applied sciences

MDPI

Article

An MLLM-Assisted Web Crawler Approach for Web Application Fuzzing

Wantong Yang, Enze Wang, Zhiwen Gui, Yuan Zhou, Baosheng Wang and Wei Xie *

College of Computer Science and Technology, National University of Defense Technology, Changsha 410073, China; yangwt0805@nudt.edu.cn (W.Y.); wangenze18@nudt.edu.cn (E.W.); guizhiwen18@nudt.edu.cn (Z.G.); yzhou@nudt.edu.cn (Y.Z.); bswang@nudt.edu.cn (B.W.)
* Correspondence: xiewei@nudt.edu.cn

Abstract: Web application fuzzing faces significant challenges in achieving comprehensive test interface (attack surface) coverage, primarily due to the complexity of user interactions and dynamic website architectures. While web crawlers can automatically access and extract critical website information—including form fields and request parameters—which are essential for generating effective fuzzing test cases, current crawler technologies exhibit three primary limitations: (i) insufficient capabilities in analyzing page relationships and determining page states; (ii) lack of functionality-aware exploration capabilities, resulting in generated inputs with poor contextual relevance; (iii) generation of unstructured operation sequences that fail to execute effectively due to their incompatibility with state-based testing logic. To address these challenges, we propose CrawlMLLM, a framework using multi-modal large language models to simulate human web browsing. It includes three core components: page state mining, functionality analysis, and automatic operation generation. Evaluations show 163% code coverage improvements over SOTA work. When integrated with vulnerability audit tools, CrawlMLLM found 44 vulnerabilities in three vulnerable web applications versus 34 by the baseline. In six real-world applications, CrawlMLLM detected 20 vulnerabilities while the next best method found six.

Keywords: multi-modal large language models; web crawlers; web application fuzzing

check for updates

Academic Editor: Douglas O'Shaughnessy

Received: 26 December 2024
Revised: 10 January 2025
Accepted: 16 January 2025
Published: 19 January 2025

Citation: Yang, W.; Wang, E.; Gui, Z.; Zhou, Y.; Wang, B.; Xie, W. An MLLM-Assisted Web Crawler Approach for Web Application Fuzzing. *Appl. Sci.* **2025**, *15*, 962. https://doi.org/10.3390/app15020962

Copyright: © 2025 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

The rapid evolution of the Internet has established web applications as critical platforms for service delivery and information exchange. However, their growing complexity introduces substantial security challenges. Critical vulnerabilities—including SQL injection, Cross-Site Scripting (XSS), and Remote Code Execution (RCE)—continue to emerge with concerning frequency, presenting significant threats to user data integrity and system security. The complexity of modern web applications, characterized by diverse technology stacks, dynamic content generation, and sophisticated user interaction patterns, poses increasing challenges for vulnerability detection and remediation. Therefore, the effective identification and prevention of these security vulnerabilities has emerged as a fundamental research priority in the field of information security. This challenge necessitates innovative approaches to enhance web application security and safeguard critical digital assets.

The escalating scale and complexity of web applications necessitate sophisticated automated vulnerability discovery approaches. Current research has established three primary vulnerability detection methodologies: white-box, black-box, and gray-box scanning. Black-box scanning [1–4] simulates external attack vectors through input-output analysis; however, it lacks visibility into internal system architecture, potentially overlooking critical

vulnerabilities. Although white-box scanning [5–7] enables comprehensive vulnerability analysis through source code access, it faces significant limitations due to challenges in code complexity and scalability. Gray-box scanning [4,8–10] emerges as an optimal solution, integrating external testing methodologies with internal system knowledge acquired through crawler interaction with operational web applications. Web crawlers serve as fundamental components in both black-box and gray-box scanners, systematically exploring web pages to identify URLs, HTML form fields, and input parameters, thereby mapping the application's complete attack surface [11]. Insufficient crawler coverage of this attack surface may result in undetected vulnerabilities, substantially elevating security risks in production environments.

Prior research has introduced various crawler methodologies to enhance web application attack surface coverage. However, traditional web crawlers [2–4] exhibit significant limitations in state mining, primarily due to their reliance on rudimentary navigation strategies and page similarity algorithms. By neglecting page states and dependencies, these crawlers generate inefficient, redundant requests. In large-scale applications, state mining complexity increases substantially, as web application states evolve dynamically through user interactions, background processes, and external events. These dynamic transitions, combined with implicit dependencies, present significant challenges in state tracking and analysis. Furthermore, traditional crawler tools struggle to handle sophisticated interaction elements in modern web applications—including form submissions, modal dialogs, and multi-level navigation. Their limited adaptability to structural changes and insufficient semantic understanding frequently result in incomplete test coverage, potentially overlooking critical security vulnerabilities in complex web applications.

We introduce CrawlMLLMs, a novel web application fuzzing approach that harnesses Multi-modal Large Language Models (MLLMs) to enhance crawler effectiveness. The framework comprises three core components: page state mining, functionality analysis, and automatic operation generation. CrawlMLLM leverages MLLMs to comprehensively extract webpage states and generate robust test code, addressing the inherent coverage limitations of traditional crawlers. The system initiates page navigation through program directory structure analysis and target URL construction. Subsequently, MLLMs process page screenshots and HTML content to identify inter-page dependencies and optimize URL access sequences. CrawlMLLM's sophisticated user interaction simulation capabilities enable precise operation prediction and behavior pattern analysis; through comprehensive multi-modal analysis (encompassing images, text, and code), it thoroughly understands component functionality and interactions, facilitating accurate prediction of rule-compliant user inputs. We developed specialized prompt templates for generating test code for user interaction simulation, implementing state evaluation mechanisms to verify code effectiveness. The system employs context few-shot learning to enhance task-specific performance, utilizing carefully curated examples and contextual information to improve output accuracy and maintain format consistency.

To evaluate CrawlMLLM's effectiveness compared to existing web application security scanning technologies, we conducted a rigorous empirical assessment of code coverage and vulnerability detection capabilities. Our experimental evaluation encompassed nine distinct web applications, demonstrating code coverage improvements ranging from 7% to 363% (averaging 163%) compared to current scanning solutions (black widow, burp, and Rnd BFS and URL path algorithms). In the experiment on vulnerability detection capability improvement, we examined nine target programs comprising two distinct application sets: six applications widely referenced in academic literature, and three vulnerable web applications specifically selected to assess the impact of crawler improvements on scanner vulnerability detection capabilities. CrawlMLLM identified 44 unique security vulnera-

bilities across three vulnerable web application environments, demonstrating superior performance and significantly outperforming other crawler-audit module combinations, which detected a maximum of 34 vulnerabilities. Furthermore, in real-world software evaluation, CrawlMLLM discovered 20 vulnerabilities across six widely deployed applications, while alternative crawler-audit module combinations identified at most six vulnerabilities. Through systematic ablation studies, we validated the effectiveness of each core component.

In summary, our research presents three primary contributions:

- New Framework: We developed a novel crawler framework guided by MLLM that simulates authentic user browsing patterns, substantially improving coverage in web crawler-based fuzzing.
- Effective Strategy: We implement three strategic approaches for integrating MLLMs into fuzzing-oriented web crawlers, addressing three critical challenges: ineffective page relationship discovery, contextually inappropriate input generation, and invalid operation sequence creation.
- High Coverage: Comprehensive experimental evaluations demonstrate that CrawlMLLM achieves superior performance, showing an increase in code coverage of 7–363% (with an average of 163%) compared to leading security scanners (Black Widow, Burp, and Rnd BFS and URL path algorithms). When integrated with vulnerability audit tools, it consistently achieves optimal results across both vulnerable web applications and real-world application evaluations.

2. Background and Motivation

This chapter establishes the foundational knowledge for understanding our work through three key concepts: crawler-based web application fuzzing, the evolution of Multi-modal Large Language Models, and their synergistic integration for intelligent crawler guidance.

2.1. Crawler-Based Web Application Fuzzing

Web crawlers serve as fundamental components in modern web application security testing, particularly during fuzzing operations. Their significance is demonstrated through four essential capabilities: (1) automated test target discovery, (2) systematic test case generation, (3) dynamic scope determination, and (4) comprehensive vulnerability analysis support. These automated systems methodically traverse web applications, enumerating all accessible resources—including static pages, dynamic content, interactive forms, API endpoints, and auxiliary elements—establishing a comprehensive testing interface for fuzzing operations. Through sophisticated path exploration algorithms, crawlers generate context-aware test cases that cover diverse testing scenarios, ensuring thorough coverage of input vectors and user interaction patterns. During operation, crawlers maintain detailed execution logs and test results, providing security analysts with critical data for identifying potential vulnerabilities and abnormal behaviors. These comprehensive logs serve as invaluable resources for analyzing fuzzing results and diagnosing complex security issues.

In the cybersecurity domain, web crawlers serve as critical tools for comprehensive state discovery and security assessment. These automated crawling systems initiate from designated seed websites and employ iterative browsing techniques to explore various pages and resources within Web applications. Crawler operations primarily rely on two distinct strategies: navigation algorithms and page similarity algorithms [12]. Within navigation algorithms, random algorithms and Breadth-First Search (BFS) represent common approaches, with random BFS emerging as an enhanced hybrid algorithm that improves upon traditional BFS through the incorporation of randomization strategies. This methodology combines systematic exploration with adaptive selection strategies, enabling crawlers

to dynamically adapt to variations in website architectures and content loading mechanisms. Empirical evidence demonstrates that random BFS algorithms, in particular, exhibit superior performance in terms of average coverage and efficiency. For content deduplication purposes, page similarity algorithms primarily utilize URL matching and DOM-based analysis techniques. URL matching has demonstrated marked advantages in large-scale data deduplication tasks through its superior processing speed, algorithmic simplicity, and scalability, consistently outperforming DOM-based methods across the majority of performance metrics.

