International Journal of Safety and Security Engineering
Vol. 16, No. 7, July, 2026, pp. 1459-1473
Journal homepage: http://iieta.org/journals/ijsse

Client-Side Real-Time Analysis of Fragmented Phishing and Cross-Site Scripting Attacks Using Large Language Models

Israa Ahmed Abbas*, Taha Mohammed Hasan

Department of Computer Science, College of Science, University of Diyala, Baquba 32001, Iraq

Corresponding Author Email: scicomphd232408@uodiyala.edu.iq

https://doi.org/10.18280/ijsse.160705

Received: 8 May 2026
Revised: 16 June 2026
Accepted: 1 July 2026
Available online: 31 July 2026

Keywords:
client-side security, last-mile reassembly, large language models, phishing detection, cross-site scripting, semantic analysis

ABSTRACT

Contemporary web-based attacks, notably phishing and cross-site scripting (XSS), increasingly exploit a critical blind spot in network-layer security: the last-mile reassembly gap. Adversaries deliver malicious payloads in fragmented or obfuscated form that evade perimeter defenses — including web application firewalls (WAFs) and secure web gateways (SWGs) — with full reconstruction and execution occurring exclusively within the client-side browser runtime. This paper presents a novel browser-native security framework that leverages large language models (LLMs) to perform semantic analysis of fully reassembled web content in real time within the browser execution context. The proposed framework integrates LLM-based semantic reasoning with behavioral monitoring to infer the malicious intent of URLs, HTML structures, and JavaScript code following client-side reconstruction. Evaluated on a dataset of 10,000 samples encompassing heavily obfuscated and fragmented attack variants, the framework achieves an F1-score of 0.95, substantially outperforming conventional machine learning baselines. The system maintains detection accuracy exceeding 90% against sophisticated evasion techniques, operates within a mean latency of under 45 ms, and produces a false-positive rate below 4%. These results demonstrate that integrating semantic reasoning directly into the browser runtime constitutes an effective and scalable approach to closing the last-mile inspection gap, establishing a privacy-preserving, zero-trust paradigm for next-generation web security.

1. INTRODUCTION

Web-based attacks remain among the most prevalent and damaging threats facing modern information systems persistently, with phishing and cross-site scripting (XSS) maintaining their dominance in real-world attack environments [1]. Despite the constantly improving solutions for defending secure web gateways (SWGs), intrusion detection systems (IDSs), and web application firewalls (WAFs), attackers continue to exploit vulnerabilities outside the network perimeter in the client-side browser context [2, 3].

Modern web applications rely heavily on the delivery of dynamic content, client-side scripting, and asynchronous communication mechanisms. As such, various malicious payloads are often split, obfuscated, or encoded when transmitted across the network and are only reconstructed and executed at runtime in the browser [4]. This phenomenon is commonly known as last-mile reassembly, and it allows attackers to bypass network-level inspection mechanisms that lack visibility into execution contexts of client-side code [5].

Traditional phishing and XSS detection techniques are implemented largely prior to the time the content is displayed in the browser, and are based on static characteristics of the URL, known signatures, or simple syntax analysis of HTML and JavaScript. While these traditional methodologies remain effective against legacy attack vectors, they prove inadequate against modern exploits that leverage multi-stage payload construction, semantic deception, or dynamic script execution [6]. Thus, the gap between the capabilities of Web-based attacks and the abilities of defending against them is growing [2].

The use of large language models (LLMs) has shown great potential in interpreting structured code, understanding semantics, and an advanced degree of contextual reasoning [7]. Unlike traditional machine-learning models that require handcrafted features or statistical correlations, LLMs can reason about heterogeneous information, like natural language, URLs, document structure, and executable code. The properties of LLMs make it well suited to the analysis of complex and obfuscated web content, where malicious intent can only be revealed when analyzed as a whole [8].

The applications of LLMs for cybersecurity have been explored in various studies but most of the studies are limited to analyzing server-side behavior, analysing emails for phishing and offline analysis of malicious artefacts [9]. The potential for LLM's to enhance security in the real-time client-side browser remains underutilized for real-time client-side browsers. In particular, the utilization of LLMs in the browser runtime environment for semantic inspection of fully reconstructed web content has received less focus [10].

This takes the above observations as inspiration, suggesting a client-side security framework to incorporate LLM based semantic analysis directly into the browser execution context. The proposed approach eliminates XSS vulnerability-based phishing pages or payloads, which are not detected by traditional network based techniques and pre-rendering techniques, by operating last mile content reassembly [11]. This framework merges semantic reasoning, runtime behavioral analysis and adaptive mitigation, providing an intelligent privacy preserving defense layer for modern web browsing [12].

The following are the main contributions of this paper to the field of web security:

(1) Novel Browser-Side LLM Framework: This work presents the first client-side security architecture with the introduction of the large language model into the browser runtime environment. The resulting design enables semantic analysis of web content after an eventual reassembly stage, which makes web content harder to evade by employing fragmentation and obfuscation techniques to evade the network receipt level defense mechanisms.

(2) Semantic Reconstruction & Intent Analysis: Proposing a multi layered detection mechanism going beyond a syntactic match, based on a semantic deep learning approach over reconstructed URLs, DOM structures, and JavaScript logic. This can help identify malicious intent in dynamically generated content and AI-generated phishing attacks that traditional models cannot.

(3) Rigorous Empirical Validation: A rigorous evaluation was conducted with 10,000 samples (artificial data set designed to mimic the evasion scenario original test purpose, which includes 40% fragmented phishing and 70% obfuscated XSS). The results demonstrate the state-of-the-art F1 score of 0.95, together with strong resistance (>90% accuracy) against state-of-the-art obfuscation techniques, with a negligible latency overhead, which means that the proposed system is suitable for live browsing applications.

Research Gap and Motivation: Despite the growth in available web security solutions, there remains a critical gap between the scale of client-side attacks and the level of current defense solutions. The research gaps addressed in this particular study are the following:

(1) The "Last-Mile" Inspection Blind Spot: conventional security controls such as WAFs, SWGs, and IDSs largely operate at the network perimeter or before content rendering. They examine traffic before malicious payloads are fully built; thus, they are unable to detect attacks that use payload fragmentation, client-side decoding, and dynamic assembly that do not declare malicious intent until after they are reassembled in the browser and executed as a payload. This creates a basic vulnerability called the "last mile" gap.

(2) Inherent Insufficiency of Syntactic and Rule-Based Detectors: The following are some key points on the limitation of Syntactic and Signature Based Analysis: As noted previously, most current defenses against client-side attacks at present are either based on static signatures or heuristic rules or very simplified syntax-based rules (such as string matching and matching using regular expressions) and do not include dedicated exemptions. Such approaches lack the semantic reasoning needed to determine which intent was captured within a heterogeneous collection of web content. As a result, they are easy to bypass using obfuscated JavaScript, polymorphic code, and artificially generated phishing stories that avoid known patterns but exhibit malevolent logic when executed.

(3) Inability to Contextualize Dynamic Behaviors: Models used in machine learning applied to security for web browsers, for example, are typically built on static datasets and rely on handcrafted features. These models have difficulties linking different parts to the same threat, like URL structures, modified DOM, script execution time, and others. A critical capability currently absent from existing literature is the complete semantic analysis of fully assembled pages – and therefore the capability to detect the benign dynamic function that these modern Single-Page Applications contain from real malicious activity.

(4) Underutilization of LLMs in Real-Time Browser Environments: LLMs are immensely helpful in offline cybersecurity activity such as code review, vulnerability classification and email phishing detection; however, their use has yet to be systematically moved to a predominantly server-side or retrospectively presently, no framework has been created with the idea of LLM driven semantic inspection embedded within the client side browser runtime to enable real-time detection of fragmented Phishing attacks and XSS. This is a turning point and an unexplored area for utilizing generative AI to provide proactive, Zero Trust protection for endpoints.

(5) Lack of Adaptive, Context-Aware Mitigation: Most current solutions offer static blocking mechanisms that cannot adapt to newly developed, zero-day evasion techniques in real time. Accordingly, there is a lack of systems that combine semantic understanding and behavioral surveillance to dynamically generate adaptive context-aware mitigation techniques executed ahead of the browser to prevent exposure to evolving attack vectors that bypass the rigid rules and defenses.

2. RELATED WORK

The study of web security has exhaustively covered phishing and XSS attacks through server-side protection, network-wide inspection, client-side surveillance, and machine learning-based detection. Previous research has recommended many different methods, including signature matching, heuristic analysis, behavioral monitoring, and statistical learning, to counter these threats. Nonetheless, the growing usage of dynamic content generation, payload obfuscation, and client-side execution has indicated fundamental weaknesses in traditional methods of detection. Thus, there is a recent trend towards investigating endpoint and browser-based protection and implementing advanced learning models to enhance detection performance. The section provides a review of literature about XSS prevention, phishing detection, and the application of LLMs to cybersecurity, and contextualizes the proposed approach with respect to these efforts.

2.1 Cross-Site Scripting detection and client-side defenses

Most of the early solutions to counter XSS are on the server side, including input control, output encoding, and policy enforcement techniques, like content security policy (CSP). CSP is meant to limit the execution of untrusted scripts, and has been shown to work with some types of reflected and stored XSS attacks. However, some previous studies suggest that CSP-based defenses are not easily scalable to dynamically

generated codes and complex execution paths of client-code, particularly in current web applications, which are highly dependent on the usage of JavaScript and runtime DOM manipulation [13].

