MIND THE BOUNDARY: DATA LEAKAGE IN NATIVE-STORE LLM BROWSER EXTENSIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

LLM browser extensions legitimately send some user content to remote models, making outbound traffic alone an inadequate privacy signal. We present a four-browser measurement study that first identifies LLM-related candidates, then uses package analysis to characterize their permissions and model-facing data surfaces, and finally executes downloadable extensions with synthetic markers to observe first-hop transmission. Among 6874 successfully analyzed candidate packages, The risks identified through static analysis, such as unnecessary browser permissions and AI-related content collection are widespread. Runtime evidence is substantially narrower: most detected transmissions carry explicit user input, while a small subset also carries context outside our predeclared profile-and-action boundary. A controlled context experiment further shows that richer and attacker-influenced pages can induce additional out-of-bound transmissions. Our results have led to the separation of static risks and risks under actual operation, and have revealed the proportion and correlation between the two in the real world.

1 INTRODUCTION

Browser extensions sit in a privileged position between websites, browsers, and users. They can read page content, inject scripts, follow navigation, bridge messages between web origins and extension pages, and attach themselves to workflows that span multiple tabs and multiple sites. That privileged position has long made extensions an attractive but risky software layer. The rise of LLM-enabled extensions(The following text is collectively referred to as LLM extension) changes the problem.

The act of "sending user data externally" seems quite reasonable within the LLM extension. A translator, summarizer, or AI writing assistant may need to send some content to a remote provider to perform exactly what the user asked it to do. As a result, neither “there is a model endpoint in code” nor “there is outbound traffic to a cloud host” is enough to establish misuse.

However, LLM extensions blur the boundary between explicit user input and ambient browser context. A selection translator should ordinarily need only the selected text. A page summarizer may reasonably need the current page body. A new-tab AI search entry point should mostly need the user query and its own local new-tab state. By contrast, hidden DOM fragments, unrelated form fields, notes, tokens, browsing history, cookies, or cross-tab state are usually beyond what those features require. The real audit question is therefore not “does the extension use AI?” but “does it bring more browser context into the AI path than the declared task justifies?”

Some model-facing transmission is legitimate; The issue we are addressing is whether the data sent out by the LLM when it receives a user’s request only includes the necessary parts to fulfill the request, or whether it also carries additional page information (which might involve user privacy). We start by constructing a candidate set from the browser extension store. Then, we use deterministic rules based on the source code (instead of an LLM classifier) to classify the functions of each LLM extension. Different categories of LLM extensions have different permitted outgoing content and additional page information that exceeds the requirements. Finally, we distinguishes requests that carry expected task markers from requests that also carry disallowed contextual markers, and a separate context experiment tests whether that distinction remains stable on richer or attacker-influenced pages. Our experiment covered the four most popular browsers(Chrome, Firefox, Edge and Opera).

This paper is organized around the following three questions:

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

RQ1: *What static analysis-based risks were exposed by the LLM extension of the native store?*

RQ2: *In a real-world operating scenario, will the LLM expansion send out unnecessary additional page data?*

RQ3: *How do more diverse context environments and page contents affect the runtime transmission and boundary stability of the original storage LLM extension?*

We confirm real runtime transmission to third parties. We also find that most observed transmissions remain aligned with explicit user-input paths such as selected text, editor text, or queries. Boundary-violating behavior exists, but it is concentrated in a much smaller subset. That is exactly why a legitimate-boundary framework is necessary: it lets us avoid labeling every LLM call as risky while still isolating the extensions that go beyond the task they advertise.

Our contributions. We provide (1) a documented collection pipeline for four native stores, with explicit distinctions between metadata candidates and package-confirmed LLM evidence; (2) a deterministic static analysis whose rules, weights, and severity thresholds are exposed rather than described as a learned model; (3) a profile-conditioned dynamic harness that separates routing from payload-boundary outcomes; and (4) a controlled robustness experiment that measures how page complexity and attacker-supplied text affect those outcomes while explicitly limiting claims about real-site prevalence.

2 PRELIMINARIES

2.1 LLM EXTENSIONS IN BROWSER ECOSYSTEMS

LLM extensions have emerged as a distinct layer in browser ecosystems: rather than exposing a standalone chat interface, they embed model-backed assistance directly into browsing surfaces such as selected text, editable documents, PDFs, new-tab pages, side panels, and site-specific workflows. This placement gives them a practical advantage, because an extension can react to local page state and user interaction without requiring users to manually copy content into a separate application, but it also grants access to permissions and contexts that ordinary web applications do not naturally obtain. Across native extension stores, this produces a heterogeneous yet structurally similar ecosystem of translators, summarizers, writing assistants, search copilots, code helpers, OCR tools, and automation agents, all of which mediate between browser-visible context and a model-facing backend.

This makes LLM extensions analytically awkward. They cannot be treated like overtly malicious exfiltration tools, but neither can they be treated like ordinary cloud clients whose inputs are always explicit and well-scoped. The audit problem therefore centers on the legitimate input boundary: what the feature plausibly needs, what extra browser context it can access, and what kind of remote routing structure it exposes.

2.2 RELATED WORK

The closest direct overlap with our application domain is the recent study of browser-based GenAI assistants by Vekaria et al. Vekaria et al. (2025), which shows that such assistants can transmit browsing context and participate in profiling or personalization. Recent work on browser agents and agent frameworks further shows that safety properties can change once an LLM is embedded in an interactive browser environment Kumar et al. (2025); Ruan et al. (2024); Wang et al. (2026); Jiang et al. (2026). These results help motivate our focus on native-store LLM extensions as a distinct application layer: unlike conventional browser extensions, they often have a legitimate reason to send some user-selected or user-entered content to remote backends, yet they also sit close enough to the page to accumulate broader contextual state.

Foundational work on browser extensions showed that declarative permission systems still leave room for serious extension-side vulnerabilities Carlini et al. (2012); Bandhakavi et al. (2010). Later studies expanded this line toward malicious-extension operations, resource-control failures, subtle history leakage, and contextual privacy flows Jagpal et al. (2015); Sanchez-Rola et al. (2017); Weissbacher et al. (2017); Fass et al. (2021); Yu et al. (2023); Ahmadpanah et al. (2025). Dynamic measurement work then showed that privacy risk cannot be understood from manifests alone: ex-

Table 1: Dataset and measurement coverage. Runtime columns use extensions with usable traces.

Browser	Candidates	Static	Runtime	Remote+	Boundary+
Chrome	4754	3955	3439	35	4
Firefox	2221	2210	1895	39	2
Edge	700	697	647	5	0
Opera	13	12	12	0	0
Total	7688	6874	5993	79	6

tensions can hide behavior behind user-triggered interactions, diffuse tracking state, or leak page content in ways storefront metadata obscures Kapravelos et al. (2014); Starov & Nikiforakis (2017); Chen & Kapravelos (2018); Xie et al. (2024); Solomos et al. (2022); Picazo-Sanchez et al. (2019); Toreini et al. (2019); Perrotta & Hao (2018); Bui et al. (2023). Ecosystem-scale governance studies further showed that over-collection, weak vetting, fake reviews, and cross-store inconsistencies remain common at store scale Hsu et al. (2024); Ling et al. (2024); Chaudhry et al. (2026); Moreno et al. (2026); Olsson et al. (2024).