While web crawlers demonstrate significant potential in vulnerability scanning applications, they face several fundamental technical challenges. These challenges primarily arise from the inherent complexity of modern Web applications, which constrains the crawlers' capability to achieve comprehensive path and state coverage, particularly when encountering sophisticated authentication mechanisms and access control policies. Furthermore, although current page similarity algorithms effectively address data redundancy issues, they face substantial challenges in optimizing the trade-off between deduplication efficiency and test coverage during intensive large-scale operations. To overcome these limitations, current research efforts focus on enhancing crawler systems through the strategic integration of artificial intelligence techniques, with the objective of achieving higher precision and efficiency in vulnerability detection.

2.2. Evolution of Multi-Modal Large Language Models

The evolution of Multi-modal Large Language Models [13] represents a significant advancement in artificial intelligence, characterized by increasingly sophisticated approaches to modality integration. Early research primarily focused on fundamental neural network architectures that enabled the basic fusion of textual and visual representations for classification and retrieval tasks. The advent of deep learning techniques catalyzed substantial progress through the implementation of Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), enabling more sophisticated cross-modal fusion capabilities. Landmark architectures, particularly VGG and ResNet, established crucial benchmarks for image processing and feature extraction in multimodal applications. The introduction of Transformer architectures (exemplified by BERT and GPT) has led to revolutionary advances in the field, where their self-attention mechanism has fundamentally transformed text processing capabilities and subsequently extended to multi-modal applications.

The introduction of CLIP by OpenAI in 2021 marked a pivotal milestone, demonstrating unprecedented efficiency in cross-modal retrieval through sophisticated joint text-image training methodologies. Concurrent advances in generative modeling, particularly exemplified by OpenAI's DALL-E, showcased remarkable capabilities in text-guided image generation. Recent developments, as demonstrated by GPT-4 post-2023, have extended the paradigm to encompass multiple modalities—including text, images, and video—during the pre-training phase, substantially expanding applications across healthcare, finance, and entertainment sectors. This multi-modal integration has significantly enhanced model comprehension and cross-domain applicability, enabling more sophisticated solutions to complex real-world challenges.

MLLMs demonstrate significant advantages through the integration of diverse data modalities (text, images, audio, etc.), exhibiting extensive potential and technical superiority across multiple dimensions. Primarily, these models excel in the comprehensive processing of heterogeneous information types, enabling sophisticated understanding and generation capabilities for complex scenarios. Furthermore, cross-modal learning facilitates synergistic effects in multi-task execution, enhancing overall performance metrics. This capability is particularly evident in multi-task learning and transfer learning contexts, where

shared modal representations strengthen inter-task correlations. Additionally, MLLMs support naturalistic user interaction paradigms, enabling system engagement through various input modalities (textual, visual, or auditory), thereby enhancing the interaction experience's intuitiveness and accessibility. Moreover, these models demonstrate exceptional proficiency in addressing data scarcity challenges through effective multimodal information complementarity, thus enhancing model robustness and adaptability. In conclusion, MLLMs provide comprehensive technical foundations for practical applications through their sophisticated information fusion capabilities, efficient task coordination mechanisms, flexible interaction methodologies, and robust data handling capabilities.

2.3. Multi-Modal Large Model Guidance for Web Crawlers

Despite significant advancements in web crawler technologies, fundamental challenges persist in achieving optimal performance. State-of-the-art random Breadth-First Search (BFS) approaches exhibit several critical limitations, including non-uniform data collection patterns, excessive content redundancy, inadequate webpage relevance assessment, and incomplete coverage resulting from missed critical pages. Traditional URL-based similarity detection mechanisms exhibit significant limitations, particularly in identifying duplicate content across dynamically generated pages and pages that share similar content structures despite having different query parameters. These limitations become particularly pronounced when handling dynamic content generation or complex URL patterns.

To overcome the inherent constraints of conventional web crawlers, we propose a novel framework that leverages MLLMs to enhance web application security testing. Our framework implements sophisticated multi-modal analysis capabilities that emulate human cognitive processes during functional and security assessments. By processing multiple concurrent data streams through MLLM—including visual elements, textual content, structural layouts, and interactive components—the framework achieves a comprehensive understanding of webpage states and their relationships. In contrast to traditional approaches, our framework transcends static analysis by incorporating real-time processing and adaptive learning mechanisms to handle dynamic web elements, such as asynchronously loaded content and sophisticated UI interactions.

MLLMs enhance state awareness and interaction capabilities in web application testing. It demonstrates advanced proficiency in form detection and completion, automated test script generation, and context-aware test strategy adaptation. The integration of diverse data modalities—visual, textual, and structural—enables superior monitoring and decision-making capabilities.

The framework's intelligent analysis capabilities facilitate dynamic strategy adaptation for complex web applications, ensuring comprehensive vulnerability detection. By effectively addressing the fundamental limitations of traditional approaches, we hope this framework establishes a new paradigm in automated security testing, offering more robust solutions to address the security challenges faced by contemporary web applications.

3. Challenges

Effective web application fuzzing crawlers encounter three fundamental challenges.

3.1. Difficulty in Complex Page Relationship Discovery

Web crawler technology encounters significant challenges in decoding the complex and implicit logical relationships within modern web applications. These interactions are inherently dynamic and context-dependent, encompassing intricate interplay between user operations, background processes, and external events. While humans can intuitively

comprehend these relationships through experience, computational systems continue to face substantial challenges in understanding and modeling such complex dependencies.

Modern web crawlers predominantly utilize depth-first or breadth-first search strategies, which fundamentally lack the capability to account for dynamic page states and historical interaction patterns. These approaches generate substantial redundant or invalid requests, particularly in large-scale, multi-component web environments characterized by nuanced, context-sensitive state transitions. The inherent static nature of these strategies renders them inadequate for the evolving landscape of modern web applications.

Web application state management constitutes a complex, inherently dynamic ecosystem where states undergo continuous transitions through diverse triggers (user inputs, background processes, and external stimuli). Critical computational challenges emerge from implicit cross-segment dependencies that frequently remain undocumented, obscuring the comprehensive understanding of state transition mechanisms and relationships.

3.2. Difficulty in Context-Aware Input Generation

Web crawler technology encounters significant challenges in generating contextually adaptive inputs. Traditional rule-based approaches consistently fail to respond dynamically to evolving page states, resulting in inefficient resource utilization and compromised testing effectiveness. The fundamental challenge lies in the inability to systematically generate inputs that correspond to the complex, multi-dimensional context of modern web applications.

In the domain of fuzzing testing, the generation of random inputs conforming to specific requirements serves as a crucial strategy for vulnerability discovery. Traditional input generation approaches rely primarily on predefined parameter specifications, yet these methods prove inadequate against the complexity and diversity of modern web application design. The inherent computational complexity of web application logic exceeds current input generation capabilities, presenting challenges beyond even experienced human operators' expertise.

Web application interactions exhibit significant computational complexity through intricate, interconnected input mechanisms. Web interface inputs require precise contextual understanding. Consider flight booking systems, where departure location, destination, and date selection fields maintain complex business logic relationships. Similarly, e-commerce filtering systems generate extensive state spaces through filter combinations. Traditional web crawlers demonstrate fundamental limitations in their lack of semantic comprehension and inability to emulate sophisticated human–computer interaction patterns, resulting in incomplete exploration of potential input scenarios.

3.3. Difficulty in Generating Valid Operation Sequences

Traditional web crawlers typically operate on rudimentary interaction strategies, including indiscriminate clicking, scrolling, and form completion. However, these approaches lack a comprehensive understanding of page content and operational logic. These automated tools execute operations without semantic comprehension, generating redundant test sequences that consume substantial computational and communication resources while failing to effectively validate security properties. The majority of generated operations fail to enhance coverage metrics for vulnerability identification, particularly in complex applications where comprehension of user interaction patterns is critical.

Modern web applications encompass sophisticated client–server interactions where certain security vulnerabilities, particularly logic flaws, are only discoverable through specific operation sequences. Traditional fuzzing approaches generate random operations without considering interaction dependencies or state transitions, significantly limiting their

effectiveness in testing multi-step processes and state-dependent interactions. The dynamic nature of contemporary web applications, characterized by asynchronous content loading and sophisticated user interactions, poses additional challenges to traditional crawlers that lack effective management of element state dependencies and complex operations, such as drag-and-drop functionality.

4. Design

To address the aforementioned challenges, we present CrawlMLLM, a framework that leverages MLLMs to enhance web application fuzzing effectiveness through sophisticated user behavior simulation-assisted web crawling. CrawlMLLM employs three distinct components to address the challenges outlined in Section 3: page state mining (top portion of Figure 1), page functionality analysis (middle portion of Figure 1), and automated operation generation (bottom portion of Figure 1).

