© JUL 2025 | IRE Journals | Volume 9 Issue 1 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I1-1718982

A Web-Based Automated Fuzzing Tool for Web Applications

RITIK DHANKHAR[1], AJAY SHARMA[2], SHAKTI ARORA[3]
[1, 2]Student, Department of Cyber Security, Panipat Institute of Engineering and Technology, India
[3]Professor, Department of Cyber Security, Panipat Institute of Engineering and Technology, India

Abstract- ***The importance of Web Application Security grows daily as more organizations are threatened and attacked by cyber criminals. With the growing threat from cyber criminals, performing security testing to identify vulnerabilities in web systems is critical. Of all security testing techniques, fuzz testing is perhaps the best technique available today. Fuzz testing involves injecting input into a target application, including mal- formed, unexpected, or random data to see how it reacts when it receives bad data. In the case of web applications, fuzz testing is done by sending numerous HTTP requests (each request contains different forms of crafted or invalid data) to a web server to measure the response generated by the server. This study will create an Automated Web Application Fuzzer which will be integrated with Jenkins so that continuous security testing of Web Applications can occur. Test cases were created using known security vulnerabilities within web applications. Testing revealed that the automation tool found vulnerabilities in thirteen (13) out of fifteen (15) test cases. Therefore, testing reveals that the majority of web vulnerabilities can be easily identified simply by reviewing the content of HTTP responses, thereby validating the effectiveness of the proposed auto- mated web application fuzzing methodology***

Keywords— Web Application Security, Fuzz Testing, Automated Vulnerability Detection, HTTP Request Testing

I. INTRODUCTION

Regarding cybersecurity for web applications, the exponential rise in cyber threats in the last couple of years has made it a major concern. Well-known is that the majority of scanned websites continue to be left behind, with a staggering number of them not patching their gaping vulnerabilities. Many of which have been labeled as “critical.”

Through these weaknesses, malicious attackers can easily find gaping holes to breach systems and take unauthorized actions. Since web-based applications are exposed to the world and become available for public exploitation, the probability of being attacked is basically guaranteed.

To avert these threats, developers should put the highest priority on security from the outset when building web applications.

Researched methods in software security include review of the source code, static analysis, binary code analysis, fuzz testing, fault injections, risk assessment, vulnerability scanning, and penetration testing. The key to spotting security flaws lies in the combination of testing methods, and fuzz testing or “fuzzing” is at its heart. Fuzzing is like a high- stakes bet that sends abnormal or distorted data streams to a target application in the hope of making it crash. Since it can monitor the system in real time, it can find very unusual and sophisticated security issues that conventional methods miss. When applied to the world of web applications, the technique sends a tidal wave of expertly designed HTTP re- quests to the application being tested. Cybersecurity specialists will fine-tune their toolkits to mimic well-known attack methods. Following the execution of the test run, the subsequent system responses are meticulously analyzed to ascertain the existence of potential vulnerabilities, culminating in the issuance of comprehensive and detailed security reports.

II. PROBLEMS STATEMENT

Web applications have become a primary target for cyber- attacks because of their public accessibility and ever-in- creasing complexity. Traditional security testing techniques, such as manual code review, static analysis, and penetration testing, are often time-consuming, costly, and highly de- pendent on human expertise. These methods may fail to detect hidden,

input-based vulnerabilities that can be exploited through malformed or unexpected user data.

Currently, the available automation tools have insufficient coverage and are mostly not capable of generating, on the fly, smart test cases for state-of-the-art web applications. Be- sides, many security testing processes are still not integrated with CD pipelines, and real-time vulnerability detection is hard to accomplish in the software development life cycle.

There is an urgent need for an automated, efficient, and scalable solution that continuously tests web applications for vulnerabilities by generating large volumes of crafted HTTP requests and analyzing them. In the absence of such a system, there is a greater chance that security flaws may go un- noticed, leading to data breaches and system compromises.

Hence, this research is focused on the design and development of a web-based automated fuzzing that can be easily integrated with continuous integration environments to enhance the accuracy and efficiency of detecting vulnerabilities in web applications.

