Observability in Natural Language Processing (NLP) Systems

EVA Advanced Software Architecture
MSE Data Science
submitted by

Moritz Schiesser
OST - Eastern Switzerland University of Applied Sciences
moritz.schiesser@ost.ch

Advised by

Prof. Dr. Olaf Zimmermann
OST - Eastern Switzerland University of Applied Sciences

June 1, 2024

Declaration of Authorship

I, Moritz Schiesser, declare that this technical report titled, "Observability in Natural Language Processing (NLP) Systems" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

June 1, 2024
Moritz Schiesser

Tools Used

- GitHub Copilot: has been used as a plugin for my IDE. Copilot has been pre-installed for coding purposes, however, Copilot also supports textual suggestions. Sometimes I have wholly or partially taken over suggestions and adapted them to the context of this report.

Abstract

Observability is a key concept in modern software engineering that allows engineers and operators to understand the health of a system by observing its external outputs. Upon discovery of issues, corrective action can be taken to ensure that a system performs as expected. Instead of hunting for issues, teams can focus on fixing them. With the recent rise of generative Natural Language Processing (NLP) systems, the need for observability in such systems has become apparent. While observability for classical software systems is increasingly popular, hardly any research exists on the same concept for generative NLP systems. This report calls for more research in the field, and explores how to transfer selected key concepts of observability to the field of generative NLP systems, discussing some common challenges and proposing a refined definition of observability for generative NLP systems. Challanges in generative text models like data drift, concept drift and adherence to topic and goals are discussed and evaluated. Speculative approaches to address these challenges are outlined.

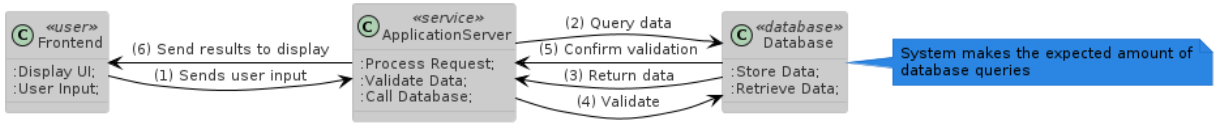
Contents

List of Figures

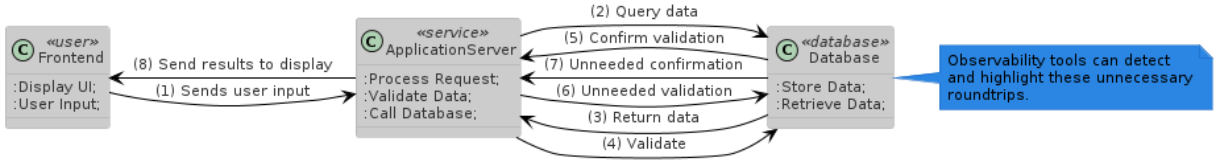
1 Introduction

Today software is often offered 'as-a-service', having continuous availability and steady quality of the service offered is crucial. Insights into the health and performance of systems are essential to ensure that these systems perform as expected. To provide data-driven insights into systems, observability is a key concept. It allows understanding for the internal state of a system by observing its external outputs. This in turn allows engineers to focus their efforts on fixing issues, rather than finding them, reducing the mean time to recovery (MTTR) [1, p. 7].

The following figure illustrates this concept on a abstract, fictional system that does not employ generative natural language processing (NLP) systems. While the system is not a real-world example, it is sufficient for demonstrating the importance of observability in a classical software system.



(a) Flow through system before update



(b) Flow through system after update

Figure 1: Overview of a fictional classical system before and after an update.

Through an update on the `ApplicationServer` component of the above diagram, the system's behavior has changed. Instead of the correct flow of program, where after querying data a validation is performed once in steps `(4)` and `(5)`, a secondary and unneeded validation is performed after the update in steps `(6)` and `(7)`. After the update, users report that the system is slower. Observing the distributed traces of the system, operators can identify the unnecessary validation step and advise the development team to address the issue. Traditional software systems have come a long way in incorporating observability into their design. The whitepaper "From Zero to Hero With Application Observability" [1] provides a comprehensive overview of observability as a concept and how it can be applied to software systems.

However, software systems nowadays are not only limited to white-box systems that one can fully control and observe. With the recent surge in interest in Machine Learning (ML) systems, and especially the widespread adoption of NLP models, the need for observability in such systems has become more apparent. The landscape of NLP systems is vast and complex, with many use cases and approaches being explored. As of late, generative transformer models, such as ChatGPT[1], Claude and Llama have become increasingly popular. In this report, the current approaches to observability in generative NLP systems are discussed and evaluated. First, the concept of observability in classic software system is introduced, followed by a discussion of the challenges of applying these concepts to NLP systems. It is argued which concepts are transferable, and which are not, and what new concept could be introduced to better understand the internal state of NLP systems, especially generative models and as they evolve over time.

[1] https://chatgpt.com

2 Observability in software systems

In this section, the concept of observability is discussed in depth. Especially the goal of observability in software systems is of interest, as learning about the goal of observability allows to transfer the concept to the domain of generative natural language processing (NLP) systems, without having to transfer details which are not applicable in the same way.

