# A Comprehensive Software Aging Analysis in LLMs-Based Systems

César Santos
*Department of Computing*
*Federal Rural University of Pernambuco*
Recife, Brazil
cesar.santos@ufrpe.br

Fumio Machida
*Department of Computer Science*
*University of Tsukuba*
Tsukuba, Japan
machida@cs.tsukuba.ac.jp

Ermeson Andrade
*Department of Computing*
*Federal Rural University of Pernambuco*
Recife, Brazil
ermeson.andrade@ufrpe.br

***Abstract*—Large language models (LLMs) are increasingly popular in academia and industry due to their wide applicability across various domains. With their rising use in daily tasks, ensuring their reliability is crucial for both specific tasks and broader societal impact. Failures in LLMs can lead to serious consequences such as interruptions in services, disruptions in workflow, and delays in task completion. Despite significant efforts to understand LLMs from different perspectives, there has been a lack of focus on their continuous execution over long periods to identify signs of software aging. In this study, we experimentally investigate software aging in LLM-based systems using Pythia, OPT, and GPT Neo as the LLM models. Through statistical analysis of measurement data, we identify suspicious trends of software aging associated with memory usage under various workloads. These trends are further confirmed by the Mann-Kendall test. Additionally, our process analysis reveals potential suspicious processes that may contribute to memory degradation.**

***Index Terms*—Software Aging, LLMs, Memory Degradation.**

## I. Introduction

Large language models are a type of generative Artificial Intelligence (AI) capable of processing and generating human-like text [1]. They are trained on vast amounts of textual data and learn to understand language patterns, semantics, and context. These models are increasingly utilized in various applications, including natural language processing, text-to-image synthesis, protein modeling, computer programming, among others [2]. Considering the significant versatility and effectiveness of LLMs across various domains, they have become indispensable tools in academia and industry. Consequently, ensuring their reliability has become a critical concern, particularly due to the potential consequences of their system failures, such as interruptions in services, disruptions in workflow, and delays in task completion.

LLMs have increasingly permeated our lives in recent years, extending their reach into critical areas such as healthcare and finance. This expansion highlights the need for more efficient and secure systems and emphasizes the importance of continuous monitoring for extended periods. Over time, LLM-based systems may experience gradual performance degradation and resource exhaustion, known as software aging [3]. While preventing software aging may not always be feasible, mitigating its effects, reversing damage, and understanding its causes are essential [4]. Despite significant attention to understanding LLMs, a research gap remains regarding their continuous operation over extended periods to detect signs of software aging. This challenge is particularly important in LLM contexts, where sustained operation is vital for effectiveness and reliability.

Existing literature on LLMs has primarily focused on exploring their capabilities, applications, and underlying mechanisms, often neglecting the long-term implications of their deployment. Although several studies have investigated aspects like model evaluation and training methodologies, none of them have explored the phenomenon of software aging and its implications for LLM reliability. For instance, in [5], the authors proposed a framework for evaluating ChatGPT. This framework encompasses ethical considerations, context adaptability, and community collaboration. In [6], Hoffmann *et al.* explored optimal transformer language model size and training data amount for a given compute budget. Therefore, it is essential to analyze the reliability aspects of these environments, especially in long-running instances.

In this study, our objective is to experimentally investigate software aging in LLM-based systems, focusing specifically on continuous execution over extended periods. We aim to explore the presence of software aging symptoms and their potential causes in these systems, utilizing prominent LLM models such as Pythia, OPT, and GPT Neo. Through statistical analysis of measurement data, our goal is to identify indicators of software aging, particularly in memory usage under varying workloads. Additionally, we intend to validate these findings using robust statistical tests, such as the Mann-Kendall test, to enhance the reliability of our results. Furthermore, our research aims to conduct a comprehensive process analysis to identify potential factors contributing to software aging in the employed LLM-based systems. Our results revealed a progressive increase in memory usage by the LLM systems over time. Additionally, Opt appears more susceptible to software aging compared to Pythia and GPT Neo, likely due to its higher RAM consumption. Our findings and analysis approach can be helpful for users or developers in making decisions to address software aging issues encountered in LLM-based systems.

This paper is organized as follows: Section II details the related work. In Section III, we provide a detailed explanation of our experiments and the approach adopted to address

software aging in LLMs. Section IV presents the results of the experiments and statistical analysis. Finally, Section V summarizes the conclusions and outlines the planned next steps for future works.

## II. Related Work

Despite the increasing popularity of LLMs, there is a lack of studies that investigate software aging symptoms specifically in LLM-based systems. Much of the existing literature on LLMs has predominantly concentrated on exploring their capabilities, applications, and underlying mechanisms, often neglecting the long-term consequences of their implementation [7], [8]. Consequently, our study begins with a review of existing methods for assessing LLMs. We then explore the concept of software aging in computing systems to provide context. Finally, we then conclude by comparing our work with those of existing literature.