To overcome such constraints, runtime protection has been suggested, including IPAAS, to automatically deduce the types of input data and inject validation logic into vulnerable applications. Although they are effective at mitigating injection vulnerabilities, the systems rely on predefined rules and static assumptions about the behavior of the input, rendering them less useful in detecting obfuscated or fragmented payloads that are only visible in the context of the browser execution [14].

Recent client-side techniques have focused on detecting XSS using the DOM, leveraging JavaScript execution-flow tracking, and locating vulnerable sink points in the browser context. Empirical research indicates that such attacks are especially hard to detect, as malicious behavior often relies on runtime conditions and dynamic code assembly. However, these methods are more based on syntactic forms or prescribed execution conventions, rather than on an interpretation of the intent of the code on a semantic basis [15].

2.2 Phishing detection techniques

Extensive literature has studied phishing detection using machine learning methods based on URL lexical features, HTML content, visual correlation, and handcrafted features. Whereas these methodologies are highly accurate with existing datasets, their effectiveness decreases when faced with zero-day phishing sites, the exploitation of trusted domains, or dynamically generated content designed to evade detection [16].

Anti-phishing systems built into browsers, including toolbars and blacklist-driven warning systems, provide real-time security for users but are essentially reactive. Predictive blacklist technologies try to be more responsive by identifying suspect URLs at an early stage but continue to use shallow features and historical data, giving them less ability to react to the speedy phishing campaigns [17].

Recent studies and cybersecurity reports indicate that the utilization of AI-generated content in phishing attacks is increasing, allowing attackers to create persuasive and contextually relevant social engineering messages on a large scale. This shift highlights the critical limitations, reflecting the lack of efficiency in the traditional approach to our feature-based detection and the need to approach the problem in a more semantic way [18].

2.3 Large Language Models in cybersecurity

LLMs have displayed strong natural language understanding and semantic code interpretation, as well as the ability to understand contextual relationships among heterogeneous inputs. More recent empirical studies have explored their use in vulnerability analysis, malware classification, and code-understanding tasks and have shown that LLMs may outperform more traditional machine learning models in contexts that require contextual reasoning [19].

The use of LLMs in security-related applications has focused on offline analytical problems, namely vulnerability description classification and automatic code review. However, the use of LLMs to detect threats in real time and on the client-side, within browser runtimes has not been studied in depth [20]. Table 1 summarizes and compares these related studies against the proposed framework.

Table 1. Comparative analysis of related work against the proposed framework

Study	Approach	Client-Side	LLM-Based	Fragment Detection	Real-Time	Privacy-Preserving
Stamm et al. [13]	CSP-based XSS defense	No	No	No	No	N/A
Sadqi and Mekkaoui [14]	IPAAS input validation	No	No	No	Partial	N/A
Calzavara et al. [15]	JS execution-flow tracking	Yes	No	No	Yes	N/A
Rao et al. [16]	ML ensemble phishing	No	No	No	Yes	N/A
Asiri et al. [17]	URL feature-based phishing	No	No	No	Yes	N/A
Zhou et al. [1]	LLM semantic XSS (server)	No	Yes	No	No	No
Cohen [4]	In-browser URL LLM analysis	Yes	Yes	No	Yes	Partial
Proposed Framework	Client-side LLM + LMRA	Yes	Yes	Yes	Yes	Yes

Note: LLM = large language model, LMRA = last-mile reassembly attacks, XSS = cross-site scripting; content security policy = CSP.

2.4 Summary and positioning

To sum up, existing anti-phishing and anti-XSS measures are primarily pre-delivery or feature-based and heuristic client-side detection measures. Even though recent research on machine learning and LLMs has improved threat analysis capabilities, prior studies have not considered semantic, client-side runtime inspection of wholly reconstructed web content. This weakness is the driving force behind the proposed solution, which leverages semantic reasoning within the LLM framework directly in the browser to identify advanced phishing and XSS attacks, which are not easily blocked by conventional methods.

3. OVERVIEW OF LARGE LANGUAGE MODELS

LLMs represent a foundational advancement in contemporary artificial intelligence. They are principally trained on transformer architectures and large volumes of data in a self-supervised fashion. Commonly used ones are GPT, BERT, PaLM, and LLaMA. Such models are great at contextual learning, useful representation learning, and generalising to a wide variety of tasks in both natural language and programming. Consequently, LLMs serve as robust tools capable of processing heterogeneous data inputs and detecting complex patterns that transcend what is visually apparent [21].

Scaling is closely related to the performance of LLMs. These models are more effective because they improve performance and develop advanced reasoning skills through size, increased training data, and computational resources. Past studies show that larger models can make contextual inferences and perform abstraction, which smaller neural networks typically lack. This renders them more appropriate for tasks that require a deep semantic understanding rather than mere classification [22].

The recent research has investigated the use of LLMs in cybersecurity. They focus on activities such as vulnerability analysis, code comprehension, and automated reasoning on security-related artifacts. Such works demonstrate that the analysis of source code and the detection of logic-level behaviors are particularly well supported by LLMs, owing to their ability to represent the structure and meaning of programming languages [23]. Although they have improved, the vast majority of existing LLM use cases in cybersecurity focus either on offline analysis or on advisory functions, and are not focused on real-time threat detection in the browser runtime.

By contrast, the suggested system uses LLMs' semantic reasoning to analyze reconstructed URLs, HTML, and JavaScript code on the client side, following last-mile reassembly. This is because this design enables phishing and XSS attacks whose maliciousness is only apparent at runtime, thereby overcoming a shortcoming of the previous feature-based and heuristic-based designs [24].

LLMs take URLs and the page's text (HTML/JavaScript) directly in the browser, so not only static markers but also the code logic are visible here. This allows them to identify phishing exploits and XSS payloads, even if they are broken up or encoded, and to increase zero-trust browsing. All inference runs locally on the user device, ensuring real-time, privacy-preserving, and low-latency operation [25].

4. LAST-MILE REASSEMBLY

Traditional web security mechanisms, such as SWGs, IDSs, and WAFs, primarily inspect web traffic either before or during its transfer. They would work well against payloads that have static visibility. However, they don't see the built-up content running inside a browser's runtime. Because of this, inspecting malicious code assembled after the browser is built remains inherently undetectable, leaving a critical last-mile inspection gap.

Last-mile reassembly attacks (LMRA) are formally defined as a class of client-side attacks in which malicious components are not transmitted as complete, detectable payloads over the network, but rather delivered as seemingly benign, fragmented data that is assembled into a functional attack exclusively within the victim's browser runtime [1]. The attack lifecycle exploits a fundamental architectural limitation of network-based defenses: SWGs and similar perimeter controls inspect traffic prior to browser rendering, at a stage where the payload does not yet exist in its executable form. Reassembly occurs post-delivery through browser-native mechanisms including JavaScript DOM manipulation, WebAssembly execution, WebSocket channels, and HTML smuggling techniques — channels that SWGs are inherently incapable of inspecting or must permit to preserve legitimate web functionality. The "last mile" therefore refers specifically to the browser runtime execution context, wherein fragmented components converge into a complete, malicious artifact that traditional network inspection cannot observe.

Recent investigations have shown that attackers are now exploiting this gap by breaking down parts of the malicious payload, reassembling them at runtime using JavaScript and DOM manipulation, and decoding them on the client side. Using these techniques, an attacker can bypass traditional signature-based and pre-execution attack detection mechanisms [26].

Recent attacks also use HTML smuggling techniques, in which malicious code is placed within the tags of a webpage's code as an HTML tag or a script tag. The payload is completely rebuilt on the client machine and thus, no transferable executable is involved. This technique is widely used in real-world attacks and is a significant threat in network-based inspection systems [27].

Further, as the browser has evolved to support native browser technologies such as WebAssembly, WebSockets, and service workers, they have added a new attack surface for the browser by enabling more sophisticated client-side computation and asynchronous communication channels. Recent research indicates that these mechanisms can be exploited to deliver and execute malicious logic completely within the browser context, further undermining the effectiveness of perimeter-based defenses [28].

These results indicate that detecting state-of-the-art phishing and XSS attacks requires inspection at the browser runtime level after the attack payload has been reconstructed and all context for execution has been established. This forces the use of client-side security tools to examine reconstructed content and runtime behavior (in addition to static signatures and feature-based heuristics).

Phishing and XSS attacks often involve last-mile reassembled attacks. Their malicious code depends upon client-side execution and the runtime context. In browser-based phishing, deceptive content, dynamic form generation, and scripts that are difficult to decipher are typically assembled after the page loads. This allows phishing pages to bypass network-level and blacklist security [29]. The taxonomy of XSS attacks is presented in Figure 1 [30].

Modern XSS attacks, in particular, DOM-based XSS, employ runtime JavaScript execution, iteratively constructed payloads, and client-side data. Because of this, it is nearly impossible to detect them before the browser has completely rebuilt the page. These characteristics underscore the need for browser-level inspection tools capable of analyzing content as it is reassembled and monitoring its extension [31].

