applied sciences

MDPI

Article

Self-Adaptable Software for Pre-Programmed Internet Tasks: Enhancing Reliability and Efficiency

Mario Martínez García [1], Luis Carlos G. Martínez Rodríguez [2] and Ricardo Pérez Zúñiga [3,*]

[1] Universidad de Guadalajara Campus CUValles, Computer Sciences and Engineering Department, Ameca Highway Km. 45.5, Ameca 46600, Jalisco, Mexico; mario.mgarcia@academicos.udg.mx
[2] Universidad de Guadalajara Campus CUCEI, Computer Science Department, Blvd. Gral. Marcelino García Barragán 1421, Olímpica, Guadalajara 44430, Jalisco, Mexico; luis.martinez4092@alumnos.udg.mx
[3] Universidad de Guadalajara, System of Virtual University Department, Guadalajara Street Mezquitán No. 302. Central Colony Barranquitas, Guadalajara 44100, Jalisco, Mexico
* Correspondence: perezuniga@udgvirtual.udg.mx; Tel.: +52-3328100572

Abstract: In the current digital landscape, artificial intelligence-driven automation has revolutionized efficiency in various areas, enabling significant time and resource savings. However, the reliability and efficiency of software systems remain crucial challenges. To address this issue, a generation of self-adaptive software has emerged with the ability to rectify errors and autonomously optimize performance. This study focuses on the development of self-adaptive software designed for pre-programmed tasks on the Internet. The software stands out for its self-adaptation, automation, fault tolerance, efficiency, and robustness. Various technologies such as Python, MySQL, Firebase, and others were employed to enhance the adaptability of the software. The results demonstrate the effectiveness of the software, with a continuously growing self-adaptation rate and improvements in response times. Probability models were applied to analyze the software's effectiveness in fault situations. The implementation of virtual cables and multiprocessing significantly improved performance, achieving higher execution speed and scalability. In summary, this study presents self-adaptive software that rectifies errors, optimizes performance, and maintains functionality in the presence of faults, contributing to efficiency in Internet task automation.

Keywords: software; self adaptable; internet; tasks; artificial intelligence

check for updates

Citation: Martínez García, M.; Martínez Rodríguez, L.C.G.; Pérez Zúñiga, R. Self-Adaptable Software for Pre-Programmed Internet Tasks: Enhancing Reliability and Efficiency. *Appl. Sci.* **2024**, *14*, 6827. https://doi.org/10.3390/app14156827

Academic Editors: Rui Araújo and Lykourgos Magafas

Received: 13 June 2024
Revised: 16 July 2024
Accepted: 29 July 2024
Published: 5 August 2024

Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

In the rapidly evolving digital landscape, software systems play a crucial role in automating various tasks on the Internet [1]. However, ensuring the reliability and efficiency of such systems remains a significant challenge. The emergence of errors or failures in their code can hinder their proper functioning, leading to service disruptions or incorrect results. To address this issue, a new generation of software has emerged that offers self-adaptive capabilities [2–5] that can rectify errors and optimize performance automatically. In this article, we present an innovative software solution that not only executes pre-programmed tasks on the Internet but also has the ability to autonomously reconfigure itself in the face of code failures.

As technology advances, the importance of robust and reliable software systems cannot be underestimated [6]. Organizations heavily rely on automated processes to optimize their operations, ranging from data collection and analysis to web scraping [7–10] and information retrieval. Any disruption or malfunction in these critical processes can result in substantial financial losses, compromise data integrity, and diminish the user experience.

The current state of research in the field of self-adaptive software systems [11,12] underscores the need for innovative approaches to ensure the reliable and efficient execution of pre-programmed tasks. While traditional software often requires manual intervention

for debugging and updates, the concept of self-adaptation promises to mitigate these challenges [13]. By integrating intelligent mechanisms such as fault detection, diagnosis, and recovery into the software architecture, self-adaptive systems can dynamically respond to emerging errors and autonomously adjust their behavior to ensure continuous functionality [14,15].

Currently, we are in the era of automation for repetitive processes. Using artificial intelligence (AI) techniques contributes to saving time and resources that can be utilized for other purposes [16].

Within the field of AI, there are a number of commonly used learning methods, including neural networks [17] and expert systems. We will base our later discussion on the latter. Neural networks, as the name suggests, are designed to mimic the behavior of neurons, achieved by programming layers on demand. On the other hand, expert systems operate through pre-programmed logical reasoning [18], like a program that replicates the judgment of a human expert.

Imagine for a moment that you need to extract information from a web page to store it in a database. Hiring people could be an option, but it would not be very cost-effective or efficient if you want something quick and economical. So, programming an 'expert system' with pre-programmed logic [19] to do it for you is a better choice. However, what if the web page changes some elements of its structure? Your software would no longer work, and you would have to reprogram it.

This is where SiReTAAPP Version 1.0, created by Mario Martínez García and Luis Carlos Guadalupe Martínez Rodriguez, in Ameca, Jalisco. (Self-Adjusting Pre-Programmed Task Execution System) comes in. It is a web-based automated work software [20–22] that self-modifies based on changes or errors that have been pre-configured. It functions by having multiple processes working concurrently in parallel, monitoring the software. It utilizes a multi-master-single-slave communication architecture, where the slave, using the allowed processor cores, performs various tasks simultaneously, while the masters monitor the processes. In the event of a failure, the slave reports the error to the master modules. The master modules, with pre-programmed error information, resolve the conflict, rewrite the code of the slave module with the repaired error, and set it in motion again [23–25]. Depending on the work the slave will perform, there is a series of steps that must be followed to train the master modules, which are as follows:

- Pre-programming Work Phase
- Debugging and Error Storage Phase
- Polishing and Common Error Phase
- Monitoring Startup Phase

Once these four phases are completed, the software will be fully trained to work and resolve errors on its own. Figure 1 shows the basic operation diagram of the SiReTAAPP software.

The pre-programming phase of work is when the user enters information into the slave module program for the task they want to perform for subsequent analysis.

The debugging and error storage phase is the stage where the master modules will run the slave module, evaluate its performance, detect errors, and save each error with a unique ID for the future.

Once the second phase is completed, the polishing and frequent error phase begins. This is where the AI is trained to find ways to resolve the errors that occurred during the execution of the slave module so that if the error occurs again in the future, the AI knows how to resolve it. Errors are stored in a database from which the user can view and resolve each one. Once the error is identified, it is classified. Subsequently, using predefined abstractions—where an abstraction involves the identification of certain characteristics that define an object or an action, and represents a sequence of actions to follow in order to resolve it—the machine is informed about how that error was resolved. The structure of the IRP (Information and Problem Resolution) modules is organized as follows in Figure 2.

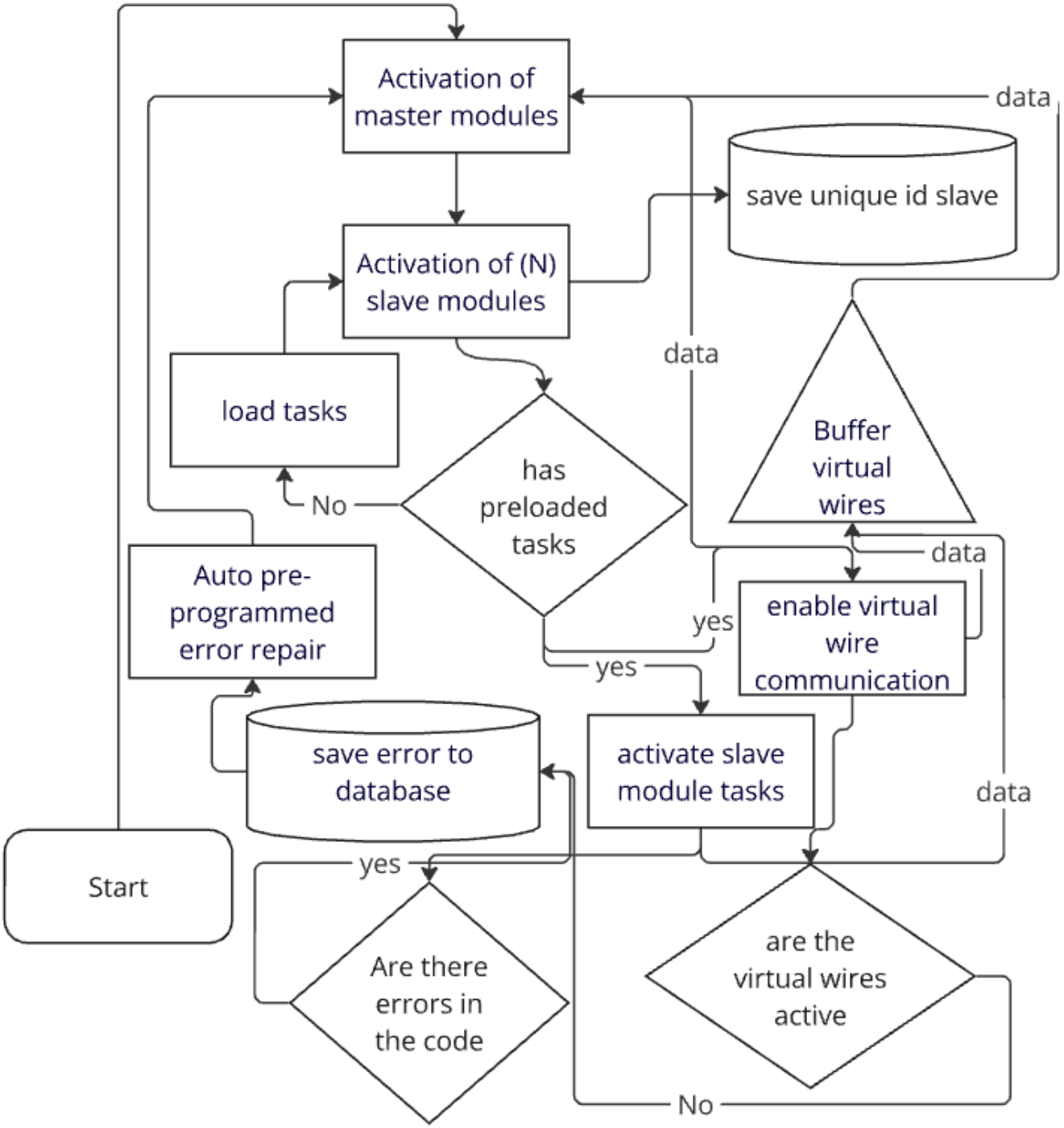


Figure 1. Basic operating diagram of the system.

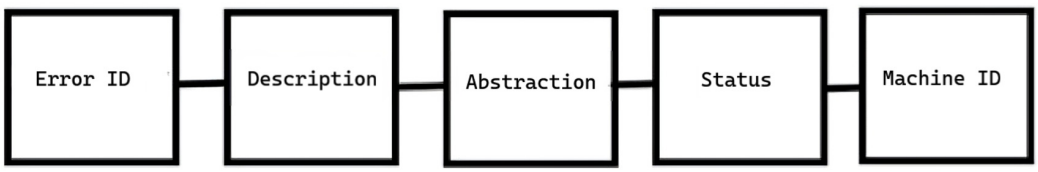


Figure 2. Sequence Diagram of Error Saving in the Software.

The error ID is the unique identification number for each error, which the machine can use to apply the abstraction that resolves that specific error. Abstractions are pre-programmed steps that the machine follows to resolve the error. They can range from inspecting the web page you are working on to changing the reference values used by the slave module, searching and downloading necessary drivers, and rewriting the slave module.

The description is the error message, the status indicates whether the error has already been resolved, and the machine ID is the identifier in case there are multiple computers in operation.

Table 1 describes the main features of the proposal, and Table 2 outlines the advantages of the proposal.

Table 1. Characteristics of the self-adaptive software proposed.

Feature	Description
Self-adaptation	Capability to reconfigure and adjust automatically in the face of code failures
Automation	Automation of pre-programmed tasks on the Internet
Fault Tolerance	Capacity to withstand and recover from emerging errors
Efficiency	Optimization of system performance and efficiency
Robustness	Capacity to maintain continuous functionality even in the presence of unforeseen errors

Table 2. Advantages of Self-Adaptive Software Systems.

Advantage	Description
Performance Improvement	Autonomous adjustment of software behavior to ensure continuous functionality
Fault Tolerance	Enhanced ability to withstand and recover from errors, avoiding service interruptions
Increased Reliability	Reduction in errors and failures, enhancing user experience and data integrity
Efficient Automation	Optimization of automated operations, from data collection and analysis to information retrieval
Scalability	Increased distributed processing capacity to accelerate response times and enhance system scalability

In this context, our work aims to contribute to the advancement of self-adaptive software systems by introducing an innovative solution that can effectively address code failures in pre-programmed Internet tasks. Leveraging cutting-edge techniques from the fields of artificial intelligence [26,27], machine learning [28,29], and automated reasoning [30,31], our software possesses the unique ability to self-rewrite its code when it encounters anomalies. By reconfiguring its underlying logic, the software ensures the continuous execution of pre-programmed tasks, even in the presence of unforeseen errors.