The static literature largely targets general extension vulnerabilities or contextual privacy flows; the dynamic literature emphasizes behavioral elicitation, taint tracking, or disclosure inconsistencies; and the GenAI-assistant literature studies a much smaller application slice than the four native stores we examine here. Our paper combines those threads around one narrower question: how much browser context an LLM extension is structurally able to couple into a model-facing path, and whether that coupling stays inside the legitimate boundary implied by the extension's function.

3 Methodology

3.1 Dataset construction

The four stores require different public acquisition paths. Chrome combines 132,356 unique manifests from a 2025-01-10 snapshot with 33 current-store queries; Firefox combines 84,235 AMO snapshot records with AMO search; Edge combines its locale-specific default catalog with Microsoft-curated AI collections and query results; Opera uses a full public-directory enumeration plus search. Records are deduplicated by store identifier. Because snapshots are point-in-time, searches are bounded, and the Edge index is locale-specific, this is a recall-oriented candidate set rather than a census of every installable extension.

Extension selection is deterministic and does not invoke an LLM. Weighted regular expressions over names, descriptions, categories, URLs, and tags assign higher weight to concrete provider/model evidence and explicit LLM descriptions than to generic terms such as *assistant* or *summarizer*; records scoring at least five are retained. Package analysis subsequently separates metadata candidates from stronger client-visible evidence: 4,157 of 6,874 scored packages contain a known model/API endpoint, and 5,151 contain either an endpoint or an LLM-specific code shape.

3.2 Profiles and Legitimate-Input Boundaries

Each candidate receives one of eight operational profiles: general copilot, automation/agent, new-tab/search, site-specific assistant, developer/code, page/document summarizer, vision/OCR, or selection/editor. An ordered regular-expression classifier uses storefront and manifest fields; unmatched records fall back to general copilot. This deterministic primary-profile assignment makes the downstream boundary reproducible, but it can simplify multifunction extensions.

Following least privilege and data minimization OWASP Foundation (n.d.a;n), each profile defines a task proxy(that is, a certain type of work). Explicit user input and context intrinsically required by the advertised action are allowed; unrelated hidden DOM, form fields, credentials, cross-site state, and cross-task context are disallowed by default. Thus, selected text is legitimate for selection translation, current-page text is legitimate for page summarization, and code in an active editor is legitimate for a code assistant. The same content may be out of bound for a new-tab query tool. The

metadata-versus-manifest agreement and cross-browser Jensen–Shannon analyses in Appendix A test robustness to evidence availability and browser ecology.

3.3 Static Rules and Runtime Labels

The static analyzer extracts manifest authority, execution primitives, browser-data access, model endpoints, relay/provider structure, and code-level couplings between browser context and model-facing requests. Forty-three named rules are grouped into generic extension risks, AI data-surface indicators, and profile-boundary indicators. Each receives a 0–10 triage weight based on ambient authority, directness of sensitive-context coupling, and boundary exceedance. A verified endpoint receives zero because it is evidence of functionality rather than misuse. The additive score prioritizes review; it is not a calibrated probability. Appendix E gives every rule, weight, and count.

The dynamic harness installs downloadable packages in isolated browser profiles and performs profile-oriented actions: page clicks and selections, prompt-like input, and, when available, popup, options, or side-panel interaction. Synthetic sentinels distinguish selected text, query, editor/code input, page context, hidden text, form fields, password fields, credentials, and *attack-bait*. Attack-bait is visible attacker-authored page text urging an assistant to inspect or transmit extra page regions; it tests whether surrounding content can redirect the extension beyond the user's action.

We capture client-visible first-hop requests. An event is *remote-positive* when a sentinel reaches a non-local destination, *task-consistent* when all transmitted sentinels are allowed by the profile/action template, and *profile-boundary-positive* when at least one is disallowed. Destination and content are independent axes: a call to a disclosed provider can still carry out-of-bound content, while an opaque relay can carry only legitimate input.

For context robustness, an enriched baseline-responsive cohort is replayed across article, editor, code, business, search, and dashboard pages at low, medium, and high complexity. Low pages contain task-essential surfaces; medium pages add realistic clutter; high pages add hidden/contextual fields and attack-bait. Newly induced positives compare the same extension/page-family pair and require usable traces at both baseline and target conditions. The design supports causal context comparisons within this cohort, not prevalence claims about the public Web.

4 Experiments and Results

4.1 RQ1: Static Indicators and AI Data Surfaces

Of 7,688 candidates, 6,874 packages are successfully scored. Static exposure is widespread: 2,898 packages (42.16%) enter the high/critical triage buckets and 2,508 (36.49%) carry profile-boundary indicators. Dynamic HTML/code execution appears in 77.41% of packages, insecure backend transport in 62.16%, tab-context capability in 61.11%, page-content-to-LLM coupling in 53.64%, and form-input-to-LLM coupling in 41.17%. These observations describe latent design surfaces, not executed leakage.

General copilots account for 2,913 candidates (42.37%) and automation/agent tools for 1,322 (19.23%). The remaining 2,639 candidates span new-tab/search, site-specific, developer/code, summarization, vision/OCR, and selection/editor workflows. This long tail matters because narrower advertised functions provide stronger legitimate-input constraints: a selection assistant offers a clearer basis for identifying unexpected page capture than a broadly described copilot.

The distribution is uneven. Edge has the highest high/critical concentration (56.53%), followed by Opera (50.00%, but only 12 packages), Chrome (43.84%), and Firefox (34.57%). New-tab/search and selection/editor profiles repeatedly combine narrow advertised inputs with broad contextual authority: their high/critical rates reach 75.36% and 84.31% on Edge and 60.17% and 70.18% on Chrome. Relay/provider groupings are descriptive rather than causal; for example, OpenRouter relays co-occur with an 81.88% high/critical rate, but broader permissions and feature complexity may explain the association.