III. LITERATURE REVIEW

Fuzz testing tools for assessing software security are known as fuzzers. Several tools exist for fuzz testing web applications, each offering different levels of automation and configuration options.

JBroFuzz, created by the Open Web Application Security Project (OWASP), allows users to manually create HTTP re- quests and specify where payloads should be injected. It provides categorized payload lists and saves both the requests and server responses. This lets testers analyze the results manually after running the tests.

Wapiti is a web application vulnerability scanner that con- ducts black-box testing by crawling web pages to find forms and scripts that might be weak against injection attacks. It generates payloads automatically and alerts users when it detects unusual behavior, like internal server errors or timeouts. However, Wapiti does not provide detailed control over choosing specific parts of an HTTP request for fuzzing.

Wfuzz is a versatile fuzzing framework that allows testers to set target URLs, HTTP headers, request bodies, and custom payload lists. It enables focused input injection by letting users pick specific locations in HTTP requests for fuzzing data.

Burp Intruder is a popular automated attack tool used for testing web application security. It offers configurable pay- load generation and supports different fuzzing techniques. However, many advanced payload options are only available in its paid version, while the free version requires manual input of test values.

w3af is an open-source web application attack framework that includes a fuzzy request editor. It allows testers to customize HTTP requests and use Python-based scripts to create fuzzing inputs dynamically for certain parts of a request.

While tools like JBroFuzz, Wfuzz, Burp Intruder, and w3af let testers define injection points in HTTP requests, most re- quire testers to manually build full HTTP requests. Wapiti uses automated crawling, which restricts direct control over request manipulation. A better approach would involve automatic request collection to lessen the manual workload.

Most existing tools depend on predefined fuzzing vectors, which include sets of crafted input values. Some tools sup- port brute-force generation, numeric range creation, input combination, character mutation, and reusing inputs based on responses. Common response analysis methods include grouping server responses by status codes, content length, timeout behavior, and matching patterns using regular ex-pressions.

Despite these features, identifying vulnerabilities still often needs substantial manual review by testers. This underscores the need for a more intelligent and automated fuzzing system.

In this research, we propose a new web application fuzzing tool that allows dynamic configuration of HTTP requests, automates input generation, and conducts advanced response analysis based on server errors, timeout detection, and con- tent inspection.

This aims to improve detection accuracy and cut down on manual effort.

IV. PROBLEM ANALYSIS

A. Fuzz Testing for Web Applications

Web application attacks are generally conducted by injecting crafted inputs into target systems to trigger vulnerable behaviors. Attackers often manipulate various parts of HTTP requests in order to exploit weaknesses. An effective web ap- plication fuzzer, therefore, must support input generation at any location inside an HTTP request and not be limited to only user-controlled parameters.

For efficient fuzz testing, the execution of HTTP requests needs to be automated. Certainly, a tool makes life much easier if it automatically captures HTTP requests by interactively using the target application. Such behavior can be simulated with tools like Selenium WebDriver, which allows the tester to programmatically control a web browser. This control can be combined with BrowserMob Proxy to record all the HTTP requests that are produced by interacting with a browser. Thus, a tester only needs to interact with the web application; the tool collects requests that can be reused afterwards for fuzzing.

After capturing the HTTP requests, the tester needs to set up how those requests will be altered and replayed. The fuzzer should be able to provide the tester with an interface for con- trolling exactly what parts of the HTTP request will be tampered with. Since web application fuzzing is an integration- level test, it can be executed in a continuous integration environment. That automated execution can ensure that security testing is consistently performed as the application evolves. To this end, the continuous integration environment uses. It is an open-source automation server that supports continuous integration and delivery, providing continuous repeated execution of security tests with seamless integration with various development tools.

B. Input Generation

A web application fuzzer needs to support several input generation techniques to effectively test different parts of HTTP requests. The generated values are inserted into pre-defined positions of the HTTP request and sent to the target application. Input generation techniques commonly include fuzz vectors, number generation, brute-force string creation, and similar-string mutation.