The main objective of this work is to present the architecture of our self-adaptive software, emphasizing its unique features and discussing its potential applications. Furthermore, we will showcase the results of comprehensive experiments, demonstrating the effectiveness and reliability of the software in real-world scenarios. Our findings indicate that self-adaptation significantly enhances fault tolerance and overall performance of pre-programmed Internet tasks [32,33], making it a promising avenue for future advancements in the field.

Moreover, this software distinguishes itself by its ability to work with multi-task multiprocessing modules [34,35], which implies greater efficiency and performance in executing pre-programmed tasks. Harnessing distributed processing capability [36,37] accelerates response times and enhances system scalability, which is particularly relevant in high-performance environments and applications that require intensive data processing.

Currently, many services are offered online; however, using them can become tedious, cumbersome, and overwhelming for users. Additionally, these services are often provided by third parties, such as hospitals, where you need to schedule an appointment to donate blood, and for that, you have to provide personal information. Similarly, the tax administration office requires you to make an appointment and enter personal data, only to find out that there are no available appointments for the next 3 or 4 months. This also applies

to services offered by certain private prepaid companies, where you have to provide your personal information for each prepaid service you want to obtain or visit an authorized establishment, resulting in a significant waste of time.

Moreover, companies and institutions often innovate and update their services for various reasons, changing buttons, text fields, or anything they consider convenient for their interests.

Therefore, developing an application that automatically and periodically utilizes third-party services or schedules appointments becomes complicated. You develop the application to automate the service, and then the institution updates its portal and/or the required information to access its service changes. This is why adaptive software designed for pre-programmed tasks on the Internet is needed, which is what we have developed.

In this article, we propose a self-adaptive software system to address the challenges faced by Internet-based automated processes [38–40]. The software's ability to reconfigure its code when encountering errors represents a significant advancement in the field [41–44], opening doors to more reliable and efficient Internet automation. By providing insights into the current research landscape and highlighting the key findings of our work, we hope to encourage further exploration and adoption of self-adaptable software across various domains.

In this work, we present a novel approach to developing self-adaptive software capable of autonomously rewriting its code in response to errors. This significantly improves fault tolerance and overall performance in pre-programmed tasks on the Internet. By leveraging artificial intelligence techniques and integrating comprehensive fault detection and recovery mechanisms, this software represents a substantial advancement in the field of automated task execution on the Internet. The innovative use of a communication architecture between the master module and the slave module utilizing distributed processing allows for greater execution speed and scalability.

2. Materials and Methods

The self-adaptive software developed for pre-programmed tasks on the Internet described in this study is designed to automatically rewrite its own code if it detects faults [45,46]. We use a methodology to design self-rewriting algorithms that include the following: (1) a pre-programming work phase, which consists of inputting information into the program for analysis; (2) a debugging and error storage phase, where we evaluate performance and detect and store errors with a unique ID; (3) a polishing and frequent errors phase, where we train the AI to resolve known errors; and (4) a monitoring startup phase, where we continuously monitor the processes and adjust the software's behavior based on the detected errors [47,48]. The software is primarily programmed in Python 3.9.5, making use of its extensive libraries and frameworks for efficient and flexible programming. It utilizes MySQL 8.0.23 and Firebase Realtime Database for storing information related to assigned tasks, detected errors, and corresponding solution methods. These databases allow seamless data management and retrieval, ensuring efficient analysis and the execution of appropriate actions without the need for human intervention. Image analysis is performed using OpenCV 4.5.2, while web scraping is done using BeautifulSoup 4.9.3 and Selenium 4.0.0.

The research questions are as follows:

Is it possible to design software to fulfill a specific objective in an environment where variables change, and for the software to adjust automatically to continue fulfilling the objective?

What elements must be considered for the software to adjust automatically to changes in its environment and remain useful, achieving the purpose for which it was designed?

In interaction with the operating system, the following Python standard libraries are used:

- Os Module
- Subprocess Module
- Pythorch Module
- Keras Module
- Numpy Module
- TensorFlow Module

The software architecture follows a master–slave model, with the master component acting as the central supervisor. The master module receives information from the slave modules and coordinates their activities. This hierarchical structure allows for efficient communication and ensures that the software functions smoothly as a cohesive unit.

To enhance the capabilities of the software, it incorporates image analysis algorithms that allow the work performed by the slave modules to be examined. These algorithms leverage state-of-the-art computer vision techniques, such as deep learning frameworks like TensorFlow Version 2.16.1, created by the Google Brain team, in Mountain View, California or PyTorch Version 2.3.1, created by the Facebook AI Research (FAIR) team, in Menlo Park, California. This image analysis functionality enables advanced quality control and ensures that tasks executed by the slave modules meet predefined criteria.

You just need to download the trained model to detect the desired language from Tesseract-Trained models and then save the file within the 'tessdata' folder where Tesseract is installed, as shown in Figure 3.

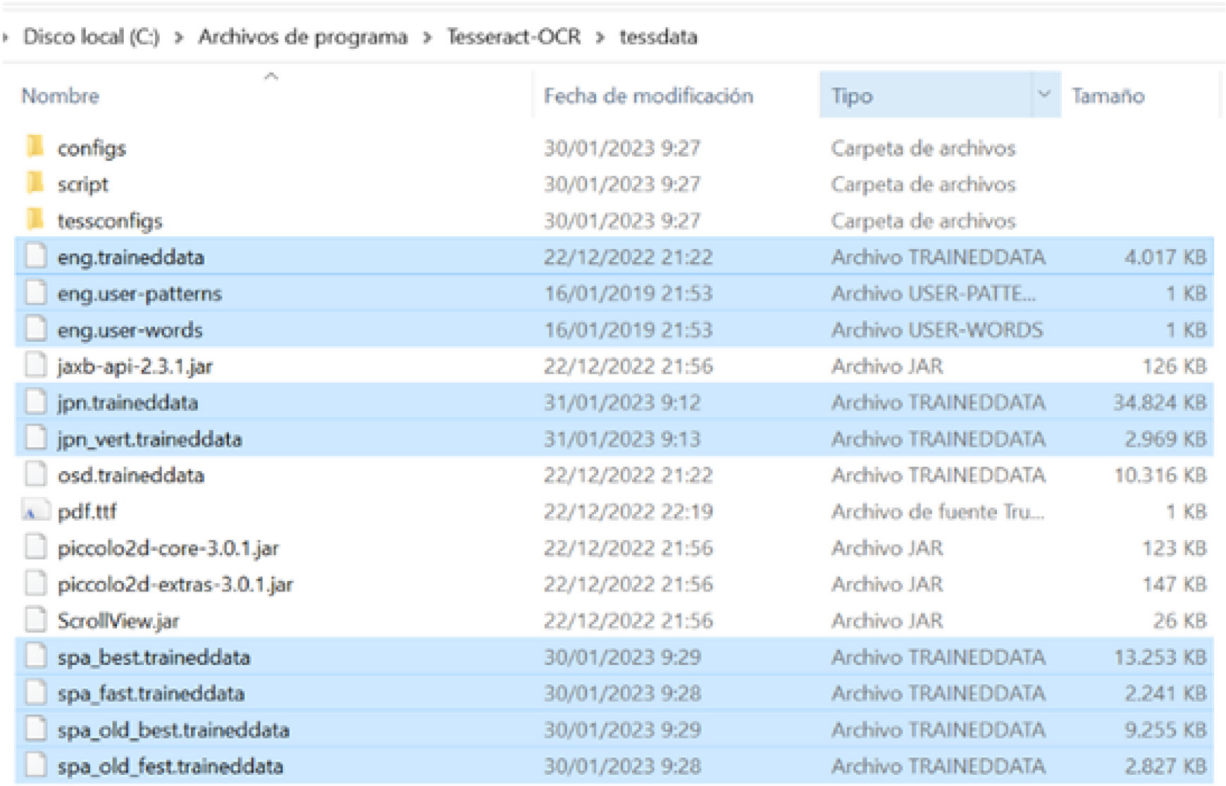


Figure 3. Test image of a webpage from which information is extracted for the modules to perform an activity analysis.

As can be observed in Figures 3 and 4, the raw data extraction process has an accuracy rate of 90% in terms of gathering information from the image.

```
(384, 211)
8 31 392 242
"Tan solo debes descargar el modelo entrenado para detectar el idioma que desees di
Jesseract
rained models
Después, debes poner este archivo dentro de carpeta tessdata donde está instalado Tesseract:
Disco local (C) > Archivos de programa > Tesseract-OCR > tessdata
PS
Nombre
Fecha de modificación
Tipo
Y
Tamaño
E configs
30!01!2023 9:27
Carpeta de archivos
E script
30!01!2023 9:27
Carpeta de archivos
tessconfigs
30!01!2023 9:27
Carpeta de archivos
22!12!2022 21:22
Archivo TRAINEDDATA
4.017 KB
eng.traineddata
16!01!2019 21:53
Archivo USER-PATTE.
1KB
```

Figure 4. Raw data extraction carried out by the artificial vision module in Figure 3.

These data are subsequently converted into vectors, as shown in Figure 5, and sent to the master modules to verify if the tasks are being carried out correctly. In the event of detecting any anomalies such as data stagnation, progress failure, query errors, etc., the diagnostic process will be triggered, and depending on the severity, the software will assess if it is necessary to self-rewrite in order to proceed with the assigned task.

The UUIDs are the identifiers of the machines from which the information was received or sent, as seen in Figures 5 and 6.

In this case, a pattern can be observed within the files presented in Figures 4 and 5, as both slave and master modules store information in vector form, which facilitates our work compared to plain text. Values are assigned to different text connections based on their relevance and sent to the modules so that they can interpret them according to the operation carried out with the vector modules.

Furthermore, the software incorporates web scraping capabilities to gather relevant data from web pages and APIs. This feature relies on well-established libraries such as BeautifulSoup or Scrapy to extract and analyze desired information. By harnessing web scraping, the software can autonomously retrieve data from various online sources, further expanding its capabilities for automated tasks.

```
"message": "RAVEN: Tuesday, April 11, 2023 at 04:28PM",
"speaker": "RAVEN",
"time": "1681252108.7645166",
"timestring": "Tuesday, April 11, 2023 at 04:28PM ",
"uuid": "290986be-7292-4ecf-8965-e182eb8ba48a",
"vector": [
    0.02359672449529171,
   -0.019401751458644867,
    0.020607806742191315,
   -0.04803244397044182,
   -0.017448466271162033,
    0.038960814476013184,
    0.09659725069999694,
    0.008527593687176704,
   -0.020135872066020966,
   -0.015049466863274574,
    0.0992158317565918,
   -0.0005682058399543166,
    0.013712318614125252,
    0.010611971840262413,
    0.09903230160474777,
    0.011097014881670475,
    0.04213326424360275,
   -0.013417359441518784,
    0.028106320649385452,
   -0.002892237389460206,
   -0.018982253968715668,
    0.01989990472793579,
    0.022731510922312737,
```


Figure 5. Example of the raw extracted information by the slave modules after being vectorized, which will be sent to the master modules for interpretation through machine learning modules.

```
"notes": "- User: Monday, April 10, 2023 at 09:36PM",
"time": "1681251204.6040165",
"times": [
    1681184176.0627828,
    1681184198.2944765,
    1681184225.6986268,
    1681184267.4183366,
    1681184435.043221,
    1681250776.8129776,
    1681250782.4258602,
    1681250841.1825042,
    1681251013.811653,
    1681251039.0772169,
]
"uuid": "287aa70e-2564-478c-b24c-d8a871bdd59a",
"uuids": [
    92cb8cc6-8dd6-4248-8db7-5fbdc5962d52,
    1c5dd63d2-ba31-40fa-b46f-e4f085a20ea7,
    51cd9d47-d4de-4ecb-bc3c-7ea2103e4a43,
    5ff81091-1853-4484-82f2-e183ceee035e,
    6aa1e742-33b7-4843-84d9-4eaea5cd34d8,
    d2cd04ca-fca8-4a54-a88c-b027e9337217,
    0c26781e-b229-4a79-b4bd-e10140c9a63,
    b105900e-cd8d-461b-950b-02630d74eecf,
    798e7aaa-5ad9-49c6-b711-9500f55221c8,
    83ac0471-4f55-4f44-a71d-572f4b9f3a65,
]
```


Figure 6. Internal notes that the master modules write and share with each other based on their training and error and status information obtained from their communication with the slave modules.