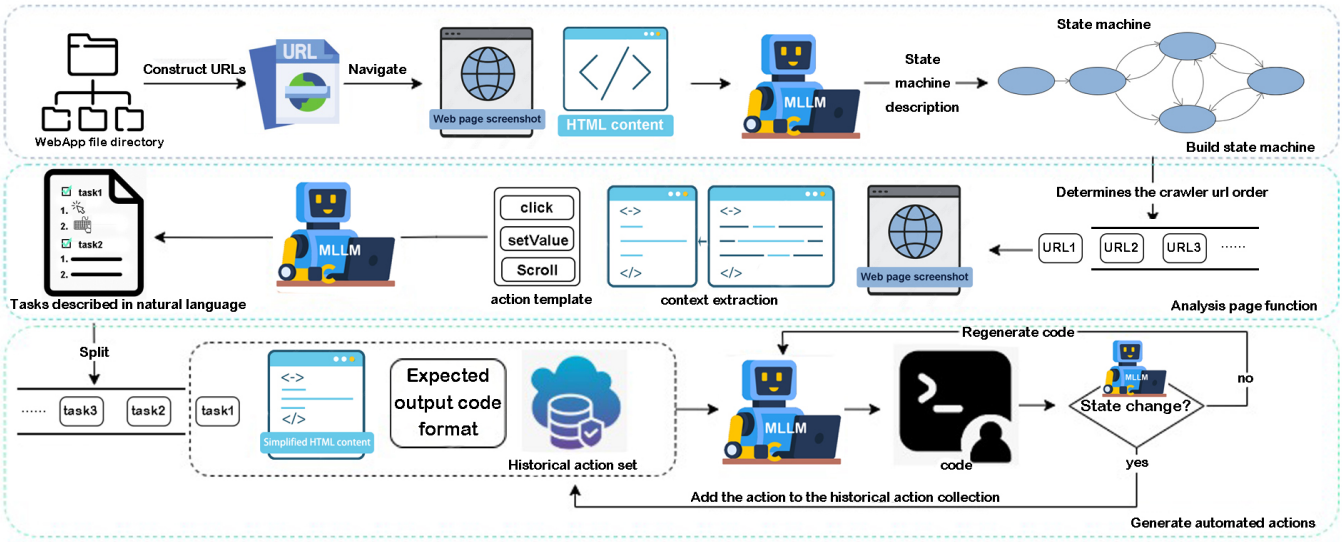

Figure 1. Workflow of CrawlMLLM. The framework comprises three main components. The page state mining component (**top**) generates state machine descriptions that guide the crawler's URL navigation sequence. The page functionality analysis component (**middle**) performs both macro-level analysis of page business logic and micro-level widget identification to determine achievable tasks and their required operations. The automated operation generation component (**bottom**) sequentially generates and validates the code necessary for task completion.

4.1. Mining Page States

Effective web application security testing necessitates a comprehensive understanding of page states and their interdependencies. Our methodology leverages MLLMs to construct this information through parallel analysis of visual and code-based representations, integrating computer vision capabilities with traditional code analysis techniques to transcend the limitations of conventional static analysis approaches.

The system initiates page navigation through program directory structure analysis and target URL construction. Subsequently, MLLM analyzes page screenshots and HTML content to identify inter-page dependencies and optimize URL traversal sequences. We implemented a systematic data collection and sanitization pipeline using Puppeteer. The system first captures both the HTML structure and visual representation of web pages. The data sanitization phase encompasses multiple steps: (1) elimination of dynamically generated identifiers and session-specific data that lack semantic value; (2) HTML structure normalization, including handling malformed tags and nested elements; (3) detection and removal of non-essential elements such as advertisements and tracking scripts; (4) stan-

dardization of encoding schemes and character sets. This preprocessing pipeline ensures robust subsequent analysis. The dual-modal analysis framework then provides complementary information streams: the sanitized HTML content reveals structural relationships, while the processed screenshots capture critical visual state information, including dynamic UI elements and styling properties that might be imperceptible through static code analysis alone.

The responses generated by MLLMs are inherently in natural language, demonstrating substantial flexibility. While a common approach is to manually transform MLLM outputs into desired formats, this method potentially compromises the automated nature of fuzzing. Therefore, it becomes crucial to develop mechanisms enabling MLLMs to generate responses directly in the expected format. Prompt engineering presents an effective solution to this challenge, as prompts significantly influence the structural output of MLLMs without requiring additional training or code implementation.

To effectively structure MLLM's output, we leveraged context few-shot learning as an efficient model fine-tuning approach. Figure 2 illustrates the model prompting framework for mining web application page states. The instruction component directs the MLLM to analyze inter-page relationships within a web application. The information section comprises three elements: page URLs, sanitized HTML content, and page screenshots. The latest state machine description component provides the current state machine representation of the model. The desired format section presents two exemplar state machine descriptions—a shopping application and a flight booking application—emphasizing that each page represents a distinct state with action-dependent transitions. The task instructs the model to analyze relationships between the three new pages and existing pages in the state machine, and then incorporate these relationships into the state machine representation.

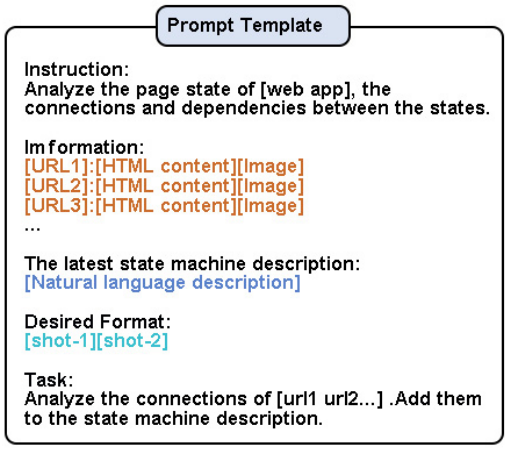

Figure 2. Mining Page Status Model Prompts. The framework consists of four components: (1) instructions for MLLM to analyze web application inter-page relationships, (2) input information comprising URLs, sanitized HTML, and screenshots, (3) current state machine description, and (4) example state machine descriptions and integration task specification.

This methodology enables the automatic generation of machine-processable state descriptions without manual intervention, preserving automation efficiency. The MLLM analyzes sanitized page data to identify valid interactive elements and state transitions. For each page, the model evaluates meaningful interactive functionality through analysis of sanitized HTML structure and visual characteristics. Pages lacking interactive elements are excluded from the state machine representation. For valid pages, the model assesses their relationships with existing states and updates the state machine accordingly. The implementation employs iterative refinement across multiple interaction rounds to systematically construct a comprehensive state machine representation. During each interaction

round, we provide two format exemplars and conduct focused dialogues with the MLLM. This iterative process continues through multiple rounds, with each iteration incorporating approximately three new states while optimizing existing descriptions.

Figure 3 illustrates the state machine diagram constructed from MLLM's natural language descriptions of page transitions in a shopping web application. Each node in the diagram represents a distinct page state. Note that variations in page content resulting from different user interactions (such as empty versus populated shopping cart states) are temporarily consolidated into a single state in this section. This simplification serves solely to establish the navigation URL sequence for the crawling phase. In Section 4.3, following the execution of MLLM-generated code, we perform a detailed analysis of these state variations and differentiate them into distinct states.

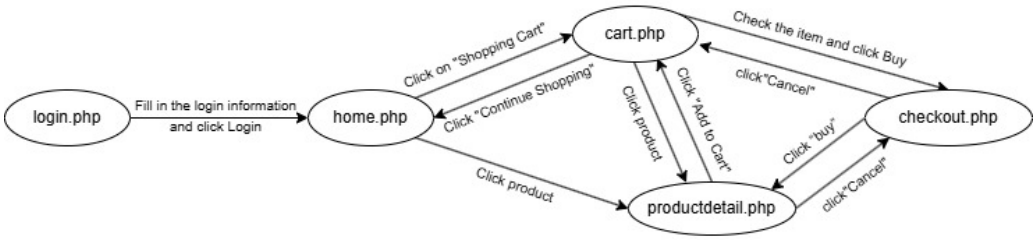

Figure 3. Example State Machine for a Shopping Web Application. In this representation, each page corresponds to a unique state, where nodes denote distinct states and edges represent the required operations for state transitions.

In our state machine representation, nodes denote distinct states while edges represent the operations necessary for state transitions. From this state machine, we can clearly observe the workflow of the web application: users initially access the login page (login.php), and upon password submission, are redirected to the homepage (home.php). Subsequently, users can navigate and perform various operations across the homepage (home.php), product detail page (productdetail.php), shopping cart (cart.php), and checkout page (checkout.php). This workflow exemplifies our methodology's application to e-commerce systems, capturing the state transitions across authentication, product browsing, shopping cart management, and checkout processes. Based on this state machine, we established the preliminary URL navigation sequence and traversal paths for our web crawler.

While MLLMs may exhibit stochastic behavior during response generation, such occurrences are relatively infrequent. To mitigate the impact of this randomness on our results, we implemented a multi-round dialogue strategy, wherein we collect multiple responses and extract the consistently occurring primary answers as our final output. This approach enables us to effectively leverage MLLMs for mining state information from web applications.

4.2. Page Functionality Analysis

Leveraging MLLM's superior capabilities in context comprehension and text generation, we investigated its potential application in web crawling processes for automated web application testing. MLLMs demonstrate robust capabilities in analyzing web page structures, precisely identifying various interactive elements and their associated attributes. Furthermore, MLLM exhibits exceptional prowess in interaction analysis, accurately interpreting both the functionality of these elements and their post-execution effects. In processing web pages, MLLMs effectively synthesize complementary information from both textual and visual modalities: textual data provide rich semantic and contextual information, while visual data capture detailed visual characteristics and spatial relationships. This multi-modal analytical approach significantly enhances the effectiveness of web testing procedures.