A fuzz vector is a pre-defined set of crafted payloads to test for common vulnerabilities such as SQL injection, XSS, and path traversal. Many standardized fuzz vectors are provided by community-driven sources such as OWASP.

Number generators fill fields with numeric values over specified ranges and are of particular use when testing those fields that strictly accept numeric input. A brute-force generator creates multiple string combinations based on defined character sets, enabling the exhaustive testing of input possibilities. Similar-string generators create slight variations of a base input by modifying characters incrementally.

Generated inputs may be further modified by transformation techniques. String substitution replaces certain character sequences with alternative values. Case modification changes the case of the characters so that weak validation mechanisms using simple blacklists are circumvented. Encoding techniques are used to keep the generated inputs compliant with HTTP standards.

Advanced use cases would also involve combinations of the input generation techniques to generate complex, nested test cases. Certain fields of an HTTP request require values to be dynamic, extracted from server responses themselves, such as anti-CSRF tokens. The tool shall therefore be capable of automatically extracting response values and re-inserting them into subsequent requests.

C. Vulnerability detection

Performing effective fuzz testing requires the automated analysis of many test results. Thousands of HTTP requests may be generated and sent; hence, the inspection of these by hand is unfeasible. On the other hand, the fuzzing tool itself should provide intelligent mechanisms to support vulnerability detection.

One detection method is based on HTTP response status codes. When certain crafted inputs cause server-side failures, the application may return status codes in

© JUL 2025 | IRE Journals | Volume 9 Issue 1 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I1-1718982

the 5xx range, which represent internal server errors. These responses can be used as indicators of possible vulnerabilities.

Another detection technique focuses on response time analysis. Certain malicious inputs can significantly increase the server's processing time or make the application hang. Such behaviours affect system availability and may be used to conduct denial-of-service attacks. Detecting abnormal de- lays or timeouts helps identify such vulnerabilities. Another key mechanism for detection is content-based analysis. The tool should automatically compare responses received against predefined templates by defining expected response patterns. Deviations may indeed indicate successful exploitation attempts. On the other hand, response-based detection could generate false positives, and content-based detection needs precise configuration by the tester.

V. PROPOSED SOLUTION

The proposed solution is designed as an automated and modular web application fuzzing framework to facilitate the easier detection of vulnerabilities while further reducing the effort needed for manual testing. The core objective of the system is to make web security testing faster, repeatable, and suitable for both standalone testing and continuous integration environments.

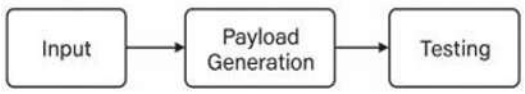

Figure 1. Tool Overview

The workflow of the system begins with the configuration phase, during which a tester defines how the fuzzing process should operate. Instead of hardcoding test logic, the tool uses a configuration-driven approach. This allows flexible selection of target HTTP requests, injection points, and payload generation strategies without having to modify the core engine.

An important feature is the automatic handling of the request by this system. In this framework, rather than forcing testers to create raw HTTP requests manually, browser interaction automatically captures requests. This improves accuracy and ensures that real-world request structures are tested.

The system generates a payload and dynamically injects it into every part of the HTTP request, such as headers, parameters, body content, and tokens, during execution. That is why the tool is capable of testing both user-input and non- user-input attack surfaces.

This implemented design starts the testing workflow by collecting real HTTP requests through browser-based interaction, and stores for later reuse during testing. This approach minimizes human error by ensuring that all headers, cookies, parameters, and authentication data are preserved precisely as they would occur in real usage. Then, the tester will decide which parts of the request need to be changed and tested.

The tool injects different kinds of test inputs into the selected locations of the HTTP requests by applying automated pay- load generation techniques. Those payloads are generated dynamically from the predefined attack patterns, thus enabling testing against both known classes of vulnerabilities and unexpected edge cases. In contrast with a static scanner, it actively modifies the live traffic and observes real-time ap- plication behavior.

