Brilliance
Research of Artificial Intelligence

E-ISSN : 2807-9035
Volume 3, **Number** 2, November 2023
https://doi.org/10.47709/brilliance.v3i2.3275

Design and Implementation Plagiarism Checker Application with DetectGPT using Scheduler Algorithm

Arif Supriyanto[1], Hendrik Setyo Utomo[2], Oky Rahmanto[3*]
[1,2,3]Politeknik Negeri Tanah Laut, Indonesia
[1]arif@politala.ac.id, [2]hendrik.tomo@politala.ac.id, [3]oky.rahmanto@politala.ac.id,

open access

***Corresponding Author**

Article History:
Submitted: 10-12-2023
Accepted: 15-12-2023
Published: 26-12-2023

Keywords:
DetectGPT, AI, worker, scheduler, plagiarism

ABSTRACT

The increasing use of Generative AI in the field of writing in the academic world creates problems in the field of plagiarism. this problem has prompted an urgent need for plagiarism detection tools. However, not all universities are able to implement a system that is able to detect sentences or paragraphs resulting from generative AI. Recent research seeks to overcome this obstacle using AI itself, specifically through the DetectGPT model. However, the implementation of this technology has limitations, such as requiring large computing resources and quite a long examination time. Using a GPU can speed up the process, but not all users can implement it. Solutions to minimize processing time include using more than one worker, however, scheduling is essential to maximize plagiarism detection efficiency. This research proposes the implementation of the above system with the help of the scheduler. The results obtained in this system prototype are an average waiting time for checking of 457,778 seconds to complete 10 tasks with the help of 3 workers running at the same time.

INTRODUCTION

Plagiarism through generative AI has become a serious problem in the academic world (Khalil and Er 2023), where generating text using technology such as GPT (Generative Pre-trained Transformer) can be considered a form of unethical copying. Several commonly used tools, namely iThenticate and Turnitin, are considered less capable of detecting text resulting from GPT (Khalil and Er 2023). To overcome this problem, several approaches have been developed, one of which is using a plagiarism detection tool called DetectGPT. The main advantage of DetectGPT lies in its ability to identify and distinguish text generated by generative AI (Mitchell et al. 2023), thereby enabling academic institutions to effectively track and prevent such acts of plagiarism.

Although DetectGPT can be considered an effective solution, its implementation is not always smooth. One of the main obstacles is the need for specific and quite large computing resources, which is one of the characteristics of AI itself. This is often a challenge, especially for academic institutions or researchers with limited budgets or technological infrastructure. Therefore, this research aims to overcome this problem by designing a DetectGPT implementation that can run with resources that can be divided into smaller amounts, allowing flexibility and scalability in detecting plagiarism using generative AI.

Practical and measurable steps need to be taken in the DetectGPT implementation process so that it can operate optimally with limited resources. By breaking down resource requirements into smaller units which are then called workers, it is hoped that academic institutions and researchers can access this plagiarism detection tool without having to face obstacles in the form of limited resources. In this way, increased academic security and protection of intellectual integrity can be achieved without sacrificing technological efficiency or sustainability.

LITERATURE REVIEW

This section discusses the use of several scientific knowledge used to solve the problems of this research. Knowledge includes DetectGPT, Scheduling in Cloud Computing, Workers in Docker Container and First Come First Serve Scheduling Algorithms.

DetectGPT

DetectGPT is a plagiarism detection tool developed to identify and distinguish text generated by generative AI (Mitchell et al. 2023). This tool uses a probability curve from a model probability function to determine whether a text was generated by generative AI or not (Mitchell et al. 2023). DetectGPT does not require separate classification training, collection of original or generated text datasets, or watermarking of generated text (Mitchell et al. 2023). The tool simply uses the log of probabilities(Yang et al. 2023), calculated by the model of interest and random noise of text from another generic pre-trained language model (Mitchell et al. 2023).