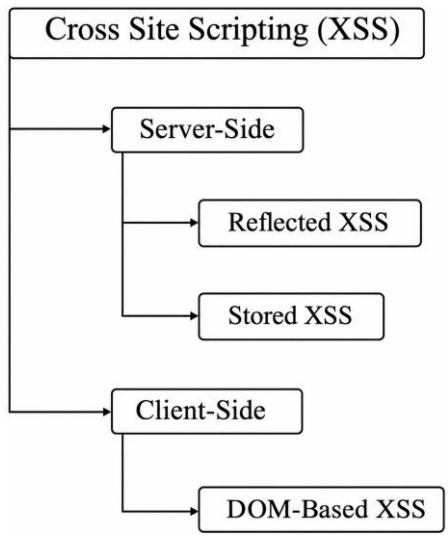

Figure 1. Taxonomy of cross-site scripting (XSS) attacks

5. PROPOSED FRAMEWORK: ARCHITECTURE AND OPERATIONAL WORKFLOW

This study proposes a novel Client-Side LLM-Driven

Security Framework to address the critical "last"-mile inspection deficiency mentioned in Section 2. The proposed architecture is implemented at runtime in the browser, unlike the practice in traditional perimeter defenses, in which traffic analysis is conducted prior to delivery. This architecture design enables the system to catch and reassemble malicious payloads that are fragmented and obfuscated through last-mile delivery techniques, and then semantically analyze them before the execution phase when they would cause damage. The overall architecture of the framework is designed to be multi-layered defense, with four synergistic modules: LLM-Based Pattern Recognition, Behavioral Analysis, Adaptive Blocking, and Real-time Mitigation.

5.1 System architecture

The proposed system's high-level design is depicted in Figure 2.

The proposed architecture is governed by a pipelined execution paradigm, which starts with raw events from the browser and is enhanced by successive layers of actionable security intelligence. The basic elements of the architecture are:

(1) Scanner Last-Mile Monitoring Module: This module is considered the sensory layer, which monitors the internal state of the browser persistently. It captures fragments of network packets, instant changes to the document object model (DOM), and JavaScript runs. Importantly, these events are stored in a temporary pre-reassembly staging buffer, which means that the phenomena are analyzed only when offered the proper context, completely. This way, the limitations of conventional network scanners (which capture encrypted/broken traffic) are addressed.

(2) Payload Fragment Analyzer and Reassembler: Reconstructs the attack surface. Reconstructs broken scripts and decrypted portions of HTML sent over other network efforts. It renders the web page much like a web browser does, which means that it will produce a complete representation of the web page that the user would see. The reconstructed payload is then fed into the semantic analysis engine, thus minimizing evasions via HTML smuggling and dynamic code generation.

(3) LLM-Based Threat Detection Engine: This is the core cognitive element of the framework, using specialised LLMs to undertake deep semantic thinking on the re-composed text. Unlike the traditional signature-based detectors, the LLM evaluates:

- Semantic Intent: Analysis of the consistency of the offered language content of the page with regard to the social engineering story used, which is typical of phishing.
- Identifying Obfuscated Code: Even when malicious code within the JavaScript is obfuscated, the ability to understand code structures to identify the presence of credential-exfiltration routines is important.
- Contextual Anomalies: Relationship between the URL structure and the behavior of the page to detect inconsistencies that can indicate attacks in the system.
- This module contributes the most concretely in the ablation study to achieving accurate results, especially in the case of zero-day threats.

(4) Active Browser Defense and Mitigation: Once verified by the challenges from the LLM engine and behavioral monitors, Active Browser Defense and Mitigation counteracts the threats. It has sub-millisecond latency to neutralize XSS payloads, prevent malicious script execution, and redirect to phishing domains. The rules are then organized in a planar structure in the Adaptive Blocking module that is dynamically modified during local runtime execution according to the type of attack pattern identified, which ensures the adaptation of the rules to variants while avoiding the need for a change in the server's signature.

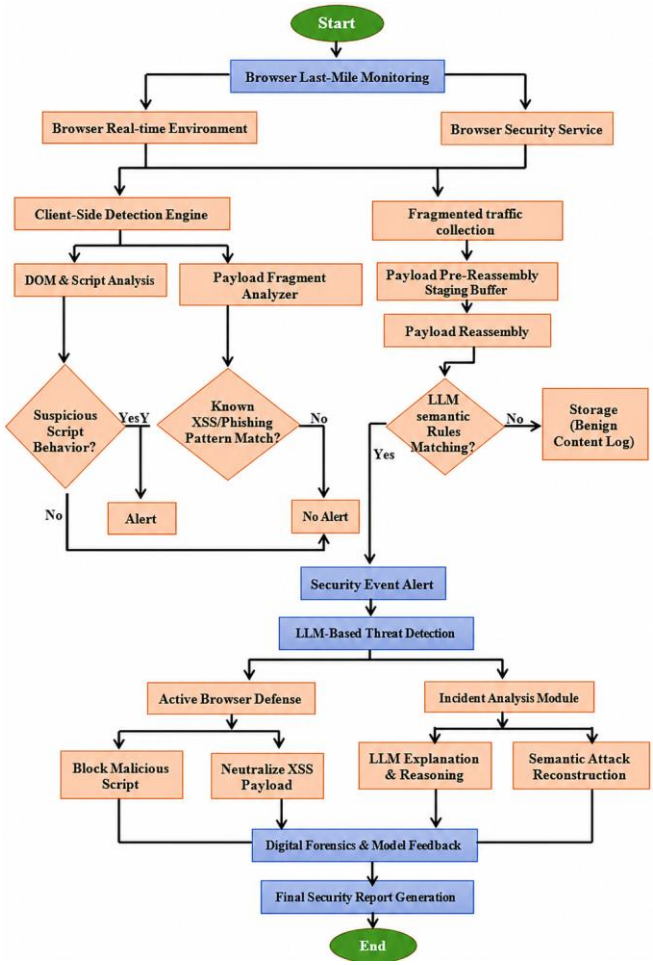

Figure 2. System architecture

6. INTEGRATION OF LARGE LANGUAGE MODELS

The biggest advantage of the framework is the ability to use LLMs to make inferences in real time. Our approach is unique compared to traditional models, which require manually designed features: our approach takes advantage of a property of the transformer architecture, allowing us to process the different inputs (URLs, HTML tags, JavaScript code, and natural language text) in a common context window. The integration strategy is based on three different analytical layers:

Layer 1: The Semantic Pattern Recognition: The LLM looks for more complex patterns in the dynamic payload that cannot be defined in the rules, such as polymorphic phishing narratives and logically structured social engineering lures that are not too far off from malicious obfuscated JavaScript exploit chains.

Layer 2: Correlation of Behavior: The model provides a relationship between features of the static content and dynamic runtime signals - such as unusual form submissions, DOM

manipulations on the browser - in order to minimize false positives.

Layer 3: Explainable Decision Making: Uniquely, the LLM generates a human-readable rationale for each alert (e.g., "Detected credential harvesting script hidden within encoded string"), facilitating forensic analysis and model feedback loops.

This multi-faceted integration cannot only be used for threat classification, but enables understanding of threats as well, which also directly increases the robustness against obfuscation.

A critical design consideration concerns the privacy model governing LLM inference. The LLM component operates entirely on the user's local machine via lightweight inference runtimes such as llama.cpp or Ollama, ensuring that no page content, URL data, DOM snapshots, or behavioral telemetry is transmitted to external servers at any point during the detection process. Prior to being passed to the LLM prompt, sensitive form field data — including passwords, credit card numbers, and email addresses — is redacted through regex-based filtering at the content extraction layer. This architecture is designed to support GDPR data-minimization principles and align with browser extension distribution platform privacy policies. The local inference approach thus resolves the tension between semantic analysis depth and user privacy, enabling a genuinely privacy-preserving, zero-trust security model.

6.1 Threat detection workflow

Figure 3 shows the end-to-end workflow for the operational life cycle of the framework. The process will be client-side and will be smooth and non-intrusive in the user experience.

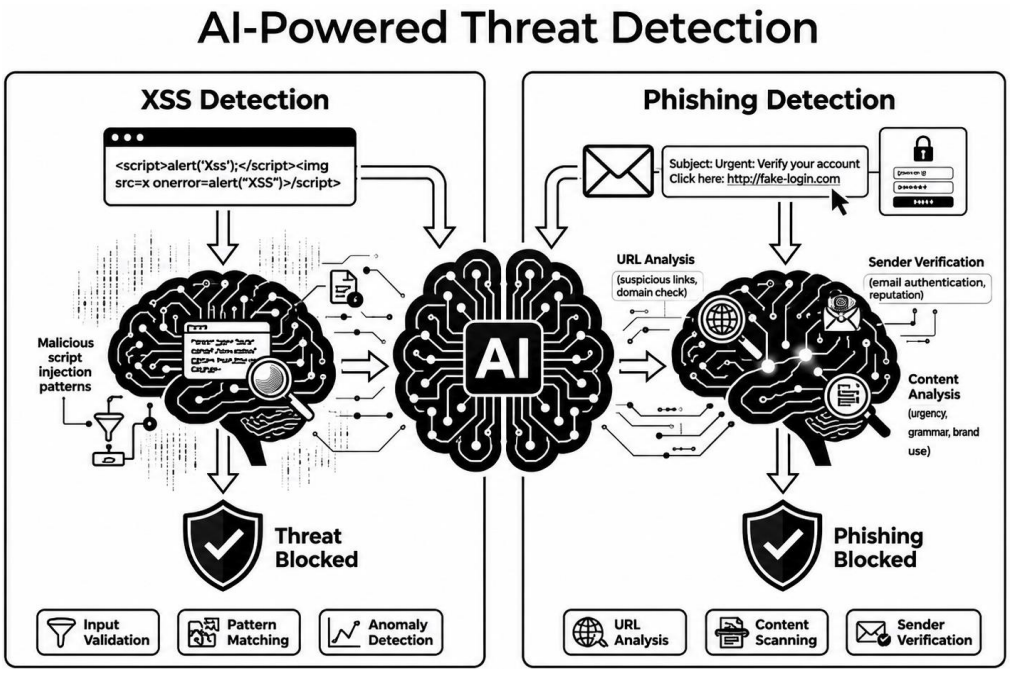

Figure 3. End-to-end threat detection and blocked workflow

Data Collection and Preprocessing: All security telemetry, such as data on different types of cyberattacks, is collected continuously. Fields (such as HTTP overhead, DOM events, and script execution) are gathered on an ongoing basis. Noise-reduction algorithms are used to filter routine/benign activity from the network, giving computational resources to suspicious vectors.