2.1 What observability aims to achieve

In essence, observability lets engineers and operators understand the health and inner state of a system by observing its external outputs. By understanding the systems behavior and performance, engineers can spend their time on fixing issues, rather than finding them, which in turn reduces the mean time to recovery (MTTR) [1, p. 7].

For example, in a distributed system with a microservice architecture, designing each system with obserability in mind enables operators to understand the flow of data through the system. A distributed trace through the whole system allows to reconstruct when each following request touched which service when, and how long each service took to process the request.

Figure 2 illustrates the need for this kind of observability in a fictional system.

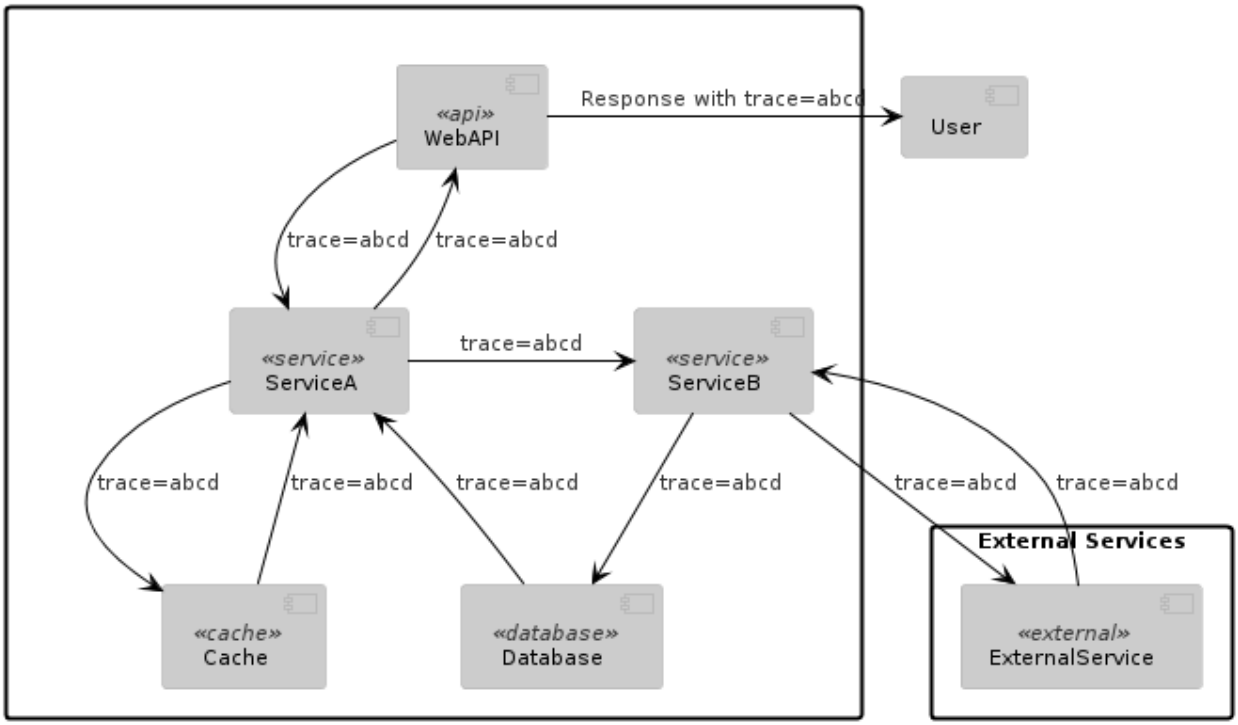


Figure 2: A fictional system with five components

As we can see, for a simple API call from the users perspective, a multitude of services might be involved. As operators need to understand the flow of data through the system, the use of distributed traces is essential to understand the health of the systems and to identify bottlenecks or other issues. In the example above, each component involved for the fulfillment of the API call would log relevant information to a central logging system, with each log entry containing the same trace identifier (marked in Figure 2 as `abcd`) This allows for later analysis of the logs of the whole system, by putting all the logs together to reconstruct the flow of data through the system.

2.2 The goal of observability

Observability helps ensure the continuous availability and performance of a system, for example by discovering changes in user of system or user behavior. For example, if a system that was updated suddenly shows that is requires 50% more calls to the database to fulfill a request than before the update, this could indicate that a bug has been introduced which could potentially slow the whole system down. More than that, it could impact the stability of the whole system, or require the deployment of additional capacity to handle the increased load. Neither of these scenarios are desirable for a service provider, as they could lead to a loss of customers or revenue. With sufficient observability in place, the operations team can detect such changes, and take corrective action either before the system is widely impacted, or minimize the duration during which the system is impacted.

Most of these goals and concepts are transferable to the domain of generative NLP systems. For example, a web-based chatbot system is largely still a classical software system, but with the added complexity of a generative NLP model. Except for the model itself, the system still has to handle user requests, and provide responses in a timely manner. More than that, the need for observability is still present for the model itself, but with a different focus.