Because the nature of DetectGPT does not require training a separate classifier model, DetectGPT can be used directly without spending resources on training for each change in the dataset. This fits well with the limitations of this

Brilliance
Research of Artificial Intelligence

E-ISSN : 2807-9035
Volume 3, **Number** 2, November 2023
https://doi.org/10.47709/brilliance.v3i2.3275

research which requires low resources. Apart from that, DetectGPT can be run without using a special component in the form of a GPU which is usually needed to speed up model training

Scheduling in Cloud Computing

Scheduling in cloud computing is a crucial aspect that has been studied extensively in recent years. According to (Murad et al. 2022)., scheduling is the process of mapping tasks to resources in a way that optimizes one or more objectives. Various algorithms have been proposed to solve scheduling problem. One such algorithm is the Genetic Algorithm (GA), which is a metaheuristic optimization algorithm that mimics the process of natural selection. Another algorithm is the Particle Swarm Optimization (PSO) algorithm, which is a population-based optimization algorithm inspired by the social behavior of birds and fish (Murad et al. 2022). These algorithms have been shown to be effective in solving the scheduling problem in cloud computing.

In addition to these algorithms, researchers have also proposed various scheduling policies to improve the performance of cloud computing systems. One such policy is the First-Come-First-Serve (FCFS) policy, which is a non-preemptive scheduling policy that schedules tasks in the order in which they arrive (Siahaan 2016). Another policy is the Round-Robin (RR) policy, which is a preemptive scheduling policy that allocates a fixed time slice to each task in a cyclic order. Researchers have also proposed hybrid scheduling policies that combine the advantages of different policies to achieve better performance (Siahaan 2016).

Scheduling in cloud computing is not only important for optimizing resource utilization but also for ensuring Quality of Service (QoS) for cloud users. According to a study published in IEEE Transactions on Cloud Computing, QoS-aware scheduling policies can significantly improve the performance of cloud computing systems (He 2016). The study proposed a QoS-aware scheduling policy that considers the deadline, priority, and resource requirements of tasks to schedule them effectively 3. The policy was evaluated using a simulation, and the results showed that it outperformed other scheduling policies in terms of QoS metrics such as response time and throughput (He 2016).

Worker in Docker container

Worker in Docker container for Cloud computing is an important concept in modern software development. Docker containers are isolated environments that allow developers to run applications anywhere without worrying about compatibility or dependency issues (Singh and Singh 2016). Workers, on the other hand, are basic units of processing that perform specific tasks in a system (Marathe, Gandhi, and Shah 2019). In other words, the concept worker containers for Cloud computing refers to the use of Docker containers to run workers in a cloud computing environment (Bernstein 2014).

In this context, Docker containers allow developers to create isolated environments to run workers (Hardikar, Ahirwar, and Rajan 2021). These environments allow workers to run with isolated and limited resources, thereby allowing developers to optimize application performance. In addition, Docker containers also allow developers to manage application dependencies more effectively, making application development and testing easier

Using Docker containers to run workers in a cloud computing environment has several advantages(Sharma, Saxena, and Singh 2020). First, Docker containers allow developers to create isolated environments for running workers, thereby allowing developers to optimize application performance. Second, Docker containers allow developers to manage application dependencies more effectively, making application development and testing easier. Third, Docker containers allow developers to run workers anywhere, without worrying about compatibility or dependency issues.

First Come First Serve Algorithm

FCFS or FIFO can be defined as a process that arrives first will be served first. If there is a process to arrive at the same time, the services they carried through their order in the queue. The process in the queue behind had to wait until all the process in front of him is complete. Any process that is on ready status put in FCFS queue according to the time of arrival. (Siahaan 2016). The advantage of the FCFS algorithms is that they are simple and easy to implement. Because this algorithm does not require much interaction between processes, it is suitable for systems that do not require fast response times. Additionally, this algorithm ensures that each job/task is processed in the same order as their arrival into the queue.

