WWW.IJRPR.COM

International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

BROZER: A Local Privacy-Boundary Design for Browser Agents Using On-Device PII Detection, Context Sanitization, and Visual Perception, with an Audit of Its Prototype

Padmanabhan Suresh Babu[1], Mithuneesh K[2], Nishanth KR[3], Nitish RG[4]

[1] Department of Computer Science and Engineering Sri Shakthi Institute of Engineering and Technology Coimbatore 641062, India
padmanabhanonly@gmail.com
[2] Department of Computer Science and Engineering Sri Shakthi Institute of Engineering and Technology Coimbatore 641062, India
mithuneesh.k@gmail.com
[3] Department of Computer Science and Engineering Sri Shakthi Institute of Engineering and Technology Coimbatore 641062, India
nishanthkr775@gmail.com
[4] Department of Computer Science and Engineering Sri Shakthi Institute of Engineering and Technology Coimbatore 641062, India
nitishrg.8220psgps2020@gmail.com

ABSTRACT—

Browser agents that send page screenshots and serialized DOM trees to remote vision-language models can expose passwords, financial records, and personal messages to third parties. We present Brozer, a browser-agent design that moves page perception, sensitive-data classification, sanitization, and action validation onto the client, so that only a sanitized page context and a structured action schema leave the device. The design is specified as a six-stage pipeline with a deterministic policy gate and three security invariants. We audit the prototype code printed in the project report with controlled tests. As printed, its detection path does not classify the fields it reads, its sanitizer matches elements by identifier so that unrelated elements can share redactions, and its action validator accepts malformed selectors. Accuracy and latency figures in the report could not be matched to logs, datasets, or scripts, and they are excluded. The contributions are a design, a threat-model analysis, a code-level defect analysis, and an evaluation plan. The privacy properties of Brozer remain hypotheses until a reproducible evaluation is available.

***Index Terms*—browser agents, privacy, PII detection, context sanitization, trust boundary, on-device perception, prompt injection, policy enforcement**

I. INTRODUCTION

Autonomous browser agents carry out multi-step web tasks from natural-language instructions [1], [2], [3]. A common design is capture-then-reason: the agent sends the screen or the DOM to a remote model, which returns the next action. Recent work has begun to examine the privacy practices of such agents. In an external preprint, Ukani et al. examined eight deployed browser agents and reported 30 vulnerabilities; seven of the eight support only off-device models [4]. These findings come from that study and are not results of Brozer's experiments.

Pages mix navigational controls with confidential data such as credentials, account numbers, and private messages. Sending unredacted context to a third party conflicts with data-minimization principles, for example Article 5(1)(c) of the GDPR [5] and the minimum-necessary standard of the HIPAA Privacy Rule [6]. Page content is also attacker-controlled, and indirect prompt injection can carry instructions into a model that reads it [7], [8], [9].

Few designs make the trust boundary between page perception and remote reasoning an explicit, inspectable component. Brozer takes this approach. Perception, classification, sanitization, and action validation run on the client. The remote model sees only a sanitized context and returns actions in a fixed schema. Brozer builds on the open-source WebBrain browser-extension framework [10]; the licence terms of the inherited code must be confirmed before any public release. The prototype code audited here is the set of code excerpts printed in the authors' project report, which we call "the report" below.

This paper makes three contributions.

1. A design specification: a trust boundary, a six-stage pipeline, a deterministic policy gate, and a structured action schema (Section V).
2. Three security invariants, each with its enforcement point and the conditions under which it can fail (Section V-D).
3. An audit of the prototype code printed in the report, which identifies defects in the detection and sanitization path and in the action validator (Section

IX), together with an evaluation plan (Section X).

We make no claim of zero leakage or of any particular detection accuracy. Unverified figures from the report are not used (Section XII).

II. BACKGROUND AND PROBLEM FORMULATION

Let *W* be a web page with DOM *D* and viewport *S*, and let *I* be a user instruction. The agent must produce actions that accomplish *I* on *W*. The client is trusted: it holds the page, the user's credentials, and the execution environment. The remote reasoner is untrusted. It may be wrong, it may be manipulated by page content, and it observes everything sent to it.

We assume that the client browser, the extension, and local storage are not compromised, that the user can respond to confirmation prompts, and that the reasoner may produce arbitrary output. Under these assumptions we state three objectives.

- **O1 (containment).** Critical secrets (passwords, one-time passwords, card verification values, and authentication tokens) do not leave the client. Personal identifiers are replaced before transmission.
- **O2 (utility).** The sanitized context retains enough structure for the reasoner to choose actions. This objective is not evaluated in this paper.
- **O3 (controlled execution).** Every executed action uses a fixed verb set and passes a local gate.

Brozer's problem is therefore not only detection. It is the placement of the privacy decision: the system must decide what leaves the device before the reasoner sees it, and it must constrain what the reasoner's output can do.