The middle section of Figure 1 illustrates the page functionality analysis pipeline. Building upon the state machine constructed in Section 4.1, the MLLM determines the crawler's URL navigation sequence. For each URL, the system captures page screenshots and implements a comprehensive data sanitization protocol. This includes heuristic-based advertisement filtering, and removing DOM nodes with common ad identifiers and iframe frameworks. Given MLLM prompt length constraints, we perform targeted HTML content sanitization. To enhance MLLM's analysis accuracy for element functionality and form input requirements, we developed detailed operation templates as prompt enrichments. This approach enables dual-level analysis: macro-level page functionality inference and micro-level element role identification. The MLLM then generates natural language descriptions specifying executable tasks and their operational sequences.

Figure 4 illustrates MLLM's natural language descriptions of webpage-executable tasks, comprising task overviews and operational procedures. We implemented several optimizations to enhance MLLM's comprehension: (1) designed specialized prompt templates with the system prompt "You are a browser automation assistant" to improve contextual understanding; (2) leveraged Puppeteer to capture webpage screenshots and HTML content, providing comprehensive multi-modal input; (3) implemented unique "element-id" attributes for precise element localization. To optimize MLLM's attention mechanism, we introduced a context extraction pipeline that includes the following: removing advertisement and tracker nodes through DOM tree analysis, eliminating invisible elements, extracting interactive elements, and preserving only functionally critical attributes (name, type, aria-label, etc.). This multi-tiered sanitization strategy substantially improved data quality.

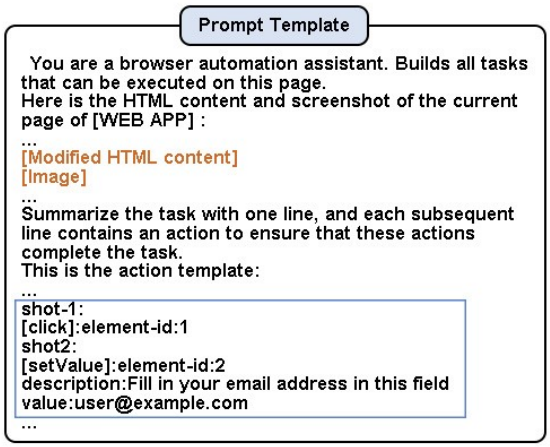

Figure 4. Example State Machine for a Shopping Web Application. Configure system prompt as "You are a browser automation assistant". Input consists of sanitized HTML, screenshots, and example operations (clicks and form submissions). The MLLM generates natural language task descriptions with overviews and step-by-step procedures.

To address the stochastic nature of large language model outputs, we implemented a structured methodology to standardize MLLM output. Specifically, we developed action templates to normalize task descriptions, requiring MLLMs to enumerate all executable tasks and their corresponding operational procedures on the current page. Within these templates, we differentiated between two fundamental operation types: click operations, which require only the target element's unique identifier, and form completion operations, which necessitate element identification, operation description, and input values. To enhance MLLM's comprehension of actual input content, we incorporated relevant contextual information into the templates. This structured prompting approach enables MLLMs to generate standardized and comprehensive page task descriptions.

MLLMs frequently encounter limitations in generating comprehensive responses within a single interaction. To address this constraint, we implemented an automated iterative dialogue mechanism where the system utilizes previously generated content as context to guide MLLMs in supplementing omitted information. To accurately determine response completeness, we implemented a completion signal protocol requiring MLLMs to append an end marker upon finishing content generation. The system monitors this signal to determine whether to continue the dialogue for additional content or conclude the current interaction and proceed to the next phase. This automated iterative supplementation mechanism ensures the comprehensiveness of the generated responses.

4.3. Automated Operation Generation

Given MLLM's inherent instability in code generation, we opted against direct test code generation and instead decomposed the process into two distinct phases, as described in Section 4.2 and the current section. In this section, we first transform MLLM's natural language task descriptions into automated test scripts, followed by defining and validating the success of web page state transitions. We provide a detailed discussion of both the transformation process and the state verification mechanism.

As illustrated in the bottom section of Figure 1, the automated operation generation phase begins with the decomposition of tasks generated in Section 4.2. This decomposition is motivated by two key considerations: MLLM's relatively low accuracy in directly generating multi-task test code, and the challenge of accurately evaluating individual task code generation through state transitions. For each decomposed task, we provide MLLM with multiple inputs: the natural language task description, HTML content processed through context extraction, the desired code format, and historical operation records. These inputs enable MLLM to generate the specific code required for task execution.

We leverage MLLM to validate state transitions through the analysis of page screenshots and content following code execution. A state represents the visual and functional characteristics of a web page during user interaction, encompassing specific content, layout, and interactive elements. State transitions comprise both full-page navigations and dynamic intra-page updates triggered by user actions (inputs, clicks, scrolls). For validation, MLLM analyzes pre- and post-execution page data (URLs, HTML content, screenshots), tasks, and executed code. State transitions are considered successful in two scenarios: (1) intra-page state changes, such as appointment confirmation appearing without page navigation, and (2) inter-page transitions, such as redirecting to an order confirmation page after purchase. When MLLM validates a successful state transition, the operation is logged in the history collection for future reference; otherwise, feedback is provided to MLLM for code regeneration.

Figure 5 presents our prompt template designed for multi-round dialogues with MLLMs. The template incorporates context-extracted page content, decomposed individual tasks with their operational procedures, and exemplar Puppeteer interaction code snippets (encompassing both click and form completion operations). This context-based few-shot learning approach enhances both MLLM task comprehension capabilities and ensures standardized code generation, facilitating the automated execution of the entire process.

In web application fuzzing, adhering solely to conventional user operation patterns may inadequately identify potential vulnerabilities. To enhance test coverage, the system must execute non-conventional exploratory operations to detect security vulnerabilities that might remain unidentified under normal usage scenarios. Consequently, following the completion of our primary testing workflow, we introduce a limited set of randomized operations to augment crawler results, thereby expanding test scenarios and improving vulnerability detection efficacy.

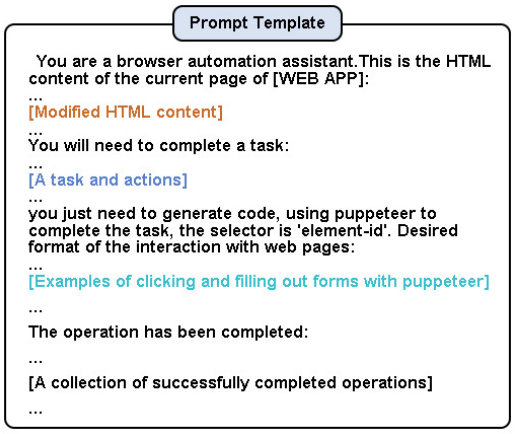

Figure 5. Prompt template for automated operation generation. Configure system prompt as "You are a browser automation assistant" with cleaned HTML input. Generate single-task code with corresponding operations, referencing Puppeteer examples for clicks and form submissions. Include completed operation history.

5. Evaluation

We evaluate CrawlMLLM's performance through the following three Research Questions (RQs):

- **RQ1. Code Coverage Assessment:** To what extent can CrawlMLLM achieve code coverage compared to state-of-the-art vulnerability scanners?
- **RQ2. Impact on Vulnerability Detection:** How does CrawlMLLM compare to state-of-the-art vulnerability scanners in terms of vulnerability discovery enhancement?
- **RQ3. Ablation Study:** What is the relative contribution of each system component to CrawlMLLM's overall performance?

5.1. Configuration Parameters

In this research, we improved upon open-source traditional crawler frameworks [14] by incorporating the three strategies mentioned above. In our study, we employ GPT-4o as our multi-modal large language model, with temperature parameters adjusted according to specific task phases. During page state mining and automated operation generation (Sections 4.1 and 4.3), we utilize a lower temperature value of 0.5 to ensure output stability and consistency. For page function analysis (Section 4.2), we set the temperature to 1.0 to enhance content creativity and diversity, enabling more comprehensive functional testing. To maintain system efficiency, we implemented two operational constraints: the system automatically proceeds to the next task in the queue after three consecutive failures to trigger expected state transitions and random crawler execution time is capped at one minute per URL.

This study evaluates a diverse set of nine PHP-based web applications (detailed in Table 1) that are widely adopted as benchmarks in web application fuzzing research. The proposed approach spans diverse application domains, demonstrating its broad applicability and generalizability, encompassing content management systems, e-commerce platforms, electronic health record systems, identity management systems, scheduling systems, forum platforms, and security training environments. These applications were chosen not only for their representation of primary web application categories but also for their established vulnerability profiles, making them ideal candidates for assessing fuzzing tool effectiveness. Table 1 presents comprehensive metadata for each application, including version information, GitHub popularity metrics (measured by star count), codebase size (LOC), and their adoption in prior research efforts.

Table 1. Web Applications Used in Evaluation.

Application	Ver.	GitHub Stars	Lines of Code	Prior Research
OpenEMR	5.0.1.7	1.6 k	9443	[4,15,16]
Wordpress	5.7.1	15 k	253,183	[1,3,4,17–19]
osCommerce	2.3.4.1	272	44,355	[1,4,20–23]
phpBB	3.3.3	1.4 k	318,104	[1,3,4,17,18]
DVWA	2.3	10 k	5826	[24]
XVWA	-	1.7 k	14,717	[24]
bWAPP	-	n/a	31,794	[24]
User Login Management System	2.1	n/a	1490	[4]
Doctor Appointment Booking System	1.0	n/a	3981	[4]

5.2. Code Coverage Evaluation

In evaluating code coverage, three representative tools were selected as baselines for comparison: First, Burp, the most advanced commercial black-box vulnerability scanner for Web applications. Its crawler module supports dynamic pages, form submissions, and authentication, with flexible configuration options. Second, Black Widow, an open-source black-box Web crawler tool that builds on traditional crawling capabilities by incorporating navigation modeling, traversal, and state-dependent tracking features. The third baseline, Rnd BFS and URL path was chosen based on the research by Stafeev and Pellegrino [12]. This study showed that combining random breadth-first search with a page similarity algorithm based on URL paths (including domain names, paths, and query strings) achieves optimal performance across various scenarios.