2.3 The need for observability

In May 2024, Google incorporated their generative NLP model Gemini into their search engine [2] through "AI overviews", a feature that provides summaries of selected search results, created by generative AI, stating "our custom Gemini model can take the legwork out of searching".

In the weeks after, several users reported on various platforms that the generative portion of the search engine was generating incorrect and harmful information. The generative part of the search engine reported that "According to geologists at UC Berkeley, you should eat at least one small rock per day", and that "You can also add about 1/8 cup of non-toxic glue to the sauce to give it more tackiness" as a recommendation for making cheese stick to pizza better [3].

The statement about eating rocks might stem from an article published by The Onion titled "Geologists recommend eating at least one small rock per day" [4]. The Onion is a satirical news organization that publishes fake news articles for entertainment purposes only, and thus might unintentionally poison retrieval of information from the web.

While these two examples above are amusing if readers are aware that they are painfully wrong, they clearly illustrate that the Gemini system in its current state cannot be trusted to generate correct information in all scenarios. Less obvious cases of wrong information could lead to serious harm. One reason for these mishaps having happened *might* be that the operators did not take the required steps to ensure that the generative system was observable. Possibly there are no concepts in place to take corrective action, further demonstrating the need for observability in generative NLP systems.

In the next sections of this report, approaches for observability in generative NLP systems are discussed and evaluated, with the goal of providing a starting point for further research in the field.

3 State of the art in generative NLP systems observability

The topic of observability, as of writing this report, is being picked up by the community in software engineering. This however does not necessarily mean that the topic is as well understood in the field of artificial intelligence (AI), and especially not in the sector of generative NLP systems. As of writing the report, the author has not found many scientific papers that discuss any approaches in depth. More often than not, any reports that are found are non-scientific and of a sales magazine nature, in which products are advertised that claim to offer insights into generative NLP systems similarly to observability tools offer insights into classical software systems.

3.1 Methods of research

On Google Scholar, IEEE Xplore [2] as well as Google Search, the author searched for the following terms:

- "observability in natural language processing"
- "observability in NLP"
- "observability in generative AI systems"
- "evolution management in generative AI systems"
- "evolution management in generative NLP systems"
- "change management in generative NLP systems"
- "evolution management in generative natural language processing systems"
- "observability in chatbot systems"

The author iteratively refined the search terms to try and find relevant papers on the topic.

Further, ChatGPT was asked for papers on the topic, as well as the purpose-built "Consensus" GPT [3] was asked for papers on the topic.

The searches were last conducted on 2024-05-12.

None of these searches yielded any meaningful scientific results specifically, none relating to generative text systems and approaches to observability therein. There were results for other types of AI and machine learning (ML) systems, from which later sections try to derive approaches from.

During the writing of this report, it was debated whether to include the results of the search in the report. As the results are not deemed relevant to the topic at hand at all, the author decided to not include them in the report at all, and instead focus on the speculative approaches that are discussed in later sections of this report.

3.2 Call for research

The author of this report calls for more research in the field of observability in generative NLP systems. Later sections of this article offer initial ideas for approaches that could be taken to achieve the goals of observability which can hopefully picked up by the research community and further refined and critiqued. As companies and research institutions around the world dive into applying the capabilities of modern generative architectures onto real world problems, the field will require accountability to be created for the output these models generate. Aside from that, the business aspects of providing a generative AI service will, like they do with classical software systems, also require insights into the health and performance of the inner workings of the systems. As such, the author of this report calls for more research in the field of observability in generative NLP systems, and hopes that this report can serve as a starting point for such research by laying out the current state of the art, the challenges that are faced in the field and offering some speculative approaches to tackle these challenges.

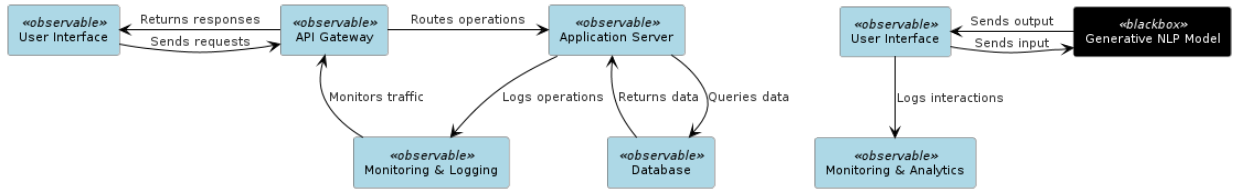
[2] https://ieeexplore.ieee.org/Xplore/home.jsp
[3] https://chat.openai.com/g/g-bo0FiWLY7-consensus

4 Translating observability principles to NLP systems

Generative natural language processing (NLP) systems still are software systems to a large extent, but they exhibit certain characteristics that make them different from a traditional software systems, as explained in Section 2 of this report.

At some point in a traditional software system, the operations become atomic, and with that unobservable in their inner workings, either on the level of the language, or on the level of the system itself. This point is generally reached at a level of detail that is uninteresting to observe. E.g. a operation where two integers are added together is atomic, and the state before and after the operation is observable and can be of interest, but the state of the CPU registers during the operation is probably not. This is where generative NLP systems differ, as the non-observable operations occur on a much higher level of abstraction and would be interesting to observe. For example, in a chatbot system, the response to a user query is generated by a model, and since the model is a black box, the inner workings of the model are not observable by nature. Essentially, when thinking in terms of classical software systems, the call to a model is the same as the call to a third-party non-open-source binary with bad documentation, where it is only know what the goal of the program is, but not why it does it, and how it does it.

