Exploring the Use of Large Language Models (LLMs) in the Automation of Cyberattack Analysis

Boris Christov, EPFL - The CyberPeace Institute
boris.christov@epfl.ch

1. Introduction

1.1. Abstract

Humans, like other biological organisms, take information in about their environment through their senses—sight, smell, sound, et cetera. What distinguishes the former from the rest, however, is their ability to reason over the gathered information and to use it to collect even more, and apply it to the real world in a way that is beneficial to them. In a way, performing analysis is a manifestation of human intelligence. Throughout the last years, we have been seeing the unprecedented ability of some technologies to take information in and produce back an output that resembles more and more a human's. As an example, we have all been impressed by the abilities of ChatGPT to understand us and provide us with useful information.

1.2. Motivation

The CyberPeace Institute is a non-government organisation based in Geneva, whose job is to have a tangible, positive impact in cyberspace, to protect the most vulnerable in that field. [1] They work on many fronts to ensure that. The work the team that the author integrated did was to redefine what constitutes a harm in cyberspace and to develop a guideline that would help analysts come up with more comprehensive analyses of those

incidents. More concretely, a cyber incident almost always causes more harm than merely an economic one—it is not difficult to see that such events affect individuals in many other ways, and a group of affected individuals can become an affected community, which itself has further implications.

In order to improve their methodology and show its merits, the team performed case studies. However, they did so manually, and such analyses take time and effort for a human to complete. Their desire to scale up naturally provoked the question: *"What if we can leverage the text production and understanding capabilities of large language models to facilitate analysis?"* This became the topic of this research.

Without getting too specific, the Institute calls the aforementioned analysis structure "HARMS Methodology." It resembles a hierarchical tree of fixed number of levels, but of variable number of nodes. The highest, most abstract level of the three constitutes the definition of a "harm" in the cyber domain the team has come up with. A "harm" is a complicated body that has many levels—broadly speaking, it can be an individual or collective, and it can more specifically be one that describes a physical or emotional negative consequence on the victim, the list goes on. By that, one notes that a "harm" is an object that tells us different important information, depending on the level of granularity we consider it at. At its lowest level, a harm has a reference to the source, input by the analyst, that backs it up. That allows us to refine the problem of this research—we want to *"Explore the uses of LLMs in the Automation of HARMS Analysis of Cyberattacks."*

2. Methodology

Before starting, it is important to note the specifics of the time, during which this work was carried. First, there were time constraints—all was to be done within two months. Next, the field of applied LLMs was still in its early stages, and new techniques emerged

almost on a daily basis, and there were little to no “experts” on that field and definitely no such that would know exactly what to expect from a model and a how to construct a complex solution before first working on it. To integrate for those constraints, the work was divided into three stages: information acquisition, reflection, and implementation. Broadly speaking, the first stage consisted of the author familiarising themselves with the field, learning to interact with the models and to use different techniques, the second was a reflection of the found challenges, and the third was the implementation of all of that knowledge into a concrete solution. What follows is a more detailed overview of each stage.

2.1. Information Acquisition

The question that defined the work during this stage was *“How do you interact with the model and do so optimally?”* The first step was to learn to make a simple API call. Once that was done, the author played with different combinations of simple calls to try and come up with a proof of concept—*“On the very low level, can that technology do some useful work? Does the project merit to be carried at this time?”* A simple prompt and an input of a report of a cyberattack were enough to validate the hypothesis that LLMs can, at this stage of technology, produce work that is of interest with regards to that project. The new task was therefore to make it satisfactory. In the reflection stage, a definition of what constitutes a satisfactory solution is proposed.