The ablation quantifies why conventional extension checks are insufficient. Generic rules identify 1,303 high/critical packages; AI-facing surface rules add 1,189, and profile-boundary rules add another 406. The OWASP-mapped baseline remains necessary, but ignoring model-facing

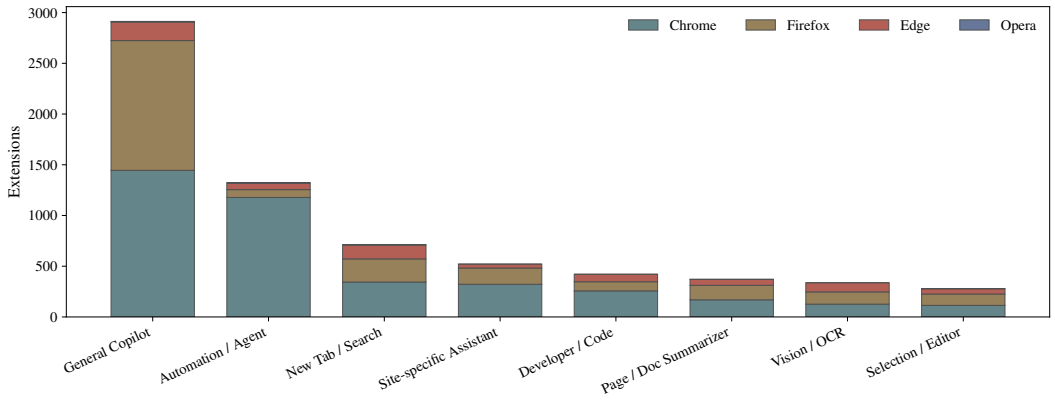

Figure 1: Functional-profile distribution across the four native stores. Broad copilots and automation assistants dominate, but narrower profiles form a substantial long tail.

Table 2: Static-score ablation over 6,874 successfully scored packages.

Score variant	High/Critical	Rate	Added
Generic only	1303	18.96%	—
Generic + AI surface	2492	36.25%	1189
Full profile-aware	2898	42.16%	406

collection and task-relative overreach materially understates the review surface. Complete signal, provider/relay, publication-time, and popularity results appear in Appendix B.

4.2 RQ2: Realized Transmission and Boundary Outcomes

Across 5,993 usable baseline traces, 84 extensions generate 330 sentinel-bearing events and 79 generate 321 remote events. Most observed traffic is legitimate under the task proxy: 314 remote events are task-consistent, whereas seven events from six extensions contain at least one disallowed sentinel. Extension-level labels overlap because an extension can first produce a task-consistent event and later a boundary-positive event. Therefore, neither the 84 observed-positive nor the 79 remote-positive extensions should be interpreted as leakage cases.

The six confirmed extensions transmit hidden text, notes, page context, code context, or editor text beyond the assigned profile/action boundary. Most remote positives instead carry explicit user input: 71 of 79 remote-positive extensions hit the explicit-input family, 11 hit a non-user-input family, and six satisfy the stricter boundary rule. This distinction is central: a summarizer can legitimately send current-page content, while the same page-context sentinel is anomalous for a selection-only translator.

Interaction richness determines whether these behaviors are observed. Page-only stimulation recovers 23.81% of the baseline-positive cohort. Opening extension-local UI raises observed-positive recall to 76.19%; full interaction with a shorter observation window also reaches 76.19%, indicating that trigger coverage matters more than simply waiting longer. Boundary-positive recall rises from 50.00% for page-only and page+UI configurations to 83.33% under full interaction with a short window.

Static scoring remains useful for queueing but cannot replace execution. Forty-eight of 77 remote-other-positive extensions fall in high/critical buckets, yet 2,600 high/critical packages remain remote-other negative under the harness; conversely, 26 remote-other positives are not high/critical. These gaps reflect both static over-approximation and dynamic under-triggering. Browser-level rates, confirmed cases, and additional runtime analysis are reported in Appendices B and C.

Table 3: Most prevalent static signals. The endpoint signal is functionality evidence and receives no risk score.

Signal	Type	Score	Rate	Interpretation
dynamic_html_or_code_ execution	Generic	9	77.41%	Dynamic HTML or code execution broadens the extension attack surface.
insecure_transport_to_ backend	Generic	8	62.16%	Non-local plaintext HTTP/WS communication exposes backend traffic.
tab_context_ permission	Generic	5	61.11%	Tab read or script-injection capability expands ambient visibility.
llm_api_endpoint_in_ code	Evidence	0	60.47%	Confirms model-facing functionality without implying misuse.
page_content_to_llm_ surface	AI	6	53.64%	Couples extracted page content to model-facing flows.
form_input_to_llm_ surface	AI	8	41.17%	Couples form/editor input surfaces to model-facing requests.
broad_message_ bridge	Generic	6	36.06%	Enlarges the page-to-extension input channel.
broad_web_accessible_ resources	Generic	6	33.33%	Increases extension-resource reachability from page contexts.

Table 4: RQ2 runtime outcomes over 5,993 usable traces.

Label	Extensions	Events	Rate
Observed-positive	84	330	1.40%
Remote-positive	79	321	1.32%
Task-consistent	75	314	1.25%
Profile-boundary-positive	6	7	0.10%

Browser-level outcomes further expose coverage differences. Firefox has the highest observed-positive rate (2.06%), followed by Edge (1.24%) and Chrome (1.08%); profile-boundary-positive rates are only 0.11% for Firefox and 0.12% for Chrome. Chrome and Firefox also have double-digit runtime-error rates, so zero observed transmission cannot be equated with successful negative evidence.

4.3 RQ3: Context Complexity and Attacker Influence

RQ3 asks whether the same extension/action remains inside its boundary as surrounding page context changes. Low-complexity boundary-positive rates are 0–1.3% across page families, and medium pages produce modest changes. High-complexity conditions produce a consistent increase: business reaches 6.8%, article and search 5.5%, editor and code 4.9%, and the Firefox-only dashboard condition 2.6%. Attack-bait positives closely track these increases, showing that visible attacker-authored instructions can enter model-facing requests even when the user did not select them.

Condition-level rates can still reflect differences in which extensions run successfully. We therefore compare each extension/page-family pair with its own usable low-complexity baseline. High complexity newly induces boundary positives in 6.21% of business pairs, 5.00% of search pairs, and 4.35–4.97% of article, editor, and code pairs. Attack-bait outcomes follow the same pattern, indicating that the effect is not merely repetition of baseline behavior.

Sensitivity varies by function. High-complexity boundary-positive rates reach 25.00% for page/document summarizers, 14.58% for automation/agent extensions, and 8.62% for vision/OCR,

Table 5: Confirmed baseline profile-boundary-positive extensions.

