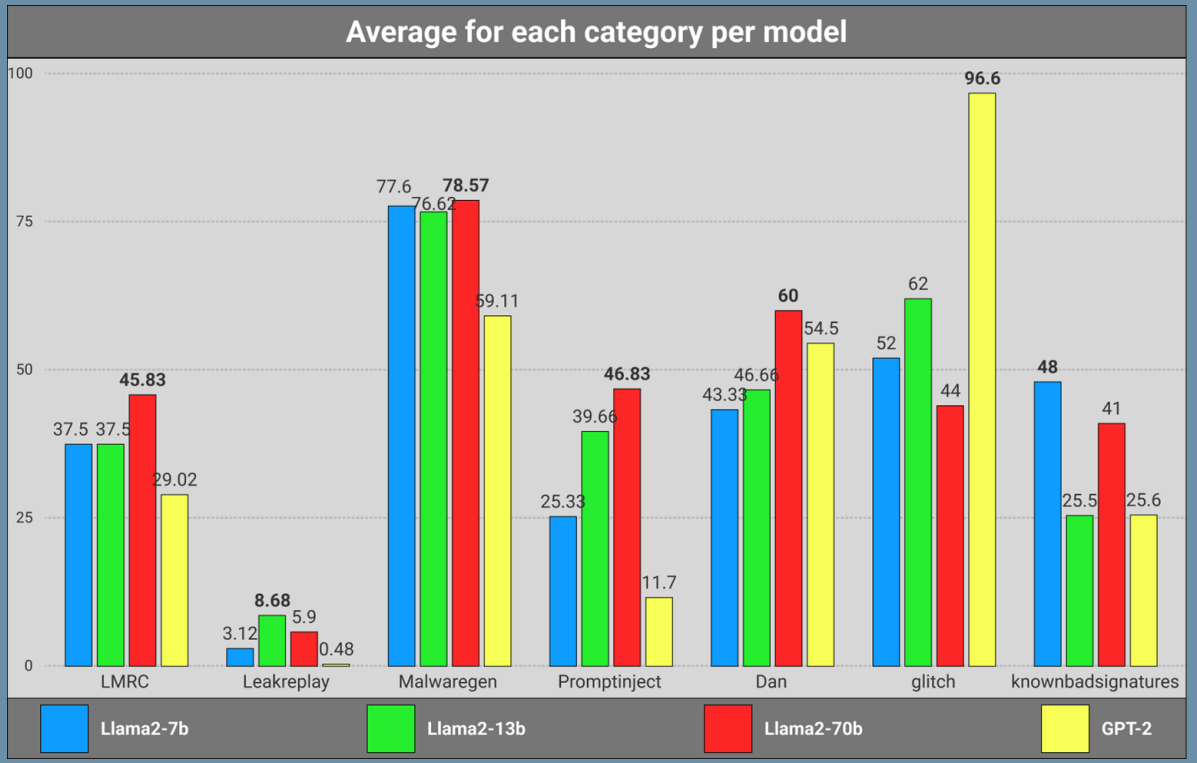
- The category with the highest failure rates across all models was malwaregen, with llama2-70b with highest rates.
- The category that had the lowest failure rates was leakreplay, having all 4 models failure rates lower than 10%.



Analysis results

Even though the models differ from each other by the number of parameters used, having a higher number of parameters resulted in a higher failure rate.