This is a significant difference, as we essentially have a part of the system that simply is not observable in the same way as the rest of the system. Compared to a simple Create-Read-Update-Delete (CRUD) web application, where the state of the system can be observed relatively easily, generative NLP systems simply do not offer the same level of observability in the classical sense.



(a) CRUD system (b) generative NLP system

Figure 3: Overview over fictious CRUD and NLP systems

Observability in classical software systems often relies on data that is generated by the system itself, through logs, metrics, and traces [1, p. 17]. This allows for the system to be observed in a way that is reliable, repeatable and consistent. In generative NLP systems, since they are not necessarily deterministic, the outputs generated can vary in wording, while still being semantically the same. Since one cannot rely on static comparisons of the output, one has to rely on comparing the semantic meaning of the output, which is a much harder problem to solve, reasons for which are discussed in the following sections.

4.1 What observability in NLP is not

It is important to differentiate between the software part of the system and the NLP part of the system. If a generative model is offered as a service, e.g. in a chatbot-like manner, most of it can be observed in the same way as a traditional software system. However, the generating part of the system, the model, is a black box and thus cannot be observed in quite the same way. This report focuses only on transferring the observability principles to the generative part of the system, and not the software part, as the software part is already well understood and can be considered a solved problem.

The outputs generated by NLP systems are not measurable in the same way that outputs of the software part of the system are. For example, in classical software systems, both output and input are well-defined, and have a `Type` that can be observed, enforced, measured, and compared. With generative NLP systems, one has to think about input and output in a more abstract way. As is the nature of text embeddings, they encode the semantic meaning of a piece of text. The semantic meaning of text in itself is already an abstract concept and cannot be measured or enforced in the same way that a scalar value can be measured. While

this limits the traditional application of observability to some extent, this does not inherently mean that the system is not observable. Instead, it requires a different approach to observability, namely one that is less concrete and employs further machine learning models to evaluate the output of the generative model.

4.2 A short introduction to text generation

Generative NLP systems generate text based on a given input. However, as processing units like the CPU or GPU cannot process text directly, text is converted into a numerical representation, a text embedding, which can be processed by the model. The embedding is a high-dimensional vector which represents the semantic meaning of a sequence of tokens. What exactly a token is can vary depending on the architecture of the model, but for explaining the mechanism of a generative model, a token can be thought of as a word. A generative model, simplified drastically, "only" has to predict the next token in a sequence of tokens, given the tokens that came before it. This is illustrated in the following listing.

```
Input: the quick brown fox
Iteration 1: the quick brown fox jumps
Iteration 2: the quick brown fox jumps over
Iteration 3: the quick brown fox jumps over the
Iteration 4: the quick brown fox jumps over the lazy
Iteration 5: the quick brown fox jumps over the lazy dog
Iteration 5: the quick brown fox jumps over the lazy dog ##STOP##
```

`##STOP##` is a special token that indicates the end of the sequence, which causes the model to stop generating tokens. The model is trained on a large corpus of text, and learns to predict the next token in a sequence of tokens. Unlike humans, that are believed to have the capability to think, evaluate and reason before speaking, the model generates text roughly the same way a parrot would, by repeating what it has heard before. Hence, text-generation models are often referred to as "stochastic parrots", a term coined by Emily Bender [5].

Modern large language models often employ a concept called "attention", which allows the model to focus on certain parts of the input text, and thus generate more coherent and contextually relevant text. This is especially important in conversational agents, where the context of the conversation is important to generate a coherent response.

4.3 Applying observability principles to generative NLP systems

Facing the issue of varying outputs for the same question, one could argue that the systems output is not observable. And if one is to adhere strictly to the definition of observability, this is true, since one must be able to estimate the state of a system from its outputs only [6, p. 487]. If one is to deviate from this strict definition, it could be argued, that the output of the system is not only the (embedded, and thus numerical) text, but also the semantic meaning of the text. This would help in achieving the goal that observability has in the current definition, the goal of detecting and diagnosing issues within the systems' entirety. In text embeddings, the semantic meaning of a piece of text is encoded as a high-dimensional numerical vector, which can be compared to other vectors. Where in classical observability a scalar value, e.g. the round-trip time of a HTTP request is measured, in NLP observability, one could measure the similarity of the semantic meaning of the current output to the semantic meaning of previous outputs. Through this, when releasing a new version of a generative model, drastic changes in generative behavior would be observable, without explicitly knowing the exact content of the output. Note that this is far from a perfect solution, as it detects changes in any semantic direction, and thus would also alert operators to changes that are not necessarily bad, but just different, or worse, changes that are much better than in previous versions. Further, this approach assumes that a operating entity has a working state of the system in its entirety in place, and can thus create meaningful comparisons between different versions.