The system periodically extracts information from the pages it is working on and even retrieves the entire page in HTML and XML format if necessary for rewriting in case of any changes to its elements, as shown in Figure 7.

Furthermore, the software includes system interaction functionalities, enabling seamless integration with the underlying operating system. It can execute commands, access file systems, and perform various operations, making it versatile and adaptable to different computing environments. This system interaction capability is achieved through the use of APIs and system-level libraries specific to the target operating system.

To establish efficient communication between slave and master modules, the software utilizes a virtual wiring system. This proprietary protocol, inspired by the I2C protocol, facilitates fast and reliable data exchange between slave and master modules. It establishes virtual connections, emulating physical cables and enabling smooth transmission of commands, data, and status information. This approach ensures secure and high-speed communication within the software ecosystem.

```
something changed
d28209965de3955af833d0dc58e4faa43c7cdf48530924282cf0b828
b'<!DOCTYPE html>\n<html lang="en">\n    <head>\n        <meta charset="UTF-8">\n
something changed
d28209965de3955af833d0dc58e4faa43c7cdf48530924282cf0b828
b'<!DOCTYPE html>\n<html lang="en">\n    <head>\n        <meta charset="UTF-8">\n
something changed
d28209965de3955af833d0dc58e4faa43c7cdf48530924282cf0b828
b'<!DOCTYPE html>\n<html lang="en">\n    <head>\n        <meta charset="UTF-8">\n
something changed
d28209965de3955af833d0dc58e4faa43c7cdf48530924282cf0b828
b'<!DOCTYPE html>\n<html lang="en">\n    <head>\n        <meta charset="UTF-8">\n
something changed
d28209965de3955af833d0dc58e4faa43c7cdf48530924282cf0b828
b'<!DOCTYPE html>\n<html lang="en">\n    <head>\n        <meta charset="UTF-8">\n
```

Figure 7. Information extracted in response to a change in the location of a critical button for the progress of the activity intended for the slave modules.

The prototype files used for the virtual wiring operation inspired by the I2C protocol are SCL and SDA files. This communication prototype allows us to lower the reading speeds of the files, since using other file types such as txt, json, etc., would have a linear algorithmic complexity of O(N), depending on the amount of data to be read. On the other hand, using virtual wires based on the I2C protocol allows us to have an algorithmic complexity of O(1) since the file structures allow us to read them line by line in 1024-bit packets, as they do not store all lines of the file in memory simultaneously. Instead, it reads one line at a time and processes each line before reading the next one. In terms of time, a file's complexity depends on its size and the number of lines it contains. In the worst case, if the file has N lines, it will be executed N times, resulting in a time complexity of O(N) because each line of the file must be read and processed.

However, we must consider that the time complexity can also be affected by the processing performed on each line. If the processing of each line has a complexity greater than O(1), then the overall complexity of the code will be determined by the processing rather than by the file reading.

In summary, the space complexity is O(1), and the time complexity is O(N), where N is the number of lines in the file. However, if we base communication on JSON files, the time and space complexity of this code is O(N), where N is the number of lines in the file.

In terms of software development practices, the team followed an agile methodology, incorporating continuous integration and deployment processes. Version control systems like Git were used to track changes and facilitate collaboration among developers. The software was thoroughly tested using unit tests, integration tests, and system tests to ensure its robustness and reliability.

To enhance the software, the following additional tools, software, and frameworks were employed during development:

Docker: The software was containerized using Docker, providing a portable and reproducible environment for seamless deployment on different platforms.

Flask: The Flask web framework was used to develop a user-friendly web interface for managing and monitoring the software's activities, allowing users to interact with the system and view real-time updates.

OpenCV: The OpenCV library was used for advanced image processing tasks, enabling the software to perform complex visual analysis and recognition.

Selenium: Selenium WebDriver was used to automate web browser interactions, allowing the software to navigate websites and perform tasks that require user-like interactions.

Scikit-learn: Scikit-learn, a popular machine learning library, was used to train and deploy machine learning models within the software, enhancing its ability to make data-driven decisions.

Pandas: The Pandas library was employed for efficient data manipulation and analysis, enabling the software to process large datasets and generate insightful reports.

Jupyter Notebook: Jupyter Notebook was used for prototyping, experimentation, and data exploration during the software development cycle, facilitating an agile and iterative development process.

The self-adaptive software developed for pre-programmed tasks on the Internet, as described in this study, runs on hardware optimized for efficient and reliable performance. The software is hosted on a computer with the following characteristics:

Processor: The central processing unit (CPU) powering this system is an AMD Ryzen 7 processor. The Ryzen 7 series is known for its multicore performance, which is essential for handling concurrent tasks and complex calculations involved in the software's operation. This CPU provides the processing power required to effectively execute the software's code.

Memory: To ensure smooth operation and the ability to handle large datasets, the system is equipped with 16 gigabytes (GB) of RAM. This RAM is configured in dual-channel mode, enhancing memory bandwidth and responsiveness. Dual-channel configuration is particularly beneficial for tasks involving data retrieval, manipulation, and analysis, as is the case with our self-adaptive software.

Storage: The primary storage for the system is a 1-terabyte solid-state drive (SSD). SSDs offer fast data access and retrieval speeds, which are crucial for efficient storage and retrieval of information related to assigned tasks, detected errors, and corresponding solution methods stored in MySQL 8.0.23 and Firebase Realtime Database. Additionally, the SSD ensures quick access to the software's code and resources for smooth execution.

Operating System: The software runs on the Windows 10 operating system, although it can also run on a compatible Linux operating system based on the system administrator's preference and specific software requirements.

The combination of an AMD Ryzen 7 CPU, 16 GB of RAM in dual-channel mode, and a 1-terabyte SSD provides a solid and responsive hardware foundation for hosting and running the self-adaptive software. This hardware configuration ensures that the software can effectively manage data, perform complex calculations, and execute tasks seamlessly, contributing to its overall reliability and effectiveness.

We use neural networks in the following way: first, we collect data on system failures and operations. This data is divided into training and testing sets. Subsequently, we transform the data into a suitable format for use by the neural network. We select an appropriate architecture for the neural network, such as convolutional neural networks or deep neural networks, depending on the type of data and the nature of the problem (we use both). We train the model using supervised learning algorithms, such as gradient descent, adjusting the weights of the network to minimize prediction error, and once trained, we integrate the model into the software system. This may involve implementing the model in a production environment using deep learning libraries such as TensorFlow or PyTorch.

The neural network model continuously monitors the data flow in real-time, and based on the model's predictions, the system can make automatic decisions to correct errors or adjust its operation. For example, if an anomaly is detected, the system can trigger automatic reconfiguration processes or send alerts to system administrators if the information in the database is insufficient. The software continuously collects new operational data which is used to update and retrain the neural network model, thus improving its accuracy and adaptability.

We use the reinforcement learning algorithm to train the master modules of the system so that they can resolve errors detected during the execution of the slave modules. Information is introduced into the slave module program for the tasks to be performed. The master modules execute the slave module, evaluate its performance, detect errors, and

store each error with a unique identification for future analysis (abstractions). We use the reinforcement algorithm to train neural networks to find ways to resolve errors that occur during the execution of the slave module. This phase uses a reinforcement learning model where the characteristics of the errors (abstractions) are identified, and the master modules are taught how to resolve them. The errors are stored in a database, and the machine learns to apply the abstractions to resolve specific errors when they reoccur.

The processes are continuously monitored, and the software adjusts its behavior based on detected errors, ensuring the continuous functionality of the system. Reinforcement learning is implemented in the polishing and common errors phase so that the neural network learns to resolve errors autonomously, optimizing the performance and efficiency of the self-adaptive software. This process includes the use of computer vision techniques and web scraping to analyze and extract data, and then vectorize it for interpretation and resolution by the master modules.

In Figure 8, we show how the self-adaptable software for enhancing the reliability and efficiency of pre-programmed Internet tasks that we developed works.

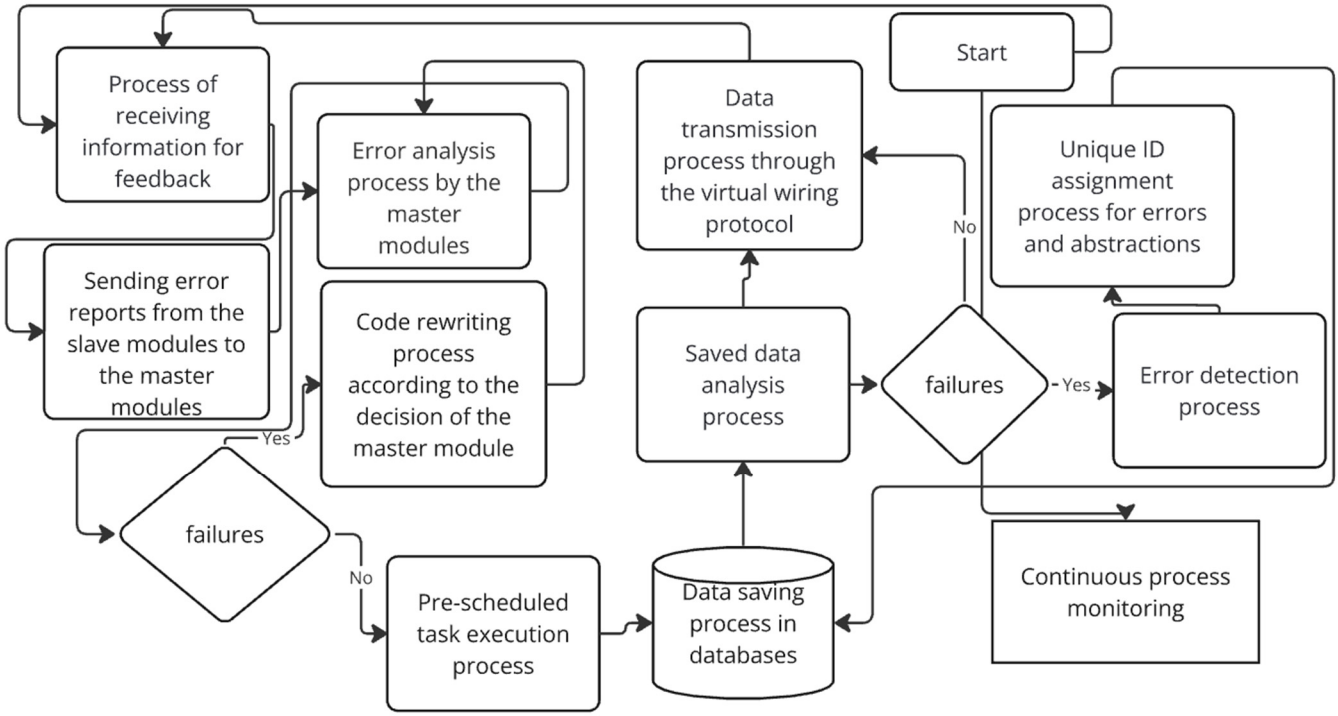


Figure 8. How the self-adaptable software for enhancing the reliability and efficiency of pre-programmed Internet tasks works.

In Figure 9, we show how the self-adaptable software for enhancing the reliability and efficiency of pre-programmed Internet tasks that we developed works.

We use these technologies because they are efficient, functional, and can run without the need for high-cost hardware, perfectly adapting to the available hardware. Python was chosen for its versatility and numerous libraries. MySQL and Firebase ensured rapid data retrieval and manipulation. Docker facilitated the creation of portable and reproducible environments. Flask provided a user-friendly interface with low resource requirements. OpenCV enabled advanced visual analyses. Selenium automated web interactions without the need for powerful hardware. Scikit-learn and TensorFlow improved the ability to make data-driven decisions, and Pandas facilitated the efficient manipulation and analysis of large datasets.