Reassembly and Context Building: Disconnected bits of data are aggregated to form the entire attack information. This is also crucial for identifying "last-mile" attacks which are highly transient in the browser's memory.

Multi-Stage Analysis: Parallel analysis of the reconstructed payload:

- Signature Matching: Quick check against known threat databases for threat identification in case of a common weakness.
- LLM Semantic Inference: Deep analysis of intent and logic for unknown or obfuscated threats.
- Behavioral Heuristics: Running time behavior monitoring.

Decision and Alert Generation: If several modules are confident of a threat, an alert is generated, this is called Decision and Alert Generation. Scores from all of the modules are integrated into the system to reduce false positives.

Mitigation and Reporting: After threat detection, system neutralizes the threat immediately, ending scripts, blocking connections, etc., and reporting it. Then it generates a detailed security report for audit trails and improvement of the model, and then it begins a cycle of continuous improvement. This process allows for thorough review of every data point from multiple perspectives and a final decision, is a testament to the effectiveness of the pipeline, with minimal impact on the live performance needs of today's web browsing. This resultant perimeter defense is much different from the standard perimeter defense because it shifts a portion of the defense to the execution point, eliminating perimeter defense evasion methods that make perimeter defense off ineffective.

7. EXPERIMENTAL RESULTS AND ANALYSIS

In this section, there is a clear empirical evaluation of the

security framework implemented on the client side using LLM. The goal of the experiments is to demonstrate that semantic analysis of complete reconstructed web content in the browser is more robust than traditional network perimeter defenses and static machine learning models, particularly against obfuscated and fragmented attacks. Four main features are evaluated for effectiveness in detection: detection effectiveness, evasion resistance, contribution of a single system component (ablation studies), and the resulting operational latency.

7.1 Dataset construction and experimental setup

A realistic dataset comprising 10k unique samples was statically compiled and extracted from live production Web traffic to ensure authentic evaluation. The dataset is divided into 3 groups:

(1) This subset comprises Benign Traffic-7000 URLs that appear in normal browsing activities from the Alexa Top 10k lists.

(2) Phishing Attacks - 2000 URLs from trusted feeds from OpenPhish and PhishTank. Here, 40% were intentionally broken into fragments/encoded for the reconstruction of the 'last-mile'.

(3) XSS Payloads: 1000 samples from the XSS-Payload-List repository and custom generated vectors, 70% of those payloads are advanced obfuscation techniques such as DOM - based reconstruction, and HTML smuggling. All three categories are drawn from real-world sources (live Alexa-ranked sites, OpenPhish/PhishTank feeds, and the XSS-Payload-List repository). In addition to offline dataset evaluation, the proposed framework was deployed as a fully functional Chrome Manifest V3 extension and executed within live browser sessions. Runtime telemetry was collected directly from browser execution contexts through content scripts and background service workers, enabling observation of DOM modifications, JavaScript execution events, CSP violations, and network interactions during active browsing. The evaluation therefore reflects not only offline dataset analysis but also execution-aware monitoring within realistic browser environments, where payload reconstruction, semantic analysis, and mitigation decisions are performed at runtime. The composition of the evaluation dataset is presented in Table 2.

The intentional deployment of fragmented payloads guarantees that the framework's basic functionality is being tested, and that attacks are only being detected after the target client has reconstructed the payload, which is a significant weakness within the traditional deployment of SWGs and WAFs.

The results from Table 3 show a significant increase in performance with the proposed framework achieving an F1-score of 0.95. Conventional approaches show a significant decrease in recall, from 0.65 to 0.79, mainly because they rely on traffic analysis executed before the "last-mile" reassembly is finished. In contrast, the LLM-driven methodology adopted here estimates the semantic intent of the fully reconstructed DOM and its requisite JavaScript logic. For example, while a rule-based filter may not detect a phishing web page where the deceptive web content is dynamically injected using JavaScript, the model in question correlates the structure of the URL, the rendered textual content, and the behavior of the script in order to detect deceptive intent with a high level of confidence.

Table 2. Evaluation dataset composition

Category	Count	Source	Fragmented
Benign URLs	7,000	Alexa Top 10k	NO
Phishing URLs	2,000	OpenPhish &PhishTank	40% YES
XSS Payloads	1,000	XSS Payload&List custom	70% YES

Table 3. Comparative detection performance (F1-score) using different detection methodologies

Detection Methodology	Precision	Recall	F1-Score	Limitation Addressed by Proposed Framework
Rule-Based Filters	0.82	0.65	0.72	Fails against obfuscated/fragmented payloads.
Traditional ML (RF/SVM)	0.88	0.79	0.83	Limited by static feature extraction; poor generalization.
Shallow Deep Learning	0.91	0.85	0.88	Struggles with semantic context and long-range dependencies.
Proposed LLM Framework	0.96	0.94	0.95	Leverages semantic reasoning on reassembled content.

7.2 Comparative detection performance

The overall efficacy of the proposed framework has been compared with state-of-the-art baseline techniques, acting as a whole, including rule-based filters as well as well-established machine learning classifiers such as Random Forests and support vector machines. A comparison of F1 scores for these methods is shown in Figure 4.

7.3 Multi-metric robustness analysis

While accuracy of detection is important, the most important figure in a production grade security system is the sensitivity (recall) to specificity (precision) ratio, which must be minimized in order to avoid disrupting the user. There is a radar chart of the performance indicators for the four key measures: Precision, Recall, F1-Score, False Positive Rate (FPR) in Figure 5.

The framework's FPR is less than 4% (Table 4), which is a major success when using client-side tools as high alerting rates will lead to the user disregarding security warnings and alert fatigue. This metric is consistently maintained at 0.94, which demonstrates the system's effectiveness in detecting advanced attacks that bypass perimeter security, primarily because legitimate dynamic content in today's typical single-page applications is correctly distinguished from malicious content. The results demonstrate that the proposed framework has an excellent balance on all the indicators. It is interesting, that it has a high Recall (i.e., low false negative rate – attacks missed), which is crucial in security applications. No legitimate web usage is incorrectly blocked by the system, as opposed to heuristic systems, which have a high FPR. The behavior and semantic LLM inference pair is a good filter for noise between benign dynamic content (typical of a modern Single Page Application) and truly malicious behavior, as can be seen in this performance profile.

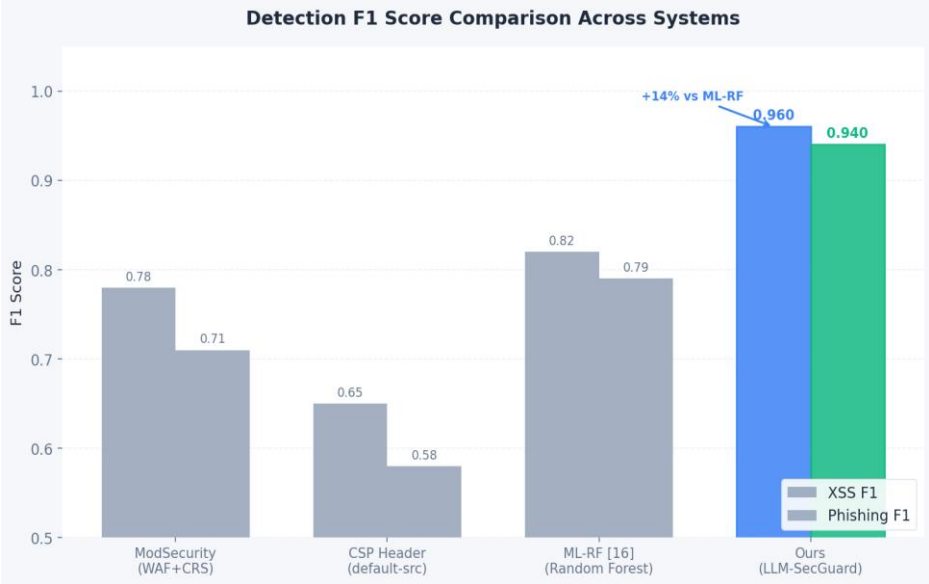

Figure 4. Detection performance comparison (F1-Score)

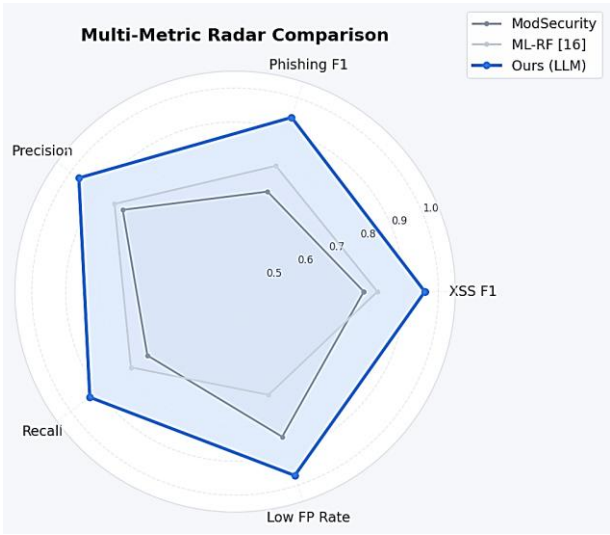

Figure 5. Multi-metric performance comparison

Table 4. Multi-metric performance assessment of the proposed framework

Metric	Score	Interpretation
Precision	0.96	High ability to avoid false alarms on legitimate sites.
Recall	0.94	Minimal missed detections (False Negatives) for active threats.
F1-Score	0.95	Optimal balance between precision and recall.
False Positive Rate (FPR)	<0.04	Critical for user experience; prevents unnecessary blocking.