To assess the performance of CrawlMLLM, we first set up a Docker container environment for each Web application, providing CrawlMLLM with the complete directory structure and login credentials. Coverage data were collected using the Xdebug extension for PHP. For nine PHP applications, we compared CrawlMLLM against Burp, Black Widow, and Rnd BFS and URL path, with a consistent test time limit of 6 h. Table 2 presents the detailed code coverage comparison results: $A \setminus B$ represents the number of code lines covered exclusively by CrawlMLLM, $A \cap B$ represents the number of code lines covered by both CrawlMLLM and the comparison tools, and $B \setminus A$ represents the number of code lines covered exclusively by the comparison tools.

Table 2. PHP Code Line Coverage Results. This figure presents the PHP code line coverage comparison results between CrawlMLLM and three baseline approaches: Burp, Black Widow, and Rnd BFS and URL path. Each scanner is compared against CrawlMLLM. The $A \setminus B$ column shows the unique lines discovered by CrawlMLLM. The $A \cap B$ shows the lines found by CrawlMLLM and the other scanner. The $B \setminus A$ column shows the unique lines found by the other tool. If CrawlMLLM has the most unique lines, the value is green. If the other tool has the most unique lines then the value is in orange.

Apllication	Burp			Black Widow			Rnd BFS and URL Path		
	$A \setminus B$	$A \cap B$	$B \setminus A$	$A \setminus B$	$A \cap B$	$B \setminus A$	$A \setminus B$	$A \cap B$	$B \setminus A$
OpenEMR	31,416	12,261	411	28,254	15,423	1622	23,638	20,039	9959
Wordpress	50,337	14,926	136	50,231	15,032	768	20,737	44,526	7051
Login Mgmt.	55	121	5	46	130	65	5	171	12
phpBB	16,589	14,159	777	12,818	17,930	1829	12,172	18,576	2115
osCommerce	7637	3958	139	5468	6127	417	4323	7272	401
Doctor Appt. Sys.	811	362	0	841	332	47	317	856	124
DVWA	1261	765	2	1333	693	8	447	1579	57
XVWA	732	1425	0	622	1536	27	279	1878	121
bWAPP	2924	801	3	2428	1297	14	1672	2053	274

Results demonstrate that CrawlMLLM consistently achieves superior code coverage compared to existing solutions: showing improvements of 51–363% over Burp, 20–209% over Black Widow, and 7–60% over Rnd BFS and URL path across eight applications. The sole exception is Login Mgmt, where Rnd BFS and URL path achieved marginally higher coverage through random operations, attributed to its minimalist interface (comprising only login and registration pages) and absence of format restrictions on form inputs.

5.3. Evaluation of Enhanced Vulnerability Detection Capabilities

To evaluate CrawlMLLM's effectiveness in vulnerability discovery, we conducted experiments using two distinct sets of Web application datasets: the first set comprised three Vulnerable Web Applications (DVWA [25], XVWA [26], and bWAPP [27]), while the second set included seven diverse widely deployed open-source Web applications. For vulnerability assessment, we employed both the commercial scanning tool Burp and the open-source scanner ZAP's audit capabilities. The crawler component utilized three different approaches: Burp's built-in crawler, the Rnd BFS and URL path algorithm, and CrawlMLLM.

Figure 6 presents a comparative analysis of vulnerability detection results across different tool combinations in the Vulnerable Web Application environment. Among all six combinations tested, CrawlMLLM+Burp demonstrated superior detection capabilities across all three testing environments, identifying 44 vulnerabilities, 10 more than the next best combination. Notably, in XVWA, which features the least complex structure and lowest page complexity, all combinations exhibited similar performance. However, as application complexity increased progressively from XVWA through DVWA to bWAPP, CrawlMLLM's contribution to vulnerability discovery became increasingly pronounced, highlighting its robust adaptability to complex Web applications and its sophisticated comprehension of intricate page structures and functionalities.

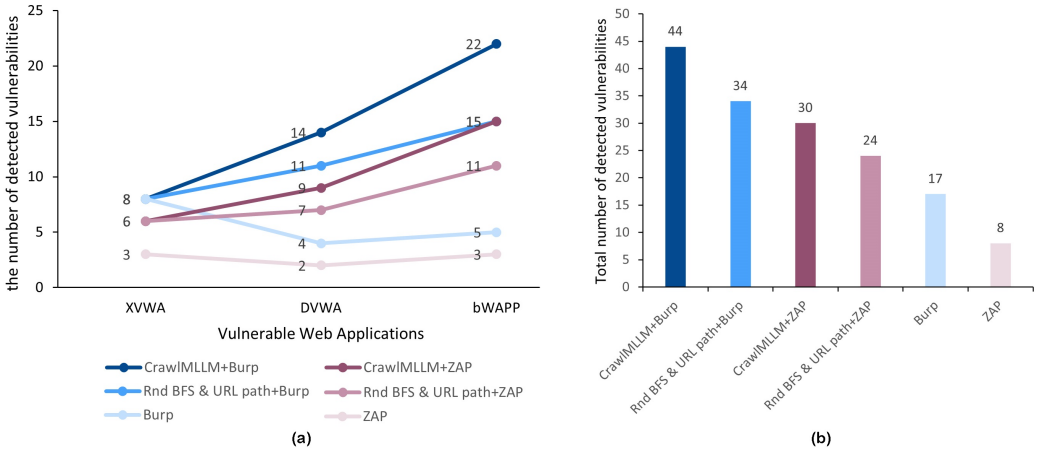

Figure 6. Comparative Analysis of Vulnerability Detection Results in Vulnerable Web Applications. (**a**) The x-axis depicts three vulnerable web applications, representing the number of vulnerabilities identified by six distinct configurations across each application. (**b**) The x-axis represents six configurations, illustrating the aggregate number of vulnerabilities discovered by each configuration across all three vulnerable web applications.

Table 3 presents the vulnerability detection results across three scanning configurations tested on six real-world web applications, with the numbers in parentheses indicating vulnerabilities uniquely discovered by each configuration. The results demonstrate that the CrawlMLLM+Burp combination identified a total of 20 vulnerabilities, with 14 being exclusive to this configuration. Consider the OpenEMR system case study: of the 28 documented vulnerabilities in the system, our approach successfully detected 13. The remaining 15 undetected vulnerabilities can be attributed to two primary limitations: (1) crawler coverage

constraints, accounting for six undetected vulnerabilities due to insufficient functional exploration within the allocated timeframe (partially addressable through extended crawl durations, though some cases are constrained by CrawlMLLM's interaction history capacity limits); (2) tool-specific limitations, where nine vulnerabilities remained undetected by Burp despite successful request data acquisition, due to the tool's inherent detection capabilities.

Table 3. Comparative Analysis of Vulnerability Detection Results in Production Web Applications. We analyzed the vulnerability detection performance of three scanning configurations across six production web applications, with parenthetical values representing the number of vulnerabilities uniquely identified by each configuration.

Application	Burp	Rnd BFS&URL Path+Burp	CrawlMLLM +Burp
OpenEMR	0(0)	3(0)	13(10)
Wordpress	0(0)	0(0)	0(0)
Login Mgmt.	1(0)	1(0)	1(0)
phpBB	0(0)	0(0)	0(0)
osCommerce	0(0)	0(0)	4(4)
Doctor Appt. Sys.	2(0)	2(0)	2(0)
	3(0)	6(0)	20(14)

5.4. Ablation Studies

The dialogue between CrawlMLLM and MLLMs implements three strategic approaches to address key challenges in the web crawler phase of fuzzing: state extraction, functionality analysis, and automated operation generation. To quantitatively assess each strategy's contribution to web application fuzzing effectiveness, we performed comprehensive ablation studies.

To determine the URL crawling sequence, we developed a state machine based on MLLM's analysis of inter-page relationships. We conducted a comparative evaluation between CrawlMLLM and its variant without page state mining capabilities (CrawlMLLM-NS) across nine previously mentioned Web applications over a 4-h period. Figure 7 illustrates the temporal progression of code coverage, with time plotted on the x-axis and coverage percentage on the y-axis. The experimental results demonstrate that CrawlMLLM's page state mining strategy significantly enhanced code coverage across six applications, attributable to their complex business logic and tightly coupled page interactions. By leveraging page state information to determine crawler navigation sequences, the system successfully accessed deeper functional layers. However, this strategy showed no measurable improvement in code coverage for three web applications, primarily due to their simplistic nature and minimal inter-page dependencies, where crawler navigation sequencing had a negligible impact on the outcome. These findings indicate that our approach effectively addresses a fundamental limitation of traditional crawlers: their excessive reliance on links and textual content while overlooking functional dependencies. Furthermore, compared to random crawling approaches, our method generates fewer redundant requests, thereby enhancing the overall efficiency of Web application fuzzing.