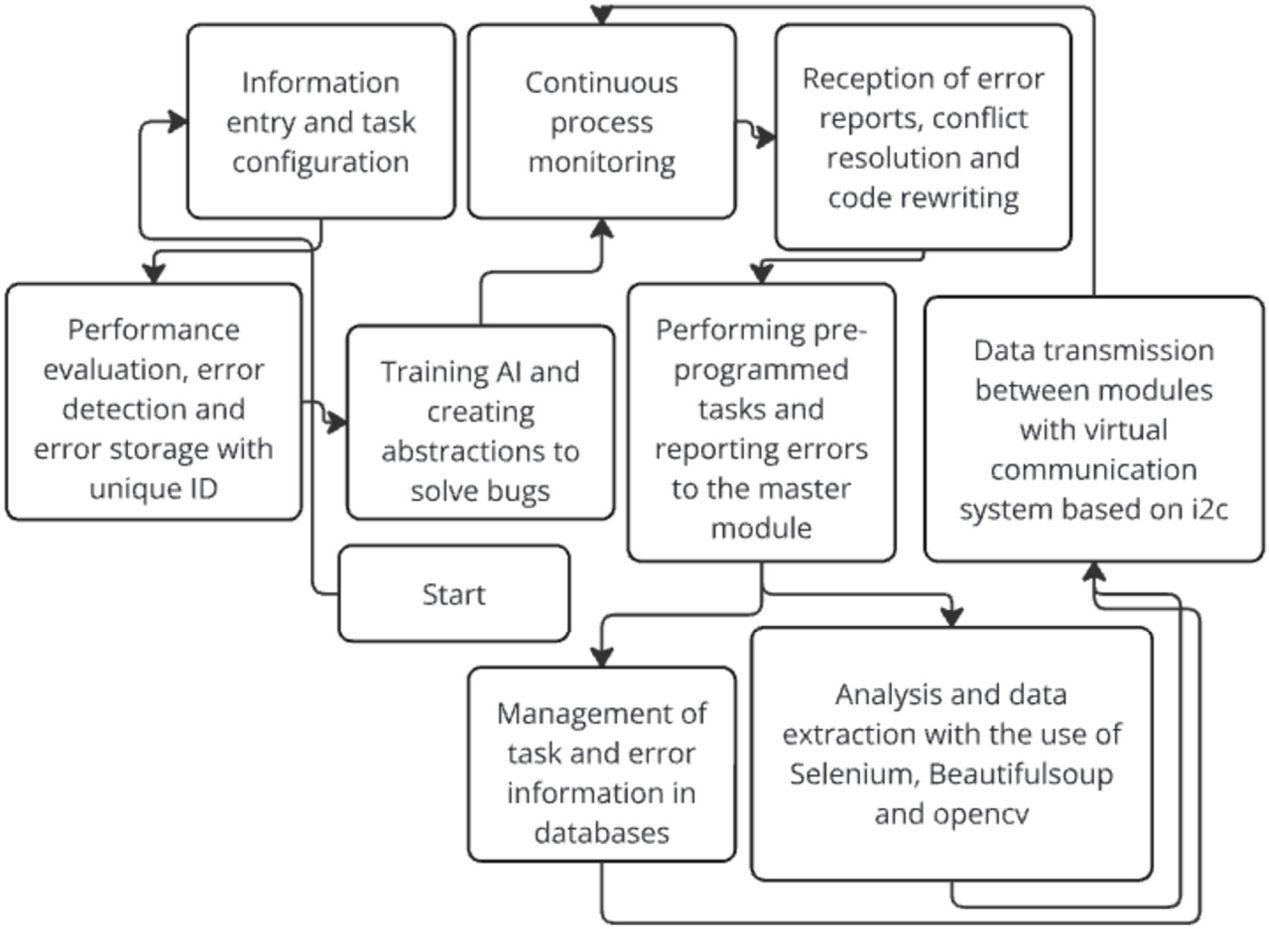


Figure 9. How the self-adaptable software for pre-programmed Internet tasks enhances reliability and efficiency.

A self-adaptive software was built to perform tasks on the web such as automatic web scraping and automatically executing activities such as services, among others. It is self-adaptive software for performing web tasks that can handle changes on pages, data extraction errors, and any errors that would normally halt its execution. For this, tools like Tensorflow and OpenCV were used to create error detection algorithms. Access is provided here: https://drive.google.com/file/d/1-rN1y2uTxFPLc_U1DE6p-6A1CJa_O6wW/view?usp=drive_link (accessed on 23 May 2024). A model was trained to accurately detect text in images, then the text was processed in another algorithm. Based on databases with all possible errors and their solutions, the software will modify its code, from editing its execution file to merely changing variables or updating drivers. Access is provided here: https://drive.google.com/file/d/1Wd8kt_DM3Ac8hrSfiqq8oveKZTe1BN0u/view?usp=sharing (accessed on 23 May 2024) and https://drive.google.com/file/d/11ambD9IYv1rMZjQZmeZgSkKFlcrpNDR1/view?usp=drive_link (accessed on 20 May 2024). Additionally, a third algorithm was implemented and trained from scratch for the interpretation of images and web pages. This uses Tensorflow to detect changes in elements on the pages and thus modify XPath records to avoid errors. Access is provided here: https://drive.google.com/file/d/1p688molo1BVStFEDfr6CQcylNz3LCyR3/view?usp=drive_link (accessed on 15 May 2024). As additional information, the labeling of the images for the models was completed with LabelImage. All are integrated, and access is provided here: https://drive.google.com/file/d/1N-wg0YSN3AXNm510Hk_JOCwRz6ArChSX/view?usp=drive_link (accessed on 23 May 2024).

3. Results

3.1. Context

There is no commercially available software capable of autonomously writing its own code completely, so the results cannot be compared with other examples.

In this development, the term 'Workloads' or 'C' was used to describe an iterative cycle used in simulating the data loading and unloading operations in the software. This notation has been adopted as an abstract representation of data processing operations.

3.2. Efficiency and Optimization

Multiprocessing Testing with Variable Workloads: Tests were conducted with different workloads (C) to evaluate how the software leveraged multiprocessing. Workloads were varied to observe how the software adjusted to different levels of demand and its performance under heavy and light loads.

This notation 'C' allows for an abstract and general representation of the operations performed in each iterative cycle of data loading and unloading, facilitating a clearer understanding of how the software adapts to different levels of demand in the tests.

In cases where the task load is not specified, it is considered that one task equals 1 Mbps.

A scenario was designed where incorrect data was deliberately injected into the software input process. This included providing variable values that did not match the specified requirements, such as negative values or text strings instead of numbers. The objective was to evaluate the software's ability to detect and correct these errors without interruptions in the process.

A scenario was simulated where online traffic significantly increased, resulting in system overload. The software was subjected to a workload (W) much higher than normal, and its ability to maintain optimal performance and adapt to increased demand without downtime was evaluated.

Tests were conducted where the structure of a target webpage experienced variations. For example, HTML tags were changed, and the order of elements on the page was modified. The software was evaluated for its ability to identify and adapt to these structural changes to complete the download task.

Integrity errors were introduced into simulated download files, such as data corruption or incomplete files. The software was evaluated for its ability to verify the integrity of downloaded data and to correct or retry the download if any errors were detected.

To achieve this, it is recommended to use expert systems and neural networks to continuously monitor and evaluate the software's performance. Error detection is based on identifying faults through automated diagnostics that compare current behavior with predefined patterns, allowing the system to react and autonomously correct faults [45,49].

Tests were conducted on different operating systems and browsers to evaluate the software's compatibility and performance across a variety of user environments.

A scenario was designed where the software's self-adaptation rate was progressively adjusted, from frequent adaptations to very rare adaptations. The objective was to evaluate how the software responded to changes in the self-adaptation rate and its ability to maintain stability and efficiency.

Tests were conducted with different workloads (C) to evaluate how the software leveraged multiprocessing. Workloads were varied to observe how the software adjusted to different levels of demand and its performance under heavy and light loads.

Deliberate code failures were introduced, such as division by zero, null pointers, or memory overflows. The software was evaluated for its ability to detect, report, and correct these errors without major system failures.

A significantly large dataset was generated to evaluate the scalability of the software. The software was subjected to a workload involving the processing of large volumes of data, and its ability to adapt and maintain efficiency in high-demand environments was evaluated.

Numerical methods and mathematical formulas were used for data approximation and model creation. Some of the most commonly used methods included:

- Nonlinear regression method for modeling complex relationships between variables. This method is especially useful for fitting data to nonlinear models.
- Principal Component Analysis (PCA) was employed to reduce the dimensionality of the data and explore patterns in multidimensional datasets. This method allows for the representation of complex data in a lower-dimensional space while preserving relevant information.
- The wavelet transform was used to analyze data in the time and frequency domains. It provided an efficient representation of non-stationary signals, aiding in the identification of patterns and hidden features in the results.

Machine learning algorithms such as polynomial regression, logistic regression, and neural networks were applied to create predictive models based on historical data. These models were used to forecast future behaviors and outcomes.

Bayesian statistics was employed to estimate probability distributions and make inferences about unknown parameters. This allowed for the evaluation of uncertainty and variability in the results.

3.3. Self-Adaptation Rate of the Self-Adaptable Software for Pre-Programmed Internet Tasks

With the following Formula (1), we define an approximation, based on time, of how long it takes for the software proposal to self-adapt to the error with the data obtained from the tests conducted.

The self-adaptation rate (A) of the software on a scale from 0 to 1 is related to the elapsed time (t) since the last self-adaptation as follows:

$$A(t) = 1 - e^{-kt} \quad (1)$$

where:

- $A(t)$ is the self-adaptation rate at time t.
- e is the base of the natural logarithm (approximately 2.71828).
- k is a positive constant that controls the speed of software adaptation.

The exponential function here models how the self-adaptation rate increases over time and gradually approaches the maximum value of 1 as shown in Table 3.

Table 3. Monitoring of the software's self-adaptation rate over 10 time intervals.

Time (t) in Hours	Self-Adaptation Rate ($A(t)$)
0	0.00
1	0.10
2	0.30
3	0.45
4	0.60
5	0.70
6	0.80
7	0.85
8	0.90
9	0.95
10	0.97

As can be observed in Figures 10 and 11, we have the actual software adaptation curve (blue points). Due to the large number of variables affecting the tests, it is not possible to have an exact adaptation graph, but rather an estimation of the software's behavior under different usage environments. In Figure 10, it underwent a light 10 h test during which changes were gradually made to allow the software to adapt. In the event of encountering the same error again, it would utilize information from the database to resolve the issue and adapt. In this case, merely changing labels and the position of elements on a test web page, solely visual changes, sufficed. As observed, the software adapted on the fly with progress closely aligning with the prediction curve (orange), performing much better from the fourth to the tenth hour, concluding with successful adaptation without stagnation.

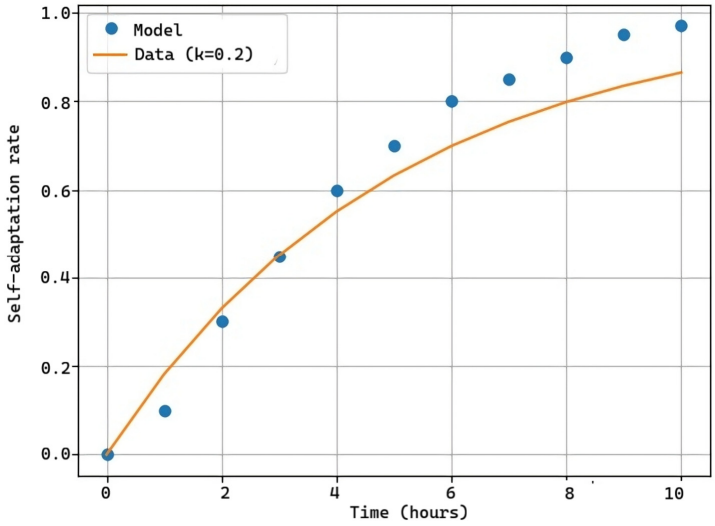


Figure 10. Self-adaptation rate $k = 0.2$.

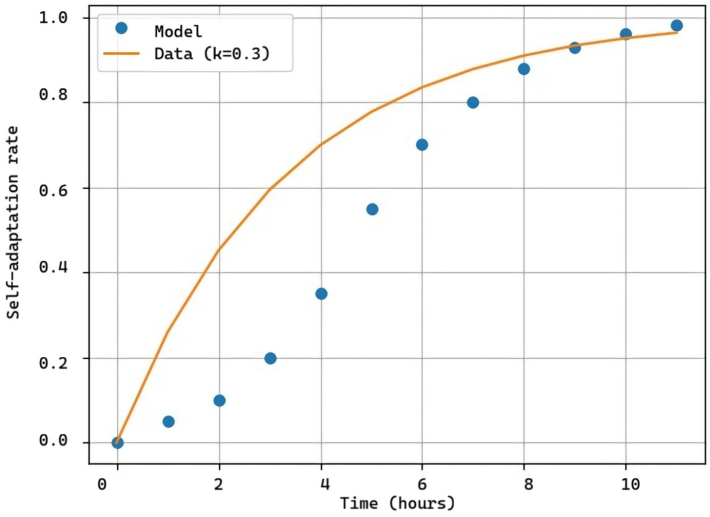


Figure 11. Self-adaptation rate $k = 0.3$.

For Figure 11, changes were made not only visually but also to Xpath, external links, and page functionality, with the same number of hours under the same configuration. As seen in Figure 11, the software experienced issues from the first hour to the fourth, deviating from the prediction curve (orange) as the solutions to these problems were much more complex than mere physical aspects changes in the test from Figure 10.