III. RELATED WORK

A. Browser agents and benchmarks. Table I compares the systems reviewed. WebGPT answers questions by browsing a text environment and is trained with human feedback [1]. Mind2Web provides a dataset of real-website tasks for generalist agents [2]. WebArena provides a self-hosted, realistic environment [3]. VisualWebArena extends evaluation to multimodal agents on visual web tasks [11]. WebVoyager builds an end-to-end agent on large multimodal models that operates on live websites [12]. BrowserGym unifies several benchmarks into one evaluation ecosystem [13]. WebLINX studies multi-turn dialogue for website navigation [14]. Gur et al. combine planning, long-context understanding, and HTML pruning in a real-world agent [15]; pruning there is motivated by context length, not privacy. ReAct interleaves reasoning traces with actions [16]. Brozer follows this reason-and-act loop and inserts sanitization and policy stages around it.

B. Privacy of browser agents. Ukani et al. evaluate eight deployed browser agents with a framework of five factors and report 30 vulnerabilities, ranging from disabled browser privacy features to sensitive personal data autofilled into form fields [4]. The study is an arXiv preprint, and its findings are not Brozer results. According to its full text, seven of the eight agents support only off-device models, and browser agents must send browser state and visited-site information to the model. Brozer addresses the same exposure at design time by asking what must leave the device for the reasoner to act.

C. Prompt injection and action control. Greshake et al. show indirect prompt injection against LLM-integrated applications, where retrieved content carries instructions [7]. Liu et al. study prompt injection attacks on such applications [8], and OWASP catalogues application-level risks for LLM systems [9]. AgentDojo provides an environment for evaluating injection attacks and defenses against tool-using agents [29], and CaMeL places a protective layer around the model that separates control flow from untrusted data and tracks capabilities [30]. These works address what an agent may do with untrusted data. Brozer additionally addresses which page data may leave the device. Its structured actions and policy gate limit what a manipulated model can do, but they do not remove instructions contained in sanitized page text (Section XI).

D. Browser isolation and least privilege. FlowFox enforces information-flow control inside a web browser [17]. Saltzer and Schroeder formulate least privilege, which motivates minimizing the data that crosses a trust boundary [18]. Chrome's Manifest V3 prohibits loading and executing remotely hosted code and restricts eval-like execution [19]. These rules constrain an extension, but they do not by themselves establish Brozer's invariants.

E. Contextual anonymization. Lison et al. survey anonymisation models for text and the trade-off between privacy protection and the utility of the text left after masking [20]. Presidio is an open-source toolkit for detecting and anonymizing personal data that combines pattern-based, rule-based, and named-entity recognizers [21]. Brozer's semantic placeholders, such as [EMAIL] and [PHONE], make the same trade-off for page structure.

F. On-device execution. WebAssembly is a portable compilation target for the web designed for efficient execution [22]. ONNX Runtime Web provides a WebAssembly backend and a WebGPU execution provider for in-browser inference [23]. WebGPU is the W3C specification for GPU access from web pages [24]. Brozer relies on these runtimes for local inference. Inference speed on Brozer's models is not measured in this paper.

G. Gap. Prior work characterizes the privacy risk of browser agents [4], attacks on LLM-integrated applications [7], [8], defenses and benchmarks for agent prompt injection [29], [30], and runtimes for local inference [22], [23], [24]. To our knowledge, the cited work does not describe a design that combines local perception, classification, sanitization, structured actions, and deterministic action gating under an explicit trust boundary, nor an audit of an implementation of such a design. Brozer targets that combination as a design. It does not yet claim that the combination has been implemented or measured.

TABLE I. Browser agents and benchmarks reviewed. "Privacy evaluated" indicates whether the cited work evaluates the privacy of agents as a contribution of its own.

System	Venue or status	Environment	Privacy evaluated
WebGPT [1]	arXiv, 2021	Text browsing for QA	No
Mind2Web [2]	NeurIPS 2023 (D&B)	Real websites, DOM	No
WebArena [3]	ICLR 2024	Self-hosted sites	No
VisualWebArena [11]	ACL 2024	Visual tasks, self-hosted	No
WebVoyager [12]	ACL 2024	Live websites, screenshots	No
BrowserGym [13]	arXiv, 2024	Evaluation ecosystem	No (infrastructure)
WebLINX [14]	ICML 2024	Real websites, dialogue	No
AgentDojo [29]	NeurIPS 2024 (D&B)	Tool-using agents, injection tests	Injection, not data exposure
Ukani et al. [4]	arXiv preprint, 2025	Eight deployed agents	Yes: 30 vulnerabilities; 7 of 8 off-device only
Brozer (design)	This paper	Local boundary	Design only; not evaluated

IV. THREAT MODEL AND SECURITY OBJECTIVES

Figure 1 shows the trust boundary, the two permitted flows, and the leak paths that the design must prevent. The adversary controls page content. Page text can contain instructions intended for the reasoner, which is indirect prompt injection [7]. The adversary may also observe traffic to the remote model, so anything sent there is treated as disclosed.