Numerous studies have explored the assessment of LLMs. Among these studies, Beyond the Imitation Game benchmark (BIG-bench) [9] is an example. This benchmark aims to assess the performance of LLMs across diverse language tasks. In [10], *Moghe et al.* evaluated LLMs considering multiple languages and domains. This evaluation aimed at providing insights into the models' global performance capabilities. In [11], the authors introduced the Holistic Evaluation of Language Models (HELM), a framework designed to improve model transparency. HELM enables the assessment of various metrics, including accuracy, calibration, robustness, fairness, bias, toxicity, and efficiency. Researchers also explore methods to enhance LLM performance through data selection techniques [12] and propose improvements to memory efficiency [13].

Some studies also have analyzed and compared software aging in computing environments. In [14], Andrade *et al.* examined how software aging affects continuously running image classification systems. They compared the phenomenon in two distinct environments: cloud and edge computing. In [15], an analysis was performed to identify the symptoms and sources of software aging in a testing framework called Jest. In [16], the authors investigated the software aging issue in a real-time object detection system that utilizes YOLOv5. In [17], Dias *et al.* investigated the root causes of software aging in the Cardano blockchain. Their analysis suggests that the cardano-node process, which is the core component of the platform, might be responsible for the observed performance degradation.

In contrast with previous studies, our research aims to examine the occurrence of software aging in LLM-based systems through a comparative analysis. Specifically, we utilized Pythia, OPT, and GPT Neo in a client-server architecture. We conducted statistical tests to detect suspicions of software aging and to compare the system environments, focusing on memory consumption. Upon identifying software aging, we conducted process analysis to investigate potential causes of the observed degradation, aiming to identify processes contributing to software aging symptoms in the adopted system environments. To the best of our knowledge, this study represents the first comprehensive comparative analysis of software aging in LLM-based systems, spanning from software aging analysis to identifying processes responsible for the degradation.

## III. Experiments

### A. Research Questions (RQs)

The objective of our experimental study is to investigate potential software aging symptoms in LLM systems. If symptoms of software aging are detected, it is important to analyze the causes and differences in their manifestations among different LLM systems. Therefore, we formulate the following research questions for our experimental study:

- RQ1: Do LLM systems encounter any software aging problems?
- RQ2: What are the potential causes of software aging if any phenomena are observed in the LLM systems?
- RQ3: How do the impacts of software aging differ among the adopted LLM systems?

To answer these questions, we conducted the following experiments.

### B. Setup

The system simulates a chat-like interaction between a client and server. The client sends requests to the server, which processes them using the deployed LLM for text generation. In the operational setup, the client periodically sends requests to the server, each carrying a character string as a parameter for text generation. The server receives and processes the requests, extracting the character string and transmitting it to the designated LLM. Upon receiving a response from the LLM, the server displays the generated text on the console.

In this work, we utilize three open-source LLMs: Pythia, OPT, and GPT-Neo. All three are comparable in size and trained on the same dataset, offering diverse capabilities [2]. Pythia, developed by EleutherAI, prioritizes interpretability, aiding scientific exploration of LLMs. OPT, from OpenAI, focuses on replicability and accessibility for researchers. Finally, GPT-Neo, also by EleutherAI, aims to replicate the capabilities of closed-source models like GPT-3.

For the execution of the chat, we employed the same example across all models to ensure consistent results. Specifically, the client sent a request with the query 'how do i make money?'. The server then utilized the LLM to generate text for the response. This interaction took place in a Local Area Network (LAN). It is worth to highlight that specific parameters were defined for the generation process. These parameters included a maximum token limit of 256, which controls the number of text units in a single input. A temperature of 1 was set to moderate randomness in the generated text, while constants for top-k (40) and top-p (1) were defined to influence word selection during text generation. The hardware specifications for both the client and server are described as follows:

- Server device: Desktop, Intel Core i58400 2.80GHz, 16 GB of memory, Kingston 120GB SSD and Ubuntu 22.04.
- Client device: Desktop, AMD A8-5500B, 8GB of memory ram, 500GB of Hard Disk and Ubuntu 22.04.

### C. Experiments, metrics, and analysis method

To address the research questions, we conducted a series of experiments aimed at investigating the possible presence of software aging in LLM systems. To achieve this goal, we applied the following workloads: low workload (1 request every 10 seconds), medium workload (1 request every 5 seconds), and high workload (1 request every 1 second). For each workload setting, we executed the experiments for 48 hours.

For each experiment, we collected system monitoring data and analyzed aging indicators. These indicators refer to system variables that can be directly measured and are associated with software aging phenomena [3]. The analysis of aging indicators can be performed both at the system level and the application level. System-level analysis investigates potential software aging phenomena in system resources. On the other hand, application-level analysis aims to identify the processes primarily responsible for resource consumption and any degradation in user-perceived performance, if present. In this study, we focused on aging indicators related to resource depletion in terms of real memory consumption, as this indicator showed the most promise in detecting aging.