3.4. Response Time of Slave and Master Modules

The response time of each module depends on the number of tasks assigned to them. We carried out three tests with the following scenarios: without any external components (software without any algorithmic enhancements), with multiprocessing, and with virtual wires based on the I2C protocol and multiprocessing, with the latter being the final model.

3.4.1. Mathematical Model without Multiprocessing or Virtual Wires

Let T_slave be the function representing the response time of the slave module in seconds.

Let T_master be the function representing the response time of the master module in seconds.

Let C be the workload (number of tasks) assigned to both modules.

N is the degree of the polynomial function used to model the response time of the slave and master modules.

The choice of N in the polynomial equations for the response times of the master and slave modules is based on the complexity of the relationship between response time and workload. Specifically, N determines the degree of the polynomial that best fits the actual data. To find the optimal value of N, model fitting and validation techniques are used.

This polynomial function allowed us to express the response time (T) of each module as a function of the workload (C) assigned.

Equations (2) and (3) are fundamental because they provide a mathematical framework for modeling the response times of the master and slave modules as polynomial functions of the workload. These models allow for predicting how response time will vary with different workloads, which is necessary for designing efficient and predictable systems.

General mathematical model:

- For the slave module: The response time of the slave module (*T_slave*) can be expressed as a polynomial function of degree N:

$$T_slave(C) = a^0 + a^1C + a_2C^2 + a_3C^3 + \ldots + a_nC^n \quad (2)$$

- For the master module: The response time of the master module (*T_master*) can also be expressed as a polynomial function of degree N:

$$T_master(C) = b_0 + b_1C + b_2C^2 + b_3C^3 + \ldots + b_nC^n \quad (3)$$

In these equations, the coefficients (a_0, a_1, a_2, . . ., a_N) and (b_0, b_1, b_2, . . ., b_N) represent the parameters of the model that need to be adjusted based on real data for each module. The choice of N will depend on the complexity of the relationship between the response time and the workload, and will be determined through model fitting techniques and validation.

3.4.2. Mathematical Model with Multiprocessing

- For the slave module: The improved response time of the slave module (*T_improved_slave*) as a function of the workload (*C*) can be modeled with an exponential function:

$$T_impruved_{_slave(C)} = ae^{-bC} \quad (4)$$

where 'a' and '*b*' are constants that determine the magnitude and rate of improvement provided by the multiprocessing algorithm.

- For the master module: The enhanced response time of the master module (*T_enhanced_master*) as a function of the workload (*C*) can also be modeled with a similar exponential function:

$$T_enhanced_{_master(C)} = ce^{-dc} \quad (5)$$

where 'c' and 'd' are constants representing the magnitude and rate of improvement for the master module.

3.4.3. Mathematical Model with Virtual Wires and Multiprocessing

By adding this improvement to the software, the communication speed between the modules increases, thus reducing transfer times.

Modification of the mathematical model with the I2C virtual wire system:

- For the slave module: The equation for the enhanced response time of the slave module (*T_improved_slave*) will now consider the additional improvement due to the I2C virtual wire system:

$$T_improved_{_slave(C)} = ae^{-bC} f(C) \tag{6}$$

where '$f(C)$' is a function representing the improvement provided by the I2C virtual wire system as a function of the workload (C).

- For the master module: The equation for the enhanced response time of the master module (*T_improved_master*) will also be adjusted to account for the additional improvement from the I2C virtual wire system:

$$T_impruved_{_master(C)} = ce^{-dC} g(C) \tag{7}$$

where '$g(C)$' is a function representing the improvement provided by the I2C virtual wire system as a function of the workload (C).

- Improvement function for the slave module: '$f(C)$': For the enhanced efficiency of the slave module. As the workload increases, the enhanced efficiency also increases, but with a lower growth rate.

$$f(C) = 1 - e^{-k_1 C} \tag{8}$$

where 'k1' is a constant that controls the improvement rate as a function of workload. As 'C' increases, the efficiency will approach 100%, but will never fully reach it.

- Improvement function for the master module '$g(C)$': Where 'k_2' is a constant that controls the initial improvement rate. As 'C' increases, the improvement slows down, but it never reaches 100% efficiency.

$$g(C) = 1 - \frac{1}{k_2 C + 1} \tag{9}$$

As observed in Figure 12, there is a difference in the successful response of the master and slave modules to workloads (data downloads) between using virtual wires, multiprocessing separately, and combining these two technologies. When the master and slave modules use only virtual wires with an average workload of 60 C at 1 Mbps, they take approximately 12 and 24 s, respectively, to respond successfully. However, the configuration that uses both wires and multiprocessing takes 5 and 6 s, respectively, reducing the time by more than half for the master modules and fourfold for the slave modules. This results in both the slave and master modules taking the same amount of time, thus avoiding delays in action processing and saturation in action transmission. The master modules will not have to wait as long to receive a response from the slave modules, enabling homogeneous communication among all modules. This is crucial in this field, considering that the software is subject to constant changes, and seamless communication without delays is necessary to resolve errors in the shortest possible time.

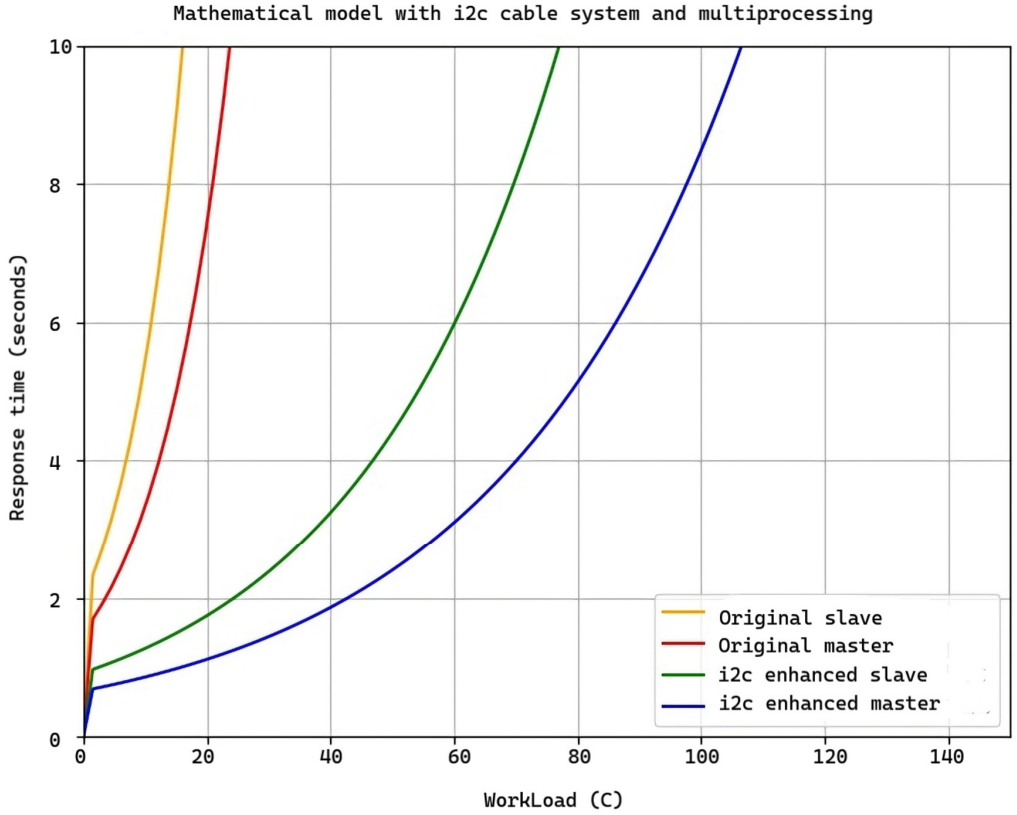


Figure 12. Comparison between response time of slave and master modules with I2C virtual wires and multiprocessing, and original slave and master modules with different workloads.

3.5. Probability of Repair and Error Resolution of the Self-Adaptable Software for Pre-Programmed Internet Tasks

In the mathematical model, we consider three key aspects:

1. The probability of failure of each module (A, B, and C), where A represents the master module, B represents the slave module, and C represents the repair module.
2. The probability that the system will repair the code in case of a fatal failure is 80%, according to the tests we conducted, with an 80% success rate.
3. The probability that the code will execute correctly after a repair.

We denote the following probabilities:

P(A): Probability of a failure occurring in module A.
P(B): Probability of a failure occurring in module B.
P(C): Probability of a failure occurring in module C.
P(R): Probability of the system repairing the code in case of a fatal failure.
P(TF): Probability of software fault tolerance (probability of the software functioning correctly).
P(RF): Probability of recovery after a failure.

Probability of a system failure:

$$P(\text{Failure}) = 1 - P(TF) \tag{10}$$

Probability of a failure occurring in modules A, B, or C (at least one failure):

$$P(A \cup B \cup C) = P(A) + P(B) + P(C) - P(A \cap B) - P(A \cap C) - P(B \cap C) + P(A \cap B \cap C) \tag{11}$$

Probability of a failure occurring in modules A and B:

$$P(A \cap B) = P(A) \times P(B) \tag{12}$$

Probability of a failure occurring in modules A and C:

$$P(A \cap C) = P(A) \times P(C) \tag{13}$$

Probability of a failure occurring in modules B and C:

$$P(B \cap C) = P(B) \times P(C) \tag{14}$$

Probability of a failure occurring in modules A and B and C:

$$P(A \cap B \cap C) = P(A) \times P(B) \times P(C) \tag{15}$$

Probability of recovery after a failure:

$$P(RF) = P(Failure) \times P(R) \tag{16}$$

Probability of software fault tolerance:

$$P(TF) = 1 - P(Failure) \tag{17}$$

Probability of code recovery after a failure in terms of the above probabilities:

$$P(R) = P(RF) \times P(Failure) \tag{18}$$

Probability of the system correctly repairing the code after a failure in terms of the above probabilities:

$$P(SuccessfulRecovery) = P(R) \times P(TF) \tag{19}$$

We represent the probability of successful code recovery after a failure as P(SuccessfulRecovery). A successful recovery is when the error repair is achieved and the master module executes the slave module again without any errors.

We use the following notations:

P(Failure): Probability of a failure occurring in the system.
P(A): Probability of a failure occurring in module A.
P(B): Probability of a failure occurring in module B.
P(C): Probability of a failure occurring in module C.
P(RF): Probability of recovery after a failure.
P(R): Probability of software fault tolerance (probability of code recovery after a failure).
P(TF): Probability of software fault tolerance (complement of the failure probability).

The unified equation is:

$$P(\text{SuccessfulRecovery}) = (1 - P(Failure))\left(\frac{P(RF)}{P(Failure)}\right)P(TF) \tag{20}$$

The above can also be expressed in the following manner, breaking down the probabilities:

$$P(\text{SuccessfulRecovery}) = (1 - P(Failure))^2\left(\frac{P(RF)}{P(Failure)}\right) \tag{21}$$

This equation considers that the probability of successful recovery occurs when the failure does not happen or is minimal ($1 - P(Failure)$), multiplied by the probability of recovery after a failure ($P(RF))/P(Failure)$.

The probability of a system failure, P(Failure), is the complement of the probability of software fault tolerance, $P(TF)$. This is derived from the fact that if a system is not

fault-tolerant, $P(TF) = 0$, the probability of failure is maximal, $P(Failure) = 1$, and vice versa. Therefore, these probabilities are complementary and their sum is 1.

In practical terms, errors cannot be controlled as in the testing environment where we test the software, which is why we cannot rely on specific numbers to determine if recovery will occur. The most accurate approach is to rely on probabilities, as observed in Figure 13, where the software has the ability to work with fault tolerances ranging from 0% to 50%. This is the probability of recovery from failures. As mentioned, the probability of recovery cannot be just a fixed value like 50% since complications outside the logic of the software can arise, reducing the probability. Therefore, the overall probability of recovery while the code is running ranges from 0 to 50 percent, with the latter being the most prevalent probability.

On the other hand, in the same Figure 13a, we can observe the probability of recovery from a code breakage. The software has the ability to function while fixing errors, but when the error is too significant that it can "break the code," the software stops all processes and prioritizes the repair of that error to prevent tasks from being compromised. When this happens, the software has a probability of successful recovery ranging from 0% to 80%, with the latter being the most prevalent in the practical tests conducted, as seen in Figure 12. It is a breakdown representation of the recovery behavior of the software concerning the degree of error, namely:

1. Normal errors for which the software was programmed, such as changes in position in elements, incorrect memory data, and file name changes, among others (0.1).
2. Errors like memory overflow (0.5).
3. Errors that cause failure in the master modules (1).

Where $P(failure)$ is the software failure probability, and $P(RF)$ is the probability of successful recovery after a failure (which in this case is 0.8), as we can see in Figure 14.

In Figure 13b, we show the statistical results of the proposal tests with a mean recovery rate of 80%, a standard deviation of 5%, and a 95% confidence interval (78.21080585628285, 81.78919414371715), with a z-score < -1.96 or $> +1.96$.