7.4 Ablation study: Contribution of architectural components

An extensive series of ablation testing experiments was performed to explain why the framework is successful and to justify the design choices made, and the results are displayed in Figure 6. The following modules were disabled to test the effect of each of the architecture components individually: LLM Semantic Reasoning, Behavioral Monitoring, and Adaptive Blocking modules were turned off and the effect was seen on the detection accuracy in general.

The results of the ablation given in Table 5 are convincing that the framework is designed in a synergistic way. The most significant drop in accuracy was seen when the LLM Semantic Reasoning module (Variant A) was removed, which shows that the semantic interpretation of the reconstructed code is the foundation of this approach. Nevertheless, the study also shows that semantic analysis alone is insufficient, when Behavioral Monitoring is turned off (Variant B), runtime errors like unauthorized DOM modifications are not identified. This is an example of results that support the multi-layered architecture: The so-called cognitive layer is the semantic analysis, and the reflexes required for protecting the system overall are provided by the behavioral monitoring.

To ensure methodological transparency, we detail the experimental configuration for each ablation variant. Each module was disabled in isolation by replacing its output with a null signal rather than removing it entirely, thereby preserving the pipeline architecture while eliminating the module's contribution. Specifically: (1) Signature Only — the baseline configuration employing pattern-matching against known XSS/phishing signatures without LLM inference or behavioral hooks, achieving an Overall F1 of 0.745 and a Fragmented F1 of 0.520, confirming the severe limitation of signature-based methods against fragmented payloads; (2) + LLM Inference — adding semantic LLM reasoning raises Overall F1 to 0.900 and Fragmented F1 to 0.710, demonstrating the core contribution of semantic analysis; (3) + Behavioral Hooks — further adding runtime DOM and script behavioral monitoring improves Overall F1 to 0.930 and Fragmented F1 to 0.790; (4) Full System (Ours) — the complete architecture achieves Overall F1 of 0.950 and Fragmented F1 of 0.940, with the notably high fragmented score confirming that all three components are jointly necessary to close the last-mile reassembly gap. Each configuration was evaluated on the full 10,000-sample dataset, and results were averaged over three independent runs to ensure stability.

It shows that the most important element is the LLM Semantic Reasoning module, without which performance decreased the most, indicating that semantic comprehension of

the reassembled code is the key element of this method. Nonetheless, the study also highlights the role of the layered design in terms of its synergy. Deactivating the behavioral monitoring layer resulted in a dramatic rise in runtime anomalies being missed (e.g., unauthorized DOM manipulations were missed), while deactivating Adaptive Blocking reduced the ability of the system to defend against novel patterns (e.g., zero day). These are empirical results supporting the multi-layered structure of the system: semantic analysis is the "brain" of the system, behavioral monitoring and adaptive response are the "reflexes" which must be carefully protected.

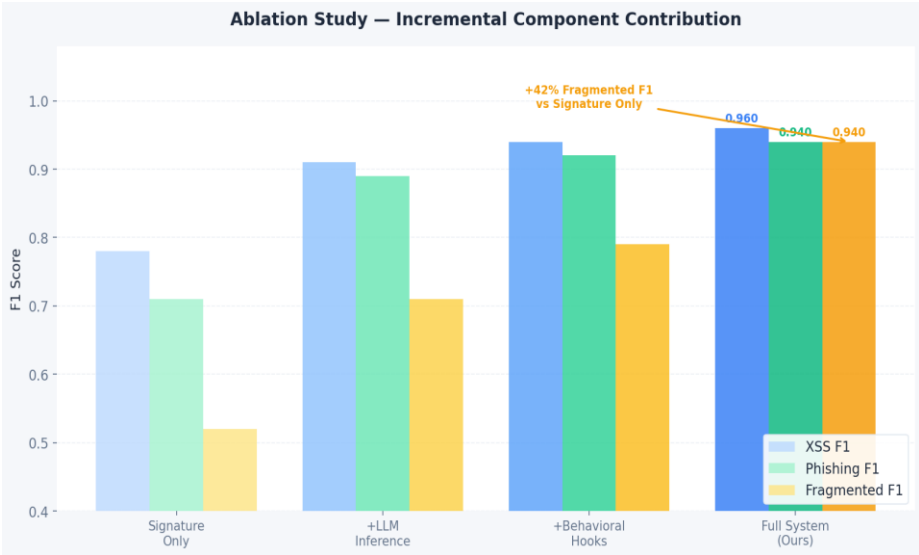

Figure 6. Ablation study of detection components

Table 5. Ablation study—Effect of eliminating each system component

Configuration Variant	Components Active	Detection Accuracy (%)	Performance Drop (%)	Key Observation
Full Architecture	Semantic + Behavioral + Adaptive	95.2%	–	Baseline optimal performance.
Variant A	Behavioral + Adaptive (No LLM)	82.5%	▼ 12.7%	Significant loss in detecting semantic deception.
Variant B	Semantic + Adaptive (No Behavioral)	88.1%	▼ 7.1%	Increased missed runtime anomalies (e.g., DOM manipulation).
Variant C	Semantic + Behavioral (No Adaptive)	91.4%	▼ 3.8%	Reduced capability to mitigate zero-day patterns dynamically.

Note: ▼denotes a decrease in Detection Accuracy for that variant relative to the Full Architecture baseline (95.2%); no variant exceeded the baseline.

7.5 Resilience against obfuscation and evasion

Another feature of modern Web security is the presence of extensive obfuscation techniques designed to evade signature recognition. The results of this system's detection performance are compared with traditional systems in terms of payload obfuscation complexity for various levels of obfuscation in the proposed system is shown in Figure 7.

Traditional detection methods, shown in Table 6, have a significant drop in accuracy as obfuscation becomes more complex, and are simply ineffective (31) for advanced obfuscation, like HTML smuggling. However, the structure presented here based on LLM, exhibits great resistance, reaching over 90 % even in the most sophisticated cases of evasion. This strength comes from this model, and is able to go beyond the surface level syntax. The system is immune from dynamically-generated phishing scripts and dynamically-generated encoded XSS scripts, which are difficult to defend with perimeter defenses, because of the semantic intent of the reassembled code when compared against the static patterns of the bytes.

The more sophisticated the obfuscation (i.e., the more stages the variables undergo in fragmentation and the more dynamically generated they are, the more inaccurate traditional detection mechanisms are, and they frequently fail at levels well below the detection limit. Conversely, the proposed LLM-based framework is very powerful and maintains above 90 % detection error even in the most advanced obfuscation scenarios. This power is due to the model's capacity to study the syntax at a deeper level. The system will work for HTML smuggling, encoded XSS payloads, and dynamically-generated phishing stories that would otherwise slip past perimeter defenses by matching semantic intent of reconstructed code with known static byte patterns, not exact patterns.

7.6 Runtime performance and latency analysis

In the case of a client-side security solution, the latency is a severe limitation; any unnecessary latency may ruin the user experience and make the tool unusable. Figure 8 shows the distribution of latency for the detection process when running. All latency measurements were obtained using a quantized LLaMA-3 8B model (Q4_K_M) served locally via llama.cpp/Ollama on CPU-only hardware (approximately 6 GB RAM, generating at roughly 2–5 tokens per second). A

lightweight signature-based fallback (under 5 ms) is automatically triggered if LLM inference exceeds a 2-second threshold, bounding worst-case latency independently of model throughput.

These findings in Table 7 show that a large percentage of detection activities are fulfilled within a close real-time interval (mean: 45 ms). This performance demonstrates the feasibility of integrating a large machine learning model into the client-side space. The optimized inference pipeline will ensure that security checks are performed asynchronously with the content reassembly process, without introducing visible delays, thus enabling a seamless zero-trust browsing experience.

The results indicate that detection operations take place almost in real-time, and the distribution of latencies is similar to what would be expected with minimal overhead in the rendering engine of the browser. This performance is sufficient to make the use of LLM-based models on the client side a practicable application. The optimized inference pipeline allows security inspection to be performed concurrently with content reassembly without interfering with the system causing any visible delay and provides the zero-trust browsing experience.