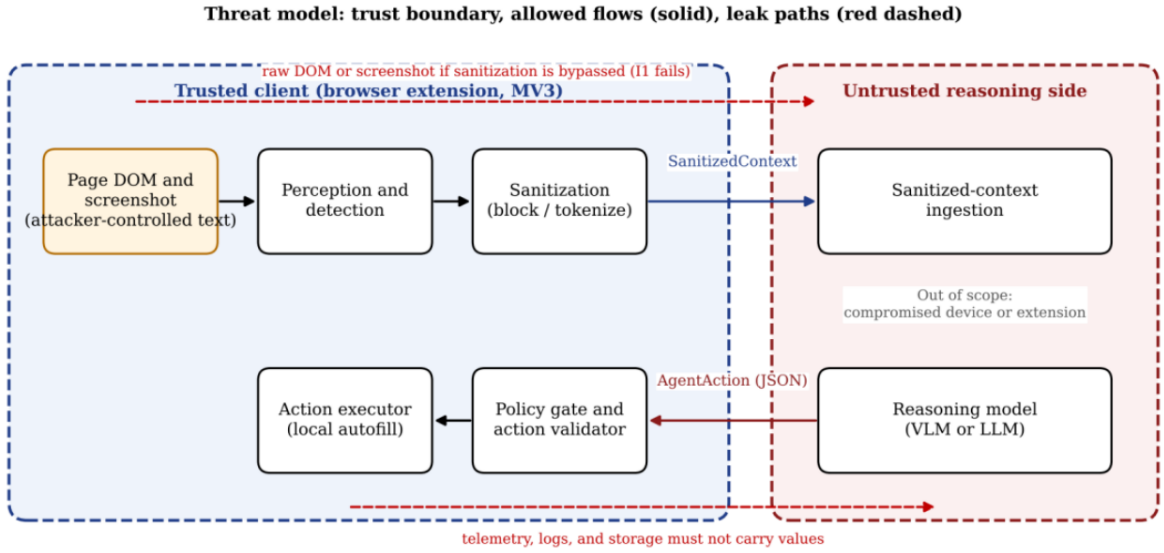


Fig. 1. Threat model. Only SanitizedContext and AgentAction cross the trust boundary. Red dashed paths are the leak routes that invariant I1 must block. A compromised device or extension is out of scope.

Browser authority adds a second risk. The agent acts with the user's session, so a request that is valid for the reasoner may still be harmful for the user. This is the confused-deputy problem, in which a privileged program is induced to misuse its authority [25]. Sanitization does not address this risk. The action validator and the consent step do, and only if they run on every path.

Adjacent threats are distinct. Membership inference and training-data extraction attack models through their training data [26]. They do not describe runtime leakage of the page context that Brozer handles, and we do not treat them as evidence for Brozer's threat model. Out of scope are a compromised device or extension, side channels such as payload size and timing, and adversaries that control the user's consent. The NIST Privacy Framework [27] and the NIST AI Risk Management Framework [28] inform the privacy and risk vocabulary used here. Neither framework is evidence that Brozer complies with them, and Brozer is not certified against either.

V. BROZER SYSTEM ARCHITECTURE

A. Components. Figure 2 shows the components. The trusted local boundary contains a browser extension layer, a local perception engine, a privacy and sanitization engine, a policy gate with an action validator, and an action executor. The remote side contains a sanitized-context ingestion endpoint and a reasoning model that emits structured actions. Exactly two messages cross the boundary: the SanitizedContext outbound and an AgentAction inbound. Credentials stay on the client.

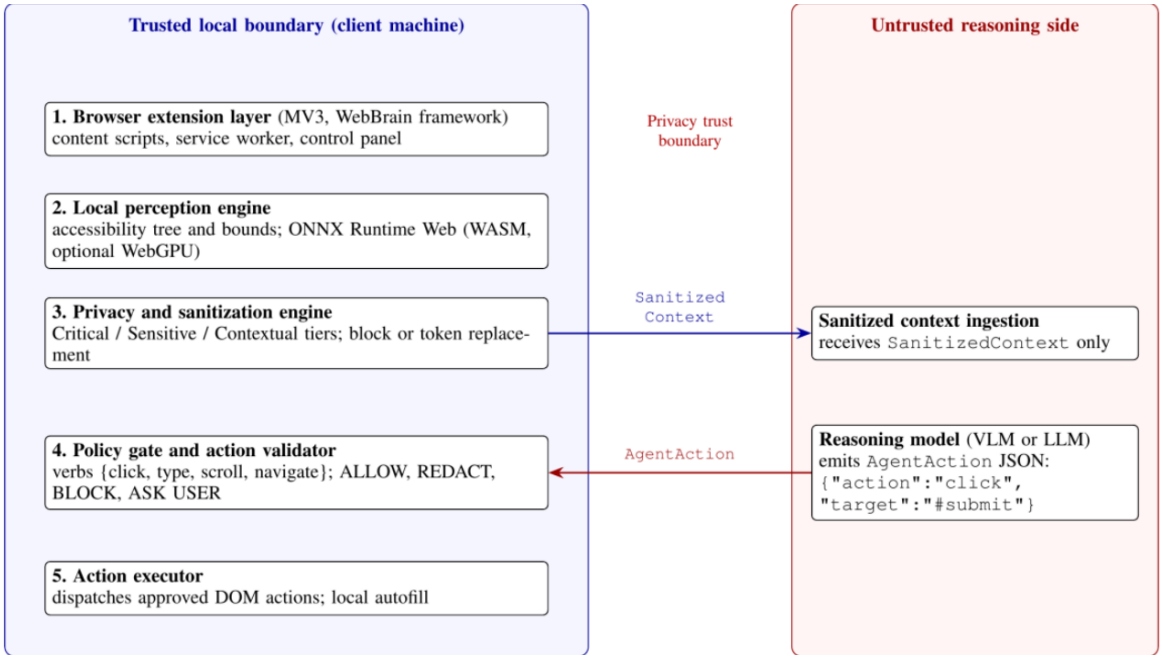


Fig. 2. Brozer architecture as specified in the report. Only SanitizedContext and AgentAction cross the privacy trust boundary (the gap between the two panels). This is the design; the implementation status of each component is given in Table III.