For the analysis of page functionality, we selected a test set of 100 pages from the Web applications enumerated in Table 1, comprising 50 simple pages and 50 complex pages (categorized based on functional complexity). We established ground truth through the manual analysis of the functional tasks within these pages and compared it against MLLM-generated results. As illustrated in Figure 8, CrawlMLLM achieved complete functional analysis and task identification for all 50 simple pages. For the complex pages, MLLM successfully provided comprehensive descriptions of executable tasks for 42 pages through iterative dialogue integration. The remaining eight pages proved challenging for complete

analysis due to multiple factors, including structural complexity, content diversity, dynamic modifications, and state transitions triggered by user interactions, which impeded the model's ability to fully capture all relevant information.

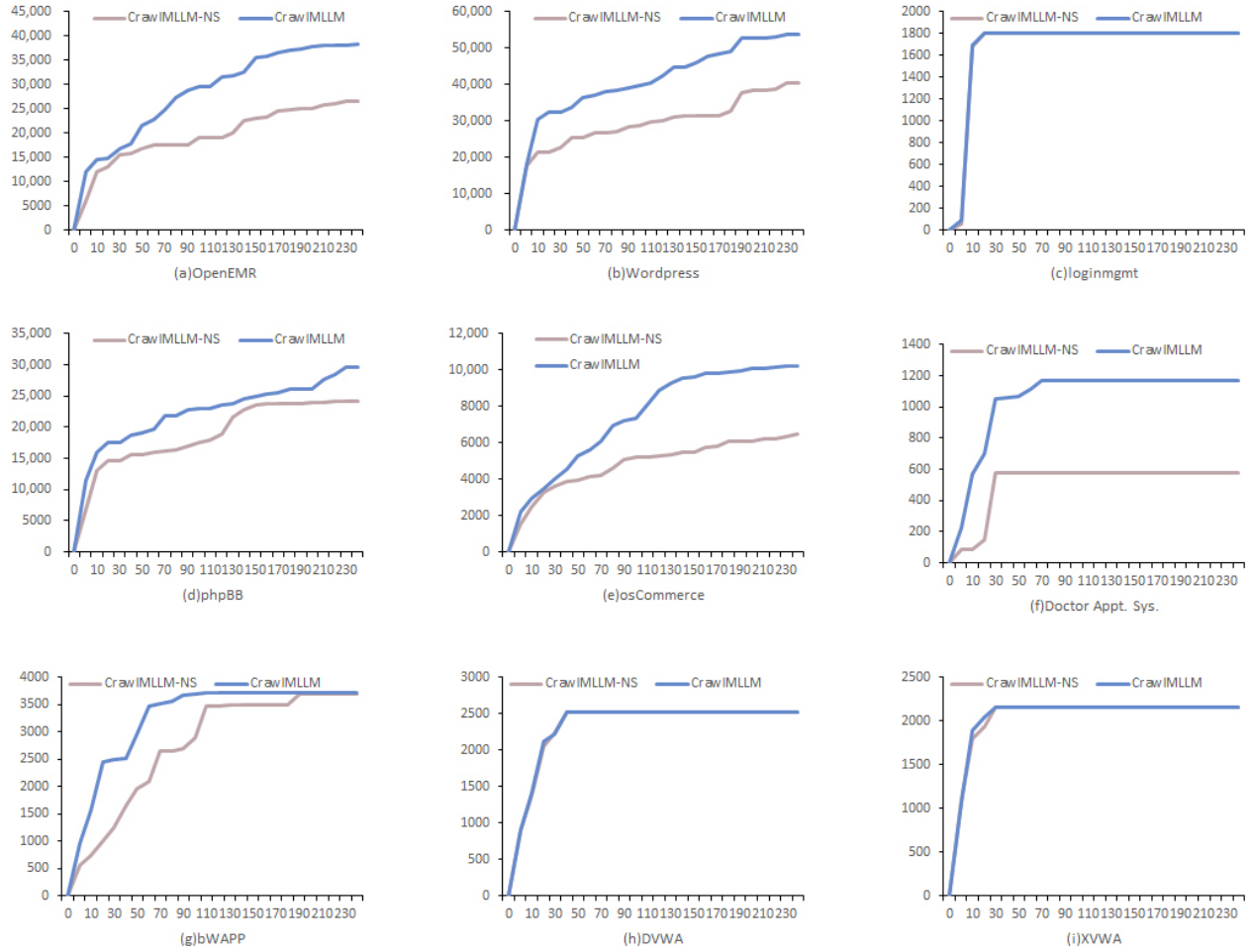

Figure 7. Code Coverage Enhancement Through Page State Mining. We evaluated CrawlMLLM against a variant without page state mining capabilities (denoted as CrawlMLLM-NS) across nine web applications, measuring code coverage progression over time. The absence of significant improvements in cases (**c**,**h**,**i**) can be attributed to the minimal inter-page dependencies in these simpler applications. However, notable improvements were observed across the remaining six complex applications(**a**,**b**,**d**–**g**).

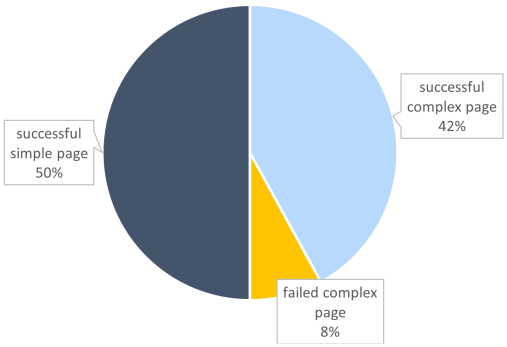

Figure 8. Evaluation of the Analytical Page Functional Modules. We selected a dataset of 100 web pages for evaluation, consisting of 50 simple pages and 50 complex pages. CrawlMLLM successfully performed comprehensive functional analysis and task extraction on all 50 simple pages. For the complex pages, the system achieved successful analysis on 42 pages, while eight pages resulted in incomplete extraction.

Following the completion of page state mining and functional analysis phases, we evaluated the success rate of automated operations generated by MLLMs. Our experimental evaluation encompassed three categories of tasks: simple tasks (completable within five operations), medium-complexity tasks (requiring 5–10 operations), and complex tasks (requiring more than 10 operations). We generated test code using the interaction process detailed in Section 4.3 and analyzed the success probability across these task categories. As illustrated in Figure 9 , the success rates were 100% for simple tasks, 87% for medium-complexity tasks, and 62% for complex tasks. The experimental results demonstrate that decomposing complex tasks into multiple simpler subtasks effectively enhances the accuracy of MLLM-generated test code. However, CrawlMLLM's generalization capabilities for certain complex tasks still require further improvement.



Figure 9. Success Rate of Operation Code Generation by MLLMs. Our experimental evaluation encompassed three categories of tasks: simple tasks (completable within five operations), medium-complexity tasks (requiring 5–10 operations), and complex tasks (requiring more than 10 operations).

6. Discussion

Advantages and Innovation. CrawlMLLM significantly enhances test request coverage through page state mining, functionality analysis, and automated operation generation during the crawling phase, providing high-quality test cases for Web application fuzzing. Evaluation results demonstrate that CrawlMLLM exhibits notable performance advantages compared to Burp, Black Widow, and Rnd BFS and URL path algorithms.

Adaptability. CrawlMLLM adapts to various web applications beyond the initial dataset, not just PHP-written web applications, but equally adapts to Java, PHP, and Node.js applications.

Intelligent Interaction Capabilities. CrawlMLLM is an intelligent crawler system capable of understanding webpage content and structure. The system not only generates appropriate access instructions for specific tasks but also simulates complex user interaction behaviors, including clicking and scrolling. Through this intelligent simulation of real user browsing patterns, CrawlMLLM's automated test scripts are more difficult for anti-crawling mechanisms to detect compared to traditional random crawlers.

Current Limitations. CrawlMLLM exhibits several constraints in webpage interaction scenarios. While demonstrating significant improvements in complex web applications, it shows a minimal advantage over traditional crawler methods when handling simple applications that require neither inter-page relationship analysis nor form content validation. The system faces challenges in effectively identifying and addressing vulnerabilities triggered by invisible parameters, primarily due to the MLLM's limited access to contextual information regarding hidden elements and dynamically generated content. The MLLM encounters difficulties in fully simulating and capturing the complete lifecycle of invisible parameters generated through user interactions. Furthermore, the model

demonstrates low accuracy in simulating time-sensitive scenarios involving parameters dynamically generated under specific temporal or conditional constraints. Additionally, CrawlMLLM presents limitations in state tracking and historical interaction data retention. Although computational overhead increases with application complexity, the framework maintains operational efficiency through its modular architecture and MLLM's capability to decompose complex pages into analyzable components. The increasing complexity of web applications presents challenges in generating comprehensive state machines for page state mining and complete task descriptions for functionality analysis.

Concerning computational resources and deployment. Our framework utilizes GPT-4o through API calls rather than requiring local MLLM deployment. This architectural choice significantly reduces computational resource requirements and simplifies deployment. The system can be effectively operated on standard development hardware, as the primary computational load is handled by the external API service.

Future Improvements. Despite CrawlMLLM's robust performance in our evaluation, several avenues for optimization remain. The primary enhancement involves strengthening state tracking capabilities through the introduction of state recovery techniques to achieve precise system state management and implementing systematic page decomposition for complex web applications. Furthermore, the system requires improved capabilities for accessing contextual information of hidden elements and dynamically generated content, coupled with enhanced reasoning capabilities for request parameters and their values. Another crucial optimization direction involves incorporating a memory mechanism for modeling historical interactions. In terms of vulnerability detection, scanning tools have inherent limitations. Although CrawlMLLM can generate high-coverage functional access requests for Web applications, actual vulnerability discovery still depends on the detection capabilities of scanning tools. Integrating LLM technology into the vulnerability detection phase presents an opportunity for further performance enhancement. Similarly, Ni, et al. [28] propose a StaResGRU-CNN architecture that incorporates biomedical domain knowledge through pre-training, significantly improving the performance of Chinese medical NLP tasks. We can train a dedicated MLLM to accomplish the related tasks mentioned in this paper.