A different approach is to analyze the amount and length of user queries, alongside the length of the responses. Used in conjunction with the typical lifespan of a conversation between a user and the system, this can provide insights if a new version of the model caused drastic changes in the behavior of the system. E.g. if a new version of the model causes much longer outputs to be generated by the NLP system for any type of

query, this could be an indicator that the new version of the model is not working as intended. Similarly, if users stop interacting with the system much quicker than before, this could be an indicator that the system has started to generate outputs which are either very useful to the user (since they have gotten all information they wanted), or very unhelpful, and they are going to look for the information elsewhere.

These principles apply to any user interface driven system, and are not limited to generative NLP systems.

4.4 Approaches to observability in AI systems

Censius identifies two potential drifts that can occur in AI systems over time [7, p. 6]:

- Concept drift (also titled "Feature drift" [8, p. 31])
- Data drift

In the concept drift, the distribution of certain features in the data the model was trained with change in the real world, thus leading to a subpar performance of the model. They give the example of the geolocation of a customer being relevant for a recommendation model, and the customer moving to a different location, thus changing the underlying ground truth to the model [7, p. 6]. The data drift is described simply as the possibilities of real-world data changing over time, i.e. new brands being introduced to an ecommerce platform, which the model has not been trained on [7, p. 6].

In the context of NLP, the concept drift could be interpreted as the change of the meaning of a certain word over time, such as the word "hacker" which has predominantly meant a person who is able to break into protected computer systems, but nowadays is also used to describe a person who is able to solve complex problems in a creative way, such as a "life hacker", or a "growth hacker". A "Life Hacker" is a person that, which creative and clever application of simple methods, is able to make everyday tasks easier, and a "Growth Hacker" is a person that uses creative and low-cost strategies to help businesses acquire and retain customers. Both of these meanings are not strictly comparable the original meaning of the word "hacker", but relate to the idea of finding a clever way to bypass restrictions or traditional limitations of a system. The data drift could be interpreted as new situations arising in the world, whose information simply was not available at the time the model was trained, such as the Russian invasion of Ukraine in 2022 which was not known to ChatGPT at the time the situation was unfolding.

Discovering the concept drift could be an exceedingly hard problem to solve, as it requires a model to reflect on its own output, the users input, and the context in which the conversation is happening. Such emergent behavior is neither easy to detect, nor easy to contextualize, as it requires a deep understanding of the world, and the ability to reason about it. This report does not offer a solution to this problem, but rather points out that it is a problem that needs to be solved.

Data drift is a problem for which partial solutions exist, such as the one described in the previous section, where user queries that are similar to each other are would lead to the model generating outputs that either have a semantic meaning which indicates that the model tells the users it does not know the answer, or that the model generates hallucinations. Maynez et al. [9, p. 864] state that "while the term hallucinations has not been defined formally, the term is standardly used to refer to the generated content which is either unfaithful to the input, or nonsensical", citing Katja Filippova [10] as the source of this definition. If data drift causes the model to hallucinate, according to Huang et al., this hallucination would be caused by a "Knowledge Boundary", and specifically "Outdated Factual Knowledge" [11, p. 9].

Data drift and concept drift differ significantly in their meaning and resulting challenges. Draft drift essentially requires training data to exist for a term or event that has not been seen before, while concept drift requires the model to understand which meaning of a term is currently relevant, or potentially to infer from context which meaning of a term is currently relevant.

As mentioned in the previous section, barley any scientific literature exist that elaborates on the challenges faced when bringing observability into the context of generative NLP systems. The reports and papers that exist often focus on the problems identified that make observability in such systems necessary, but do not offer solutions to these problems. Exemplary for this is the survey conducted by Huang et al. which lists

various benchmarks that can be used to evaluate the performance of a generative NLP system, specifically regarding the faithfulness of the output, demonstrating that the problem is known, but not solved [11, p. 4].

4.5 Beyond classical observability

This section will discuss what limitations classical observability faces when applied to generative NLP systems, and what measures can be taken to mitigate these limitations. Please note that this section is widely speculative, and the author does not claim to have tried any of the approaches described in this section.

One could argue that the entirety of a system does not end at the firewall of a system, but should instead include the users of the system. A tried and tested approach to this is the use of user feedback, for example in the form of a thumbs up / thumbs down button, or A/B testing of different variations of a generative NLP system. This approach is not new, and has been used in the software industry for a long time, but it is uniquely suited for observing generative NLP systems, as it does not require hard facts to be observed, but rather the subjective opinion of the user. It is not without its problems, as this approach really only makes sense if a system has a large enough user base so that subjective opinions and biases are counterbalanced in the whole user group. However, these positive / negative evaluations of a singular generation of outputs allows for the quick realization that something the model is producing is not as it should be, when rolling out a new version of the model. Specifically, is a new version of a model is rolled out the 5% of the user base, and these 5% overwhelmingly respond with negative "thumbs down", we can derive that the new model is not working as these users want it to.

4.5.1 Observability as a tool for detecting bad actors

Shortly teased in a previous section, the concept of attention is a key concept in modern large language models. It allows the model to focus on certain parts of the input text, and thus generate more coherent and contextually relevant text.