B. Pipeline. Algorithm 1 gives the order of operations. Perception and detection run on the client before any transmission. The reasoner receives only the sanitized context, and its proposed action passes the local gate before execution. The report leaves the REDACT re-check unspecified, so Algorithm 1 does not execute REDACT or BLOCK actions.

ALGORITHM 1. Six-stage Brozer pipeline (design, not implemented as printed).
Require: instruction I; page W with DOM D and viewport S
Ensure: executed action A_exec (or none); audit record R
```
D_tree, B <- Perceive(D, S)     # accessibility tree, bounding boxes
T <- Detect(D_tree)             # tiers: Critical, Sensitive, Contextual
C_san <- Sanitize(D_tree, T)    # block Critical, tokenize Sensitive
A_prop <- Reason(C_san, I)      # remote; structured output
d <- PolicyCheck(A_prop)        # d in {Allow, Redact, Block, AskUser}
if d = Allow then
    A_exec <- Act(A_prop)
else if d = AskUser and the user consents then
    A_exec <- Act(A_prop)
else
    A_exec <- none              # Block, Redact, or no consent
end if
append (d, A_prop, A_exec) to R
```

C. Detection and tiers. The design assigns each element to one of three tiers. Critical elements (passwords, one-time passwords, card verification values, and authentication tokens) are blocked and never transmitted. Sensitive elements (emails, phone numbers, postal addresses, and personal identifiers) are replaced by semantic tokens. Contextual elements (ordinary navigation and operational controls) are retained. Section VI details the detection signals.

D. Security invariants. Let P_{net} be the set of payload items sent across the boundary, S_{crit} the set of critical secret values on the page, A_{exec} the set of executed actions, and PG the policy gate. The design states three invariants.

$$P_{\text{net}} \cap S_{\text{crit}} = \varnothing \tag{I1}$$

$$\forall A \in \mathcal{A}_{\text{exec}} : \text{verb}(A) \in \Sigma \tag{I2}$$

$$\forall A \in \mathcal{A}_{\text{exec}} :\ \text{PG}(A_{\text{prop}}) = \textit{Allow} \ \vee\ (\text{PG}(A_{\text{prop}}) = \textit{AskUser} \ \wedge\ \text{consent} = \text{true}) \tag{I3}$$

Each invariant has an enforcement point: the sanitizer for (I1), the validator for (I2), and the gate for (I3). Each can fail in a specific way. (I1) fails when a critical secret is not classified, or when a secret appears in a node that is not classified at all. (I2) fails when a verb outside Σ reaches the executor, or when the validator does not run on some path. (I3) fails when any execution path bypasses the gate. Section IX tests these failure conditions against the printed code. The invariants are design claims. They are not proofs, and they are not verified end to end.

E. Policy gate and action schema. A proposed action is a triple of verb, target selector, and optional value, with Σ = {click, type, scroll, navigate}. The gate returns one of four outcomes. ALLOW is returned when the action passes all checks and is not on the high-risk registry. BLOCK is returned for an unknown verb, or for a target that does not exist or is hidden. ASK USER is returned for a match against a high-risk registry, such as a payment confirmation, and for a reasoner-reported confidence below a threshold of 0.80, which is a design constant; the report does not specify how that confidence is obtained. REDACT is returned when a proposed value matches a detected sensitive span; because the report leaves the re-check unspecified, this paper treats REDACT as a refusal (Algorithm 1). Local autofill of a password or OTP field injects the secret on the client, and the secret is never sent over the network. The report's text also lists select as a fifth verb, but the printed validator rejects it and Σ above excludes it (defect B3, Section IX).

VI. DOM-BASED PII DETECTION AND CONTEXTUAL CLASSIFICATION

A. Signals. The design combines four signals: DOM attributes (input type, autocomplete tokens, labels), regular-expression matches on text and values, semantic keyword labels, and a visual region classifier. It combines them as a maximum:

$$S_{\text{conf}} = \max(S_{\text{dom}}, S_{\text{reg}}, S_{\text{sem}}, S_{\text{vis}}) \tag{1}$$

The design assigns fixed values of 1.0 to a password-type attribute, 0.95 to a regex match, and 0.85 to a semantic keyword match. These constants are uncalibrated design choices, and no evaluation in this paper supports them. The report does not specify a classification threshold on S_{conf}; this is an open design parameter. A threshold on S_{conf} is not a security guarantee. Security depends on the fail-closed rule in Section IX.

B. Comparison of detection methods. Table II compares the signal types. The comparison is a design analysis and is not measured.

TABLE II. Comparison of DOM, text, and visual detection (design analysis; not measured).