Extension	First hop	Disallowed labels
Sapling Grammar Checker	api.sapling.ai	notes
Summarize: TLDR AI Buddy	aitextsummary.com	code, editor, hidden, page
Article Summarizer (Gemini)	AWS API Gateway	code, editor, page
Concentration	product server	code, editor, hidden, page
Pair Translate	Microsoft Translator	hidden text
DualView Translator	Google Translate	page context

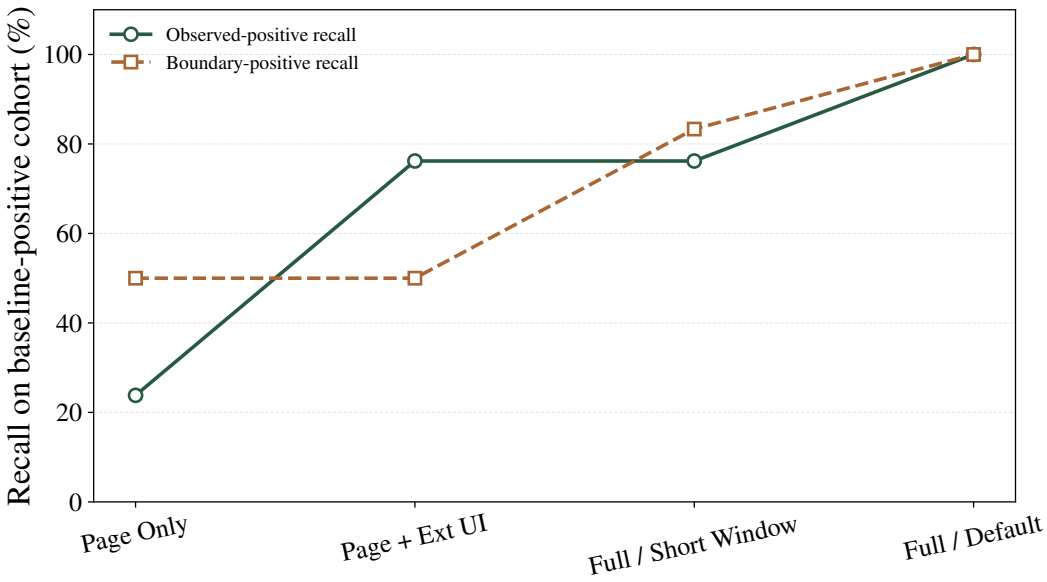

Figure 2: Interaction-richness and observation-window ablation. Extension-local UI and full task execution are necessary to recover runtime-positive behavior.

while new-tab/search and site-specific assistants remain more stable. Representative cases show translation, summarization, and OCR requests combining legitimate selected text with attack-bait, hidden text, or broader page context. The first-hop destination is often an expected provider; the risk is the additional content hitchhiking on the legitimate call. Appendix D reports auditable examples.

5 Discussion and Limitations

The findings support a staged audit. Profile and task proxies prevent every model call from being mislabeled as leakage; static rules identify packages that combine narrow legitimate inputs with broad contextual access; dynamic execution then separates latent capability from realized transmission. Extension-local UI is a first-class trigger surface, and clean-page testing is insufficient because richer or attacker-influenced context can induce new boundary outcomes. Store operators should therefore prioritize extensions whose advertised task, browser authority, routing transparency, and realized payload diverge.

This layering also clarifies how the measurements should be operationalized. Store-scale static analysis is inexpensive enough to construct a review queue, but the score should not be exposed to users as a probability of leakage. Dynamic confirmation should instead replay the advertised task, preserve explicit-input markers as legitimate, and inspect only additional sentinel families as candidate overreach. When the extension relies on a popup, side panel, or options page, those surfaces must be exercised before a negative result is considered informative.

The context experiment adds a second validation stage for extensions that remain ambiguous after clean-page execution. A page can influence an extension without exploiting browser code: visible

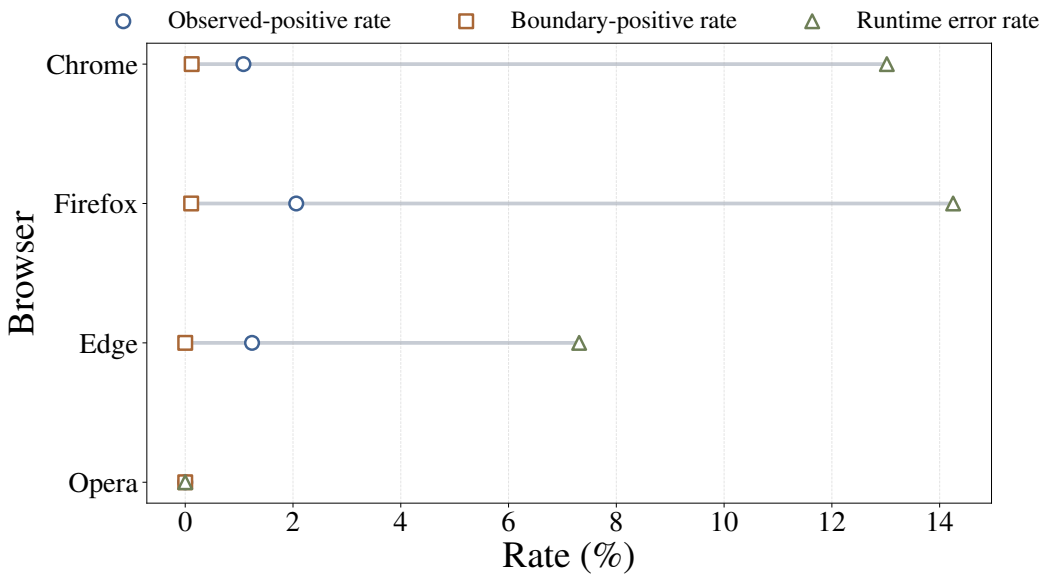

Figure 3: Browser-level runtime outcomes. Rates use usable traces; error rates expose automation-coverage limitations.

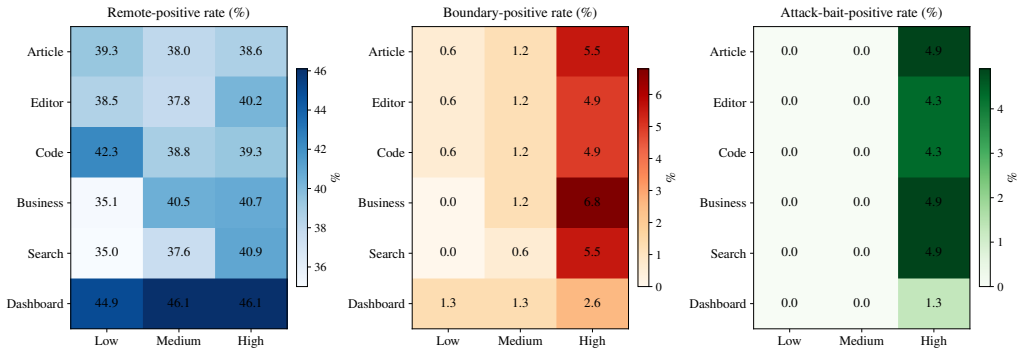

Figure 4: RQ3 outcomes across page families and context levels. Boundary and attack-bait positives rise in high-complexity contexts; rates describe the enriched controlled cohort, not Web prevalence.

instructions, hidden DOM, and semantically related form fields may simply be collected by over-broad context assembly. This is especially relevant for summarizers and agents whose advertised functionality already permits some page reading. Within-extension low-to-high comparisons separate such induced drift from baseline behavior and from differences in which packages happen to execute successfully.