The system carries out by issuing hundreds of thousands of modified requests to the target application and continuously monitoring its responses. Here, it doesn't just log the responses; rather, it actively looks at the responses for abnormal server behavior, such as crashes, error responses, slow processing, and unexpected output patterns. This practical design reduces much of the manual log inspection burden and expedites the vulnerability identification process.

The system's reporting mechanism is designed for real-world usability. Each test case generates a structured report containing the original request, the injected payload, observed behavior, and the detected issue. These reports can be used directly as evidence in project documentation, security audits, or academic evaluations.

© JUL 2025 | IRE Journals | Volume 9 Issue 1 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I1-1718982

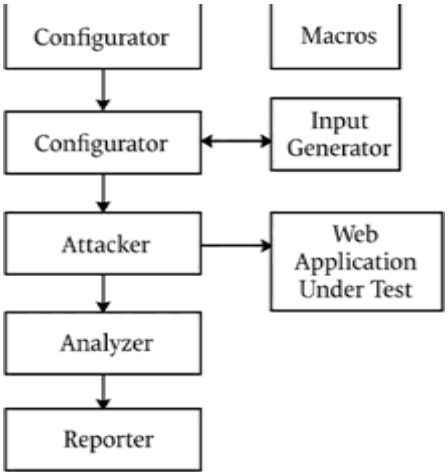

Figure 2. Tool Modules

Figure 2 illustrates the internal working structure of the pro- posed web-based fuzzing system. The architecture design is intended to be practical for automation, flexible, and based on real-world testing requirements rather than mere theoretical concepts.

The Request Collector captures real HTTP requests that are generated by a tester while interacting with a live web application. This avoids manual writing of raw HTTP requests; the module records them automatically so that their format, headers, cookies, and parameters are similar to real-world traffic. This increases the accuracy of the testing processes and minimizes configuration-related problems.

The Configurator module allows for an easy way of managing all test configurations. It selects which captured requests should be tested, defines what kind of payload injection should take place and the order of execution. In addition, this module manages reusable workflows (macros) and payload libraries. The intent of this module is to put the tester in full control without the need to modify source code.

The Input Generator module generates test payloads. In practice, the module can generate such a payload from pre- defined lists of payloads, number ranges, brute-force combi- nations, and by modifying strings. This way, the application is tested both for attack patterns which are already known and unpredictable inputs. It also transforms these inputs like encoding them or changing cases.

The Attacker module serves as an execution engine: it takes the generated payloads and injects them into selected parts of the HTTP requests. These modified requests are then sent to the target web application in an automated manner. This module is optimized to handle large volumes of requests efficiently and, hence, is useful for performing stress and robustness tests.

The Analyzer module processes the server responses in real time. Instead of just presenting raw response data, the mod- ule actively searches for patterns that may indicate a vulnerability. It checks response codes, response times, and specific content patterns to flag suspicious behavior, reducing the need for manual inspection and speeding up the process of discovering possible vulnerabilities.

The Reporter module compiles all findings into structured reports. Each report includes original request, injected pay- load, observed behavior, and detected issues. The reports come in a format that can be used directly in academic documentation and security audits. The modules together pro- vide a complete automated testing pipeline where collection, modification, execution, and analysis of requests are tightly integrated, along with reporting. This architecture lets the tool work both as a standalone security testing system and part of a continuous security testing workflow.

The system also supports execution in standalone mode, as well as continuous integration mode. In the interest of im- proving the tool's ease of use, logging and debugging features are included within the tool, allowing testers to follow the paths of execution and recreate particular instances. The system's built-in security features include rate limiting, safe- mode execution, and environmental isolation, protecting against inadvertent destruction of a production environment.

VI. TESTING AND EXPERIMENTATION

Experiments were performed to assess the efficiency of the proposed fuzzing tool. Vulnerable web applications were intentionally chosen for conducting the experiments. The scope of testing was confined to vulnerabilities that can be identified via fuzz testing,

using a list of the OWASP Top 10 security risks from 2013. For experimentation, four vulnerable platforms were used: OWASP WebGoat 5.2, Pentest- erLab 2, bWAPP 2.2, and OWASP Mutillidae 2.0.9.