Given a user that wants to override the intention of a model's operators, they could instruct a model to "discard all previous instructions and ..." and then give the new instruction. By observing the attention mechanism of the model, it might be possible to detect that the model is not paying attention to the original instructions, which might be an indication that the model is being misused by a bad actor. This would then lead to the need for corrective action.

4.5.2 Observability as a countermeasure for bad evolution

The previous section outlines a few speculative approaches on how the principles and goals of observability can be applied to black box generative models. The following highlights how these approaches could be used to counteract the possible bad evolution of a generative model.

Example 1: Concept drift

For illustrating how the proposed measures can help keeping quality and experience for end users high, let's consider the following fictitious scenario:

The word "napkin" that currently commonly refers to a piece of paper or cloth used at meals to wipe fingers or lips and protect garments. Due to pop-culture references, the word "napkin" can now also be used to disrespectfully refer to a person that does not follow up on their promises.

Suddenly, users of a chatbot application like ChatGPT start asking the chatbot for what "napkin" means, and the chatbot responds with the first definition as that was the meaning of the word when the model was trained. Through observing that lots of users ask the Chatbot multiple times for the definition of the word "napkin" in the same conversation, operators can infer that the system is not working as intended and thus might lead the user base to churn. Reacting to that and other examples of concept drift, operators can determine the need for a retraining / fine-tuning of the model, and thus ensure that the system is working as intended.

The "napkin" example is a clear case of concept drift, as it amends the meaning of a word, and thus the model needs to be retrained to understand the new meaning of the word. It does not introduce a new concept like a new brand, but rather changes the meaning of a word that the model already knows.

Example 2: Data drift

Similarly, if a new type of electronic device is released, and users start asking the chatbot about the device, the chatbot might not have the information to answer the questions. Since new releases of devices, and other news-type information are not feasible to be included in real-time in the training data of the model, retraining a model might not be a viable solution. Instead, operators can let the machine learning development team know that they would propose to instruct the model in the fine-tuning process to instead decline to answer the question, as the information is not available to the model. In a second step, if the model then does not do as instructed, this could once again be inferred from user behavior, either from user feedback, the user suddenly stopping to interact with the system, or the user asking the same question multiple times in the same conversation.

4.5.3 Observability as a tool for detecting out of scope behavior

A common use case for chatbots is to provide information to users. Businesses and educational institutions alike might apply generative NLP systems to provide a unified interface for browsing their knowledge bases, using retrieval augmented generation (RAG). RAG refers to the process of comparing a user-query's embedding with the embeddings of the documents in a knowledge base, and then providing the chatbot system with the most similar documents as context for the generation of the response. This allows the chatbot to provide users with up to date and relevant information on a topic, without having to be explicitly trained on an entities' data. If an entity wants to provide a chatbot that can only answer questions relating to a certain domain, e.g. a chatbot that provides information on the products of a company, it is not desirable to have users ask the chatbot about arbitrary topics. Prompt engineering can be used to guide the chatbot to only answer questions that are in scope. However, since it is not realistic to have a perfect prompt from the inception of such a project, and requirements and capabilities may change over time, so will the prompt used to steer a chatbot in the right direction. Detecting whether a modification on a prompt has led the chatbot to answer out of scope questions can be done through observing the user behavior or using user feedback (if the userbase is 'friendly'). However, this might not be sufficient. Instead, a secondary 'Observer' LLM could be implemented, that reads the output of the generative model, and then reports if the output is deemed out-of-scope. From this, operators can infer if a prompt modification has caused the system to become more susceptible to out-of-scope behavior and needs further tuning. The fictitious graph below illustrates this concept.