The claims have three principal limits. First, heterogeneous store interfaces, metadata heuristics, and one-primary-profile assignment introduce coverage and construct error; the corpus is neither a complete store census nor human-validated intent ground truth. Second, static indicators and additive scores denote triage priority, not exploitation probability. Third, dynamic tests observe only client-visible first hops under scripted actions. Login, paid features, prolonged use, dormant triggers, backend retention, and final model routing remain outside scope, while execution failures bias usable traces. RQ3's synthetic pages isolate context effects safely, but its enriched cohort and Firefox-only dashboard condition cannot estimate real-Web prevalence.

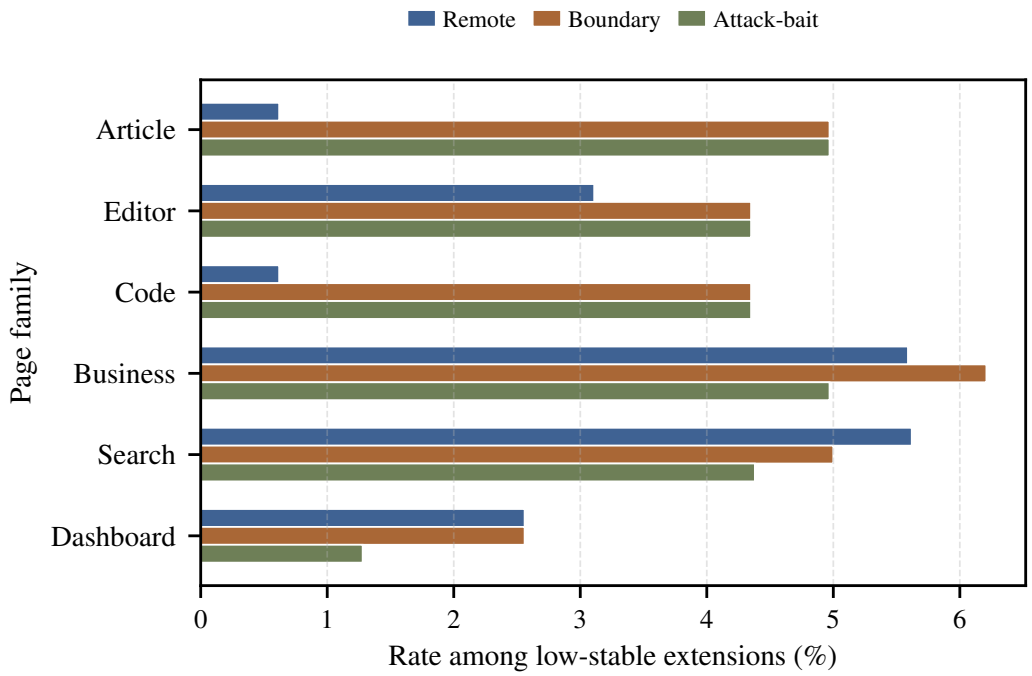

Figure 5: Within-extension positives newly induced by high complexity relative to the same page family's usable low baseline.

6 Conclusion

Across four stores, we identify 7,688 candidates, statically score 6,874 packages, and obtain 5,993 usable baseline traces. Static context-coupling indicators are widespread, but most observed model-facing transmissions carry explicit task input; six extensions produce seven confirmed profile-boundary-positive events. Controlled high-complexity and attacker-influenced pages induce additional boundary outcomes in the responsive cohort. Privacy risk is therefore best measured as the gap between what a feature asks the user to provide, what its package can couple into the AI path, and what actually leaves the client.

Ethical Considerations

Stakeholders and potential harms. The directly affected stakeholders are extension users and developers, browser-store operators, LLM and relay providers, and researchers who may reuse the dataset or harness. Users could benefit from clearer distinctions between task-necessary model calls and broader contextual transmission. Potential harms include accidental collection of sensitive user data during testing, disruption of extension or provider services, reputational harm from false-positive findings, and misuse of the harness to probe extensions at scale.

Risk minimization. All dynamic tasks use unique synthetic markers on locally served pages. We do not use real credentials, private user content, production accounts, paid features, or data obtained from non-consenting users. The harness does not submit transactions or attempt to bypass authentication, store controls, anti-abuse mechanisms, or provider access restrictions. Tests use bounded interactions and observation windows rather than load generation. Store metadata and packages are obtained through publicly accessible interfaces; analysis is limited to client-visible code and first-hop traffic.

Interpretation and reporting. We separate static indicators from observed behavior, treat provider endpoints as functionality evidence rather than proof of wrongdoing, and condition payload findings on a documented profile-and-action proxy. We do not infer developer intent, backend retention,

or final model routing from a first-hop request. Named examples are included only to make measured payload classes auditable; conclusions concern observed executions under our harness and not maliciousness. Aggregate results are used wherever individual identification is unnecessary.

Benefits, residual risk, and oversight. The anticipated benefit is an auditable method for identifying extensions that deserve focused privacy review without labeling every legitimate LLM request as leakage. Residual risks remain from measurement error, automation bias, and downstream misuse of released artifacts. These are mitigated by publishing rule definitions, confidence limits, non-prevalence caveats, and synthetic rather than personal traces. The study does not recruit human subjects or intervene in user accounts; nevertheless, the authors remain responsible for following applicable institutional requirements and for reassessing disclosure needs before releasing extension-level evidence.

AI USE STATEMENT

In this work, generative AI tools are used for language editing and improving the readability of papers, providing feedback on some experimental processes and results. All AI-based content and code are manually reviewed and verified by the author. The author assumes full responsibility for the content, statements, results and achievements presented in this paper.

REFERENCES

Mohammad M. Ahmadpanah, Matías F. Gobbi, Daniel Hedin, Johannes Kinder, and Andrei Sabelfeld. CodeX: Contextual flow tracking for browser extensions. In *Proceedings of the Fifteenth ACM Conference on Data and Application Security and Privacy*. ACM, 2025. doi: 10.1145/3714393.3726495.

Sruthi Bandhakavi, Samuel T. King, P. Madhusudan, and Marianne Winslett. VEX: Vetting browser extensions for security vulnerabilities. In *19th USENIX Security Symposium (USENIX Security 10)*. USENIX Association, 2010.

Duc Bui, Brian Tang, and Kang G. Shin. Detection of inconsistencies in privacy practices of browser extensions. In *2023 IEEE Symposium on Security and Privacy*. IEEE, 2023. doi: 10.1109/SP46215.2023.10179338.

Nicholas Carlini, Adrienne Porter Felt, and David Wagner. An evaluation of the Google Chrome extension security architecture. In *21st USENIX Security Symposium (USENIX Security 12)*. USENIX Association, 2012.

Abdullah Hassan Chaudhry, Valentino Dalla Valle, and Aurore Fass. Two heads are better than one: Analysing browser extensions across stores. In *Network and Distributed System Security Symposium (NDSS)*, 2026.