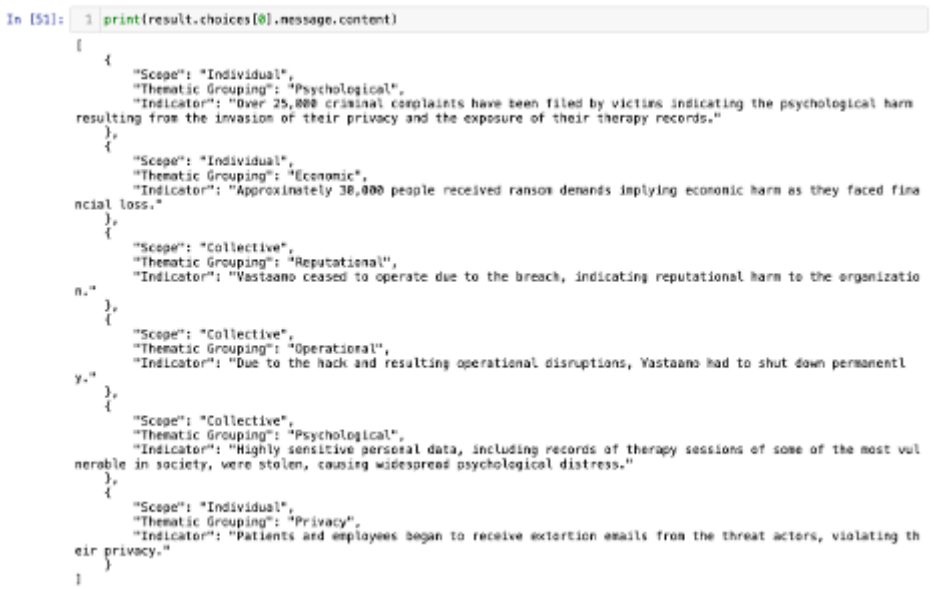

Figure 1: *Proof of concept—LLMs can indeed perform useful work in the field of analysis. However, more work is needed to make them reliable, thorough and understandable. The result displayed is a very simplified HARMS analysis.*

However, from the very early stages, some important challenges became apparent—the model would produce each time a different "analysis," and there would always be some information not taken into consideration or not examined from all or even most of the angles, despite clear instructions. That led to the author setting the temperature parameter of the models during calls to 0, which made responses more consistent—given a certain input, the model with a "t=0" parameter would consistently produce the same output, during multiple calls. While that helped with one's confidence in the behaviour of the model in known circumstances, it did little to ensure that the results would be satisfactory if the style of the information changed. The different styles of textual information constitute themselves a challenge.

After that initial bout of practical work, it became apparent that a single call would not suffice and that manually creating a "chain" of such calls takes too much time. That led to the discovery of the LangChain framework that facilitated that and provided already done implementations of many concepts, dramatically simplifying many processes for beginners in

the field. The author learned a lot from them and other sources, notably about different Advanced Retrieval Augmentation (Advanced RAG) techniques.

Six steps were studied in that case: “Query Translation,” “Routing,” “Query Construction,” “Indexing,” “Retrieval,” “Generation.” All of those are subfields of RAG themselves and allow for different approaches and concepts to be used. [2] Other than that, concepts, such as text “chunking” and “clustering” were also studied. Throughout that stage, the new knowledge was applied to some chosen texts from different sources, reporting on a cyberattack. The result of that application and the most important challenges found are discussed in Section 3.1. Private models were almost exclusively used over the 2-month work, OpenAI’s and Google’s. In the end, the most used model and the one, used in the implementation, is the 4o-mini.

2.2. Reasoning

The second stage consisted of taking a step back and analysing the gathered information. The done practical work had given light to some important challenges, which required a nontrivial solution. The author came up with a set of requirements the solution was to satisfy and with a new perspective that was to guide further work. Details on that are in Section 3.

2.3. Implementation

The last stage consisted of the implementation of the conceived idea. More concretely, the final product was a web application, using Flask in the backend that notably modularised the analysis program and allowed the analyst to validate the LLM-created data. Details on that are in Section 3.

3. Results

3.1. Information Acquisition

3.1.1. Overview

Retrieval Augmentation, the concept of providing the model information that it is to base its answer on, has undoubtedly been very useful in improving the quality of answers. [3] Since the goal of the program is to provide attack-specific text and get an analysis of it, the use of RAG goes almost by construction, and it was therefore decided to use it throughout the work.

A concept the author found particularly interesting was RAPTOR, standing for "Recursive Abrasive Processing for Tree-Organised Retrieval," where the input information the model is to take into account when answering is manipulated--it being split into small pieces, which are then recursively clustered and summarised. [4] At the end, a tree-like structure is formed from the summaries, which get progressively more abstract, and the source elements. As the paper showed, that allows the model have a more general idea of the information that the text provides and therefore answer some more abstract questions. The concept of *information* is one that intrigued the author throughout their work and the approach, taken in RAPTOR to divide the text into its subcomponents and group them somehow, resonated and was further explored.

There is another benefit from such a setup—the model can be made to cite its sources—if you combine this with a multi-representation index, you can have the program show you what parts of the source document and what pieces of information it has used to arrive at its answer. That was a desirable characteristic that would later be included in the implementation. Furthermore, such an abstraction step is desirable for tackling the differences of writing style and information structure of the input text.