3.6. Effectiveness Model of the Self-Adaptable Software for Pre-Programmed Internet Tasks

The software features multiprocessing, which is not only used to reduce response time between slave and master modules but also to decrease processing time for tasks assigned to the modules. In this section, we will refer to multiprocessing modules as nodes.

The model adjusted to the data obtained from the tests is an exponential function of the form:

$$T(n) = a\ e^{\wedge}(-b\ n) \quad (22)$$

where:

1. T(n) is the processing time in seconds.
2. n is the number of nodes used in the distributed system.
3. a and b are adjusted parameters of the model, representing the processing capacity and distribution efficiency, respectively.
4. Parameter a: Parameter a represents the processing capacity of the software when an infinite number of nodes is used. In other words, it is the upper limit of the processing time (T(n)) as the number of nodes (n) tends to infinity. Mathematically, as n tends to infinity, e^(−b n) tends to zero, leaving only the value of a as the theoretical maximum processing capacity.

Interpretation: A high value of a indicates that the software has a high distributed processing capacity, meaning it can handle a large number of nodes and perform complex calculations efficiently.

5. Parameter b: Parameter b represents the distribution efficiency of the software. It is a value that determines how the processing time (T(n)) varies with the number of nodes (n). Specifically, it controls the rate at which processing time decreases as more nodes

are added. A lower value of b indicates a slower decrease in processing time as the number of nodes increases.

Interpretation: A low value of b means that the software is highly efficient in distributing tasks among additional nodes. In other words, adding more nodes has a significant impact on improving processing time.

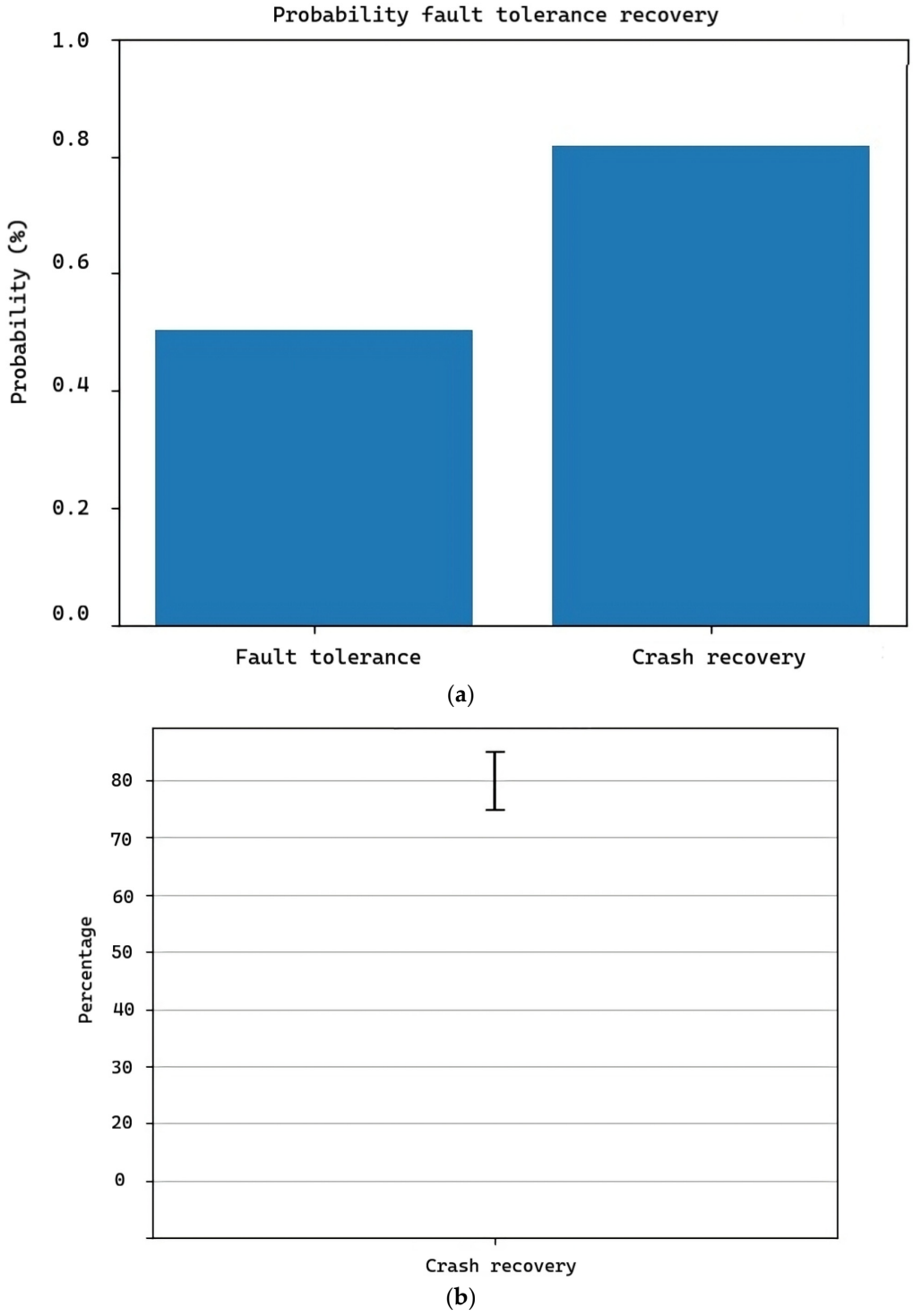


Figure 13. (**a**) Fault tolerance and software failure recovery. (**b**) Error bar—Crash recovery.

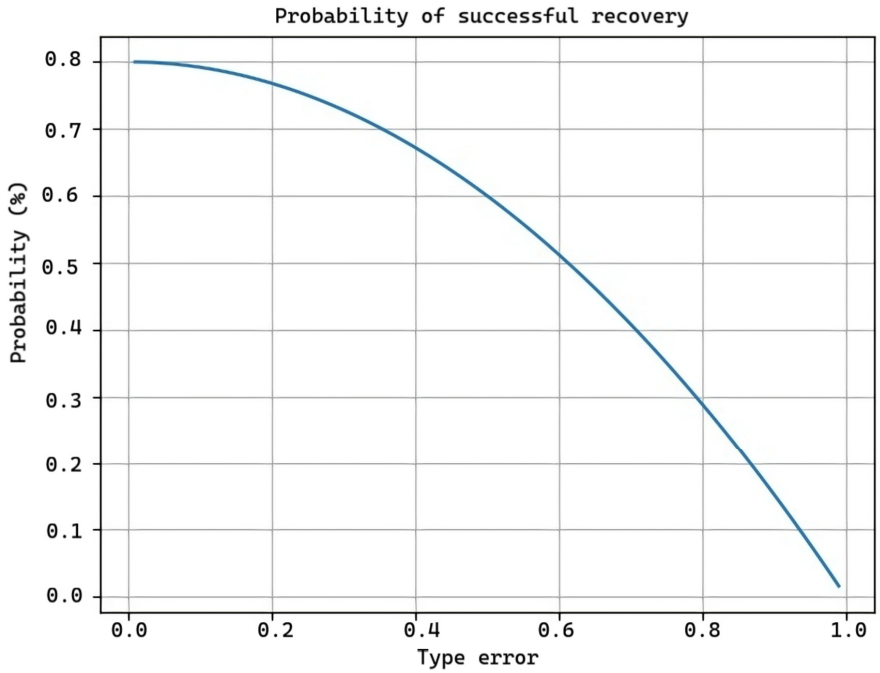


Figure 14. Probability of successful recovery.

Figure 15 displays the line of the adjusted model (in red) and the actual observed data. For this test, the database load, the start of the master and slave modules, and the load of the test web page were taken into account. As observed in the data, using only one node for execution takes too much time, but splitting the software and letting each module perform an action reduces the software startup time.

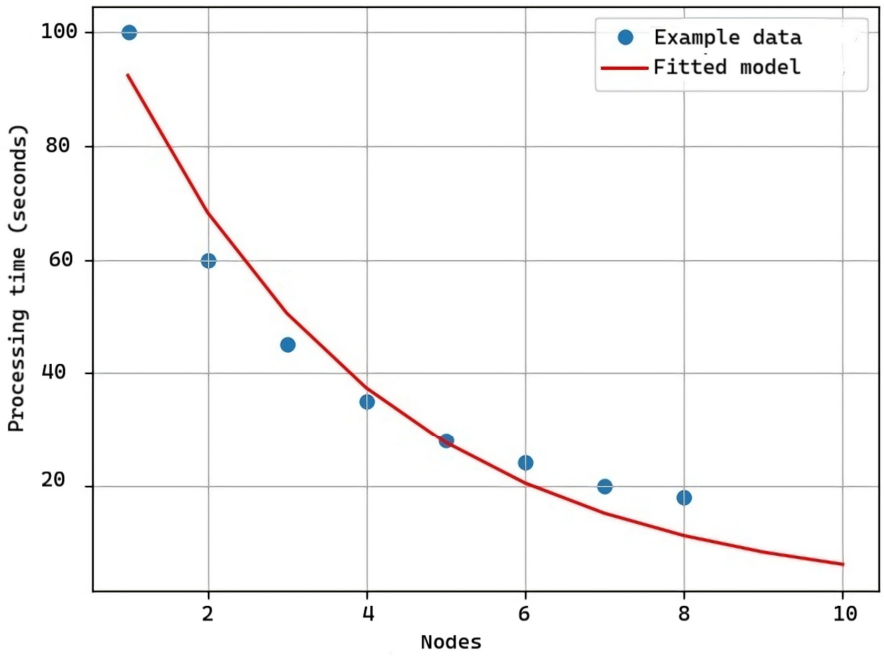


Figure 15. Graph of software data processing time according to the number of nodes.

We define some additional variables and equations to represent the system's behavior. In this case, we use a Mixed Integer Linear Programming (MILP) model to address the software task optimization problem. In this model, we will use binary variables to represent whether a task is performed or not.

We have the following data for the software tasks:

- Tasks: Task A, Task B, and Task C
- Priorities: 5, 8, 6
- Times: 4, 6, 3 (in time units)
- Total required time: 10 (in time units)
- Model variables:
- x_i (binary): 1 if Task i is performed, 0 otherwise.

Model equations:

1. Objective function: *Maximize* the sum of priorities of the tasks performed.

$$Maximize\ Z = 5x_{WorkA} + 8x_{WorkB} + 6x_{WorkC} \tag{23}$$

2. Total time constraint.

$$4x_{WorkA} + 6x_{WorkB} + 3x_{WorkC} \leq 10 \tag{24}$$

3. Binary variables constraint.

$$x_{WorkA} + x_{WorkB} + x_{WorkC} \in \{0, 1\} \tag{25}$$

With these Equations (23)–(25), we represent the Mixed Integer Linear Programming (MILP) model for the software task optimization problem. The objective function aims to maximize the sum of task priorities, and the constraints ensure that the total required time is not exceeded and that the variables are binary (representing whether each task is performed or not).

3.7. Mathematical Model for the Probability of Failures and Solutions

Failure Probability Model: assuming that the occurrence of failures follows a Poisson process. The failure rate (lambda) is represented as the average number of failures occurring in a given period of time. The probability of observing "k" failures in a period of time "t" is expressed using the Poisson probability function:

$$\text{P(k failure in t)} = (e\hat{}(-\lambda)\ \lambda\hat{}k)/k! \tag{26}$$

where:

λ (lambda) is the failure rate (average number of failures per unit of time).

Solution Effectiveness Probability Model: the effectiveness of solutions can be modeled as a conditional probability, i.e., the probability that a solution is effective given that a failure occurred. We will represent this probability as P(E | F), where E is the effectiveness of the solution and F is the occurrence of a failure.

General Effectiveness Probability Model: The overall probability that a solution is effective is obtained through the theorem of total probability. It is the probability of a failure occurring multiplied by the probability that the solution is effective given that a failure occurred:

$$\text{P(General Effectiveness)} = \text{P(Failure)P(E} \mid \text{F)} \tag{27}$$

During the 100-day trial as shown in Figure 16, we initially observe a peak because the model is practically new and may encounter various failures. As time passes, the probability of failures decreases and remains stable, as the system has adapted to the errors that arose in the assigned work area.