Quan Chen and Alexandros Kapravelos. Mystique: Uncovering information leakage from browser extensions. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1687–1700. ACM, 2018. doi: 10.1145/3243734.3243823.

Aurore Fass, Dolière Francis Somé, Michael Backes, and Ben Stock. DoubleX: Statically detecting vulnerable data flows in browser extensions at scale. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2021.

Sheryl Hsu, Manda Tran, and Aurore Fass. What is in the Chrome Web Store? In *Proceedings of the 19th ACM Asia Conference on Computer and Communications Security*. ACM, 2024.

Nav Jagpal, Eric Dingle, Jean-Philippe Gravel, Panayiotis Mavrommatis, Niels Provos, Moheeb Abu Rajab, and Kurt Thomas. Trends and lessons from three years fighting malicious extensions. In *24th USENIX Security Symposium (USENIX Security 15)*. USENIX Association, 2015.

Yukun Jiang, Yage Zhang, Xinyue Shen, Michael Backes, and Yang Zhang. "Humans Welcome to Observe": A first look at the agent social network Moltbook. *arXiv preprint arXiv:2602.10127*, 2026. URL https://arxiv.org/abs/2602.10127.

Alexandros Kapravelos, Chris Grier, Neha Chachra, Christopher Kruegel, Giovanni Vigna, and Vern Paxson. Hulk: Eliciting malicious behavior in browser extensions. In *23rd USENIX Security Symposium (USENIX Security 14)*. USENIX Association, 2014.

Priyanshu Kumar, Elaine Lau, Saranya Vijayakumar, Tu Trinh, Elaine T. Chang, Vaughn Robinson, Shuyan Zhou, Matt Fredrikson, Sean M. Hendryx, Summer Yue, and Zifan Wang. Aligned LLMs are not aligned browser agents. In *International Conference on Learning Representations (ICLR)*, 2025.

Yuxi Ling, Yun Hao, Yuyan Wang, Kailong Wang, Guangdong Bai, and Jin Song Dong. Essential or excessive? MINDAEXT: Measuring data minimization practices among browser extensions. In *2024 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pp. 964–975. IEEE, 2024. doi: 10.1109/SANER60148.2024.00104.

Jose Miguel Moreno, Narseo Vallina-Rodriguez, and Juan Tapiador. Did i vet you before? assessing the Chrome Web Store vetting process through browser extension similarity. *IEEE Transactions on Services Computing*, 2026.

Eric Olsson, Benjamin Eriksson, Pablo Picazo-Sanchez, Lukas Andersson, and Andrei Sabelfeld. FakeX: A framework for detecting fake reviews of browser extensions. In *Proceedings of the 19th ACM Asia Conference on Computer and Communications Security*. ACM, 2024.

OWASP Foundation. OWASP browser extension vulnerabilities cheat sheet. OWASP Cheat Sheet Series, n.d.a. URL `https://cheatsheetseries.owasp.org/cheatsheets/Browser_Extension_Vulnerabilities_Cheat_Sheet.html`. Accessed September 22, 2026.

OWASP Foundation. OWASP user privacy protection cheat sheet. OWASP Cheat Sheet Series, n.d.b. URL `https://cheatsheetseries.owasp.org/cheatsheets/User_Privacy_Protection_Cheat_Sheet.html`. Accessed September 22, 2026.

Raffaello Perrotta and Feng Hao. Botnet in the browser: Understanding threats caused by malicious browser extensions. *IEEE Security & Privacy*, 2018.

Pablo Picazo-Sanchez, Juan Tapiador, and Gerardo Schneider. After you, please: Browser extensions order attacks and countermeasures. *International Journal of Information Security*, 2019.

Yangjun Ruan, Honghua Dong, Andrew Wang, Silviu Pitis, Yongchao Zhou, Jimmy Ba, Yann Dubois, Chris J. Maddison, and Tatsunori Hashimoto. ToolEmu: Identifying the risks of LM agents with an LM-emulated sandbox. In *International Conference on Learning Representations (ICLR)*, 2024.

Iskander Sanchez-Rola, Davide Balzarotti, and Igor Santos. Extension breakdown: Security analysis of browsers extension resources control policies. In *26th USENIX Security Symposium (USENIX Security 17)*. USENIX Association, 2017.

Konstantinos Solomos, Panagiotis Ilia, Soroush Karami, Nick Nikiforakis, and Jason Polakis. The dangers of human touch: Fingerprinting browser extensions through user actions. In *31st USENIX Security Symposium (USENIX Security 22)*, pp. 717–733. USENIX Association, 2022.

Oleksii Starov and Nick Nikiforakis. Extended tracking powers: Measuring the privacy diffusion enabled by browser extensions. In *Proceedings of the 26th International Conference on World Wide Web*, 2017.

Ehsan Toreini, Siamak F. Shahandashti, and Feng Hao. DOMtegrity: Ensuring web page integrity against malicious browser extensions. *International Journal of Information Security*, 2019.

Yash Vekaria, Aurelio Loris Canino, Jonathan Levitsky, Alex Ciechonski, Patricia Callejo, Anna Maria Mandalari, and Zubair Shafiq. Big help or big brother? auditing tracking, profiling, and personalization in generative AI assistants. In *34th USENIX Security Symposium (USENIX Security 25)*, pp. 8115–8134. USENIX Association, 2025.

Yuhang Wang, Haichang Gao, Zhenxing Niu, Zhaoxiang Liu, Wenjing Zhang, Xiang Wang, and Shiguo Lian. A systematic security evaluation of OpenClaw and its variants. *arXiv preprint arXiv:2604.03131*, 2026. URL https://arxiv.org/abs/2604.03131.

Michael Weissbacher, Enrico Mariconti, Guillermo Suarez-Tangil, Gianluca Stringhini, William Robertson, and Engin Kirda. Ex-ray: Detection of history-leaking browser extensions. In *Annual Computer Security Applications Conference (ACSAC)*, 2017.

Qinge Xie, Manoj Vignesh Kasi Murali, Paul Pearce, and Frank Li. Arcanum: Detecting and evaluating the privacy risks of browser extensions on web pages and web content. In *33rd USENIX Security Symposium (USENIX Security 24)*, pp. 4607–4624. USENIX Association, 2024.

Jianjia Yu, Song Li, Junmin Zhu, and Yinzhi Cao. CoCo: Efficient browser extension vulnerability detection via coverage-guided, concurrent abstract interpretation. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2023. doi: 10.1145/3576915.3616584.

A RQ1 Taxonomy Robustness Ablations

A.1 Metadata Sufficiency

Table 6 reports the first **RQ1** robustness check in detail. The goal is to test whether the type assignment depends on package access that a storefront-only auditor would not yet have. It largely does not. Metadata-only and metadata+manifest inference agree on 99.62% of Chrome samples and 99.59% of Firefox samples, while the current Edge and Opera sets show perfect agreement. Only 18 Chrome samples and 9 Firefox samples change profile once manifest or static-code hints are added.