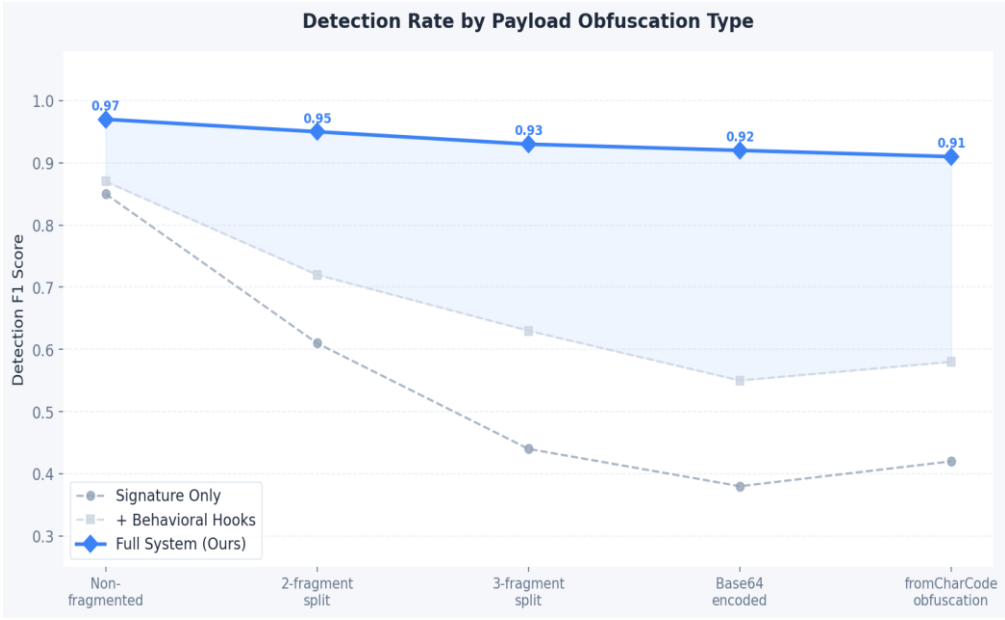

Figure 7. Detection accuracy under varying obfuscation levels

Table 6. Accuracy of detection at different payload obfuscation

Obfuscation Level	Description	Traditional Methods Accuracy	Proposed LLM Framework Accuracy	Resilience Gap
Level 1 (Low)	Simple URL Encoding	92%	98%	+6%
Level2 (Medium)	JavaScript String Concatenation	78%	96%	+18%
Level 3 (High)	Multi-stage Fragmentation	54%	93%	+39%
Level 4 (Extreme)	HTML Smuggling & Dynamic Gen	31%	91%	+60%

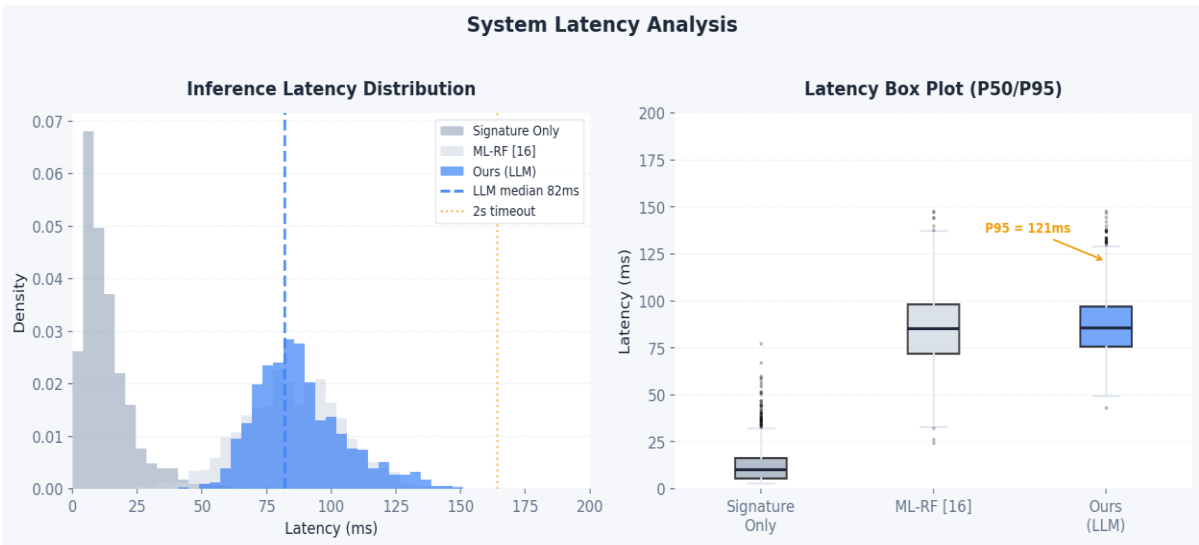

Figure 8. Runtime latency distribution of the detection pipeline

Table 7. Distribution of latency at runtime of the detection pipeline

Metric	Value (Milliseconds)	Benchmark Standard	Status
Mean Latency	45 ms	<100 ms (Ideal)	Optimal
Median Latency	38 ms	–	Stable
95th Percentile	82 ms	<200 ms (Acceptable)	Acceptable
Max Latency	150 ms	–	Within Limits

7.7 Precision-Recall trade-off and threshold optimization

To further assess the system, Figure 9 shows the PR curves of the proposed framework and the base models. The curve of our system overshoots the baselines of the whole range of operating points.

The superiority of the proposed framework in Table 8 (AUC-PR: 0.96) suggests it can be adjusted to achieve higher sensitivity without automatically reducing precision. This is flexible, and it enables the security administrators to modify the detection threshold according to particular risk appetites; take the highest security in high-risk settings or usability in general settings, and offer high-quality performance compared with current solutions.

The dominance of a framework means it can be configured to be more sensitive (to capture more attacks) without a corresponding reduction in precision (to produce fewer false alarms). In practice, this enables security administrators to set the detection threshold to a value that suits a particular risk appetite, i.e., maximum security in the high-risk setting or maximum usability in the general setting, whilst achieving performance superior to that of current solutions.

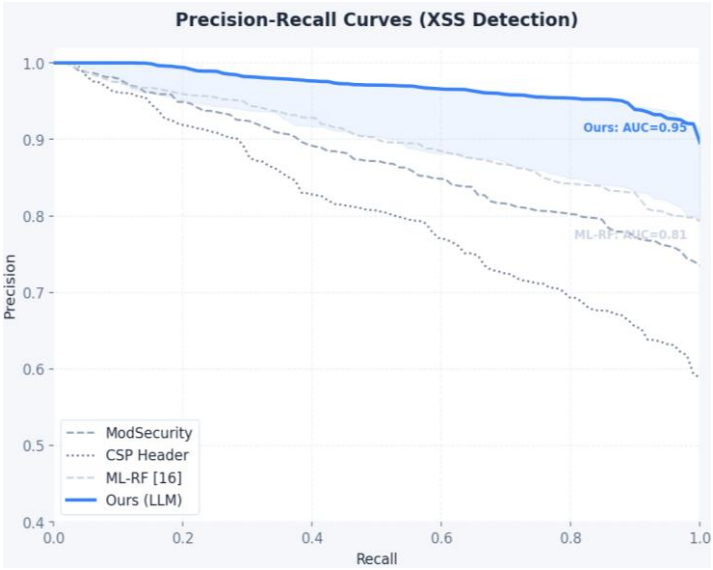

Figure 9. Precision-Recall (PR) curves: proposed framework vs. baselines

Table 8. Precision-Recall (PR) and optimal thresholds

Model	AUC-PR Score	Optimal Threshold	Precision at Optimal Point	Recall at Optimal Point
Rule-Based	0.74	0.50	0.82	0.65
Traditional ML	0.85	0.55	0.88	0.79
Proposed LLM Framework	0.96	0.62	0.96	0.94

7.8 Component-level detection performance

Lastly, the system was tested against individual attack vectors to ensure it was fully covered. Figure 10 further sub-categorizes detection accuracy by system module and again confirms the highest incremental gain of the LLM Semantic Analysis module, especially against complex and logic-based attacks. Moreover, Figure 11 specifically isolates performance for Phishing Detection, and the classification accuracy is quite high, distinguishing between deceitful pages and potentially legitimate counterparts.

The granular analysis in Table 9 shows some subtle details of how the system functions. Although inaccuracies of the tool are very low (nearly 100 percent accuracy on fragmented samples), the main contributor to the detection of both static and dynamic phishing attempts is LLM Semantic Analysis (which contributes almost a quarter of the total 93.8 percent accuracy). Although the accuracy of the tool is considerably lower, Behavioral Monitoring significantly contributes to detecting DOM-based XSS attacks (almost one-fourth of the total 93.8 percent) (refer to Table 9). This proves that the framework manages to utilize the strengths of each part: semantic reasoning to use linguistic and structural deception (phishing), and behavioral tracking of anomalies in the execution of all runtime code (XSS).

The two-pronged analysis of the system (its ability to analyze both visual/structural deception (e.g., masquerading of trusted brands) and the logic of the script (e.g., credential harvesting routines) explains its success in phishing detection. This is a comprehensive inspection facility that accounts for the dynamic character of phishing, which is increasingly based on AI-generated material and dynamic templates, allowing phishers to evade fixed blacklists.