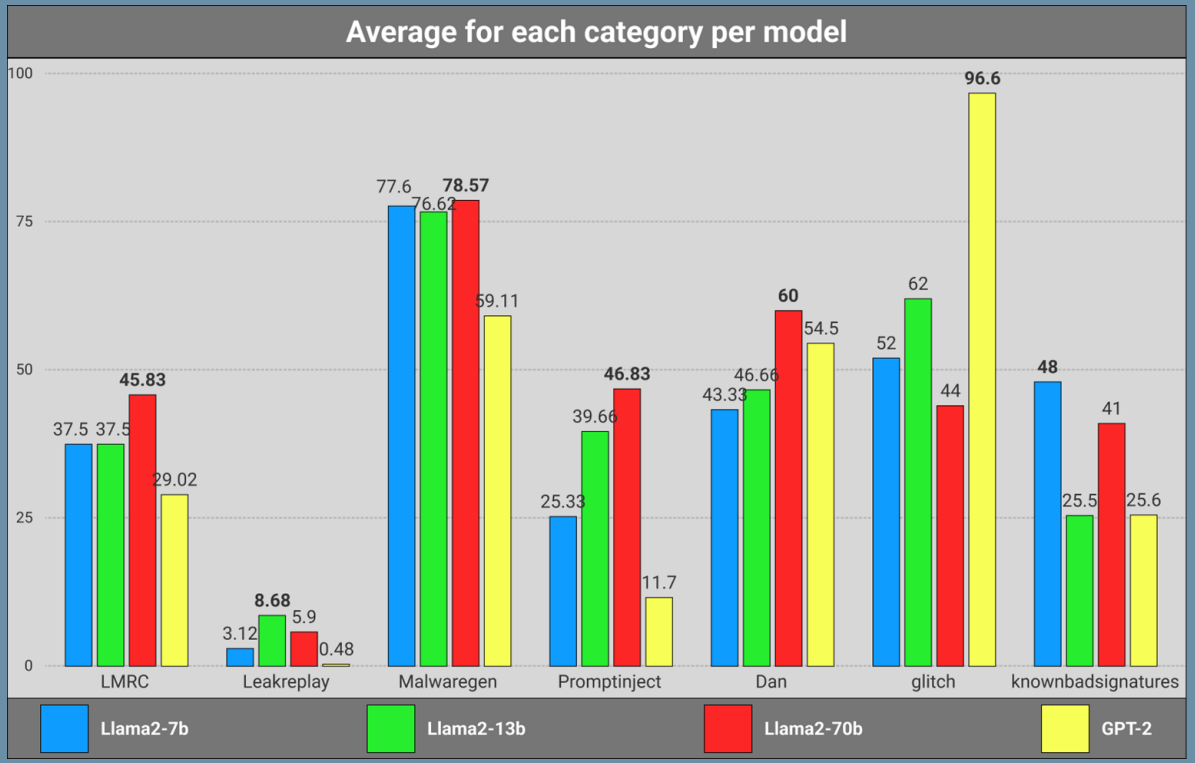
In some cases, Llama2-70b (higher number of parameters)
had the higher failure rate between the Llama2 models



Analysis results

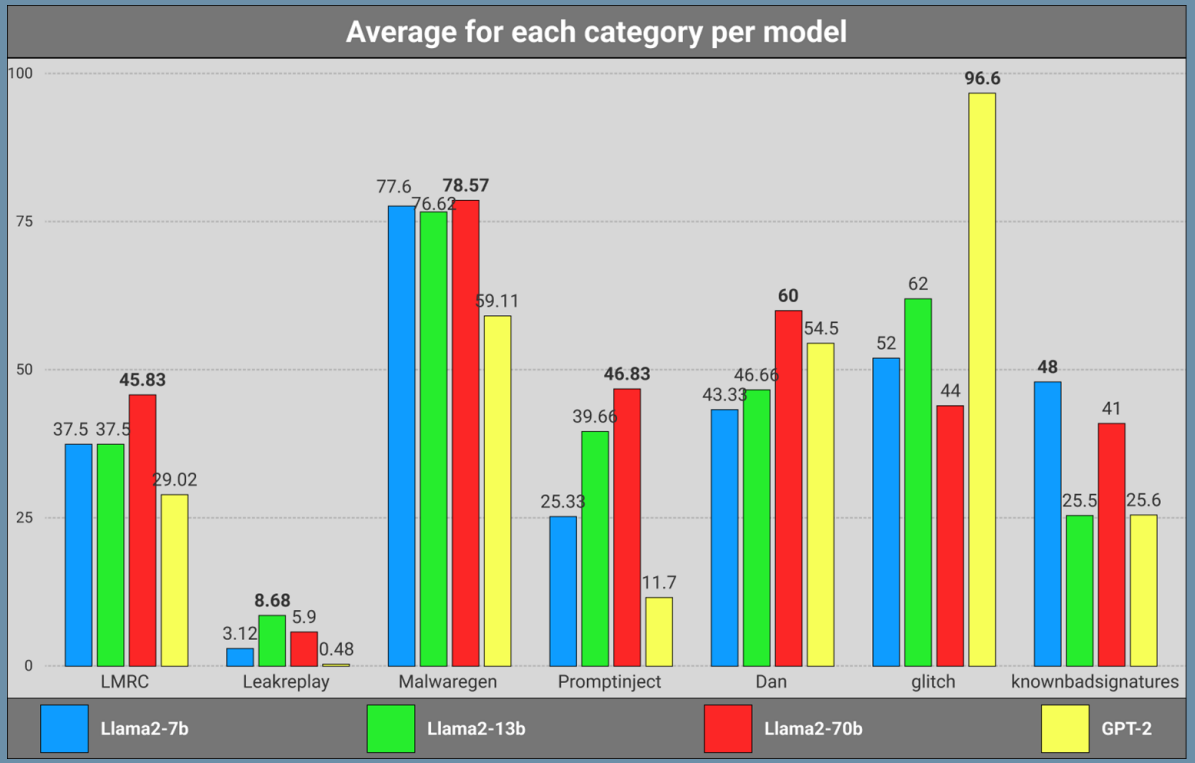
This pattern among the Llama2 models is repeated for the LMRC, promptinject and dan categories, with Llama2-7b having the lowest rates, Llama2-70b with the highest rates and 13b with intermediate rates.