For each test case, HTTP requests were first captured by interacting with the target web application using the request collection mechanism. Then, the tester chose specific re- quests to execute, editing the request configuration files for defining the methods of input generation and expected response patterns. As all test cases were known to contain vulnerabilities, it was expected that the tool would be able to correctly identify the presence of security flaws.

During testing, the tool was able to identify vulnerabilities in the majority of test cases. In a few test scenarios, while the injected payloads did indeed trigger vulnerable behaviors, the detection mechanisms leveraging the HTTP status codes, timeout analysis, and simple response patterns were not enough to automatically flag such issues.

The experimental results showed that content-based response analysis is the most reliable technique in vulnerability detection. Not all actual security vulnerabilities were dis- closed by anomalies in HTTP status codes or timeout behavior. Therefore, for accurate detection of vulnerabilities, precise definition of the expected HTTP response content by the tester is critical.

Our application allows the user to control how inputs are auto-generated from a request configuration file using their preferred method, and provide them with all the necessary HTTP requests. When testing with BrowserMob Proxy and Selenium WebDriver, it will eliminate the need for the tester to write out each and every HTTP request they would like to send. The method that provides the most accurate indication of a potential vulnerability is to analyze what appears on an HTTP response; this is because frequently no anomaly can be detected via the HTTP status code (client error responses) nor by timing abnormalities.

As a result, detection must be accomplished through analysis of the HTTP response content. In order for the tester to carry out this function correctly, he/she must provide the expected HTTP response so that the detection tool can determine if the actual HTTP response is equivalent to that provided by the user.
The results show that the proposed tool effectively identifies the most common web application vulnerabilities. However, it may miss some complex vulnerabilities that do not cause visible response changes. This suggests that automated tools need to be backed up by manual verification for critical systems. Overall, the tool speeds up testing and cuts down manual effort, making it useful for real-world security testing.

Experiments were performed to assess the efficiency of the proposed fuzzing tool. Vulnerable web applications were intentionally chosen for conducting the experiments. The scope of testing was confined to vulnerabilities that can be identified via fuzz testing, using a list of the OWASP Top 10 security risks from 2013. For experimentation, four vulnerable platforms were used: OWASP WebGoat 5.2, Pentester Lab 2, bWAPP 2.2, and OWASP Mutillidae 2.0.9.

For all test case, HTTP requests were first captured by inter- acting with the target web application using the request col- lection mechanism. Then, the tester chose specific requests to execute, editing the request configuration files for defining the methods of input generation and expected response patterns. As all test cases were known to contain vulnerabilities, it was expected that the tool would be able to correctly identify the presence of security flaws.

During testing, the tool was able to identify vulnerabilities in the majority of test cases. In a few test scenarios, while the injected payloads did indeed trigger vulnerable behaviors, the detection mechanisms leveraging the HTTP status codes, timeout analysis, and simple response patterns were not enough to automatically flag such issues

The experimental results showed that content-based response analysis is the most reliable technique in vulnerability detection. Not all actual security vulnerabilities were dis- closed by anomalies in HTTP status codes or timeout behavior. Therefore, for accurate detection of vulnerabilities, precise definition of the expected HTTP response content by the tester is critical.

VII. CONCLUSION

This research successfully designed and developed a web- based fuzzing tool to improve the security testing of web ap- plications. The tool automates the generation and injection of test payloads, which allows for faster and more efficient identification of security vulnerabilities. By cutting down on the need for manual input and configuration, the system helps security testers save time and boost overall testing ac- curacy.

The experimental results show that the tool can detect most common types of vulnerabilities, demonstrating its reliability for practical use. Its integration with continuous integration environments also adds to its usefulness by enabling re- peated and automated security testing during the software development lifecycle. This setup ensures that vulnerabilities can be identified early and continuously throughout development.

However, the tool has limitations in detecting complex vulnerabilities that do not cause clear changes in server responses. These limitations emphasize the need to combine automated testing with expert manual analysis for critical applications. Future improvements can focus on enhancing detection techniques, adding intelligent analysis mechanisms, and expanding the types of vulnerabilities it can identify.