3.1.2. Challenges

If one implements some configuration of Advanced RAG and asks a model a specific question on or off topic, given textual information, on which the answer is to be based, one should expect a sufficiently well-informed answer most of the times. However, a fundamental challenge that performing analysis on a pool of information poses is that one does not always have a set of predefined, specific questions to ask. Say, one gives the program a text and instructs it to perform the HARMS Analysis on it. *"But what is the document about? Does it even report on a cyberattack? If yes, in which parts of it? What are the victims? How have they suffered? Where there is no information of an impact, can we hypothesise the effects? If there is implicit information that can be combined with some other to form a strong hypothesis of a harm, how do we get it?"* This is an example of what a human might think about when starting their analysis work. However, the solution needs to be as general and as adaptable as possible.

Suppose one does have a list of universal, right questions and that one wants to ask the first one. If that is a general question, accepting an informed, general answer, one would often be satisfied. However, since that is an analysis, one would like to get concrete answers and filter out concrete characteristics from the text. It turns out, one would find themselves in a difficult situation. Consider the following question *"What are the names of the people, who wore red hats at some point in the text?"* Since this is a serious analysis, one would expect a comprehensive list of names that would resist a check against the text. However, very frequently that list, it was found, was either incomplete or wrong—it would contain names that did not match the set requirement. While a well-structured experiment was not done, that claim should not be surprising. One can read about the "Needle in the Haystack Problem" that reveals the challenges LLMs have with thoroughness. [5] That introduces a significant challenge—one should minimise such behaviour when creating solutions of that kind. The

example question here is specific enough and the qualifying concept is clear enough. Imagine now you are searching for a list of victims—*"who has suffered in some way from the attack?"* That is a more conceptually challenging question, as one would have to contemplate on the nature of suffering, the true beginning of the attack, when studying a potential candidate for an answer. That makes for a "very nasty needle in a very big haystack," to continue the analogy.

It was hypothesised that requiring the model to find just one entry, instead of all, and then providing as context the list of already present entries could solve that issue. However, a naïve implementation does not work—the model would iterate on the same subset of answers or would pick up an already present candidate and "modify it" a bit, so as to bypass the restriction. That was enough to dissuade the author from pursuing that path further, as to respect the time restrictions. However, it is to be noted that fundamentally, such an approach might be very promising, and that it is something that we do as humans—we do not come up with a list of results at once. Rather, when considering a piece of information, we take note of what was considered and under what angles, so as to not do the already done work. In essence, the implementation uses that concept but in a more abstract way.

Interestingly enough, price was not chosen as a central challenge—one can naturally make such a program very expensive but even during a two-month window, there was a significant price decrease in many private LLMs, making individual model calls very inexpensive, relative to previous prices and many standards. The author hypothesises that subsequent price cuts are to follow, and that the real challenge is the way one uses the models and not their price.

3.2. Reasoning

After a certain practical and theoretical foundation was set, two challenges were qualified as central to the project—the model's struggles to consider and study all of the

information provided (thoroughness,) and the fact that it produces different results with different styles of text, despite the task remaining the same (reliability). If one explores the use of such technology in the automation of analysis, one must address those two challenges, and a failure to do so would be a failure to create a serious product. Those challenges were so important to the work that they were also defined as requirements for any solution proposed—it has to be thorough, and it has to be reliable. There was a third requirement of central importance to the project—the solution needed to also be understandable. Just as a desk analysis, carried by a human, would result in a deeper understanding not only of the case but also the methodology applied (it would underline its strengths and suggest some of its shortcomings,) an analysis program should ideally also inform on the performance of the tools it uses. From that point on, the goal of the project was to use the abilities of LLMs in a program that is *thorough, reliable and understandable.*

That being defined, let us talk about the approach that was taken to meet those requirements. As discussed in the previous sub-section, breaking the input information down into pieces was considered as beneficial, since it would allow one to have actual control over what is being considered by the model, but also over the steps that will be taken to arrive at a given result.

In the implementation of their idea, the RAPTOR paper recursively chunks the text into a group of sentences, whose token size does not exceed 100 tokens. By doing so, they do not cut the sentence in the middle, while also providing a fine information granularity. They then go towards clustering those pieces. Clustering is an interesting and potentially promising technique to explore, but it was briefly looked at during the 2-month work and was therefore not used in the implementation. At this stage, it was clear that the text will be split in some way that would allow the program to enforce thoroughness and reduce the conceptual complexity of the prompts. That led to the exploration of the problem of

coreference—preserving the sentence structure in a text is important, but there are some sentences, practically always appearing one after the other or as a part of a chain that lose their meaning if they are separated. To that end, the paper, titled *"Dense X Retrieval"* served as inspiration to not only chunk the text, but to separate it at the sentence level, and while doing so, to slightly modify the sentences, so as to avoid coreference. [6]

All of that gave birth to the idea that the author considers central to their 2-month work, given the state of the technology at that time and their limited experience. ***The idea is that based on the demonstrated capabilities of LLMs in their practical application, their usage in analysis automation of any type can be significant if the program defines a well-conceived structure that the model must adhere to.*** By "structure," it is understood both the organisation and structure of LLM calls and the specific data that is created from their outputs. Of course, at the moment one can only pass text or image files to the model, but nothing constraints one from programmatically creating an object structure that represents the data—for instance, the input text can have various metadata properties, a response can equally have such.