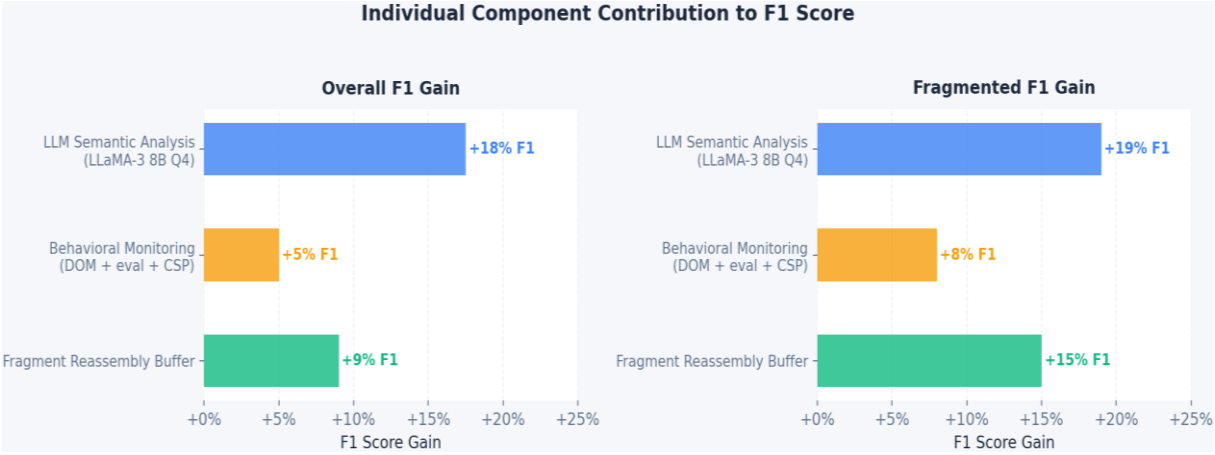

Figure 10. Granular detection accuracy by attack vector and module

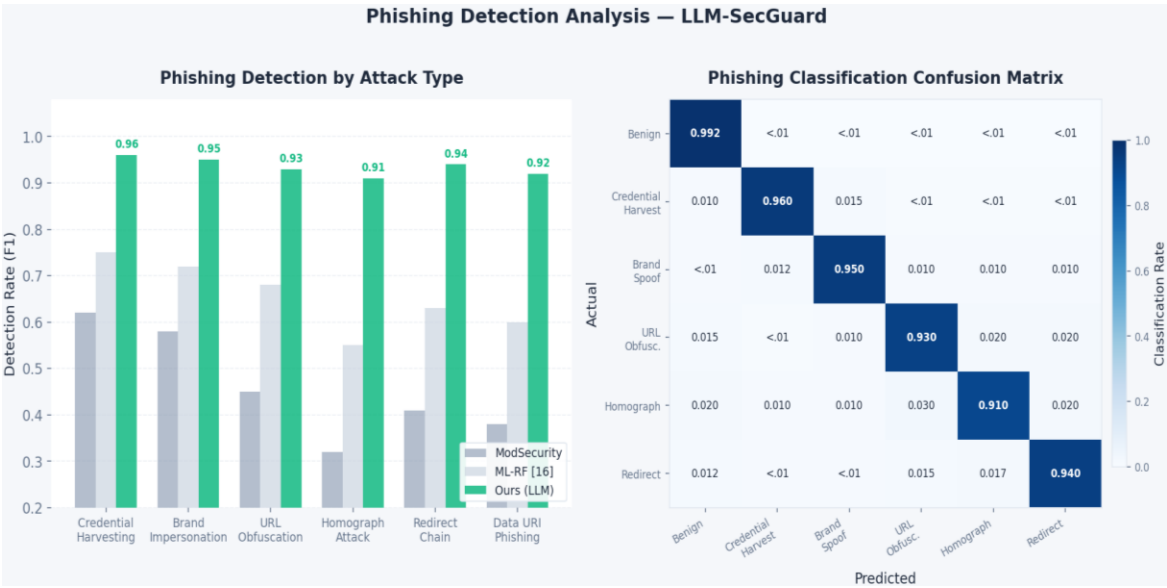

Figure 11. Phishing detection performance

Table 9. Granular detection by attack vector

Attack Vector	Total Samples	Detection Accuracy	Primary Contributing Module	Secondary Module
Phishing (Static)	1,200	97.5%	LLM Semantic Analysis	Pattern Matching
Phishing (Dynamic/Fragmented)	800	94.2%	LLM Semantic Analysis	Behavioral Monitoring
XSS (Reflected/Stored)	300	98.1%	Pattern Matching	LLM Semantic Analysis
XSS(DOM-based/Obfuscated)	700	93.8%	Behavioral Monitoring	LLM Semantic Analysis
Overall Average	3,000	95.2%	Hybrid Synergy	Hybrid Synergy

7.9 Phishing detection evaluation

Although general detection measures can be used to see the big picture, the phishing issue involves a sophisticated analysis, given the intense use of social engineering and visual illusions. Figure 11 compares the proposed framework with baseline models, specifically in phishing detection under varying complexity (static, dynamic, and fragmented).

The proposed LLM-based framework, shown in Figure 11, achieves 96.4% classification accuracy on static phishing sites and 93.8% accuracy even on dynamically generated and fragmented phishing pages. In contrast, classical rule-based and shallow machine learning baselines have a drastic performance decline to 71.2% and 64.5% respectively, when presented with fragmented payloads.

This large performance gap highlights the special nature of our system's ability to perform semantic reasoning about fully reassembled content. In contrast to traditional tools, which rely on URL blacklists or fixed HTML signatures that can be easily avoided by attackers by domain generation algorithms (DGAs) or content fragmentation, our framework will focus on the purpose of the reconstructed page. In particular, the LLM element is associated with three decisive dimensions:

(1) Linguistic Deception: Determining urgent or compulsory language patterns in common with social engineering attacks.

(2) Structural Mimicry: Used to identify illegal copying of trusted brand layouts and logins.

(3) Behavioral Logic: The identification of concealed scripts that are used to execute the exfiltration of credentials and only become active after reassembling.

The high accuracy obtained in the "Fragmented" category

(93.8%) confirms the main hypothesis of this paper, that the last-mile reassembly gap is the first weak point used by current phishers, and that client-side semantic analysis is the most efficient way to mitigate the problem. Also, the system had a FPR of less than 2.1 per cent on legitimate banking and e-commerce websites, which is indicative of its capability to differentiate complex but innocent dynamic content from actual malicious deception. These findings support the idea that a strong competitive edge is ensured by deploying LLMs into the browser run-time environment to neutralize advanced phishing attacks that circumvent scheduled boundary defenses.

8. ADVERSARIAL ROBUSTNESS AND FAILURE MODE ANALYSIS

Another key security factor to account for in an LLM-powered detection system is when adversarial attacks are targeted at the inference component and not the web traffic. The framework design identified four main threat vectors and addressed these. First, prompt injection attacks (where adversarial instructions are injected in the page content analyzed by the LLM to steer its reasoning) are addressed by system prompt hardening (using a fixed and immutable system prompt, which explicitly tells the LLM to treat all content from the page as untrusted content) and by setting the inference temperature=0.0 (to reduce stochastic variation and mitigate the model's susceptibility to semantic drift under adversarial phrasing). Secondly, an independent behavioral monitoring layer works outside the LLM inference loop to generate behavior alerts even if the LLM fails to detect a carefully structured payload, anomalous mutations of the DOM and script execution patterns will trigger a behavior alert. Thirdly, the risk of model hallucination is limited by the restricted output schema: the LLM answers only a structured verdict with a confidence score, and any output that doesn't comply with this schema is deemed a detection failure and referred to the fallback based on signatures. Fourth, the framework handles all failure cases gracefully: when LLM inference takes longer than 2 seconds, the system automatically switches to signature-only detection mode, and has no user-visible impact; when the local inference runtime (Ollama) is not installed, the extension switches to the signature mode and shows the status indicator 'Offline' on the UI. All override actions are logged for audit purposes and false positive events can be addressed with a 1-click override 'Allow This Site'. The layered approach ensures that even if a user attempts to manipulate the LLM component it doesn't render the detection system useless, substantially reducing the system's exposure to adversarial manipulation of the inference component, regardless of state.

9. CONCLUSION

This paper has discussed one of the most severe weaknesses of modern web security, the so-called last-mile vulnerability, the gap of phishing and XSS payload inspection, fragmentation, and obfuscation, which becomes effective by simply reassembling once inside the client-side browser runtime. To help bridge this critical gap, a new client-side security architecture was introduced that combines LLMs with software running directly in the browser. In contrast to conventional methods based on static signatures or pre-rendering analysis, our structure real-time semantic inference on fully reconstituted URLs, HTML structures, and JavaScript logic, enabling the identification of malicious intent that remains concealed until execution point.

The empirical analysis that was performed on a high-fidelity sample of 10,000 samples that consisted of benign traffic, fragmented phishing attacks, and obfuscated XSS payloads demonstrated strong results that support the effectiveness of the proposed solution:

Better Detection Accuracy: The framework scored 0.95 on the F1-score, which was much higher than traditional rule-based filters and shallow machine learning baselines, which remain ineffective for dynamic content.