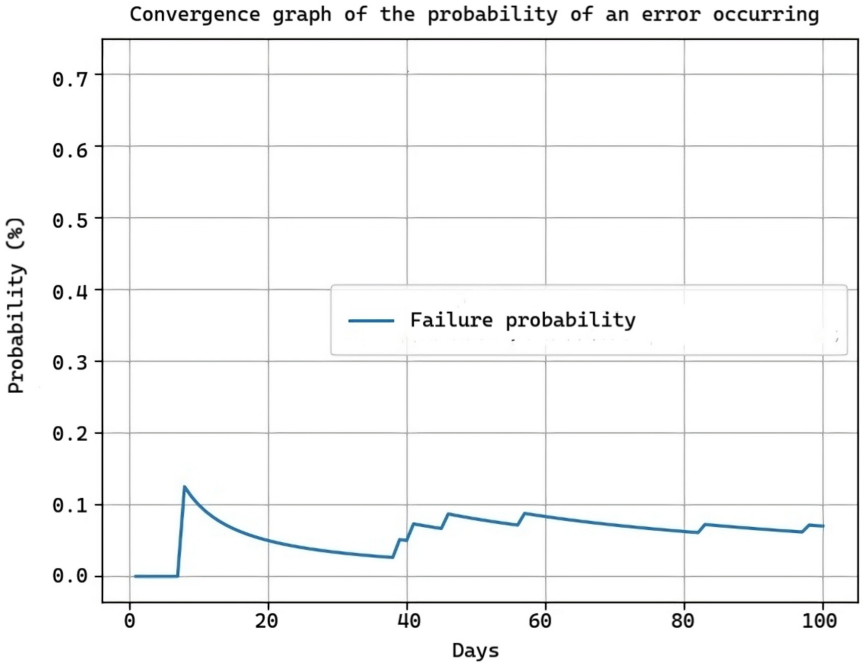


Figure 16. Failure probability and solution effectiveness during a 100-day trial period.

3.8. Reconfiguration

The software demonstrated a high capacity for autonomous reconfiguration when errors were detected. During the tests, various types of faults were introduced into the code, such as syntax errors, logical errors, and unhandled exceptions. In all cases, the software was able to identify and correct the errors efficiently, ensuring the continuous execution of pre-programmed tasks, and our results are shown in Table 4.

Table 4. Files with the information on the results of the proposed self-adaptive software.

Files with the Results of Our Work	Who Use the File	Description
ActionSummary.Release	The self-adaptative software proposal	Output information on performed actions, detected errors, and the current state of the software.
AdaptationLog.outN	The self-adaptative software proposal	Information about the modifications made by the software to prevent errors.
errorLogTestSolve.txt	The self-adaptative software proposal	Information about errors detected by the software.
SystemOperateNeuronalLog.outN	The self-adaptative software proposal	Information about the actions performed by the software (adaptation, detection, and repair).
trainingResultsNN_log.txt	Neuronal network trained without reinforcement algorithm	Information of the neural network training over 50 epochs.
rl_weights.weights.h5	Neuronal network trained with reinforcement algorithm	Weights of the neural network trained by reinforcement learning.
trained_rl_model.h5	Neuronal network trained with reinforcement algorithm	Model trained by reinforcement learning.
nn_weights.weights.h5	Neuronal network trained without reinforcement algorithm	Weights of the neural network trained without reinforcement learning algorithm.

Table 4. *Cont.*

Files with the Results of Our Work	Who Use the File	Description
trained_error_prediction_model.h5	Neuronal network trained without reinforcement algorithm	Model trained without reinforcement learning algorithm.
NnImageReader.traineddata	Neuronal network trained without reinforcement algorithm	Training data of the neural network for image information detection.

4. Discussion

At present, the automation of repetitive and tedious processes through the use of artificial intelligence and other technologies has revolutionized efficiency in various areas. This allows for a considerable saving of time and resources, which can be employed in other more strategic tasks.

In the context of a constantly evolving digital landscape, software systems play a key role in automating various tasks on the Internet. However, ensuring the reliability and efficiency of these systems remains a significant challenge. The occurrence of errors in code can lead to service interruptions or incorrect results. To address this issue, a new generation of self-adaptive software has emerged, capable of rectifying errors and optimizing performance autonomously.

It is possible to design software with adaptive capabilities that allow it to automatically adjust to changes in its environment. This approach, known as adaptive systems, uses techniques such as machine learning algorithms, expert systems, and genetic algorithms to dynamically adjust its behavior and continue meeting the desired objective even in variable environments. These systems are capable of constantly monitoring their environment, identifying significant changes, and intelligently adapting their functioning to maintain effectiveness, as mentioned by Cheng [46].

Our analysis confirms that, in addition to the mentioned techniques, the integration of deep neural networks and reinforcement learning algorithms significantly enhances the adaptability of the software. We have developed prototypes that demonstrate these systems can not only adapt to real-time changes but also predict future changes and proactively adjust their behavior to meet established objectives.

According to Salehie [45], for software to automatically adjust to changes in its environment and remain useful, it is crucial to consider several elements. These include the following: (1) the ability to collect and analyze data in real-time; (2) the implementation of efficient adaptation algorithms; (3) the integration of feedback mechanisms that allow the software to learn and improve its performance over time; and (4) the creation of flexible interfaces that facilitate interaction with other systems and users. Additionally, it is important to design the software with (5) modular and scalable architecture that facilitates the incorporation of new functionalities and adaptation to future changes in the environment.

In our study, we have identified that, in addition to the mentioned elements, it is essential to implement (6) continuous monitoring systems and predictive analysis. These systems enable the detection of patterns and trends in the environment, facilitating proactive adaptation of the software. It is also crucial to incorporate (7) automated testing and validation techniques to ensure that adaptive changes do not introduce new errors and maintain the software's functionality.

To mitigate the risks of self-rewriting software, it is essential to (1) thoroughly train the system through simulations and testing in multiple scenarios; (2) implement rollback mechanisms that allow the system to revert to a previous stable version if new errors are detected; and (3) engage in continuous monitoring and analysis to supervise system performance and adjust self-rewriting algorithms to improve accuracy and reduce the possibility of new errors, according to [50,51].

In our work, we identified that it is also necessary to implement (4) a continuous testing framework to mitigate risks. We implemented an A/B testing system that allows us to evaluate and adjust self-rewriting algorithms in a safe environment before deploying them

to production. Additionally, the integration of automatic rollback mechanisms and constant monitoring ensures that any introduced error can be quickly identified and corrected without affecting system stability.

A/B testing is an experimental method used to compare two versions of a variable to determine which is more effective. In the context of our study, these tests are used to evaluate and optimize the performance of self-adaptive software in different workload scenarios. In an A/B test, users or traffic are divided into two groups: Group A, which experiences the original (or control) version of the software, and Group B, which tests a modified (or variant) version.

The objective of A/B testing is to identify which of the two versions offers better performance in terms of specific metrics, such as response time, processing efficiency, and fault tolerance. This approach allows for incremental, data-driven adjustments, continuously improving the quality and performance of the software.

Self-rewriting significantly enhances system performance and efficiency by enabling autonomous reconfiguration in the face of errors, thus reducing downtime and improving system responsiveness. Our results demonstrate an improvement in the self-adaptation rate and faster response times due to the use of multiprocessing and automated optimization [52,53].

Our experiments show that self-rewriting not only enhances system resilience but also its overall performance. By implementing multiprocessing techniques and automated optimization, we observed a significant reduction in downtime and an improvement in response speed under variable loads, thus validating the effectiveness of self-adaptive systems in high-demand scenarios.

To ensure the coherence and integrity of data in self-rewriting software, it should employ a robust architecture that includes: (1) continuous monitoring: constant supervision of processes and data; (2) predefined abstractions: Utilization of pre-programmed steps to address errors; and (3) error handling: storage of errors with detailed descriptions and the ability for automated resolution based on previous diagnostics [54,55].

In our research, in addition to what Demirkan and Lewis suggest, we developed and tested a (4) version control system that allows the software to make changes without compromising data integrity. This system includes an automatic rollback feature that is activated when inconsistencies are detected, ensuring that the data remains coherent and accurate throughout the rewriting process.

Although the software we propose has a self-adaptation capability with an 80% success rate to errors and changes in its environment, there are still complex situations where the adaptation may not be completely effective. These may include drastic changes in web page structures or the introduction of new types of errors that were not considered in the initial models.

The use of virtual cables and multiprocessing significantly improved the speed and efficiency of the software, but the implementation and configuration of these technologies can increase system complexity, which could limit its scalability in certain operational environments.

Our proposal uses multiprocessing to improve performance, but this also implies intensive use of system resources. A robust hardware configuration is required to fully exploit the software's capabilities, which may not be available on all platforms.

Despite the fact that the use of multiprocessing and virtual cables improves scalability, there is still an inherent limitation in terms of parallelism and the efficient handling of large volumes of data. This can limit the software's effectiveness in environments that are extremely large or have very high data demands.

We suggest, as potential future research directions, the development of more advanced machine learning algorithms and artificial intelligence models that can handle a greater variety of errors and structural changes in web pages. This would include the integration of deep learning techniques to improve accuracy in error detection and correction.

Optimization methods should be implemented that reduce system complexity and improve the efficiency of multiprocessing and the use of virtual cables. This could involve

developing new techniques for task parallelization and distribution that maximize system resource usage without compromising scalability.

Finally, methods should be developed to further optimize system resource usage, possibly through the use of distributed processing techniques or cloud computing to handle large volumes of data without overloading local resources. This could include developing more efficient resource management algorithms that distribute workloads more effectively.

5. Conclusions

In this article, we present an adaptive software designed for pre-programmed tasks on the Internet, demonstrating its ability to rectify errors, optimize performance, and maintain continuous functionality in the presence of failures. The integration of technologies such as virtual cables and multiprocessing significantly contributed to its efficiency and effectiveness. This systematic and iterative approach in software development ensured a reliable and efficient solution.

It is important to note that, until now, this type of software has been limited in the commercial sphere and was less available, especially software with the ability to autonomously write its own code completely. Nevertheless, artificial intelligence is revolutionizing this field and holds great promise in this regard.

The creation of an adaptive system for pre-programmed tasks on the Internet, responsive to changes in its environment with automatic reconfiguration features in the face of possible code failures, is achievable today by combining artificial intelligence and employing technologies such as Python, MySQL, Firebase, Docker, Flask, OpenCV, Selenium, Scikit-learn, Pandas, and Jupyter Notebook.

The research addresses various fundamental aspects of software development and performance, crucial for its efficiency and reliability. Exhaustive tests were conducted to evaluate different mathematical models and implemented algorithms, allowing us to demonstrate the effectiveness of the proposed solution.

The software we developed can adapt to emerging errors and changes in its environment and has the ability to reconfigure itself autonomously, ensuring continuous and reliable operation. The adaptation rate has been modeled and monitored based on the time elapsed since the last adaptation, providing valuable insights into the system's efficiency.

The incorporation of agile development practices, along with technologies like multiprocessing and virtual cables based on the I2C protocol, significantly improved the software's efficiency. Test results indicate increased execution speed and better utilization of system resources.

During the testing phase of the proposal's development, the software's performance was evaluated in different scenarios and circumstances. Mathematical models, such as the Poisson distribution, were utilized to gain a better understanding of the software's behavior in the face of potential failures.

The implementation of multiprocessing and virtual cables allowed for greater scalability of the system and improved effectiveness in handling large volumes of data. These enhancements are crucial to meet future challenges and enable the software to adapt to the growing demand for its services.

Our study indicates that the combination of predictive analytics and AI-based systems significantly enhances error detection capabilities. We have implemented a hybrid model that uses both neural networks and reinforcement learning algorithms to identify and correct faults in real time, which has proven to be highly effective in our controlled tests.

6. Patents

The copyright registration number in Mexico is 03-2023-021514391200-01.