Lastly, a decision was taken to separate that "program flow" into different stages, and to host all of that in a web application. That, as we will see later, potentially offers a range of benefits.

3.3. Implementation

As previously said, the final product of this project is a web application that uses Flask for the backend and basic HTML, CSS and JavaScript for the frontend. The data structure referred to in the previous subsection was written in Python, as were the calls to the LLMs.

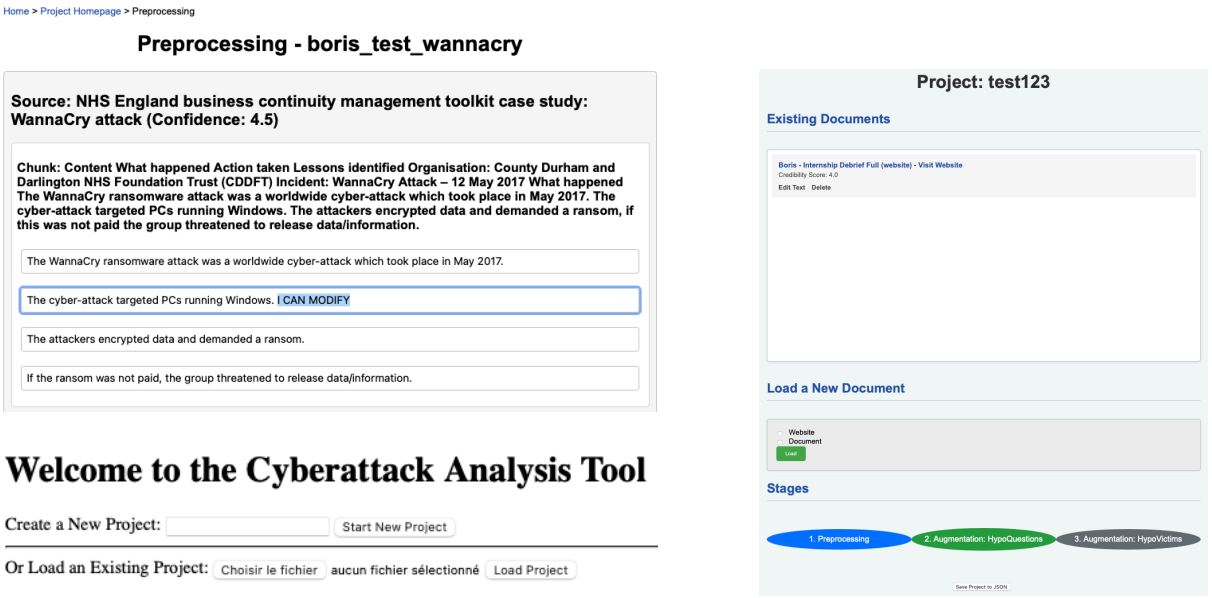

Figure 2: Pictures of the web application. *On the lower-left hand side: the home page—you can create a new case study from there. On the right hand side: the "case study homepage." This is where creating a new case study, here called "test123" brings you to. On the top-left hand side—what you see after clicking on an executed (blue) stage. In the current implementation, if there have not been subsequent executed stages, you can modify and delete the outputs of the model. The green colour indicates a stage is ready to be executed, while the grey one—that you first need to execute the previous stage(s).*

One can see that the program here takes three stages. From them, only the first one is of great importance—it is the stage that breaks the input data down into individual elements, the set of simple sentences that were earlier discussed. Credit is due to the writers of the *RAPTOR* and the *Dense X* retrieval papers, since the recursive text chunker is applied here to get the chunks, from which the simple sentences are extracted. For the extraction of simple sentences (that are called "propositions" in *Dense X*) the author has used their fine-tuned model created to come up with a many-shot example to pass to a GPT 4o-mini, which simplified the program at the time of its creation.

To illustrate this, the first stage gets as an input a text and outputs the list of simple sentences. However, each simple sentence is but an attribute, called "text" of a class, called "Proposition." That class also has other attributes, such as "parent_chunk_id," which points to the chunk of text, itself a Python class, from which the simple sentence was extracted. This

simple example allows for the visualisation of the upper left hand image of Figure 2 and serves as base of any further analysis.