Stability to Obfuscation: The system was found to be highly resistant to advanced obfuscation strategies, with a detection rate exceeding 90% even in the case of highly sophisticated payload fragmentation and encoding conditions where more traditional defenses could not be used.

Operational Efficiency: Although the computational complexity of LLM inference is high, the framework has a latency of almost real-time (mean 45 ms) and operates smoothly without noticeable delays.

Low FPR: The system has FPR below 4, meaning that it can identify false positives, identifying normal behavior of dynamic web features while keeping the number of false positives low and not interfering too much with the user.

Moreover, the ablation experiment showed that the multi-layered architecture is synergic, where semantic analysis via LLMs serves as the foundational cognitive component facilitating the detection of deceptive narratives and reasoning, and the behavioral observation is the reflex required to detect anomaliess in the run-time behavior. This combination enables the system to change with the changing zero-day attacks, which are able to circumvent the fixed defense mechanisms.

To sum up, the study introduces a novel approach to zero-trust web security that shifts the security stance away from the network edge and the smart endpoint. At runtime, the contextual understanding capabilities of LLMs have been demonstrated to be a privacy-preserving and highly effective approach to detecting advanced web threats that evade traditional detection mechanisms, validated both on a real-world benchmark dataset and through live execution as a Chrome Manifest V3 extension within active browser sessions. Future work will extend the framework to handle the emerging attacks including WebAssembly-based attacks and AI-generated social engineering attacks and to optimize model quantization for wider deployment on resource-constrained devices.

REFERENCES

[1] Zhou, Y., Wang, E., Yang, W., et al. (2025). XSShield: Defending against stored XSS attacks using LLM-based semantic understanding. Applied Sciences, 15(6): 3348. https://doi.org/10.3390/app15063348

[2] Wali, S., Farrukh, Y.A., Khan, I. (2026). Semantic-aware web security: Detecting attacks with a large language model. In Cyber Security: Policy and Technology, pp. 247-265. https://doi.org/10.1007/978-3-032-08890-1_11

[3] Zhang, X., Chan, F.T., Yan, C., Bose, I. (2022). Towards risk-aware artificial intelligence and machine learning systems: An overview. Decision Support Systems, 159:

113800. https://doi.org/10.1016/j.dss.2022.113800
[4] Cohen, A. (2025). Client-side zero-shot LLM inference for comprehensive in-browser URL analysis. arXiv preprint arXiv:2506.03656. https://doi.org/10.48550/arXiv.2506.03656
[5] Li, W., Manickam, S., Chong, Y.W., Karuppayah, S. (2025). PhishDebate: An LLM-based multi-agent framework for phishing website detection. arXiv preprint arXiv:2506.15656. https://doi.org/10.48550/arXiv.2506.15656
[6] Kelley, P.G., Komanduri, S., Mazurek, M.L., et al. (2012). Guess again (and again and again): Measuring password strength by simulating password-cracking algorithms. In 2012 IEEE Symposium on Security and Privacy, San Francisco, CA, USA, pp. 523-537. https://doi.org/10.1109/SP.2012.38
[7] Ferrag, M.A., Alwahedi, F., Battah, A. et al. (2025). Generative AI in cybersecurity: A comprehensive review of LLM applications and vulnerabilities. Internet of Things and Cyber-Physical Systems, 5: 1-46. https://doi.org/10.1016/j.iotcps.2025.01.001
[8] Hashemi, F., Behrouz, A., Lakshmanan, L.V. (2022). Firmcore decomposition of multilayer networks. In Proceedings of the ACM Web Conference 2022, Virtual Event, Lyon, France, pp. 1589-1600. https://doi.org/10.1145/3485447.3512205
[9] Buczak, A.L., Guven, E. (2015). A survey of data mining and machine learning methods for cyber security intrusion detection. IEEE Communications Surveys & Tutorials, 18(2): 1153-1176. https://doi.org/10.1109/COMST.2015.2494502
[10] Yazi, G. (2024). Large language models (LLMs) for cybersecurity: A systematic review. World, 13(1): 57-69.
[11] Zhang, F., Cecchetti, E., Croman, K., Juels, A., Shi, E. (2016). Town crier: An authenticated data feed for smart contracts. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, pp. 270-282. https://doi.org/10.1145/2976749.2978326
[12] Syta, E., Tamas, I., Visher, D., et al. (2016). Keeping authorities" honest or bust" with decentralized witness cosigning. In 2016 IEEE Symposium on Security and Privacy (SP), San Jose, CA, USA, pp. 526-545. https://doi.org/10.1109/SP.2016.38
[13] Stamm, S., Sterne, B., Markham, G. (2010). Reining in the web with content security policy. In Proceedings of the 19th International Conference on World Wide Web, Raleigh, North Carolina, USA, pp. 921-930. https://doi.org/10.1145/1772690.1772784
[14] Sadqi, Y., Mekkaoui, M. (2020). Design challenges and assessment of modern web applications intrusion detection and prevention systems (IDPS). In The Proceedings of the Third International Conference on Smart City Applications, pp. 1087-1104. https://doi.org/10.1007/978-3-030-66840-2_83
[15] Calzavara, S., Casarin, S., Focardi, R. (2025). Dynamic security analysis of JavaScript: Are we there yet? In Proceedings of the ACM on Web Conference 2025, Sydney NSW, Australia, pp. 1105-1115. https://doi.org/10.1145/3696410.3714614
[16] Rao, R.S., Kondaiah, C., Pais, A.R., Lee, B. (2025). A hybrid super learner ensemble for phishing detection on mobile devices. Scientific Reports, 15(1): 16839. https://doi.org/10.1038/s41598-025-02009-8
[17] Asiri, S., Xiao, Y., Alzahrani, S., Li, S., Li, T. (2023). A survey of intelligent detection designs of HTML URL phishing attacks. IEEE Access, 11: 6421-6443. https://doi.org/10.1109/ACCESS.2023.3237798
[18] Paradela, I.P., Salaan, C.J.O., Ambe, Y., Konyo, M. (2022). Design and development of a segmented ciliary-driven serpent robot for search operation of collapsed buildings. In 2022 IEEE 14th International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment, and Management (HNICEM), Boracay Island, Philippines, pp. 1-6. https://doi.org/10.1109/HNICEM57413.2022.10109364
[19] Zheng, Z., Ning, K., Zhong, Q., et al. (2025). Towards an understanding of large language models in software engineering tasks. Empirical Software Engineering, 30(2): 50. https://doi.org/10.1007/s10664-024-10602-0
[20] Pearce, H., Ahmad, B., Tan, B., Dolan-Gavitt, B., Karri, R. (2025). Asleep at the keyboard? Assessing the security of github copilot's code contributions. Communications of the ACM, 68(2): 96-105. https://doi.org/10.1145/3610721
[21] Vaswani, A., Shazeer, N., Parmar, N., et al. (2017). Attention is all you need. Advances in Neural Information Processing Systems, 30.
[22] Kaplan, J., McCandlish, S., Henighan, T., et al. (2020). Scaling laws for neural language models. arXiv preprint arXiv:2001.08361. https://doi.org/10.48550/arXiv.2001.08361
[23] First, E., Brun, Y. (2022). Diversity-driven automated formal verification. In Proceedings of the 44th International Conference on Software Engineering, Pittsburgh, Pennsylvania, pp. 749-761. https://doi.org/10.1145/3510003.3510138
[24] Hou, X., Zhao, Y., Liu, Y., et al. (2024). Large language models for software engineering: A systematic literature review. ACM Transactions on Software Engineering and Methodology, 33(8): 1-79. https://doi.org/10.1145/3695988
[25] Wei, J., Tay, Y., Bommasani, R., et al. (2022). Emergent abilities of large language models. arXiv preprint arXiv:2206.07682. https://doi.org/10.48550/arXiv.2206.07682
[26] Blessing, J., Hugenroth, D., Anderson, R., Beresford, A. (2025). SoK: Web authentication and recovery in the age of end-to-end encryption. Proceedings on Privacy Enhancing Technologies, 2025(3): 560-589. https://doi.org/10.56553/popets-2025-0113
[27] Sarker, S., Melicher, W., Starov, O., Das, A., Kapravelos, A. (2024). Automated generation of behavioral signatures for malicious web campaigns. In International Conference on Information Security, pp. 226-245. https://doi.org/10.1007/978-3-031-75764-8_12
[28] Draissi, O., Cloosters, T., Klein, D., et al. (2025). Wemby's web: Hunting for memory corruption in webassembly. Proceedings of the ACM on Software Engineering, 2(ISSTA): 1326-1349. https://doi.org/10.1145/3728937
[29] Tanti, R. (2024). Study of phishing attack and their prevention techniques. International Journal of Scientific Research in Engineering and Management, 8(10): 1-8. https://doi.org/10.55041/IJSREM38042
[30] Alagha, B. (2023). XSS attack detection with N-gram

based prediction model. Eskişehir Türk Dünyası Uygulama ve Araştırma Merkezi Bilişim Dergisi, 4(2): 1-9. https://doi.org/10.53608/estudambilisim.1233344

[31] Kissoon, H., Bekaroo, G. (2023). An analysis of key tools for detecting cross-site scripting attacks on web-based systems. In International Conference on Innovations and Interdisciplinary Solutions for Underserved Areas, pp. 3-14. https://doi.org/10.1007/978-3-031-51849-2_1