Signal (typical targets)	Strength	Failure mode
DOM attributes: password, OTP, card fields	Precise when present; structured	Missing or misleading attributes; shadow DOM; late-inserted fields
Text regular expressions: email addresses, card-like numbers	Simple for structured identifiers	False positives (for example, 13–16 digit runs); no context
NER or semantic models: names, postal addresses	Finds names and addresses in free text	Model errors; coverage gaps across languages
Visual (screenshot and local model): content rendered without DOM text	Sees rendered text without DOM text	Cost and model size; not in the printed code

C. Sensitivity of the signals to the page. A regular expression for email addresses misses unlabelled sensitive fields. The printed card pattern also matches digit runs that are not card numbers. Credentials and OTPs have no fixed format, so their detection depends on attribute or label signals. This is why the design uses several signals and why the printed code's reliance on regular expressions for those fields is a limitation (Section IX).

D. Evaluation definitions. For a category *c* and a labelled test set, we use the standard definitions below. They are definitions for the evaluation plan (Section X), not results.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{2}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{3}$$

$$F_1 = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{4}$$

VII. CONTEXT SANITIZATION AND DATA MINIMIZATION

Sanitization replaces each detected span or node with either a block or a typed token, according to its tier. Critical nodes are blocked. Sensitive spans are tokenized, for example as [EMAIL] or [PHONE]. Contextual nodes are kept. Minimization applies before sanitization: the collector should gather only the fields the next decision requires, and it should record categories and counts in the audit log rather than values.

Two sanitization properties follow from the design, and both are stated as requirements, not measured outcomes. First, the sanitized context must not contain a critical value for any node, including nodes that the detector did not classify. The printed code fails this requirement for unclassified text (defect T5, Section IX). Second, redactions must attach to the intended element. The printed sanitizer attaches redactions by identifier, which can fail when identifiers are empty or shared (defect T4).

The trade-off is between privacy and utility. Tokens keep structure, but over-redaction can remove cues the reasoner needs. Lison et al. describe the same trade-off for free text [20]. The right balance for Brozer is an empirical question that this paper does not answer.

VIII. ON-DEVICE VISUAL PERCEPTION AND MULTIMODAL PRIVACY

The design uses a lightweight visual model, run locally through ONNX Runtime Web on WebAssembly with optional WebGPU acceleration [23], [24]. Its role is to classify regions of the screenshot when DOM signals are missing, and its outputs would enter the same classifier as the DOM and text signals.

The printed code does not include this model, and the report does not demonstrate it, so the visual claims remain design claims. Visual perception raises its own risks. OCR can extract secrets that the DOM does not expose, and a visual classifier can miss content, so a visual detector is not a substitute for a fail-closed rule. Screenshots sent to a remote model would also defeat the design, so the visual model must run on the client and its image data must not be transmitted.

The report includes two annotated screenshots of one sign-in flow on a retail website (amazon.in), showing the extension's interface and placeholder tokens. They come from a single run, have no ground truth, and do not show the transmitted payload. They are illustrations, not measurements, and are not used as evidence of completeness.

IX. POLICY ENFORCEMENT, IMPLEMENTATION STATUS, AND TECHNICAL AUDIT

The policy gate must evaluate each proposed action before execution. The design specifies four outcomes: ALLOW, REDACT, BLOCK, and ASK USER. For a deterministic gate, the same action and the same page state must always produce the same outcome. The printed code does not include the gate, the high-risk registry, or the local autofill, so their behavior is not verified.

Detection and sanitization. The printed extractor keeps only tag, identifier, and bounds, while the detector reads type, autocomplete, inner text, and value; on the test page nothing is detected (T1). When those fields are supplied, the detector classifies correctly: a password is CRITICAL and an email is SENSITIVE (T2, a passing control). The defect is therefore in the data path, not the classifier. In addition, the global email regular expression alternates between true and false on repeated tests (T3), the sanitizer labels two unrelated elements without identifiers as [EMAIL] (T4), and text in unclassified nodes, such as "Your OTP is 123456", passes unchanged (T5). Test IDs follow the numbering of the verification scripts. Table IV lists the tests that exposed a defect.

Action validation. The printed validator checks the JSON shape and the verb. It strips quote and angle-bracket characters from the target (B2), rejects select although the report's text lists it (B3), and accepts a malformed selector (B6). It has no check that the selector exists on the page. The design places that check in the gate, which is absent. Rewriting a selector silently is a risk in its own right: the executed element can differ from the element the reasoner chose.

Consent is part of enforcement. A user who sees frequent prompts may habituate and approve without reading. The design therefore restricts ASK USER to a registry of high-risk actions, but the registry's coverage is not established.

Fail-closed behavior is a requirement. If detection, sanitization, or validation errors, the action must not proceed. Algorithm 2 states this as a proposed design. It is not implemented.

ALGORITHM 2. Proposed fail-closed pipeline (proposed design, not implemented).

```
function Step(page, instruction):
    nodes <- CollectMinimal(page)            # only fields the next step needs
    spans <- Detect(nodes)                   # DOM and text (visual if needed)
    for n in nodes:
        if n.key is missing or n not in spans.classified:
            n.tier <- CRITICAL               # fail closed for unclassified text
    ctx <- Sanitize(nodes, tier  policy)     # block Critical, tokenize Sensitive
    action <- Reason(ctx, instruction)       # remote; structured output only
    if not ValidateSchema(action): return REJECT
    if not ExistsInDOM(action.target): return REJECT
    d <- PolicyGate(action, nodes)           # ALLOW | REDACT | BLOCK | ASK  USER
    if d = ALLOW: return Execute(action)
    if d = ASK  USER and UserConsents(action): return Execute(action)
    return Refuse(d)
```

TABLE III. Design versus printed implementation. Status reflects controlled tests and code review of the printed excerpts only.