This robustness check matters because the paper's taxonomy is meant to support early triage rather than only deep reverse engineering. A practical store-scale reviewer often sees public descriptions, screenshots, and permission hints well before unpacking the package itself. If the profile assignment changed frequently once manifests or static code became available, then the entire boundary-first framing would be much harder to use as a first-pass screen. The data instead show that the public-facing identity of these extensions is usually stable enough to support profile-aware screening before full package access.

Table 6: RQ1 metadata-only versus metadata+manifest profile agreement.

Browser	Samples	Agreement	Changed profiles
Chrome	4754	99.62%	18
Firefox	2221	99.59%	9
Edge	700	100.00%	0
Opera	13	100.00%	0

This matters because **RQ1** is intentionally ecosystem-scale. If the taxonomy required privileged package access just to distinguish, for example, a selection assistant from a new-tab AI search tool, then a large portion of the static audit would become unavailable to early triage. The high agreement shows that most extensions already disclose enough functional identity in public storefront artifacts to support boundary-aware screening before full unpacking.

A.2 Cross-Browser Stability

Table 7 reports the second **RQ1** robustness check and asks whether the type system is merely a Chrome-centric projection. Among the three large ecosystems, the answer is no. Jensen–Shannon divergence between profile distributions is 0.1032 for Chrome versus Edge, 0.1122 for Chrome versus Firefox, and 0.0906 for Edge versus Firefox. These values are low enough to support the paper's cross-browser scope claim: the taxonomy captures a common ecology of LLM-extension roles rather than a store-specific artifact. Opera is more distinct, which is expected given its tiny native-store population.

This stability result is important for two reasons. First, it suggests that the profiles are not an artifact of one storefront's ranking logic or one browser's extension APIs. Second, it justifies applying the same legitimate-input-boundary idea across multiple native stores without redefining the taxonomy from scratch for each ecosystem. We do not claim the stores are identical; the point is narrower and more practical: the same high-level functional roles recur often enough that a single boundary vocabulary remains analytically useful.

Table 7: RQ1 cross-browser Jensen–Shannon divergence for profile distributions.

Browser pair	JS divergence
Chrome vs. Edge	0.1032
Chrome vs. Firefox	0.1122
Edge vs. Firefox	0.0906

Table 8: Static score ablation over all successfully scored packages.

Score Variant	Scored	High / Critical	Rate	Added vs. Previous
Generic only	6874	1303	18.96%	—
Generic + AI surface	6874	2492	36.25%	1189
Full profile-aware	6874	2898	42.16%	406

Table 9: Provider combinations and relay structures with sufficient support. "Boundary" is the profile-boundary-indicator rate, not runtime leakage.

Type	Observable set or relay	n	High	Boundary
Provider set	Anthropic + Gemini + OpenAI	58	75.86%	34.48%
Provider set	OpenRouter	33	72.73%	24.24%
Provider set	Gemini + OpenAI	41	65.85%	36.59%
Provider set	Anthropic + OpenAI + xAI/Grok	182	64.84%	42.31%
Provider set	Ollama	26	61.54%	30.77%
Provider set	OpenAI + xAI/Grok	28	46.43%	39.29%
Relay	SiliconFlow	20	100.00%	70.00%
Relay	OpenRouter + SiliconFlow	11	100.00%	81.82%
Relay	OpenRouter	138	81.88%	42.03%
Relay	None observed	6672	40.84%	36.08%

B Supplementary RQ1 and RQ2 Results

B.1 Provider, Relay, Time, and Popularity Results

The supplementary views separate three questions that are easy to conflate in the aggregate score. The ablation measures how many packages move into a high-priority bucket when AI-facing and profile-conditioned rules are added. The provider table instead groups packages by client-visible backend composition, while the cohort plot relates static priority to publication year and store popularity. None of these comparisons is a runtime leakage estimate, and the provider rows are shown only when their support passes the stated minimum.

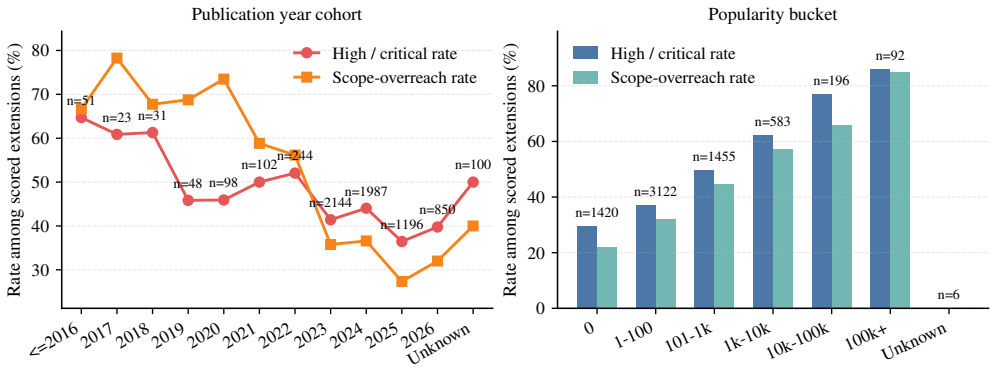

Figure 6: Publication-time and popularity relationships with static priority. Publication year is non-monotonic, while popularity has a stronger positive gradient in high/critical and profile-boundary-indicator rates.

B.2 Confirmed Profile-Boundary-Positive Cases

The six cases below retain the identifiers needed to audit the payload claim; profile-level interpretation remains in the main text. They are not a random sample and should not be used to estimate prevalence. We include every baseline case that satisfied the strict profile-and-action rule so that readers can distinguish a surprising destination from a payload that actually contains a disallowed