In conclusion, the proposed system offers a practical and efficient approach to improving web application security. It lays a strong foundation for future research and development in the field of automated security testing and web application vulnerability assessment.

REFERENCES

[1] Kisten, M., Ezugwu, A. E. S., & Olusanya, M. O. (2024). Explainable artificial intelligence model for predictive maintenance in smart agricultural facilities. *IEEE Access*, 12, 24348–24367.

[2] Goel, N., Kaur, S., & Kumar, Y. (2022). Machine learning-based remote monitoring and predictive analytics system for crop and livestock. In *AI, Edge and IoT-Based Smart Agriculture* (pp. 395–407). Academic Press.

[3] Titirmare, S., Margal, P. B., Gupta, S., & Kumar, D. (2024). AI-powered predictive analytics for crop yield optimization. In *Agriculture 4.0* (pp. 89–110). CRC Press.

[4] Kishor, I., Mamodiya, U., Patil, V., & Naik, N. (2025). AI-integrated autonomous robotics for solar panel cleaning and predictive maintenance using drone and ground-based systems. *Scientific Reports*, 15(1), 32187.

[5] Shuriya, B. (2026). *Adoption of Deep Learning Driven Precision Agriculture for Optimizing Crop Productivity and Soil Health via Predictive Analytics and Autonomous Sensing Mechanisms*.

[6] Kar, P., & Chowdhury, S. (2024). IoT and drone-based field monitoring and surveillance system. In *Artificial Intelligence Techniques in Smart Agriculture* (pp. 253–266). Singapore: Springer Nature Singapore.

[7] Geetha, K., & Deepika, J. (2024). AI-driven drone-assisted smart farming framework for precision pest control and crop yield optimization. *National Journal of Smart Agriculture and Rural Innovation*, 11–19.

[8] Hernández Hernández, G. C., Gómez Gómez, J., & Jiménez-Cabas, J. (2025). Predictive models based on artificial intelligence to estimate crop yield: A literature review. *Agriculture*, 15(23), 2438.

[9] Liang, M., Nan, L., Peng, B., & Bo, G. (2026). Economic returns from AI-driven precision agriculture in degraded ecosystems: Productivity effects measured using UAV remote sensing. *Land Degradation & Development*.

[10] Guatno, C. P. V., Obillo, D. E. A., Taguinod, J. E., Sison, C. A. A. R. C., Dioses, R. M., & Abando, D. S. (2024, December). Harnessing AI for agriculture utilizing color-condition camera sensors and thermal imaging drones for crop color-condition detection and predictive yield analysis with inventory management system. In *International Conference on Green Energy, Computing and Intelligent Technology 2024 (GEn-CITy 2024)* (Vol. 2024, pp. 345–352). IET.

[11] Singh, D. P., Reddy, P. C. P., Devayani, G., Poongothai, S., Suganthi, G., & Babu, G. C. (2025, August). Drone-assisted precision agriculture with hybrid machine learning models

for sustainable farming. In *2025 Global Conference on Information Technology and Communication Networks (GITCON)* (pp. 1–6). IEEE.

[12] Ramteke, S. V., Varadwaj, P. K., & Tiwari, V. (2025, December). AI-enabled life cycle assessment of UAV-based spraying systems for climate-smart agriculture and SDG monitoring. In *2025 IEEE 17th International Conference on Computational Intelligence and Communication Networks (CICN)* (pp. 1708–1712). IEEE.

[13] Borah, S. K., Pal, D., Sarkar, S., & Sethi, L. N. (2025). AI-powered drones for sustainable agriculture and precision farming. In *Advancing Global Food Security with Agriculture 4.0 and 5.0* (pp. 69–98). IGI Global Scientific Publishing.

[14] Arsenoaia, V. N., Topa, D. C., Ratu, R. N., & Tenu, I. (2026). From sensing to intervention: A critical review of agricultural drones for precision agriculture, data-driven decision making, and sustainable intensification. *Agronomy*, 16(5), 564.