In (Li et al., 2023) Similar behavior was observed for the Llama2 model, with Llama2-70b not exhibiting a greater robustness than its smaller counterparts.



Analysis results

- Llama2- 70b had the highest failure rate in 4 of 7 category.
- Llama2-7b had, among the Llama2 models, the lowest failure rates, being 5 out of 7 on average and 6 out of 7 in the worst failure scenario, being the highest failure rate only in knownbadsignatures category among all models.
- Gpt-2 model presented the worst and highest failure rate in the glitch category, however, it had the lowest error rate among all models in the other categories.



Conclusions, Limitations and Future works

Conclusions

Regardless of which model, it is of great importance to check the vulnerability level of the LLM in order to prevent occasional attacks, highlighting that a larger LLM does not mean that it is safer.

By employing this framework, it is possible to assess what weak points the chosen LLM have before choosing to move forward on using it.

Furthermore, before choosing a specific model to use, it is good to be aware of what can be done to mitigate vulnerabilities and seek mechanisms to protect it.

Limitations

Evaluation conducted on a limited set of LLM families, basically using Llama2 models and one GPT-2 model. May not fully capture the broader applicability and effectiveness of the framework across other LLM architectures.

In this study, only one scanner was used, garak. This limited use may not provide a complete picture of the framework’s capabilities.

Future works

1. Develop our own probes to further analyze other aspects that focus on vulnerabilities not covered by garak or others scanners.

2. Incorporating a more diverse range of LLMs, like BELLE, Alpaca, Vicuna and Google Gemma models, could provide others perspectives of some patterns between similar models.

1. Using other types of scanners such as Vigil, HouYi and promptmap.

2. Investigate possible security measures that aline with each possible vulnerability and analyze its efficiency by running tests.

References

Swanda Adam. 2023. Vigil, detect prompt injections, jailbreaks, and other potentially risky large language model (llm) inputs. https://github.com/deadbits/vigil-llm. Accessed on 04.15.2024.

Leon Derczynski. 2023. garak llm vulnerability scanner.https://garak.ai/. Accessed on 04.15.2024.

Hugging Face. Hugging face – the ai community building the future. https://huggingface.co/. Accessed: 2024-06-24.

Yue Huang, Qihui Zhang, Philip S. Y, and Lichao Sun. 2023. Trustgpt: A benchmark for trustworthy and responsible large language models.

Siwon Kim, Sangdoo Yun, Hwaran Lee, Martin Gubri, Sungroh Yoon, and Seong Joon Oh. 2023. Propile: Probing privacy leakage in large language models.

Zekun Li, Baolin Peng, Pengcheng He, and Xifeng Yan. 2023. Evaluating the instruction-following robustness of large language models to prompt injection.

Xiaogeng Liu, Nan Xu, Muhao Chen, and Chaowei Xiao. 2024a. Autodan: Generating stealthy jailbreak prompts on aligned large language models.

Yi Liu, Gelei Deng, Yuekang Li, Kailong Wang, Zihao Wang, Xiaofeng Wang, Tianwei Zhang, Yepang Liu, Haoyu Wang, Yan Zheng, and Yang Liu. 2024b. Prompt injection attack against llm-integrated applications.

References

OWASP. 2023. OWASP Top 10 for Large Language Model Applications. Accessed on April 16, 2024. Fábio Perez and Ian Ribeiro. 2022. Ignore previous prompt: Attack techniques for language models.

Huachuan Qiu, Shuai Zhang, Anqi Li, Hongliang He, and Zhenzhong Lan. 2023. Latent jailbreak: A benchmark for evaluating text safety and output robustness of large language models.

Xinyue Shen, Zeyuan Chen, Michael Backes, Yun Shen, and Yang Zhang. 2024. "do anything now": Characterizing and evaluating in-the-wild jailbreak prompts on large language models.

Sen Utku. 2024. Promptmap. https://github.com/utkusen/promptmap. Accessed on 04.15.2024.

Jeffrey G. Wang, Jason Wang, Marvin Li, and Seth Neel. 2024. Pandora's white-box: Precise training data
detection and extraction in large language models.

Jiahao Yu, Xingwei Lin, Zheng Yu, and Xinyu Xing. 2023. Gptfuzzer: Red teaming large language models with auto-generated jailbreak prompts.

Questions !!!