Author Contributions: Conceptualization, M.M.G. and L.C.G.M.R.; Data curation, L.C.G.M.R.; Formal analysis, R.P.Z.; Funding acquisition, M.M.G., L.C.G.M.R. and R.P.Z.; Investigation, M.M.G.; Methodology, L.C.G.M.R.; Project administration, M.M.G.; Resources, L.C.G.M.R.; Software, L.C.G.M.R. and R.P.Z.; Supervision, L.C.G.M.R.; Validation, M.M.G. and L.C.G.M.R.; Visualization, R.P.Z.; Writing—original draft, M.M.G.; Writing—review and editing, L.C.G.M.R. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The data created can be observed in the results section.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Wei, X.; Wang, Z.; Yang, S. An Automatic Generation and Verification Method of Software Requirements Specification. *Electronics* **2023**, *12*, 2734. [CrossRef]
2. Filieri, A.; Hoffmann, H.; Maggio, M. Automated design of self-adaptive software with control-theoretical formal guarantees. In Proceedings of the 36th International Conference on Software Engineering, New York, NY, USA, 31 May–7 June 2014; pp. 299–310. [CrossRef]
3. Filieri, A.; Hoffmann, H.; Maggio, M. Automated multi-objective control for self-adaptive software design. In Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, New York, NY, USA, 30 August–4 September 2015; pp. 13–24. [CrossRef]
4. Kokash, N. Self-Adaptive Systems: A Survey of Current Approaches and Challenges. *J. Syst. Softw.* **2020**, *159*, 110430. [CrossRef]
5. Zeadally, S.; Sanislav, T.; Mois, G.D. Self-Adaptation Techniques in Cyber-Physical Systems (CPSs). *IEEE Access* **2019**, *7*, 171128. [CrossRef]
6. Manzhos, Y.; Sokolova, Y. A Software Verification Method for the Internet of Things and Cyber-Physical Systems. *Computation* **2023**, *11*, 135. [CrossRef]
7. Barbera, G.; Araujo, L.; Fernandes, S. The Value of Web Data Scraping: An Application to TripAdvisor. *Big Data Cogn. Comput.* **2023**, *7*, 121. [CrossRef]
8. González, F.; Torres-Ruiz, M.; Rivera-Torruco, G.; Chonona-Hernández, L.; Quintero, R. A Natural-Language-Processing-Based Method for the Clustering and Analysis of Movie Reviews and Classification by Genre. *Mathematics* **2023**, *11*, 4735. [CrossRef]
9. Santos, F.; Acosta, N. An Approach Based on Web Scraping and Denoising Encoders to Curate Food Security Datasets. *Agriculture* **2023**, *13*, 1015. [CrossRef]
10. Oancea, B. Automatic Product Classification Using Supervised Machine Learning Algorithms in Price Statistics. *Mathematics* **2023**, *11*, 1588. [CrossRef]
11. Huang, H.; Du, R.; Wang, Z.; Li, X.; Yuan, G. A Malicious Code Detection Method Based on Stacked Depthwise Separable Convolutions and Attention Mechanism. *Sensors* **2023**, *23*, 7084. [CrossRef]
12. García-Grao, G.; Carrera, Á. Extending the OSLC Standard for ECA-Based Automation. *Electronics* **2023**, *12*, 3043. [CrossRef]
13. Pejić Bach, M.; Topalović, A.; Krstić, Ž.; Ivec, A. Predictive Maintenance in Industry 4.0 for the SMEs: A Decision Support System Case Study Using Open-Source Software. *Designs* **2023**, *7*, 98. [CrossRef]
14. Barakat, C.S.; Sharafutdinov, K.; Busch, J.; Saffaran, S.; Bates, D.G.; Hardman, J.G.; Schuppert, A.; Brynjólfsson, S.; Fritsch, S.; Riedel, M. Developing an Artificial Intelligence-Based Representation of a Virtual Patient Model for Real-Time Diagnosis of Acute Respiratory Distress Syndrome. *Diagnostics* **2023**, *13*, 2098. [CrossRef]
15. Nancy, A.A.; Ravindran, D.; Vincent, D.R.; Srinivasan, K.; Chang, C.-Y. Fog-Based Smart Cardiovascular Disease Prediction System Powered by Modified Gated Recurrent Unit. *Diagnostics* **2023**, *13*, 2071. [CrossRef] [PubMed]
16. Oliveira, R.C.d.; Silva, R.D.d.S.e. Artificial Intelligence in Agriculture: Benefits, Challenges, and Trends. *Appl. Sci.* **2023**, *13*, 7405. [CrossRef]
17. Lv, X.; Zhang, G.; Bai, Z.; Zhou, X.; Shi, Z.; Zhu, M. Adaptive Neural Network Global Fractional Order Fast Terminal Sliding Mode Model-Free Intelligent PID Control for Hypersonic Vehicle's Ground Thermal Environment. *Aerospace* **2023**, *10*, 777. [CrossRef]
18. Panayiotou, K.; Tsardoulias, E.; Zolotas, C.; Symeonidis, A.L.; Petrou, L. A Framework for Rapid Robotic Application Development for Citizen Developers. *Software* **2022**, *1*, 53–79. [CrossRef]
19. Wang, D.; Yang, M.; Zhang, W. Wind Power Group Prediction Model Based on Multi-Task Learning. *Electronics* **2023**, *12*, 3683. [CrossRef]
20. Dominguez, X.; Mantilla-Pérez, P.; Gimenez, N.; El-Sayed, I.; Díaz Millán, M.A.; Arboleya, P. Web-Based Simulation Environment for Vehicular Electrical Networks. *Energies* **2021**, *14*, 6087. [CrossRef]
21. Moreno, V.; Génova, G.; Alejandres, M.; Fraga, A. Automatic Classification of Web Images as UML Static Diagrams Using Machine Learning Techniques. *Appl. Sci.* **2020**, *10*, 2406. [CrossRef]
22. Challenger, M.; Tezel, B.T.; Alaca, O.F.; Tekinerdogan, B.; Kardas, G. Development of Semantic Web-Enabled BDI Multi-Agent Systems Using SEA_ML: An Electronic Bartering Case Study. *Appl. Sci.* **2018**, *8*, 688. [CrossRef]

23. Stergiou, C.L.; Koidou, M.P.; Psannis, K.E. IoT-Based Big Data Secure Transmission and Management over Cloud System: A Healthcare Digital Twin Scenario. *Appl. Sci.* **2023**, *13*, 9165. [CrossRef]
24. Krichen, M. A Survey on Formal Verification and Validation Techniques for Internet of Things. *Appl. Sci.* **2023**, *13*, 8122. [CrossRef]
25. Jiang, T.; Chen, L.; Chen, W.; Meng, W.; Qi, P. ReliaMatch: Semi-Supervised Classification with Reliable Match. *Appl. Sci.* **2023**, *13*, 8856. [CrossRef]
26. Amiri, Z.; Heidari, A.; Darbandi, M.; Yazdani, Y.; Jafari Navimipour, N.; Esmaeilpour, M.; Sheykhi, F.; Unal, M. The Personal Health Applications of Machine Learning Techniques in the Internet of Behaviors. *Sustainability* **2023**, *15*, 12406. [CrossRef]
27. Barbedo, J.G.A. Deep Learning for Soybean Monitoring and Management. *Seeds* **2023**, *2*, 340–356. [CrossRef]
28. Eivazi, H.; Tröger, J.-A.; Wittek, S.; Hartmann, S.; Rausch, A. FE2 Computations with Deep Neural Networks: Algorithmic Structure, Data Generation, and Implementation. *Math. Comput. Appl.* **2023**, *28*, 91. [CrossRef]
29. Poměnková, J.; Malach, T. Optimized Classifier Learning for Face Recognition Performance Boost in Security and Surveillance Applications. *Sensors* **2023**, *23*, 7012. [CrossRef]
30. Kammüller, F.; Satija, D. Explanation of Student Attendance AI Prediction with the Isabelle Infrastructure Framework. *Information* **2023**, *14*, 453. [CrossRef]
31. Gong, C.; Ma, J. Automated Model Selection of the Two-Layer Mixtures of Gaussian Process Functional Regressions for Curve Clustering and Prediction. *Mathematics* **2023**, *11*, 2592. [CrossRef]
32. Alshdadi, A.A. Evaluation of IoT-Based Smart Home Assistance for Elderly People Using Robot. *Electronics* **2023**, *12*, 2627. [CrossRef]
33. Al-Naime, K.; Al-Anbuky, A.; Mawston, G. Internet of Things Gateway Edge for Movement Monitoring in a Smart Healthcare System. *Electronics* **2023**, *12*, 3449. [CrossRef]
34. Zhang, W.; You, H.; Wang, C.; Zhang, H.; Tang, Y. Parallel Optimization for Large Scale Interferometric Synthetic Aperture Radar Data Processing. *Remote Sens.* **2023**, *15*, 1850. [CrossRef]
35. Vo, C.P.; Jeon, J.h. An Integrated Motion Planning Scheme for Safe Autonomous Vehicles in Highly Dynamic Environments. *Electronics* **2023**, *12*, 1566. [CrossRef]
36. Chen, Z.; Cui, H.; Wu, E.; Yu, X. Computation and Communication Efficient Adaptive Federated Optimization of Federated Learning for Internet of Things. *Electronics* **2023**, *12*, 3451. [CrossRef]
37. Du, Z.; Peng, C.; Yoshinaga, T.; Wu, C. A Q-Learning-Based Load Balancing Method for Real-Time Task Processing in Edge-Cloud Networks. *Electronics* **2023**, *12*, 3254. [CrossRef]
38. Zhang, D.; Zhong, Z.; Xia, Y.; Wang, Z.; Xiong, W. An Automatic Classification System for Environmental Sound in Smart Cities. *Sensors* **2023**, *23*, 6823. [CrossRef]
39. Palanisamy, P.; Padmanabhan, A.; Ramasamy, A.; Subramaniam, S. Remote Patient Activity Monitoring System by Integrating IoT Sensors and Artificial Intelligence Techniques. *Sensors* **2023**, *23*, 5869. [CrossRef]
40. Aslan, Ö. Separating Malicious from Benign Software Using Deep Learning Algorithm. *Electronics* **2023**, *12*, 1861. [CrossRef]
41. Galan-Uribe, E.; Morales-Velazquez, L.; Osornio-Rios, R.A. FPGA-Based Methodology for Detecting Positional Accuracy Degradation in Industrial Robots. *Appl. Sci.* **2023**, *13*, 8493. [CrossRef]
42. Zhou, P.; Tang, S.; Sun, Z. Simulation Research on the Grouser Effect of a Reconfigurable Wheel-Crawler Integrated Walking Mechanism Based on the Surface Response Method. *Appl. Sci.* **2023**, *13*, 4202. [CrossRef]
43. Aguayo, O.; Sepúlveda, S. Variability Management in Dynamic Software Product Lines for Self-Adaptive Systems—A Systematic Mapping. *Appl. Sci.* **2022**, *12*, 10240. [CrossRef]
44. Mao, Y.; Migliore, V.; Nicomette, V. MATANA: A Reconfigurable Framework for Runtime Attack Detection Based on the Analysis of Microarchitectural Signals. *Appl. Sci.* **2022**, *12*, 1452. [CrossRef]
45. Salehie, M.; Tahvildari, L. Self-adaptive software: Landscape and research challenges. *ACM Trans. Auton. Adapt. Syst. (TAAS)* **2009**, *4*, 1–42. [CrossRef]
46. Cheng, B.H.C.; de Lemos, R.; Giese, H.; Inverardi, P.; Magee, J.; Andersson, J. Software Engineering for Self-Adaptive Systems: A Research Roadmap. In *Software Engineering for Self-Adaptive Systems*; Springer: Berlin/Heidelberg, Germany, 2009; pp. 1–26. [CrossRef]
47. Krupitzer, C.; Roth, F.M.; VanSyckel, S.; Schiele, G.; Becker, C. A Survey on Engineering Approaches for Self-Adaptive Systems. *Pervasive Mob. Comput.* **2015**, *17*, 184–206. [CrossRef]
48. Macedo, N.; Cunha, J.; Pacheco, H. A Survey on Automated Debugging Systems. *ACM Comput. Surv. (CSUR)* **2018**, *51*, 15–39.
49. Ghosh, S.; Chakraborty, S. Neural Network-Based Fault Detection Methods: A Survey. *J. Comput. Secur.* **2018**, *7*, 14–24.
50. Dobson, G.; Sawyer, P.; Hall, R. Using scenarios for runtime requirements monitoring: A position paper. In Proceedings of the 14th IEEE International Requirements Engineering Conference (RE'06), Minneapolis/St. Paul, MN, USA, 11–15 September 2006; IEEE: New York, NY, USA; pp. 354–357. [CrossRef]
51. Menascé, D.A.; Kephart, J.O. Guest Editors' Introduction: Autonomic Computing. *IEEE Internet Comput.* **2007**, *11*, 18–21. [CrossRef]
52. Brun, Y.; Di Marco, A.; Serugendo, G.M.; Gacek, C.; Giese, H.; Kienle, H.M.; Zambonelli, F. Engineering Self-Adaptive Systems through Feedback Loops. In *Software Engineering for Self-Adaptive Systems*; Springer: Berlin/Heidelberg, Germany, 2009; pp. 48–70. [CrossRef]

53. Garlan, D.; Cheng, S.W.; Huang, A.C.; Schmerl, B.; Steenkiste, P. Rainbow: Architecture-based self-adaptation with reusable infrastructure. *Computer* **2004**, *37*, 46–54. [CrossRef]
54. Demirkan, H.; Delen, D. Leveraging the capabilities of service-oriented decision support systems: Putting analytics and big data in cloud. *Decis. Support Syst.* **2013**, *55*, 412–421. [CrossRef]
55. Lewis, G.A.; Morris, E.J.; Simanta, S.; Wrage, L. Common principles in dynamic system adaptation. In Proceedings of the ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems, Vancouver, BC, Canada, 18–19 May 2009; IEEE: New York, NY, USA, 2009; pp. 50–57. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.