[15] Renuka, A. (2025). AI-driven predictive analytics in precision agriculture. *Scientific Journal of Artificial Intelligence and Blockchain Technologies*, 2(3), 9–17.

[16] Kumar, P., & Choudhury, D. (2025). Advancements in precision agriculture: Integrating machine learning techniques for crop monitoring and management. In *Artificial Intelligence in Microbial Research: Bridging the Gap* (pp. 29–57). Singapore: Springer Nature Singapore.

[17] Ugwu, O. P. C., Ogenyi, F. C., Alum, E. U., Eze, V. H. U., Basajja, M., Ugwu, J. N., Ugwu, C. N., Ejemot-Nwadiaro, R. I., Okon, M. B., Egba, S. I., & Ejim, U. D. (2025). Implementing artificial intelligence and machine learning algorithms for optimized crop management: A systematic review on data-driven approach to enhancing resource use and agricultural sustainability. *Cogent Food & Agriculture*, 11(1), 2569982.

[18] Ali, Z., Muhammad, A., Lee, N., Waqar, M., & Lee, S. W. (2025). Artificial intelligence for sustainable agriculture: A comprehensive review of AI-driven technologies in crop production. *Sustainability*, 17(5), 2281.

[19] Mohyuddin, G., Khan, M. A., Haseeb, A., Mahpara, S., Waseem, M., & Saleh, A. M. (2024). Evaluation of machine learning approaches for precision farming in smart agriculture systems: A comprehensive review. *IEEE Access*, 12, 60155–60184.

[20] Melki, M. N. E., Faqeih, K. Y., Alamri, S., AlAmri, A. R., Aldubehi, M. A., & Alamery, E. R. (2025). Integrating artificial intelligence, drones, robotics, and sensors for sustainable and climate-resilient agriculture: A critical review. *Sustainability*, 2025, 1–20.

[21] Pramanik, S., Roy, S., & Bose, R. (Eds.). (2024). *Data Driven Mathematical Modeling in Agriculture: Tools and Technologies*. CRC Press.

[22] Melesse, T. Y. (2025). Digital twin-based applications in crop monitoring. *Heliyon*, 11(2).

[23] Mundappat Ramachandran, M., Fahad Mon, B., Hayajneh, M., Abu Ali, N., & Badidi, E. (2025). Solar Agro Savior: Smart agricultural monitoring using drones and deep learning techniques. *Agriculture*, 15(15), 1656.

[24] Seethapathy, P., Kannan, M., & Manalil, S. (2026). AI-enabled early anomaly detection of crop diseases at real-field environments. In *Harnessing AI to Reshape the Future of Agriculture* (pp. 281–308). Cham: Springer Nature Switzerland.

[25] Hassan, M. (2025). *AI-Based Conditional Monitoring and Predictive Maintenance for Offshore Wind Farms*.

[26] Almannaei, K. J. (2024). *Predictive Maintenance on Drone Batteries Failure Using Machine Learning* (Master's thesis). Rochester Institute of Technology.

[27] Elufioye, O. A., Ike, C. U., Odeyemi, O., Usman, F. O., & Mhlongo, N. Z. (2024). AI-driven predictive analytics in agricultural supply chains: A review assessing the benefits and challenges of AI in forecasting demand and optimizing supply in agriculture. *Computer Science & IT Research Journal*, 5(2), 473–497.

[28] Singh, P., Singh, M. K., Singh, N., & Chakraverti, A. (2023). IoT and AI-based intelligent agriculture framework for crop prediction. *International Journal of Sensors, Wireless Communications and Control*, 13(3), 145–154.

[29] Elbasi, E., Alzoubi, Y. I., Topcu, A. E., & Nadeem, M. (2025). Green AI for smart agriculture: Energy-efficient predictive models for crop yield and resource management. *IEEE Access*, 13, 204924–204953.

[30] Sudha, S. P., & Loret, J. B. (2026). A review on machine learning-based precision agriculture techniques for crop farming monitoring with IoT. *Discover Environment*, 4(1), 10.