To analyze potential software aging, we used two tests: the Mann-Kendall test and Sen's slope calculation. The Mann-Kendall test checks if there's a trend in the data [18], while Sen's slope measures the size of this trend [19]. If the test's p-value is below 0.05, it means there is a significant trend. Software aging is a gradual process, and these tests help us identify patterns of degradation. However, finding a trend does not necessarily mean the software is aging [20]. If the system experiences issues after prolonged use, it suggests software aging. Additionally, Sen's slope provides insight into the strength of the trend, with a positive slope indicating growth and a negative one indicating decline.

To analyze the processes, a Python script was developed to collect information on the memory usage of the server's running processes. Data was collected every minute using the 'ps' command, which is a tool for displaying detailed information about the physical, virtual, and shared memory usage of a system, including memory usage by individual processes. This command provides metrics such as USS (Unique Set Size), PSS (Proportional Set Size), and RSS (Resident Set Size). In this study, we utilized RSS values to monitor the memory usage growth over time by the processes of the adopted LLMs and identify the processes that exhibited significant memory growth.

## IV. Results

In this section, we present the results of the experiments, as well as the statistical analysis we performed to answer the research questions.

TABLE I: RAM Memory Usage: Mann-Kendall and Sen's Slope Test Results and Mean.

| Model | Workload | p-value | Slope | Mean |
|---|---|---|---|---|
| Pythia | Low | 2.22e-16 | 0.874587e-3 | 2.33974 |
| | Medium | 2.22e-16 | 5.989796e-3 | 2.415752 |
| | High | 2.22e-16 | 3.355995e-3 | 2.443616 |
| Opt | Low | 2.22e-16 | 6.535879e-3 | 1.997343 |
| | Medium | 2.22e-16 | 6.325496e-3 | 2.164912 |
| | High | 2.22e-16 | 12.63217e-3 | 2.359757 |
| Gpt Neo | Low | 2.22e-16 | 6.172007e-3 | 2.199209 |
| | Medium | 2.22e-16 | 6.736925e-3 | 2.312415 |
| | High | 2.22e-16 | 6.640713e-3 | 2.280273 |

### A. Aging analysis (RQ1)

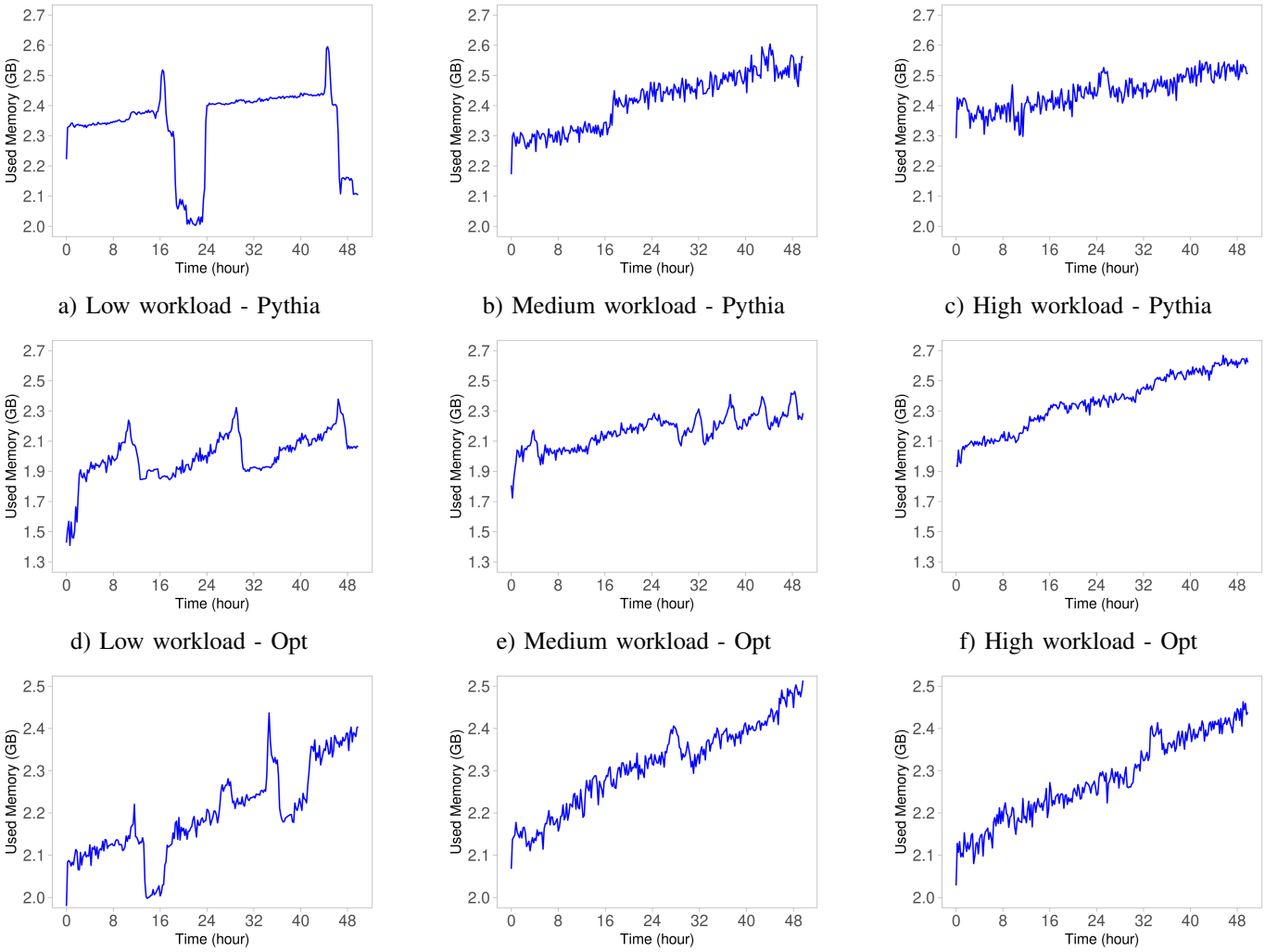
To address RQ1 regarding whether LLM systems encounter any software aging problems, we analyzed signs of software aging in the RAM memory. Figure 1 provides a comprehensive comparison of RAM usage across the LLM systems, showing RAM usage levels under low, medium, and high workloads for Pythia (Figures 1 (a), (b), and (c)), Opt (Figures 1 (d), (e), and (f)), and GPT Neo (Figures 1 (g), (h), and (i)). Across all scenarios and systems, there is a consistent trend of increasing memory consumption over the 48-hour period. However, for the low workload, distinct peaks followed by sharp declines are noticeable. While these peaks are particularly pronounced for Pythia, they are still observable, but less pronounced, for Opt and GPT Neo.