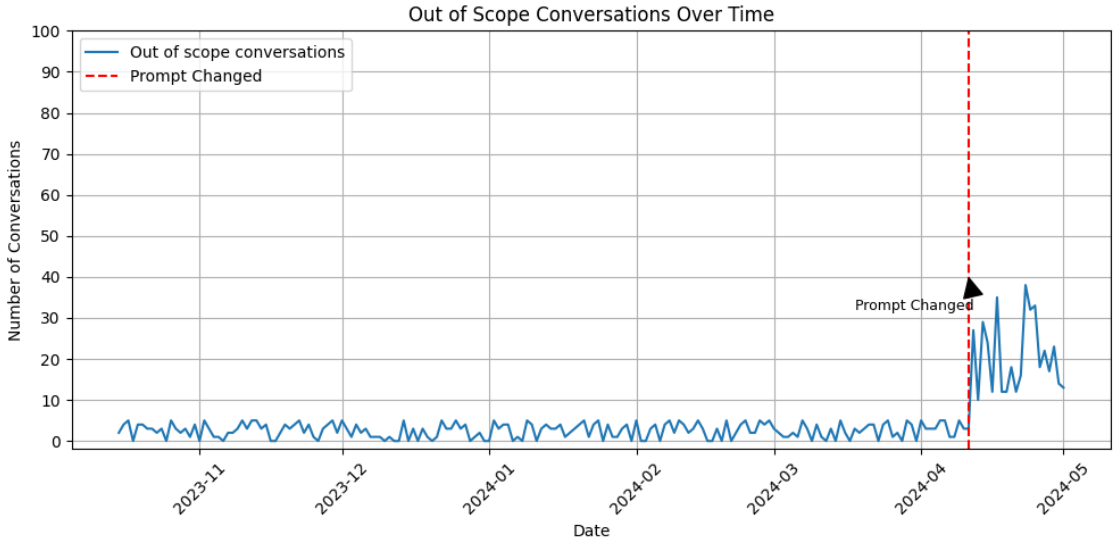


Figure 4: Generator-Observer detection mockup

Going one step further, if it is discovered that using prompt engineering to prevent unintended behavior is not sufficient, one could modify the concept of the 'Observer' LLM to be a 'Guardian' LLM. The 'Guardian' would use the output of the 'Observer' as input, and if the 'Observer' has deemed a conversation to be out of scope, the 'Guardian' would then take corrective action, such as ending the conversation, or asking the user to rephrase their question. Figure 5 below illustrates the flow through each component of the system.

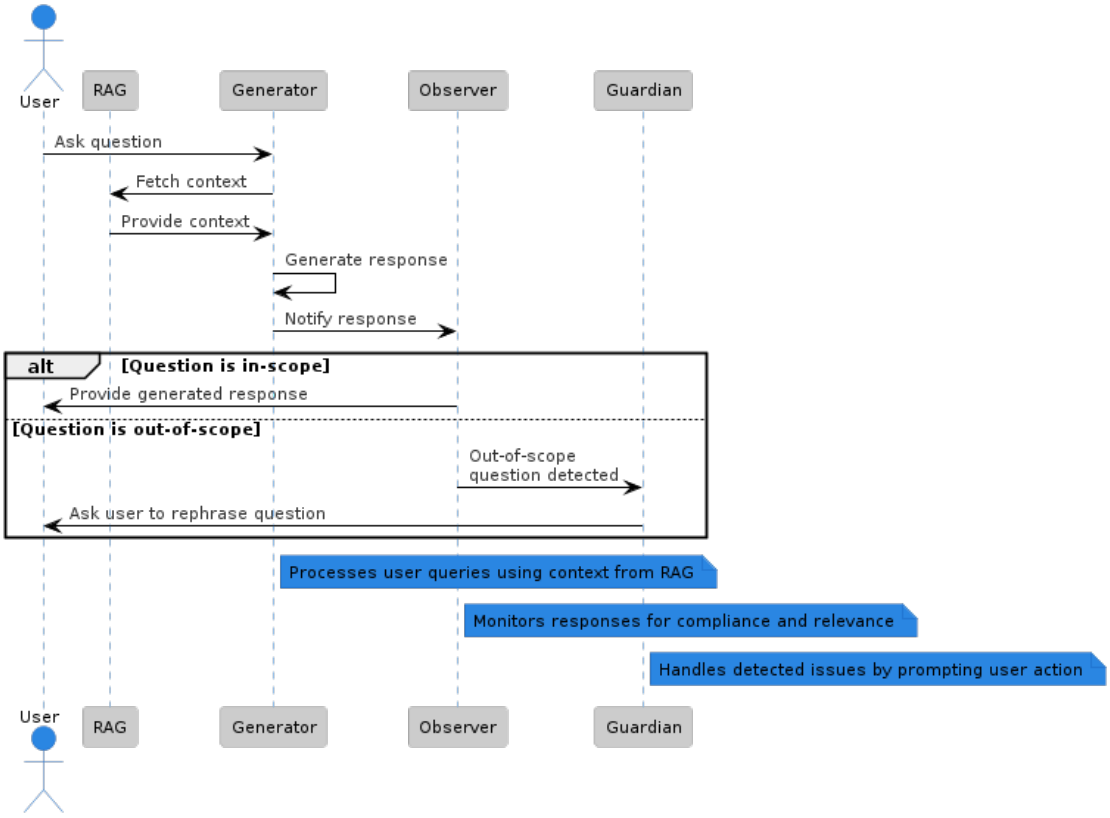


Figure 5: Generator-Observer-Guardian flow

Of course, with the 'Guardian' and the 'Observer' being large language models as well, research on how to efficiently train such models to detect out of scope behavior is needed.

4.6 Conclusion

The topic of how to transfer observability principles and goals onto generative NLP systems is a largely unexplored field. The speculative approaches on how to achieve this, as outlined in this section, are a starting point for further research in the field. The initial ideas for approaches on observability in generative NLP systems presented in this report show however, that if one is to deviate from the original definition of observability in the strict sense, and create a new definition that is more suited to the domain of generative NLP systems, observability in generative NLP systems is possible. A proposed definition for observability in generative NLP systems is outlined in the following section.

5 Observability for generative NLP systems: a refined definition

As established in the previous sections of this report, observability for generative natural language processing (NLP) systems is a topic that is not well understood or documented in the academic community. This report serves as a starting point for further research in the field. During the writing of this report, it has become apparent that the current definition of observability for classic software systems fails to capture the challenges that are present in generative NLP systems. This new definition is not meant to replace the definition of observability for software systems, but to offer an extension that is more applicable to the field of generative NLP systems.