Component	Design	Printed code	Status
Perception	Accessibility tree and bounds	Keeps tag, id, and bounds only	Partial
Visual perception	Lightweight ONNX/WASM localization	Not included	Unverified
Detection	Four signals, three tiers	Reads fields the extractor does not supply (T1)	Defective
Sanitization	Block or tokenize by element	Matches by id; collisions (T4); unmatched text passes (T5)	Defective
Reasoning service	Constrained JSON output	Express route and Ollama call; not executed	Shown
Action validator	Verb and target checks	Verb and value checks; no DOM check (B2, B3, B6)	Partial
Policy gate	Four outcomes and rules	Not included	Unverified
Local autofill	Secret injected locally	Not included	Unverified
Extension UI	Live panel and badges	Screenshots only	Qualitative

TABLE IV. Defects found in the printed code, consequences, and proposed remedies. Evidence is from controlled tests of the printed excerpts (Node.js v22.23.4). Remedies are proposed and not yet applied.

ID	Defect (evidence)	Consequence	Proposed remedy
T1	Extractor keeps only tag, id, and bounds; detector reads type, autocomplete, innerText, value. No detections on the test page.	Sensitive fields are transmitted; I1 fails.	Extend the extractor to pass the fields the detector reads.
T3	Global email regex alternates true, false, true on repeated tests.	Intermittent misses.	Remove the global flag or reset state before each test.
T4	Two elements without identifiers; one sensitive; both labelled [EMAIL].	Unrelated content is redacted (over-redaction).	Assign unique, stable node keys.
T5	Text in unclassified nodes (“Your OTP is 123456”) passes unchanged.	I1 is not enforced for such text.	Fail closed on unclassified nodes; add an OTP signal.
B2	Selector quotes and angle brackets are removed silently.	The executed target can differ from the proposed one.	Parse and reject, or document the rewrite.
B3	Verb select is rejected, but the report’s text lists it as a verb.	Design and code disagree.	Decide which is correct; update the other.
B6	Malformed selector div:::broken(((is accepted.	Invalid targets reach the executor.	Syntax check, plus an existence check in the gate.
—	Policy gate, visual model, and local autofill absent from the printed code.	I3 cannot be tested; visual and autofill claims unsupported.	Implement and test as new work.

X. EXPERIMENTAL METHODOLOGY AND EVALUATION FRAMEWORK

A. Status. No evaluation of Brozer has been performed. The results in Section IX are controlled tests of printed excerpts, and they test defects, not accuracy. The framework below is a proposal. It specifies what would be measured and reports no outcomes.

B. Test environment. Use synthetic sign-in, checkout, and profile pages with known sensitive fields. Include unlabelled fields, dynamically inserted fields, shadow DOM, cross-origin iframes, multilingual labels, obfuscated values, and adversarial page text. Use synthetic identities only. Candidate task sets, including Mind2Web, WebLINX, and VisualWebArena, must have their licences and suitability for sensitive fields checked before use.

C. Metrics. Detection quality uses precision, recall, and F1 (Equations 2 to 4), reported per category and per signal type, with confidence intervals on proportions. Leakage is defined as follows. Let S_{crit} be the critical values on a test page, and let $L(t)$ be the set of critical values that appear, in any form, in the output of trial t at one of the monitored locations: model request, tool arguments, browser logs, action output, screenshots and OCR output, persistent storage, or telemetry. The leakage rate over N trials is

$$\text{Leak} = \frac{1}{N} \sum_{t=1}^{N} \frac{|L(t)|}{|S_{\text{crit}}|} \tag{5}$$

Task success over *M* attempts is the number of attempts completing the goal divided by *M*. Utility also records unnecessary blocking, user interventions, and failure reasons. Latency is reported end to end and per stage (collection, detection, visual inference, sanitization, reasoning, validation, execution), with hardware, browser version, model size, and median, mean, variability, and 95th percentile.

D. Baselines. The planned comparison has six conditions: an unprotected agent; a DOM-only detector; a text-only sanitizer; a DOM-plus-visual configuration, if implemented; Brozer without the policy gate; and Brozer with the policy gate.

E. Robustness. Planned robustness tests cover PII in unlabelled elements, dynamically inserted fields, obfuscated values, multilingual content, adversarial webpages, prompt injection, malformed actions, ambiguous fields, and detector failure.

F. Statistical reporting. Each condition is repeated over multiple runs and pages, and the number of runs is stated before any result is shown. Failure cases are reported in full. No significance claim is made without a stated test.

G. What can be claimed after each stage. Before measurement, the paper can claim only a design and a defect analysis. After leakage tests on the synthetic set, it can report leak rates on that set, with intervals. A zero-leakage claim requires a stated threat model and a test set where no leak is observed, together with a bound on the failure rate. Even then the claim is empirical, not a guarantee.

XI. SECURITY ANALYSIS AND DISCUSSION

Brozer changes where the privacy decision is made. In the agents studied by Ukani et al., seven of the eight support only off-device models, so page context reaches a remote model unless it is filtered earlier [4]. Brozer decides before transmission, which is the only point where minimization can apply to data the reasoner would otherwise receive. This is a design argument. The amount of data the design keeps off the device is not measured here.