To confirm the suspicions of software aging in RAM usage results, the analysis involved computing the Mann-Kendall statistic, Sen's slope estimator, and the mean of memory usage. Table I presents the obtained values, confirming suspicions of software aging and a consistent trend of increasing RAM usage across all scenarios. The results also reveal that Opt exhibited one of the highest slope rates, particularly notable in the high workload case, followed by GPT Neo, with Pythia showing the lowest rate. However, the results also indicate that the rates do not correspond consistently to the workload. Lastly, it is notable that although Opt had one of the highest slopes, it had one of the lowest average memory consumption.

### B. Process analysis (RQ2)

In this subsection, we present in detail the process analyses conducted in this study. The primary objective of these analyses was to identify potential processes responsible for the symptoms of software aging in the utilized system environments and, consequently, to address RQ2. First, we identified the processes that showed the highest increase in RAM consumption during the tests. Subsequently, we present a comprehensive comparison of the LLM processes' growth over time.

Table II presents the ranking of these processes for the Pythia system. The Pythia processes demonstrated the most substantial rise in RAM consumption across all workload cases, with growth rates of 459%, 506%, and 873% for the low, medium, and high workloads, respectively, suggesting




a) Low workload - Pythia b) Medium workload - Pythia c) High workload - Pythia

d) Low workload - Opt e) Medium workload - Opt f) High workload - Opt

g) Low workload - GPT Neo h) Medium workload - GPT Neo i) High workload - GPT Neo

Fig. 1: RAM Memory Consumption.

it could be the primary suspect of software aging. Table III displays the ranking of processes experiencing the most growth for the Opt system. Opt processes for the low, medium, and high workloads exhibited the second most significant increase in RAM consumption during the tests, with growth rates of 115%, 20%, and 28%, respectively, indicating they might also contribute to software aging in the Opt environment. Note that the processes with the highest growth were *fwupd*, *packagekitd*, and *snap-store*. Additionally, Table IV shows the ranking of processes with the most growth for the GPT Neo system, where the GPT Neo process displayed the most significant increase in RAM consumption during the tests, with growth rates of 267%, 317%, and 238% for the low, medium, and high workloads, respectively, suggesting it could also be a primary suspect of software aging. Table V presents a detailed description of all processes.

TABLE II: Processes that grew the most for the Pythia system.

| Ranking | PROCCESS | Growth |
|---|---|---|
| Low / Pythia | | |
| 1st | Pythia.py | 459% |
| 2nd | fwupd | 141% |
| 3rd | gnome-terminal-server | 16% |
| 4th | snap-store | 13% |
| 5th | systemd-journald | 5% |
| Medium / Pythia | | |
| 1st | Pythia.py | 506% |
| 2nd | fwupd | 143% |
| 3rd | gnome-terminal-server | 22% |
| 4th | snap-store | 17% |
| 5th | gnome-shell | 12% |
| High / Pythia | | |
| 1st | Pythia.py | 873% |
| 2nd | gnome-terminal-server | 21% |
| 3rd | tracker-miner-fs-3 | 15% |
| 4th | snap-store | 13% |
| 5th | gnome-shell | 8% |

TABLE III: Processes that grew the most for the Opt system.