7. Related Work

Traditional web crawlers leverage page similarity algorithms and navigation algorithms to explore and identify webpage states. Their navigation algorithms primarily consist of random algorithms [29–31], breadth-first search (BFS) [32,33], and depth-first search (DFS) [34,35]. While these methods are straightforward to implement, they exhibit serious limitations when handling modern Web applications: random algorithms struggle to systematically cover application states, and BFS and DFS are susceptible to state explosion problems when dealing with large-scale applications. Regarding page similarity assessment, URL matching [1,36] and DOM-based algorithms [37–40] represent the most widely adopted methods, but these approaches excessively rely on static features and fail to accurately identify dynamically generated content and complex state transitions. Traditional crawlers primarily employ static methods for webpage content extraction, focusing solely on HTML structure and links, lacking the capability to comprehend page semantics and interaction logic, which results in insufficient coverage and missed critical functional states when handling modern Web applications. In contrast, MLLM-assisted crawlers excel at understanding page UI structure, interaction logic, and state management, enabling them to effectively process dynamic content and complex Web applications, thereby achieving comprehensive application coverage and identifying potential security vulnerabilities.

Recent years have witnessed significant advancements in machine-learning approaches for application testing. In 2023, Hu, et al. [41] introduced a testing technique that integrates multiple machine learning methods to simulate tester verification processes by capturing interaction intentions aligned with human cognition. While this approach demonstrates exceptional performance in mobile application testing, it encounters three critical limitations: (1) the requirement for extensive labeled data to support model training, which elevates practical implementation costs; (2) the adoption of multimodal self-attention deep learning methods, involving complex model design and training processes, which compromises scalability; (3) the method's specialized design for mobile applications, with architectures and assumptions that do not address the unique characteristics of Web applications. Drawing inspiration from the research of Hu, et al. [41] and Liu, et al. [42], who introduced human interaction simulation in mobile GUI testing through functionality-aware decisions, this study employs MLLM methodology. This approach effectively reduces the computational overhead of multiple independent models through shared model weights and computing resources, while demonstrating superior performance in multi-task and multimodal data processing for simulating human interaction intentions in web applications.

In the domain of fuzzing, researchers have begun exploring the potential applications of LLM. ChatAFL [43] enhances AFLNet by leveraging LLM for grammar extraction and test case generation, though its scope remains primarily confined to protocol-level testing. FuzzGPT [44] pioneered the application of LLM to fuzzing deep learning libraries through the implicit learning of API constraints for test input generation, but their methodology proves challenging to extend to complex Web application testing scenarios. Fuzz4All [45] employs LLM as an input generation and mutation engine, introducing specialized fuzzing prompt techniques, yet fails to address the state exploration challenges in Web applications. CODAMOSA [46] utilizes Codex for generating example test cases during coverage plateaus but focuses solely on single-state test generation. While these works demonstrate LLM's potential in test generation, they do not address the challenges of intelligent navigation and state comprehension in Web application crawling. This research presents the first comprehensive intelligent crawler solution for Web application fuzzing by leveraging MLLM to understand page UI structure, interaction logic, and state management.

8. Conclusions

Fuzzing testing of Web applications presents unique challenges compared to traditional binary fuzzing. Primarily, Web application fuzzing heavily depends on crawler technology, where the quality of crawler requests directly determines testing effectiveness. Additionally, the stateful nature of Web applications introduces another layer of complexity to the testing process. To address these challenges, we introduce a crawler technology assisted by MLLMs. This approach generates high-quality test code through a comprehensive understanding of page structure, interaction logic, and state management, significantly enhancing both the coverage and efficiency of fuzzing testing.

Our experimental results demonstrate that the proposed CrawlMLLM effectively performs page state mining, functionality analysis, and automated operation instruction generation. When compared to state-of-the-art tools, CrawlMLLM achieves superior code coverage within equivalent time periods while substantially improving vulnerability discovery capabilities.

Author Contributions: Conceptualization, W.Y. and W.X.; methodology, W.Y.; software, W.Y.; validation, W.Y., W.X. and E.W.; formal analysis, W.Y.; investigation, W.Y.; resources, W.Y.; data curation, W.Y.; writing—original draft preparation, W.Y., Y.Z., W.X. and B.W.; writing—review and editing,

E.W.; visualization, W.Y.; supervision, W.Y.; project administration, Z.G. and Y.Z.; funding acquisition, W.X. and B.W. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The original contributions presented in this study are included in the article. Further inquiries can be directed to the corresponding author.

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

References

1. Eriksson, B.; Pellegrino, G.; Sabelfeld, A. Black Widow: Blackbox Data-driven Web Scanning. In Proceedings of the 2021 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 24–27 May 2021. [CrossRef]
2. Mesbah, A.; Bozdag, E.; van Deursen, A. Crawling AJAX by Inferring User Interface State Changes. In Proceedings of the 2008 Eighth International Conference on Web Engineering, Yorktown Heights, NY, USA, 14–18 July 2008. [CrossRef]
3. Pellegrino, G.; Tschürtz, C.; Bodden, E.; Rossow, C. jÄk: Using Dynamic Analysis to Crawl and Test Modern Web Applications. In Proceedings of the 18th International Symposium on Research in Attacks, Intrusions, and Defenses (RAID 2015), Kyoto, Japan, 2–4 November 2015; Volume 9404, pp. 295–316. [CrossRef]
4. Trickel, E.; Pagani, F.; Zhu, C.; Dresel, L.; Vigna, G.; Kruegel, C.; Wang, R.; Bao, T.; Shoshitaishvili, Y.; Doupé, A. Toss a Fault to Your Witcher: Applying Grey-box Coverage-Guided Mutational Fuzzing to Detect SQL and Command Injection Vulnerabilities. In Proceedings of the 2023 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 21–25 May 2023; pp. 2658–2675. [CrossRef]
5. Felmetsger, V.; Cavedon, L.; Kruegel, C.; Vigna, G. Toward automated detection of logic vulnerabilities in web applications. In Proceedings of the 19th USENIX Security Symposium, Washington, DC, USA, 11–13 August 2010.
6. Huang, Y.W.; Yu, F.; Hang, C.; Tsai, C.H.; Lee, D.T.; Kuo, S.Y. Securing web application code by static analysis and runtime protection. In Proceedings of the 13th International Conference on World Wide Web, New York, NY, USA, 17–20 May 2004; pp. 40–52. [CrossRef]
7. Jovanovic, N.; Kruegel, C.; Kirda, E. Static analysis for detecting taint-style vulnerabilities in web applications. *J. Comput. Secur.* **2010**, *18*, 861–907. [CrossRef]
8. Olsson, E.; Eriksson, B.; Doupé, A.; Sabelfeld, A. Spider-Scents: Grey-box Database-aware Web Scanning for Stored XSS. In Proceedings of the 33rd USENIX Security Symposium (USENIX Security 24), Philadelphia, PA, USA, 14–16 August 2024; pp. 6741–6758.
9. Hassanshahi, B.; Lee, H.; Krishnan, P. Gelato: Feedback-driven and Guided Security Analysis of Client-side Web Applications. In Proceedings of the 2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), Honolulu, HI, USA, 15–18 March 2022; pp. 618–629. [CrossRef]
10. Steinhauser, A.; Tůma, P. Database Traffic Interception for Graybox Detection of Stored and Context-sensitive XSS. *Digit. Threat.* **2020**, *1*, 1–23. [CrossRef]
11. Manès, V.J.; Han, H.; Han, C.; Cha, S.K.; Egele, M.; Schwartz, E.J.; Woo, M. The Art, Science, and Engineering of Fuzzing: A Survey. *IEEE Trans. Softw. Eng.* **2021**, *47*, 2312–2331. [CrossRef]
12. Stafeev, A.; Pellegrino, G. SoK: State of the Krawlers–Evaluating the Effectiveness of Crawling Algorithms for Web Security Measurements. In Proceedings of the 33rd USENIX Security Symposium (USENIX Security 24), Philadelphia, PA, USA, 14–16 August 2024; pp. 719–737.
13. Li, C.; Gan, Z.; Yang, Z.; Yang, J.; Li, L.; Wang, L.; Gao, J. Multimodal Foundation Models: From Specialists to General-Purpose Assistants. *Found. Trends® Comput. Graph. Vis.* **2024**, *16*, 1–214. [CrossRef]
14. BoB-WebFuzzing. Request-Crawler. 2024. Available online: https://github.com/BoB-WebFuzzing/Request-Crawler (accessed on 23 October 2024).
15. McGee, Z.; Acharya, S. Security Analysis of OpenEMR. In Proceedings of the 2019 IEEE International Conference on Bioinformatics and Biomedicine (BIBM), San Diego, CA, USA, 18–21 November 2019; pp. 2655–2660. [CrossRef]
16. Akowuah, F.; Lake, J.; Yuan, X.; Nuakoh, E.B.; Yu, H. Testing the security vulnerabilities of OpenEMR 4.1.1: A case study. *J. Comput. Sci. Coll.* **2015**, *30*, 26–35.
17. Doupé, A.; Cavedon, L.; Kruegel, C.; Vigna, G. Enemy of the State: A State-Aware Black-Box Web Vulnerability Scanner. In Proceedings of the 21st USENIX Security Symposium (USENIX Security 12), Bellevue, WA, USA, 8–10 August 2012; pp. 523–538.