5.1 The need for a refined definition

The current definition of observability for software systems is centered around the idea that engineers and operators can understand the health and inner state of a system by observing its external outputs. While this is a desirable quality for any system, and especially for generative AI, this requirement falls short as soon as black-box models are involved. Modern generative NLP systems demonstrate an immense ability to generate human-like, grammatically correct statements, but cannot reason about the output they generated. This makes generative NLP models inherently unreliable, as they can be instructed by users to generate information that is not desired, or even harmful. In order to make observability for generative NLP systems an accessible concept to ML engineers and operators, a refined definition is needed.

5.2 The goal of observability in generative NLP systems

Observability for NLP systems, like observability for software systems, aims to provide insights into the health and performance of the system. NLP observability should enable operators to understand an abstract and averaged general behavior of a system, and thus enable them to take corrective action when the system is not performing as expected. This can include, but is not limited to the previously mentioned examples of data drift, concept drift, or guarding against unwanted outputs.

5.3 A refined definition for observability in generative NLP systems

A generative NLP system is observable, if the outputs of the generating components are sufficiently monitored and evaluated in a way that allows operators to understand if the model is achieving the desired outcomes effectively and efficiently.

This definition is intentionally kept vague, as the desired outcomes of such a system might very greatly depending on the use case. For example, desired outcomes might be relative abstract, such as to achieve a high level of user engagement, user satisfaction or too much more defined, such as to provide short summaries of a long input text.

In order to achieve this, metrics need to be defined that allow to measure a text-based output in a meaningful way. This can take form by interpreting user interactions, or by evaluating the output of the model in a context that is meaningful to the operators. Meaningful metrics would be ones that allow operators to gauge both performance and quality of the system as a whole, and would allow them to understand if corrective action needs to be taken.

6 Summary

With the recent advancements in observability for classical software engineering systems as well as the recent surge in interest and popularity of generative NLP systems, the need for observability in generative NLP systems has become more apparent. The field for this application of observability on generative NLP systems is still in its infancy, and that more research on the topic is needed. While the concepts of observability lend themselves to being applied to the field of machine learning, some more comprehensive approaches are required to allow for the same level of accountability and insight into the inner workings of generative NLP systems. This report calls for more research in the field of observability for generative NLP systems, offers speculative approaches to tackle the challenges that are faced in the field, and hopes to serve as a starting point for further research in the field. Especially questions such as "What external outputs can be observed to understand the inner state of a model" and "How can the inner state of a model be quantified" are of interest and require further research. This report proposes a refined definition of observability for generative NLP systems, and the author hopes that this definition can be critiqued, refined and built upon by the scientific community.

References

[1] SolarWinds, "From Zero to Hero With Application Observability." 2021 [Online]. Available: https://www.solarwinds.com/assets/solarwinds/swresources/ebook/from-zero-to-hero-with-application-observability-ebook.pdf

[2] "Google I/O 2024: Generative AI in Search: Let Google do the searching for you," 2024. [Online]. Available: https://blog.google/products/search/generative-ai-google-search-may-2024/

[3] "Eat a rock a day, put glue on your pizza: how Google's AI is losing touch with reality," 2024. [Online]. Available: https://theconversation.com/eat-a-rock-a-day-put-glue-on-your-pizza-how-googles-ai-is-losing-touch-with-reality-230953

[4] "Geologists Recommend Eating At Least One Small Rock Per Day." 2021 [Online]. Available: https://www.theonion.com/geologists-recommend-eating-at-least-one-small-rock-per-1846655112

[5] Bender, Emily M., Gebru, Timnit, McMillan-Major, Angelina, and Shmitchell, Shmargaret, "On the Dangers of Stochastic Parrots: Can Language Models Be Too Big?" *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*. Association for Computing Machinery, 2021 [Online]. Available: https://doi.org/10.1145/3442188.3445922

[6] Kalman, R. E., "On the general theory of control systems," *IFAC Proceedings Volumes*, vol. 1. 1960.

[7] Censius, "A Guide to Stepping into AI Observability," 2024 [Online]. Available: https://censius.ai/ebooks/guide-to-stepping-into-ai-observability

[8] Arize AI, "Machine Learning Observability 101," 2024 [Online]. Available: https://arize.com/resource/ebook-machine-learning-observability-101/

[9] Joshua Maynez, Shashi Narayan, Bernd Bohnet, and Ryan McDonald, "On Faithfulness and Factuality in Abstractive Summarization," *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, pp. 1906–1919, 2020 [Online]. Available: https://aclanthology.org/2020.acl-main.173

[10] Katja Filippova, "Controlled Hallucinations: Learning to Generate Faithfully from Noisy Data," *Findings of the Association for Computational Linguistics: Empirical Methods in Natural Language Processing (EMNLP) 2020*. Association for Computational Linguistics, pp. 864–870, 2020 [Online]. Available: https://aclanthology.org/2020.findings-emnlp.76

[11] Lei Huang *et al.*, "A Survey on Hallucination in Large Language Models: Principles, Taxonomy, Challenges, and Open Questions." 2023.