| Ranking | PROCCESS | Growth |
|---|---|---|
| Low / Opt | | |
| 1st | fwupd | 141% |
| 2nd | Opt.py | 115% |
| 3rd | snap-store | 29% |
| 4th | packagekitd | 11% |
| 5th | gnome-terminal-server | 6% |
| Medium / Opt | | |
| 1st | packagekitd | 48% |
| 2nd | Opt.py | 20% |
| 3rd | systemd-journald | 14% |
| 4th | gnome-shell | 8% |
| 5th | snap-store | 7% |
| High / Opt | | |
| 1st | snap-store | 91% |
| 2nd | Opt.py | 28% |
| 3rd | packagekitd | 25% |
| 4th | snapd-desktop-integration | 12% |
| 5th | gnome-terminal-server | 8% |

TABLE IV: Processes that grew the most for the GPT Neo system.

| Ranking | PROCCESS | Growth |
|---|---|---|
| Low / GPT Neo | | |
| 1st | GPT Neo.py | 267% |
| 2nd | snap-store | 29% |
| 3rd | gnome-terminal-server | 20% |
| 4th | systemd-journald | 6% |
| 5th | gnome-shell | 5% |
| Medium / GPT Neo | | |
| 1st | GPT Neo.py | 317% |
| 2nd | packagekitd | 54% |
| 3rd | gnome-terminal-server | 21% |
| 4th | snap-store | 18% |
| 5th | gnome-shell | 7% |
| High / GPT Neo | | |
| 1st | GPT Neo.py | 238% |
| 2nd | fwupd | 142% |
| 3rd | packagekitd | 49% |
| 4th | snap-store | 23% |
| 5th | gnome-terminal-server | 17% |

TABLE V: Description of the processes that consumed the most memory.

| Process | Description |
|---|---|
| GPT Neo.py | Process related to GPT Neo. |
| Pythia.py | Process related to Pythia. |
| Opt.py | Process related to Opt. |
| tracker-miner-fs-3 | Filesystem tracker process. |
| snap-store | Snap Store process. |
| gnome-terminal-server | GNOME Terminal Server process. |
| systemd-journald | Systemd Journal Daemon process. |
| gnome-shell | GNOME desktop environment shell. |
| packagekitd | PackageKit daemon process. |
| fwupd | Firmware update daemon process. |

As indicated by the previous analysis, the processes associated with the LLMs could be among the factors contributing to the degradation of the adopted system environments. In this analysis, we further explore the examination of these processes. Figure 2 offers a comprehensive comparison of the growth of LLM processes over time, illustrating their usage levels under low, medium, and high workloads for Pythia (Figures 2 (a), (b), and (c)), Opt (Figures 2 (d), (e), and (f)), and GPT Neo (Figures 2 (g), (h), and (i)). In all scenarios, there is a consistent trend of the processes exhibiting increasing memory consumption over the 48-hour period.

TABLE VI: Processes Memory Usage: Mann-Kendall and Sen's Slope Test Results and Mean.

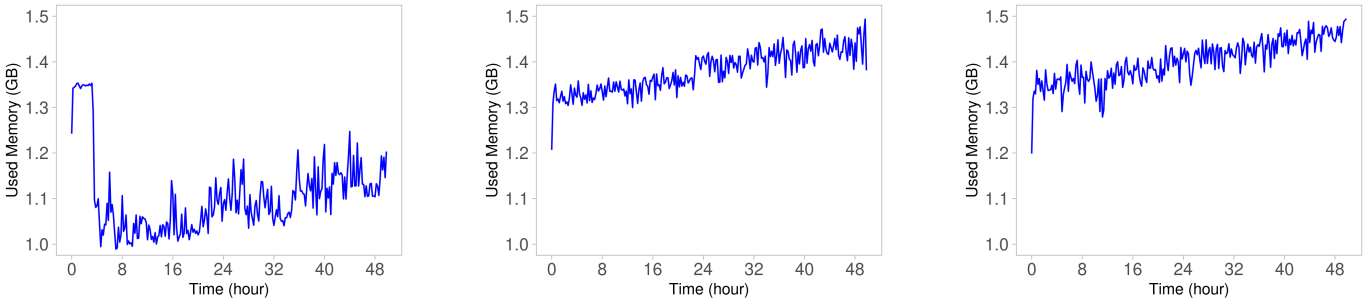
| Model | Workload | p-value | Slope | Mean |
|---|---|---|---|---|
| Pythia | Low | 2.22e-16 | 0.2208466e-3 | 1.104081 |
| | Medium | 2.22e-16 | 2.713036e-3 | 1.379944 |
| | High | 2.22e-16 | 2.726568e-3 | 1.401264 |
| Opt | Low | 2.22e-16 | 6.142107e-3 | 0.9554188 |
| | Medium | 2.22e-16 | 5.275099e-3 | 1.183878 |
| | High | 2.22e-16 | 7.085329e-3 | 1.277751 |
| GPT Neo | Low | 2.22e-16 | 2.016777e-3 | 1.108548 |
| | Medium | 2.22e-16 | 4.858558e-3 | 1.271167 |
| | High | 2.22e-16 | 4.021265e-3 | 1.278351 |

To confirm suspicions of software aging in process usage, as well as the magnitude and mean consumption of these processes, we computed the Mann-Kendall statistic, Sen's slope estimator, and the mean memory usage. Table VI presents the obtained values, confirming suspicions of software aging and a consistent trend of increasing memory usage for these processes across all scenarios. The results also reveal that Opt exhibited the highest slope rates, followed by GPT Neo, with Pythia showing the lowest rate. Additionally, it is important to highlight that although Opt had one of the highest slopes, it displayed one of the lowest average memory consumption rates. These findings align with the results obtained for RAM memory.