18. Duchene, F.; Rawat, S.; Richier, J.L.; Groz, R. KameleonFuzz: Evolutionary fuzzing for black-box XSS detection. In Proceedings of the 4th ACM Conference on Data and Application Security and Privacy (CODASPY '14), Darmstadt, Germany, 18–20 July 2014; Association for Computing Machinery: New York, NY, USA, 2014; pp. 37–48. [CrossRef]
19. Bau, J.; Bursztein, E.; Gupta, D.; Mitchell, J. State of the Art: Automated Black-Box Web Application Vulnerability Testing. In Proceedings of the 2010 IEEE Symposium on Security and Privacy, Oakland, CA, USA, 16–19 May 2010; pp. 332–345. [CrossRef]
20. Gupta, S.; Gupta, B.B. PHP-sensor: A prototype method to discover workflow violation and XSS vulnerabilities in PHP web applications. In Proceedings of the 12th ACM International Conference on Computing Frontiers (CF '15), Ischia, Italy, 18–21 May 2015; Association for Computing Machinery: New York, NY, USA, 2015. [CrossRef]
21. Pellegrino, G.; Balzarotti, D. Toward Black-Box Detection of Logic Flaws in Web Applications. In Proceedings of the Network and Distributed System Security Symposium, San Diego, CA, USA, 23–26 February 2014.
22. Deepa, G.; Thilagam, P.S.; Praseed, A.; Pais, A.R. DetLogic: A black-box approach for detecting logic vulnerabilities in web applications. *J. Netw. Comput. Appl.* **2018**, *109*, 89–109. [CrossRef]
23. Li, X.; Xue, Y. BLOCK: A black-box approach for detection of state violation attacks towards web applications. In Proceedings of the 27th Annual Computer Security Applications Conference (ACSAC '11), Orlando, FL, USA, 5–9 December 2011; Association for Computing Machinery: New York, NY, USA, 2011; pp. 247–256. [CrossRef]
24. Güler, E.; Schumilo, S.; Schloegel, M.; Bars, N.; Görz, P.; Xu, X.; Kaygusuz, C.; Holz, T. Atropos: Effective Fuzzing of Web Applications for Server-Side Vulnerabilities. In Proceedings of the 33rd USENIX Security Symposium (USENIX Security 24), Philadelphia, PA, USA, 14–16 August 2024; pp. 4765–4782.
25. digininja. Damn Vulnerable Web Application (DVWA). 2024. Available online: https://github.com/digininja/DVWA (accessed on 23 October 2024).
26. s4n7h0. Xtreme Vulnerable Web Application (XVWA). 2020. Available online: https://github.com/s4n7h0/xvwa (accessed on 23 October 2024).
27. raesene. bWAPP (Buggy Web Application). 2021. Available online: https://github.com/raesene/bWAPP (accessed on 23 October 2024).
28. Ni, P.; Li, G.; Hung, P.C.; Chang, V. StaResGRU-CNN with CMedLMs: A stacked residual GRU-CNN with pre-trained biomedical language models for predictive intelligence. *Appl. Soft Comput.* **2021**, *113*, 107975. [CrossRef]
29. Hong, G.; Yang, Z.; Yang, S.; Zhang, L.; Nan, Y.; Zhang, Z.; Yang, M.; Zhang, Y.; Qian, Z.; Duan, H. How You Get Shot in the Back: A Systematical Study about Cryptojacking in the Real World. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS '18), Toronto, ON, Canada, 15–19 October 2018; Association for Computing Machinery: New York, NY, USA, 2018; pp. 1701–1713. [CrossRef]
30. Konoth, R.K.; Vineti, E.; Moonsamy, V.; Lindorfer, M.; Kruegel, C.; Bos, H.; Vigna, G. MineSweeper: An In-depth Look into Drive-by Cryptocurrency Mining and Its Defense. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS '18), Toronto, ON, Canada, 15–19 October 2018; Association for Computing Machinery: New York, NY, USA, 2018; pp. 1714–1730. [CrossRef]
31. Pan, X.; Cao, Y.; Chen, Y. I Do Not Know What You Visited Last Summer: Protecting users from stateful third-party web tracking with TrackingFree browser. In Proceedings of the Network and Distributed System Security Symposium, San Diego, CA, USA, 8–11 February 2015.
32. Drakonakis, K.; Ioannidis, S.; Polakis, J. The Cookie Hunter: Automated Black-box Auditing for Web Authentication and Authorization Flaws. In Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (CCS '20), Virtual Event, USA, 9–13 November 2020; Association for Computing Machinery: New York, NY, USA, 2020; pp. 1953–1970. [CrossRef]
33. Steffens, M.; Rossow, C.; Johns, M.; Stock, B. Don't Trust The Locals: Investigating the Prevalence of Persistent Client-Side Cross-Site Scripting in the Wild. In Proceedings of the 2019 Network and Distributed System Security Symposium, Bangalore, India, 4–7 January 2019. [CrossRef]
34. Djeric, V.; Goel, A. Securing script-based extensibility in web browsers. In Proceedings of the USENIX Security Symposium, Washington, DC, USA, 11–13 August 2010.
35. Jueckstock, J.; Snyder, P.; Sarker, S.; Kapravelos, A.; Livshits, B. Measuring the Privacy vs. Compatibility Trade-off in Preventing Third-Party Stateful Tracking. In Proceedings of the ACM Web Conference 2022, Lyon, France, 25–29 April 2022. [CrossRef]
36. Khodayari, S.; Pellegrino, G. JAW: Studying Client-side CSRF with Hybrid Property Graphs and Declarative Traversals. In Proceedings of the USENIX Security Symposium, Online, 11–13 August 2021.
37. Kim, I.L.; Wang, W.; Kwon, Y.; Zheng, Y.; Aafer, Y.; Meng, W.; Zhang, X. AdBudgetKiller. In Proceedings of the 2018 World Wide Web Conference on World Wide Web (WWW '18), Lyon, France, 23–27 April 2018; pp. 297–307. [CrossRef]
38. Subramani, K.; Melicher, W.; Starov, O.; Vadrevu, P.; Perdisci, R. PhishInPatterns: Measuring elicited user interactions at scale on phishing websites. In Proceedings of the 22nd ACM Internet Measurement Conference (IMC '22), Nice, France, 25–27 October 2022; Association for Computing Machinery: New York, NY, USA, 2022; pp. 589–604. [CrossRef]

39. Yang, R.; Wang, X.; Chi, C.; Wang, D.; He, J.; Pang, S.; Lau, W. Scalable Detection of Promotional Website Defacements in Black Hat SEO Campaigns. In Proceedings of the USENIX Security Symposium, Online, 11–13 August 2021.
40. Zeng, E.; Wei, M.; Gregersen, T.; Kohno, T.; Roesner, F. Polls, clickbait, and commemorative $2 bills: Problematic political advertising on news and media websites around the 2020 U.S. elections. In Proceedings of the 21st ACM Internet Measurement Conference (IMC '21), Virtual Event, 2–4 November 2021; Association for Computing Machinery: New York, NY, USA, 2021; pp. 507–525. [CrossRef]
41. Hu, Y.; Gu, J.; Hu, S.; Zhang, Y.; Tian, W.; Guo, S.; Chen, C.; Zhou, Y. Appaction: Automatic GUI Interaction for Mobile Apps via Holistic Widget Perception. In Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2023), San Francisco, CA, USA, 3–9 December 2023; Association for Computing Machinery: New York, NY, USA, 2023; pp. 1786–1797. [CrossRef]
42. Liu, Z.; Chen, C.; Wang, J.; Chen, M.; Wu, B.; Che, X.; Wang, D.; Wang, Q. Make LLM a Testing Expert: Bringing Human-like Interaction to Mobile GUI Testing via Functionality-aware Decisions. In Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE), Lisbon, Portugal, 14–20 April 2024.
43. Meng, R.; Mirchev, M.; Böhme, M.; Roychoudhury, A. Large Language Model guided Protocol Fuzzing. In Proceedings of the 31st Annual Network and Distributed System Security Symposium (NDSS), San Diego, CA, USA, 26 February–1 March 2024.
44. Deng, Y.; Xia, C.S.; Yang, C.; Zhang, S.D.; Yang, S.; Zhang, L. Large Language Models are Edge-Case Generators: Crafting Unusual Programs for Fuzzing Deep Learning Libraries. In Proceedings of the IEEE/ACM 46th International Conference on Software Engineering (ICSE '24), Lisbon, Portugal, 14–20 April 2024; Association for Computing Machinery: New York, NY, USA, 2024. [CrossRef]
45. Xia, C.S.; Paltenghi, M.; Tian, J.L.; Pradel, M.; Zhang, L. Fuzz4All: Universal Fuzzing with Large Language Models. In Proceedings of the 46th International Conference on Software Engineering (ICSE '24), Lisbon, Portugal, 14–20 April 2024.
46. Lemieux, C.; Inala, J.P.; Lahiri, S.K.; Sen, S. CodaMosa: Escaping Coverage Plateaus in Test Generation with Pre-trained Large Language Models. In Proceedings of the 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE), Melbourne, Australia, 14–20 May 2023; pp. 919–931. [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.