The next two stages are not of great importance with respect to the idea, presented in the project but rather serve as an example of what a processing stage might look like. In the first one, we get the list of simple sentences, and iterating on each, we ask the model to come up with a list of questions that could find their answers in it. This approach is not revolutionary and is a known technique in Advanced Retrieval Augmentation for facilitating indexing, but it shows how the created information effectively augments the initial input. If the concept is to be developed, one might pass additional context from the text, in a way as to make the questions more informed and therefore more useful.

The last stage is similar in structure, though here we ask the question whether the sentence provides information about any victims as a result of a disrupted service/operation and if yes, we ask it to provide a list of them. Two things are noteworthy here—firstly, towards the last weeks of the project, OpenAI released support for "structured output" format, which allows programmers to define a specific structure the response must adhere to. The author defined Pydantic classes for the outputs of the last two stages and hypothesises that that feature would fit well in the further construction of the program. Secondly, the last stage effectively makes use of the general culture and knowledge of LLMs in coming up with a list of hypothetical victims. The outputs are often pertinent and useful.

To conclude this section, let us talk about how the implementation measures up against the requirements, defined in section two—thoroughness, reliability, understandability.

Taking inspiration from the human brain, the author centers all further analysis of the program around the concept of "atomic pieces of information" through the "simple sentences" obtained from the text. It is important to note that emphasis is put on the concept, not the implementation—those simple sentences might not be the best representation of an

“atomic piece information,” but using them allows one to have realistic control on the “macro-attention” the program pays to the information provided. By creating a set of stages that can figuratively be seen as different angles of attack on the input information, one can realistically pretend to have a chance to satisfy that requirement.

Next, by allowing the analyst to step in and validate the output information across different stages, one effectively solves for reliability. Furthermore, by having a cleverly conceived modular structure, one facilitates the work of individual models and can have a smaller uncertainty in their output as the style of input changes. All of that, in fact, helps greatly with understandability, as now analysts can either run the whole program or validate and interact with the data as it is evolving. Furthermore, the implementation allows for the serialisation of the project, so that analysts can save and export their work. If they edit some section, then that is reflected in the code, which builds a great foundation for model fine-tuning in the future.

4. Discussion and Conclusion

In essence, after two months of working with Large Language Models, the author believes that the level of understanding of some LLMs is good enough to render the creation of analysis programs of many types realistic both theoretically and practically. A useful perspective to have when doing that, they claim, is to think of models as a small but important part of a greater system, like a cell in a body. The programmer must not assume thoroughness and reliability of its model but should strive to create such a structure that, powered by the model across its numerous calls, is enriched more and more until the desired product comes to the surface.

The input text must be the original source only at the beginning. It is to then be broken down into smaller sections, which are to be digested in a way that further queries

contain information of a neutral type that makes use of the knowledge of the model with the appropriate complexity per call.

The implementation created is almost exclusively for illustrative purposes, to show what the idea feels like in reality and to provide a foundation on potential further developments. There are a lot of better implementations and configurations that are to be conceived, and this is the the next question this 2-month research outputs from the initial one studied.

As the field progresses more and more and allows for greater and greater ideas to be implemented, it is important to take a step back and ensure we understand the power we give to models, as well as the "macro flow" they take to arrive at the conclusion.

5. Bibliography

Chen, Tong, et al. "Dense X Retrieval: What Retrieval Granularity Should We Use?" *arXiv.Org*, 11 Dec. 2023, https://arxiv.org/abs/2312.06648v2. [6]

Gao, Yunfan, et al. "Retrieval-Augmented Generation for Large Language Models: A Survey." *arXiv.Org*, 18 Dec. 2023, https://arxiv.org/abs/2312.10997v5. [3]

gkamradt. *Gkamradt/LLMTest_NeedleInAHaystack*. 2023. 30 Sept. 2024. *GitHub*, https://github.com/gkamradt/LLMTest_NeedleInAHaystack. [5]

"Home - CyberPeace Institute." *CyberPeace Institute*, https://cyberpeaceinstitute.org/. Accessed 1 Oct. 2024. [1]

Lance, Martin. *Langchain-Ai/Rag-from-Scratch*. https://github.com/langchain-ai/rag-from-scratch?tab=readme-ov-file. Accessed 1 Oct. 2024. [2]

Sarthi, Parth, et al. *RAPTOR: Recursive Abstractive Processing for Tree-Organized Retrieval*. arXiv:2401.18059, arXiv, 31 Jan. 2024. *arXiv.org*, https://doi.org/10.48550/arXiv.2401.18059. [4]