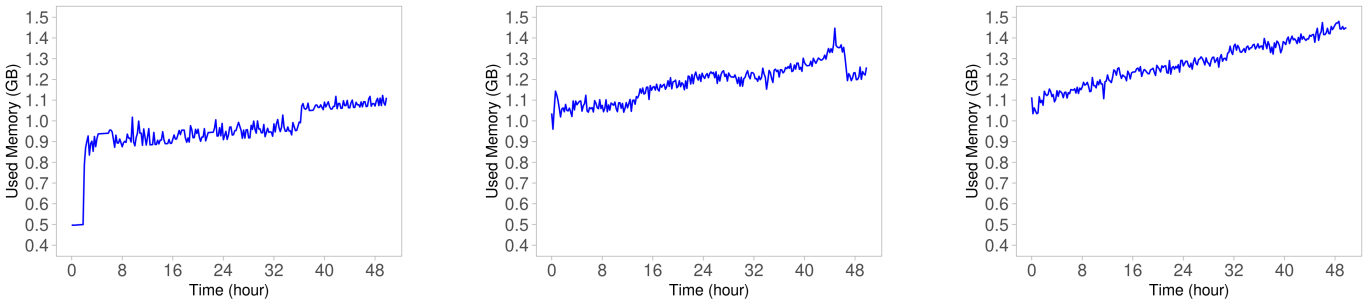
### *C. Comparison*

While our experiments were the same across different LLM systems, we found similar trends in memory degradation and identified potential causes. To address RQ3, we conducted statistical analyses to explore any differences in trends among these LLM systems. We calculated 95% confidence intervals (CI) for the slopes of both RAM usage and the memory usage of the LLM processes. Overlapping slopes indicate no significant differences between the cases.

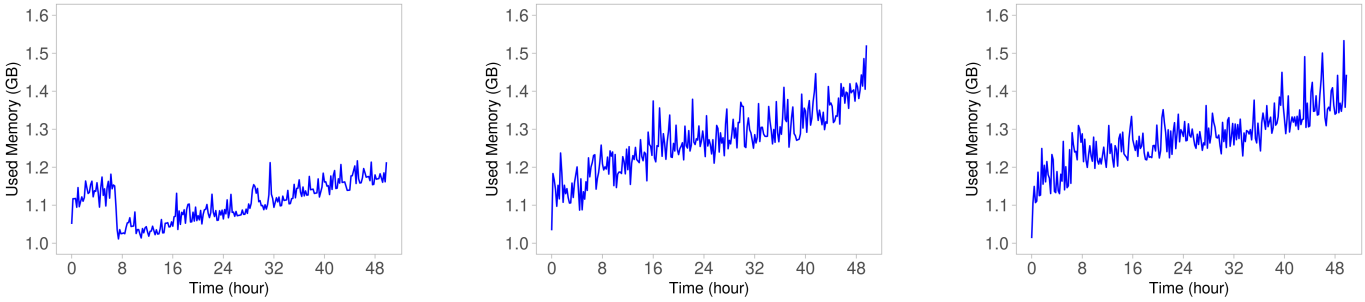
Table VII presents the 95% confidence intervals for the trends in RAM memory usage depicted in Figure 1. First, we proceed to compare the LLM systems separately. Opt exhibits overlapping trends for both low and medium workloads, while GPT Neo shows overlapping trends for medium and high workloads. In contrast, no overlapping trends were identified for Pythia. These findings suggest that Opt and GPT Neo have similar memory degradation patterns across different workload levels, while Pythia behaves differently in terms of memory




a) Low workload - Pythia b) Medium workload - Pythia c) High workload - Pythia




d) Low workload - Opt e) Medium workload - Opt f) High workload - Opt




g) Low workload - GPT Neo h) Medium workload - GPT Neo i) High workload - GPT Neo

Fig. 2: Processes Memory Consumption.

usage. When comparing trends among all systems, Opt and GPT Neo overlap for low and medium workloads. Additionally, Pythia and Opt overlap for the medium workload. These findings suggest similarities in memory degradation patterns between Opt and GPT Neo across different workloads, as well as a potential relationship between Pythia and Opt in terms of memory usage under medium workload conditions.

It is worth noting that the Opt model exhibited the highest memory degradation rates, with slopes of 6.535879e-3 for low workload, 6.325496e-3 for medium workload, and 12.63217e-3 for high workload. This suggests a more rapid increase in memory degradation over time for Opt, especially under high workloads. In contrast, Pythia and GPT Neo displayed lower and more consistent slopes across all workloads, indicating a slower rate of memory degradation compared to Opt.