METHOD

The flow of this research follows waterfall SDLC software development (Olorunshola and Ogwueleka 2022). There are Requirements Analysis; This stage aims to identify user needs and determine the features required in the software. At this stage, observations and literature studies are carried out to understand the needs of the system to be produced. Design; This stage aims to design the software architecture and determine the technology that will be used. At this stage, user interface design, database design, and software architecture design are carried out. Software Development; This stage aims to implement the software according to the design that has been made. At this stage, program code creation, program code integration, and unit testing are carried out. Implementation; This stage aims to

Brilliance
Research of Artificial Intelligence
E-ISSN : 2807-9035
Volume 3, Number 2, November 2023
https://doi.org/10.47709/brilliance.v3i2.3275

install the software in the production environment. At this stage, software installation is carried out on the server and software configuration. The result of this stage is software that is ready to use. Testing; This stage aims to ensure that the software created functions well and meets user needs. At this stage, functional testing, integration testing and system testing are carried out. The result of this stage is software that has been tested and is ready to use.

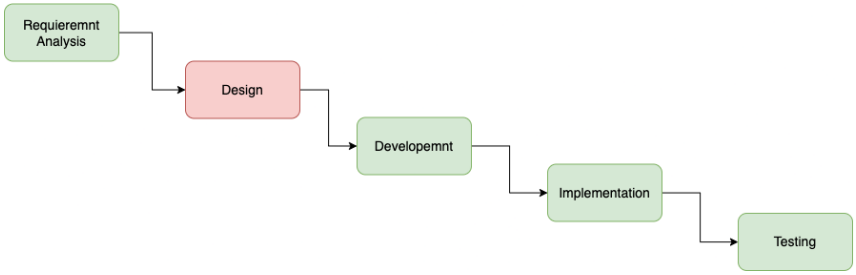

Fig. 1 Research flow

In the context of implementation, using containers as the basis or place for workers to run is a strategic choice. Containers were chosen because they provide adequate isolation while allowing shared access to required resources. Implementation with containers can be illustrated through worker and scheduler images. Workers are entities that carry out tasks in containers, while schedulers are responsible for managing scheduling and resource allocation for workers(Ibrahim and Al 2021). The use of containers provides advantages in terms of isolation, where each worker operates independently but can share the necessary resources and also give cross-OS compatibility (Saeki et al. 2020).

In this research, the algorithm chosen for the implementation process is First-Come-First-Serve (FCFS). The selection of FCFS (Yates and Kaul 2019) is based on the consideration that in plagiarism checking, the time required for each task cannot be predicted in advance. FCFS is considered appropriate because it gives each worker an equal opportunity to carry out available tasks. The main advantage of FCFS is that it is simple and fair, where the first incoming task is executed first.

The FCFS sequence stage begins by accepting tasks or jobs that are entered into the queue. Then, the available workers will carry out these tasks in the order they arrive. Each task is completed before the worker can move on to the next task in the queue. This approach ensures that each task is given fair processing time and that each worker has an equal opportunity to complete the task.

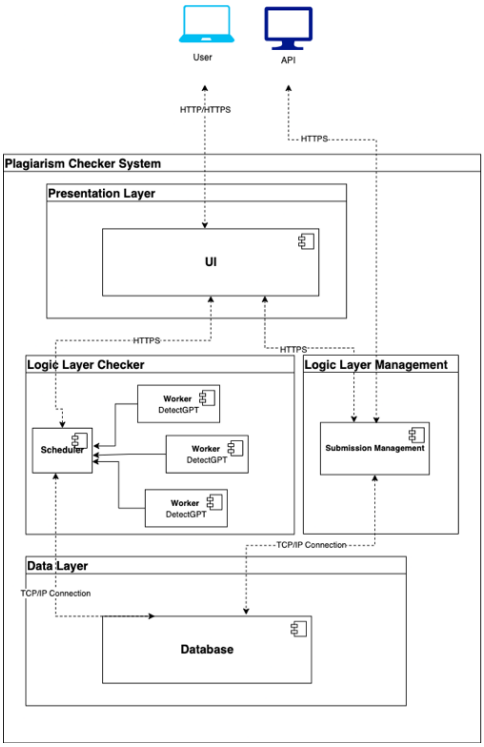

Fig. 2 Application Architecture

Brilliance
Research of Artificial Intelligence

E-ISSN : 2807-9035
Volume 3, **Number** 2, November 2023
https://doi.org/10.47709/brilliance.v3i2.3275