The design does not solve injection. Sanitization removes the secrets and identifiers that the detector finds. It does not remove instructions written in visible page text, so a manipulated reasoner can still propose any allowed action on a valid target. The high-risk registry and the ASK USER step limit the damage, but the registry's coverage is not established, and the printed code contains neither the registry nor the gate. Indirect injection [7], [8] is therefore a residual risk, not an addressed one.

Platform rules help but do not establish invariants. Manifest V3 prevents remotely hosted code and restricts eval-like execution [19]. These rules narrow the attack surface. The invariants depend on the sanitizer, the validator, and the gate, and none of those is verified end to end.

Residual leakage. Invariant I1 covers critical secrets only. Non-secret page content, page structure, and request metadata such as timing and size still leave the device. These are not addressed by the design.

XII. LIMITATIONS, ETHICS, AND REPRODUCIBILITY

A. Limitations. The threats to validity are as follows.

1. **Audited material.** The audit covers the code excerpts printed in the report. The full repository, datasets, evaluation scripts, and logs were not part of the audited material, so the released code may differ.
2. **No quantitative evaluation.** Accuracy, latency, and leakage claims cannot be supported, and the report's figures are excluded. They are also inconsistent with each other: task completion is 94.6% in one place and 94.2% in another; PII recall is 97.2% and 98.7%; PII precision is 98.4% and 99.4%; latency is given as 138 ms, under 150 ms, under 180 ms, and under 200 ms; visual content accuracy is 97.8% without documented methodology; and a zero-leakage claim has no supporting test. None of these values is selected or reported as a result.
3. **Defects in the printed code** (Table IV) undermine I1 and I2 as printed.
4. **Unverified components.** The visual model, policy gate, and local autofill are absent from the printed code.
5. **Injection** through sanitized text reaches the reasoner (Section XI).
6. **Anecdotal visual evidence.** The screenshots come from one run on one site and carry the authors' annotations.
7. **Upstream dependency.** Brozer builds on the WebBrain framework [10]; the licence of any inherited code must be confirmed before release.
8. **Scope.** A compromised device or extension and side channels are not analyzed.

B. Ethics and responsible data handling. The evaluation plan uses synthetic identities and synthetic pages only. No real personal data is needed or should be collected. Privacy-event logs should record categories and decisions, never values. The paper does not claim GDPR or HIPAA compliance. Principles such as data minimization [5], [6] motivate the design, and whether a deployment complies is a separate question that depends on the deployment.

C. Reproducibility. The verification artifacts are the transcribed code excerpts, the test scripts, and their outputs. They are available from the corresponding author on request. The repository, datasets, scripts, and logs behind the report's own figures are not available, so no experiment in this paper can be reproduced from that material. Reproducing the defect analysis requires only Node.js and the transcribed excerpts. Reproducing the evaluation requires the components that do not yet exist.

XIII. FUTURE WORK

Future work should proceed in this order. First, release the code, the datasets, and the logs. Second, fix the detection data path, the identifier handling, the unclassified-text path, and the validator defects, with a regression test for each defect in Table IV. Third, implement the policy gate and test invariant I3, including the selector check. Fourth, measure detection precision, recall, and F1 per tier on a documented benchmark with labelled ground truth, reporting confidence intervals. Fifth, evaluate resistance to injected instructions on adversarial pages. Sixth, measure leakage and latency on documented hardware.

XIV. CONCLUSION

We specified Brozer as a local privacy boundary for browser agents: a six-stage pipeline with a deterministic policy gate and three security invariants. An audit of the prototype code printed in the report found that its detection path, as printed, classifies nothing on the test input, and that its sanitizer can attach redactions to unrelated elements. The report's accuracy and latency figures could not be verified, and this paper excludes them. The contribution is therefore a design, a threat-model analysis, a code-level defect analysis, and an evaluation plan. Brozer's privacy properties remain hypotheses until they are measured.

ACKNOWLEDGMENT

The authors thank their project supervisor, Mr. B. Sanjay Krishna, Assistant Professor, Department of Computer Science and Engineering, for technical guidance during the project.

REFERENCES

[1] R. Nakano, J. Hilton, S. Balaji, et al., "WebGPT: Browser-assisted question-answering with human feedback," arXiv:2112.09332, 2021.

[2] X. Deng, Y. Gu, B. Zheng, et al., "Mind2Web: Towards a generalist agent for the web," in *Advances in Neural Information Processing Systems 36 (NeurIPS 2023), Datasets and Benchmarks Track*, 2023.

[3] S. Zhou, F. F. Xu, H. Zhu, et al., "WebArena: A realistic web environment for building autonomous agents," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2024.

[4] A. Ukani, H. Haddadi, A. S. Shamsabadi, and P. Snyder, "Privacy practices of browser agents," arXiv:2512.07725 [cs.CR], Dec. 2025.

[5] European Parliament and Council of the European Union, "Regulation (EU) 2016/679 (General Data Protection Regulation)," *Official Journal of the European Union*, L 119, 2016.