Table VIII presents the 95% confidence intervals for the trends in RAM memory usage depicted in Figure 1. Pythia exhibits overlapping trends for both medium and high workloads, while no overlapping trends were identified for Opt and GPT Neo. Additionally, there is an overlap between GPT Neo and Opt for the medium workload, indicating some similarity in their RAM memory usage trends under these conditions. This suggests that while Pythia's RAM usage trends remain consistent across medium-level workloads, Opt and GPT Neo may exhibit more variability in their memory consumption

TABLE VII: Confidence intervals for the trends in RAM memory usage depicted in Figure 1.

| Model | Workload | Slope | Confidence interval |
|---|---|---|---|
| Pythia | Low | 0.874587e-3 | 0.550452e-3 1.198722e-3 |
| | Medium | 5.989796e-3 | 5.749500e-3 6.230092e-3 |
| | High | 3.355995e-3 | 3.129853e-3 3.582137e-3 |
| Opt | Low | 6.535879e-3 | 6.166154e-3 6.905605e-3 |
| | Medium | 6.325496e-3 | 6.053809e-3 6.597184e-3 |
| | High | 12.63217e-3 | 12.42881e-3 12.83553e-3 |
| GPT Neo | Low | 6.172007e-3 | 6.001180e-3 6.342834e-3 |
| | Medium | 6.736925e-3 | 6.587521e-3 6.886328e-3 |
| | High | 6.640713e-3 | 6.489275e-3 6.792151e-3 |

TABLE VIII: Confidence interval of the trends for the LMM processes illustrated in Figure 2.

| Model | Workload | Slope | Confidence interval | |
|---|---|---|---|---|
| Pythia | Low | 0.2208466e-3 | -0.1057519e-3 | 0.5474451e-3 |
| | Medium | 2.713036e-3 | 2.485960e-3 | 2.940113e-3 |
| | High | 2.726568e-3 | 2.497172e-3 | 2.955964e-3 |
| Opt | Low | 6.142107e-3 | 5.887276e-3 | 6.396939e-3 |
| | Medium | 5.275099e-3 | 5.057626e-3 | 5.492572e-3 |
| | High | 7.085329e-3 | 6.917412e-3 | 7.253246e-3 |
| GPT Neo | Low | 2.016777e-3 | 1.850319e-3 | 2.183235e-3 |
| | Medium | 4.858558e-3 | 4.549498e-3 | 5.167617e-3 |
| | High | 4.021265e-3 | 3.710932e-3 | 4.331598e-3 |

patterns across different workloads. It is worth noting that the Opt process exhibited the highest memory degradation rates, followed by GPT Neo and Pythia processes.

### D. Threats to validity

Threats to the validity of this work are described below.

- **Generalizability of the Experimental Study.** As with any experimental investigation, our findings might not readily apply to all LMM system environments. However, the analysis procedure and experimental architecture demonstrated in this work are easily adaptable to various other LLM system environments. Researchers interested in this field can replicate our study using the same methodology.
- **Limitation of Duration and Number of Experimental Runs.** Software aging experiments are time-intensive, and like any experimental work, they are subject to constraints on duration. While our tests ran for 48 hours under the selected workloads, it's essential to recognize that this timeframe may not have captured all degradation trends.
- **Representativeness of the adopted workload.** The workloads considered in our study consist of three levels of constant intensities (low, medium, and high) that may not represent real user behaviors. Since the aim of this research was to expose potential software aging trends in the studied system environments, we believe this does not significantly limit our observations.

## V. Conclusion

LLM systems have been widely used in the world in recent years, varying from natural language processing tasks such as text generation and sentiment analysis to more complex applications like machine translation and question-answering systems. This work aimed to investigate and compare software aging symptoms in LLM systems. To achieve this, the Mann-Kendall test and Sen's slope estimator were used to perform a trend analysis on the collected data. The results obtained in this work show indications of software aging in the adopted systems. The results revealed that various processes, including *fwupd*, *packagekitd*, and *snap-store* as well as processes related to LLM systems, exhibited a growth in memory consumption over time. Additionally, the results suggest that Opt demonstrates a higher susceptibility to software aging compared to Pythia and GPT Neo due to its elevated utilization of RAM resources. As for future work, we plan to explore other LMMs and consider additional aging indicators such as cache and energy consumption. Additionally, we aim to investigate whether software aging in LLMs leads to hallucinations.


**César Santos** is an undergrad student at the Federal University of Pernambuco, Brazil. His research interest is focused in Artificial Inteligence, Performance, and Software Aging.

**Fumio Machida** is an associate professor of Computer Science Department in University of Tsukuba. He was a principal researcher at NEC Corporation until 2019. He received his Ph.D. in 2018 from the Department of Mathematical and Computing Science, Tokyo Institute of Technology. He was a recipient of the young scientists' prize of Japan in 2014. He was a visiting scholar in the Department of Electrical and Computer Engineering at Duke University in 2010. His research interests include modeling and analysis of system dependability, software aging and rejuvenation, and reliability of machine learning systems.