RESULT

Test conducted on computer with 32 GB of memory and 1 TB storage and no GPU is used. The worker were implemented and runs inside docker container. Containerization of worker were choosen because it gives more interoperability for worker to be runs in any Operating System that support containerization. It also give more chances to duplicate the worker as long as the computer still have resources or do horizontal scaling. The example text used for test were about 260 words and about 12 sentences. Three workers were deployed.

Table 1. Result of checker total time

Iteration	Jobs Worker 1	Jobs Worker 2	Jobs Worker 3	Average Time (s)
Iteration 1	4	3	3	467.547
Iteration 2	4	3	3	449.396
Iteration 3	4	3	3	452.783
Iteration 4	4	3	3	448.624
Iteration 5	4	3	3	448.338
Iteration 6	4	3	3	448.910
Iteration 7	4	3	3	464.350
Iteration 8	4	3	3	462.683
Iteration 9	4	3	3	466.563
Iteration 10	4	3	3	468.584
Average	**4**	**3**	**3**	**457.778**

DISCUSSION

The conducted test on a computer with 32 GB of memory and 1 TB storage, utilizing three workers implemented within Docker containers, provides valuable insights into the system's performance. The absence of GPU usage suggests a reliance on CPU and memory resources. The decision to containerize workers is strategic, offering increased interoperability across different operating systems supporting containerization. This approach allows for seamless execution in diverse environments and provides the flexibility to duplicate workers, ensuring adaptability as long as computer resources permit or for horizontal scaling when needed.

The presented table, outlining the results of the checker's total time across ten iterations, reflects a consistent workload distribution among the three workers. Each worker consistently completes four jobs in each iteration, indicating a balanced allocation of tasks. The average time for job completion across all workers and iterations is approximately 457.778 seconds, providing a baseline for the overall efficiency of the system.

The choice of Docker containers proves advantageous not only for cross-OS compatibility but also for the ease of duplication and potential horizontal scaling. This flexibility aligns with the evolving demands of computing environments and workload variations. As a foundation, this data provides a starting point for further analysis and optimization, ensuring the adaptability of the system to changing requirements and enhancing its overall efficiency. Ongoing monitoring and potential adjustments will be essential for maintaining optimal performance as the system evolves over time.

CONCLUSION

This research results showcase the efficacy of the implemented workers within Docker containers on a computer with 32 GB of memory and 1 TB storage, revealing consistent job completion and a well-distributed workload. The decision to leverage containerization without GPU usage aligns with a strategy emphasizing adaptability and ease of duplication. The stability observed in the average time across iterations suggests a reliable and consistent performance of the workers. Monitoring resource utilization during the test would be crucial for understanding the scalability of the solution and identifying potential areas for optimization.

However, it is essential to acknowledge the limitation that further testing is warranted, especially exploring the performance under alternative scheduling algorithms such as round-robin or others. This would provide a more comprehensive understanding of the system's versatility and guide potential optimizations for varied workloads and scheduling scenarios.

ACKNOWLEDGMENT

This research is funded by Research by DIPA Fund Lecturers at Politeknik Negeri Tanah Laut.

Brilliance
Research of Artificial Intelligence

E-ISSN : 2807-9035
Volume 3, **Number** 2, November 2023
https://doi.org/10.47709/brilliance.v3i2.3275

REFERENCES

Bernstein, David. 2014. "Containers and Cloud: From LXC to Docker to Kubernetes." *IEEE Cloud Computing* 1(3):81–84. doi: 10.1109/MCC.2014.51.