[6] U.S. Department of Health and Human Services, "HIPAA Privacy Rule," 45 C.F.R. Parts 160 and 164, Subparts A and E. [Online]. Available: https://www.hhs.gov/hipaa/for-professionals/privacy/index.html

[7] K. Greshake, S. Abdelnabi, S. Mishra, C. Endres, T. Holz, and M. Fritz, "Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection," in *Proc. 16th ACM Workshop Artif. Intell. Secur. (AISec)*, 2023, pp. 79–90.

[8] Y. Liu, G. Deng, Y. Li, et al., "Prompt injection attack against LLM-integrated applications," arXiv:2306.05499, 2023.

[9] OWASP Foundation, "OWASP Top 10 for large language model applications." [Online]. Available: https://owasp.org/www-project-top-10-for-large-language-model-applications/

[10] E. Sokullu, "WebBrain," Mozilla Add-ons listing. [Online]. Available: https://addons.mozilla.org/en-GB/firefox/addon/webbrain/

[11] J. Y. Koh, R. Lo, L. Jang, et al., "VisualWebArena: Evaluating multimodal agents on realistic visual web tasks," in *Proc. 62nd Annu. Meeting Assoc. Comput. Linguistics (Vol. 1: Long Papers)*, 2024, pp. 881–905, doi: 10.18653/v1/2024.acl-long.50.

[12] H. He, W. Yao, K. Ma, et al., "WebVoyager: Building an end-to-end web agent with large multimodal models," in *Proc. 62nd Annu. Meeting Assoc. Comput. Linguistics (Vol. 1: Long Papers)*, 2024, pp. 6864–6890, doi: 10.18653/v1/2024.acl-long.371.

[13] T. Le Sellier De Chezelles, M. Gasse, A. Drouin, et al., "The BrowserGym ecosystem for web agent research," arXiv:2412.05467, 2024.

[14] X. H. Lù, Z. Kasner, and S. Reddy, "WebLINX: Real-world website navigation with multi-turn dialogue," in *Proc. 41st Int. Conf. Mach. Learn. (ICML)*, 2024; arXiv:2402.05930.

[15] I. Gur, H. Furuta, A. Huang, et al., "A real-world WebAgent with planning, long context understanding, and program synthesis," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2024; arXiv:2307.12856.

[16] S. Yao, J. Zhao, D. Yu, et al., "ReAct: Synergizing reasoning and acting in language models," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2023.

[17] W. De Groef, D. Devriese, N. Nikiforakis, and F. Piessens, "FlowFox: A web browser with flexible and precise information flow control," in *Proc. ACM Conf. Comput. Commun. Secur. (CCS)*, 2012.

[18] J. H. Saltzer and M. D. Schroeder, "The protection of information in computer systems," *Proc. IEEE*, vol. 63, no. 9, pp. 1278–1308, 1975.

[19] Google Chrome Developers, "Improving security in Manifest V3," Chrome for Developers documentation. [Online]. Available: https://developer.chrome.com/docs/extensions/develop/migrate/improve-security

[20] P. Lison, I. Pilán, D. Sánchez, M. Batet, and L. Øvrelid, "Anonymisation models for text data: State of the art, challenges and future directions," in *Proc. 59th Annu. Meeting Assoc. Comput. Linguistics and 11th Int. Joint Conf. Natural Lang. Process. (Vol. 1: Long Papers)*, 2021, pp. 4188–4203.

[21] Microsoft, "Presidio: Data protection and de-identification SDK," documentation. [Online]. Available: https://microsoft.github.io/presidio

[22] A. Haas, A. Rossberg, D. L. Schuff, et al., "Bringing the web up to speed with WebAssembly," in *Proc. 38th ACM SIGPLAN Conf. Program. Lang. Design Implement. (PLDI)*, 2017.

[23] ONNX Runtime Developers, "Using the WebGPU execution provider, ONNX Runtime Web," ONNX Runtime documentation. [Online]. Available: https://onnxruntime.ai/docs/tutorials/web/ep-webgpu.html

[24] World Wide Web Consortium (W3C), "WebGPU," W3C technical report. [Online]. Available: https://www.w3.org/TR/webgpu/

[25] N. Hardy, "The confused deputy (or why capabilities might have been invented)," *ACM SIGOPS Oper. Syst. Rev.*, vol. 22, no. 4, pp. 36–38, Oct. 1988, doi: 10.1145/54289.871709.

[26] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," in *Proc. IEEE Symp. Secur. Privacy (SP)*, 2017, pp. 3–18; arXiv:1610.05820.

[27] K. Boeckl and N. Lefkovitz, "NIST Privacy Framework: A tool for improving privacy through enterprise risk management, Version 1.0," NIST CSWP 01162020, Jan. 2020, doi: 10.6028/NIST.CSWP.01162020.

[28] E. Tabassi, "Artificial Intelligence Risk Management Framework (AI RMF 1.0)," NIST AI 100-1, Jan. 2023.

[29] E. Debenedetti, J. Zhang, M. Balunović, L. Beurer-Kellner, M. Fischer, and F. Tramèr, "AgentDojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents," in *Advances in Neural Information Processing Systems 37 (NeurIPS 2024), Datasets and Benchmarks Track*, 2024; arXiv:2406.13352.

[30] E. Debenedetti, I. Shumailov, T. Fan, et al., "Defeating prompt injections by design," arXiv:2503.18813, 2025.