**Ermeson Andrade** is an associate professor at the Department of Computing at the Federal Rural University of Pernambuco, Brazil. In 2014, he completed his PhD in Computer Science from the Federal University of Pernambuco. His research interest is focused in Performance, Dependability, and Modeling.

## REFERENCES


[1] M. Kraus, J. A. Bingler, M. Leippold, T. Schimanski, C. C. Senni, D. Stammbach, S. A. Vaghefi, and N. Webersinke, "Enhancing large language models with climate resources," *arXiv preprint arXiv:2304.00116*, 2023.

[2] S. Biderman, H. Schoelkopf, Q. G. Anthony, H. Bradley, K. O'Brien, E. Hallahan, M. A. Khan, S. Purohit, U. S. Prashanth, E. Raff *et al.*, "Pythia: A suite for analyzing large language models across training and scaling," in *International Conference on Machine Learning*. PMLR, 2023, pp. 2397–2430.

[3] K. S. Trivedi, M. Grottke, and E. Andrade, "Software fault mitigation and availability assurance techniques," *International Journal of System Assurance Engineering and Management*, vol. 1, pp. 340–350, 2010.

[4] E. Andrade, F. Machida, R. Pietrantuono, and D. Cotroneo, "Memory degradation analysis in private and public cloud environments," in *2021 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE, 2021, pp. 33–39.

[5] P. P. Ray, "Benchmarking, ethical alignment, and evaluation framework for conversational ai: advancing responsible development of chatgpt," *BenchCouncil Transactions on Benchmarks, Standards and Evaluations*, vol. 3, no. 3, p. 100136, 2023.

[6] J. Hoffmann, S. Borgeaud, A. Mensch, E. Buchatskaya, T. Cai, E. Rutherford, D. d. L. Casas, L. A. Hendricks, J. Welbl, A. Clark *et al.*, "Training compute-optimal large language models," *arXiv preprint arXiv:2203.15556*, 2022.

[7] W. X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Dong *et al.*, "A survey of large language models," *arXiv preprint arXiv:2303.18223*, 2023.

[8] Y. Chang, X. Wang, J. Wang, Y. Wu, L. Yang, K. Zhu, H. Chen, X. Yi, C. Wang, Y. Wang *et al.*, "A survey on evaluation of large language models," *ACM Transactions on Intelligent Systems and Technology*, 2023.

[9] A. Srivastava, A. Rastogi, A. Rao, A. A. M. Shoeb, A. Abid, A. Fisch, A. R. Brown, A. Santoro, A. Gupta, A. Garriga-Alonso *et al.*, "Beyond the imitation game: Quantifying and extrapolating the capabilities of language models," *arXiv preprint arXiv:2206.04615*, 2022.

[10] N. Moghe, E. Razumovskaia, L. Guillou, I. Vulić, A. Korhonen, and A. Birch, "Multi3nlu++: A multilingual, multi-intent, multi-domain dataset for natural language understanding in task-oriented dialogue," *arXiv preprint arXiv:2212.10455*, 2022.

[11] P. Liang, R. Bommasani, T. Lee, D. Tsipras, D. Soylu, M. Yasunaga, Y. Zhang, D. Narayanan, Y. Wu, A. Kumar *et al.*, "Holistic evaluation of language models," *arXiv preprint arXiv:2211.09110*, 2022.

[12] M. Li, Y. Zhang, Z. Li, J. Chen, L. Chen, N. Cheng, J. Wang, T. Zhou, and J. Xiao, "From quantity to quality: Boosting llm performance with self-guided data selection for instruction tuning," *arXiv preprint arXiv:2308.12032*, 2023.

[13] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. Gonzalez, H. Zhang, and I. Stoica, "Efficient memory management for large language model serving with pagedattention," in *Proceedings of the 29th Symposium on Operating Systems Principles*, 2023, pp. 611–626.

[14] E. Andrade, R. Pietrantuono, F. Machida, and D. Cotroneo, "A comparative analysis of software aging in image classifiers on cloud and edge," *IEEE Transactions on Dependable and Secure Computing*, 2021.

[15] D. Dias, F. Machida, and E. Andrade, "Software aging analysis in a testing framework," in *2023 IEEE 34th International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE, 2023, pp. 222–229.

[16] K. Watanabe, F. Machida, E. Andrade, R. Pietrantuono, and D. Cotroneo, "Software aging in a real-time object detection system on an edge server," in *Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing*, 2023, pp. 671–678.

[17] D. Dias, F. Machida, and E. Andrade, "Analysis of software aging in a blockchain platform," in *2022 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE, 2022, pp. 170–177.

[18] H. B. Mann, "Nonparametric tests against trend," *Econometrica: Journal of the econometric society*, pp. 245–259, 1945.

[19] P. K. Sen, "Estimates of the regression coefficient based on kendall's tau," *Journal of the American statistical association*, vol. 63, no. 324, pp. 1379–1389, 1968.

[20] F. Machida, A. Andrzejak, R. Matias, and E. Vicente, "On the effectiveness of mann-kendall test for detection of software aging," in *2013 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE, 2013, pp. 269–274.