Hardikar, Sanjay, Pradeep Ahirwar, and Sameer Rajan. 2021. "Containerization: Cloud Computing Based Inspiration Technology for Adoption through Docker and Kubernetes." Pp. 1996–2003 in *2021 Second International Conference on Electronics and Sustainable Communication Systems (ICESC)*.

He, Hong. 2016. "A Resource Reservation Based Framework for QoS-Aware Resource Provision in Cloud Computing." *International Journal of Grid and Distributed Computing* 9:193–204. doi: 10.14257/ijgdc.2016.9.9.17.

Ibrahim, Ibrahim Mahmood, and Et Al. 2021. "Task Scheduling Algorithms in Cloud Computing: A Review." *Turkish Journal of Computer and Mathematics Education (TURCOMAT)* 12(4):1041–53.

Khalil, Mohammad, and Erkan Er. 2023. "Will ChatGPT Get You Caught? Rethinking of Plagiarism Detection."

Marathe, Nikhil, Ankita Gandhi, and Jaimeel M. Shah. 2019. "Docker Swarm and Kubernetes in Cloud Computing Environment." Pp. 179–84 in *2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI)*.

Mitchell, Eric, Yoonho Lee, Alexander Khazatsky, Christopher D. Manning, and Chelsea Finn. 2023. "DetectGPT: Zero-Shot Machine-Generated Text Detection Using Probability Curvature."

Murad, Saydul Akbar, Abu Jafar Md Muzahid, Zafril Rizal M. Azmi, Md Imdadul Hoque, and Md Kowsher. 2022. "A Review on Job Scheduling Technique in Cloud Computing and Priority Rule Based Intelligent Framework." *Journal of King Saud University - Computer and Information Sciences* 34(6, Part A):2309–31. doi: 10.1016/j.jksuci.2022.03.027.

Olorunshola, Oluwaseyi Ezekiel, and Francisca Nonyelum Ogwueleka. 2022. "Review of System Development Life Cycle (SDLC) Models for Effective Application Delivery." Pp. 281–89 in *Information and Communication Technology for Competitive Strategies (ICTCS 2020), Lecture Notes in Networks and Systems*, edited by A. Joshi, M. Mahmud, R. G. Ragel, and N. V. Thakur. Singapore: Springer.

Saeki, Takaya, Yuichi Nishiwaki, Takahiro Shinagawa, and Shinichi Honiden. 2020. "A Robust and Flexible Operating System Compatibility Architecture." Pp. 129–42 in *Proceedings of the 16th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, VEE '20*. New York, NY, USA: Association for Computing Machinery.

Sharma, Vivek, Harsh Kumar Saxena, and Akhilesh Kumar Singh. 2020. "Docker for Multi-Containers Web Application." Pp. 589–92 in *2020 2nd International Conference on Innovative Mechanisms for Industry Applications (ICIMIA)*.

Siahaan, Andysah Putera Utama. 2016. "Comparison Analysis of CPU Scheduling FCFS, SJF and Round Robin." doi: 10.31227/osf.io/6dq3p.

Singh, Sachchidanand, and Nirmala Singh. 2016. "Containers & Docker: Emerging Roles & Future of Cloud Technology." Pp. 804–7 in *2016 2nd International Conference on Applied and Theoretical Computing and Communication Technology (iCATccT)*.

Yang, Xianjun, Kexun Zhang, Haifeng Chen, Linda Petzold, William Yang Wang, and Wei Cheng. 2023. "Zero-Shot Detection of Machine-Generated Codes."

Yates, Roy D., and Sanjit K. Kaul. 2019. "The Age of Information: Real-Time Status Updating by Multiple Sources." *IEEE Transactions on Information Theory* 65(3):1807–27. doi: 10.1109/TIT.2018.2871079.