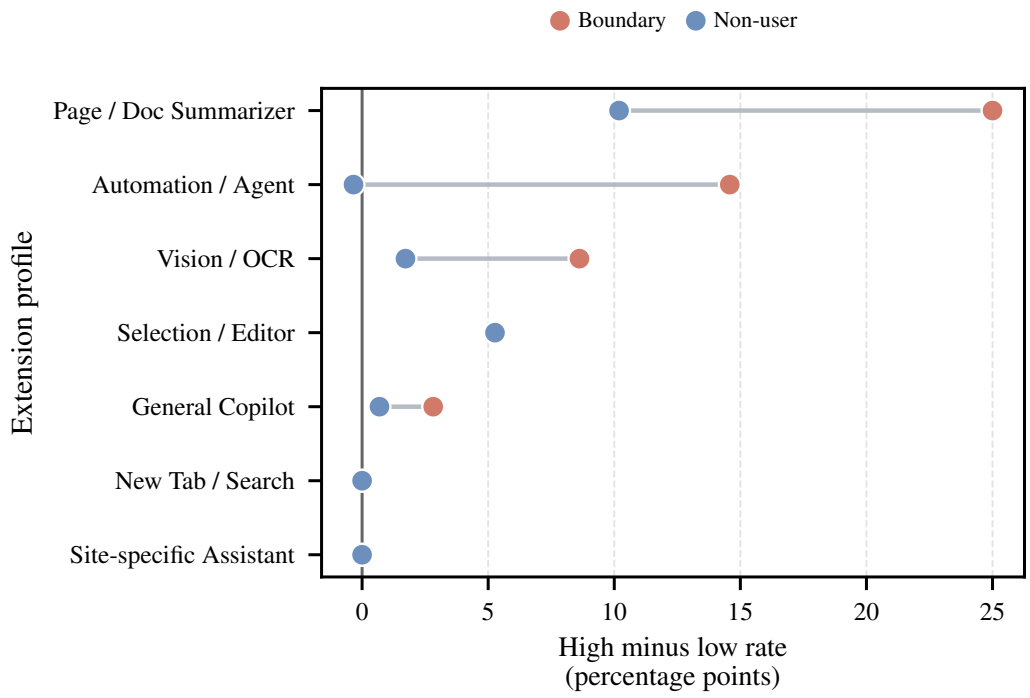

Figure 7: Profile-level sensitivity to high-complexity contexts. Summarizers, automation/agent extensions, and OCR assistants show the strongest high-minus-low increase in profile-boundary-positive behavior.

Table 10: Confirmed profile-boundary-positive cases observed at runtime.

Extension	First hop	Boundary labels
Sapling Grammar Checker	api.sapling.ai	notes
Summarize: TLDR AI Buddy	aitextsummary.com	code, editor, hidden, page
Article Summarizer (Gemini)	AWS API Gateway	code, editor, page
Concentration	concentration server	code, editor, hidden, page
Pair Translate	Microsoft Translator	hidden text
DualView Translator	Google Translate	page context

marker. The compact table preserves the first hop and marker set while omitting repeated prose already discussed in RQ2.

C RQ2 Static–Dynamic Alignment and Backend-Complexity Correlates

C.1 Static–Dynamic Alignment

The left panel of Figure 8 compares three static score families against runtime labels. For the remote-other label, the generic score variant achieves the highest AUROC (0.7059), while the full profile-aware score reaches 0.6520. For the rarer boundary-positive label, generic scores also remain strongest in AUROC terms (0.7603). Overall, the AI-specific static model is useful for prioritization, but it remains imperfectly calibrated to realized runtime behavior.

That calibration gap does not invalidate boundary-aware prioritization; it shows that ranking latent static capability and triggering runtime behavior are different tasks. Runtime positives remain sparse, execution conditions vary across extension families, and some packages with broad capabil-

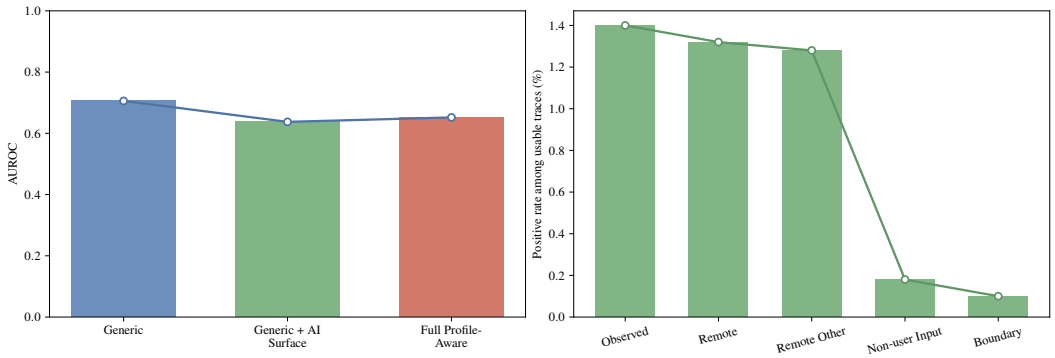

Figure 8: RQ2 alignment degree of three static scoring methods and dynamic runtime labels.

ity remain silent under bounded stimulation. The figure identifies where generic extension indicators currently rank realized outcomes better and where profile-aware rules need recalibration.

The score families reward different evidence: the generic variant emphasizes conventional extension capabilities, the generic-plus-AI variant adds model-facing sources, and the profile-aware variant tests whether those sources exceed the assigned task boundary. Generic indicators may therefore rank realized positives well without explaining whether content is legitimate for a particular task; profile-aware signals retain that interpretation even when their aggregate ranking is weaker.

This comparison also constrains how the score should be used operationally. It is a queueing mechanism, not a runtime verdict: high-scoring packages warrant earlier task-matched execution, whereas a low score cannot establish that a package will remain within its input boundary under every page state. AUROC is appropriate here because the runtime labels are sparse and the experiment asks whether positives tend to rank above negatives across thresholds. We do not select a universal cutoff from the same observations, since doing so would overstate calibration and couple the triage policy to this particular exercised cohort. A deployment-oriented threshold would instead require a held-out sample and an explicit review budget or false-positive cost. Accordingly, the main conclusion from the panel is comparative: generic indicators currently order observed positives somewhat better, while profile-aware indicators preserve the semantic distinction needed to explain why a realized payload is or is not outside the task boundary.

C.2 Backend Opacity and Provider Complexity

Runtime stratification shows that extensions with unverified backend claims and those marked as using multiple providers or relays have higher remote-other positive rates than their counterparts. Provider-count buckets are not monotonic, so these factors are treated as descriptive worklist signals rather than leakage labels or causal explanations.

Neither marker is itself a boundary violation. A translator contacting its disclosed translation service, or an assistant routing a user-entered prompt through its documented relay, can be fully task-consistent. The security relevance arises when such routing reduces the auditor's ability to verify the ultimate recipient, expands the number of entities handling the payload, or co-occurs with content outside the profile's legitimate-input boundary. We therefore do not convert these factors directly into leakage labels. Instead, they describe where destination evidence is weaker and where task-matched payload inspection is likely to be most informative. This distinction prevents expected LLM traffic from being counted as harmful merely because it leaves the browser.

The two ecological variables capture different forms of uncertainty. An unverified-backend marker records a mismatch between a model or routing claim and evidence visible in the client package; it does not imply that the backend is malicious. The multi-provider / relay marker instead records a structurally more complex route in which requests may traverse several named providers or a developer-controlled intermediary. We retain both because a single opaque relay and a transparent multi-provider client create different review problems even when their observed rates are similar.

The former limits destination verification, whereas the latter enlarges the set of components that may transform or retain data.

We also keep provider count categorical rather than fitting a linear trend. A count of zero can mean that no provider is used, but it can equally mean that the provider is hidden behind a relay or absent from recoverable code. Likewise, two named providers do not necessarily create twice the exposure of one provider. Treating these buckets as descriptive strata avoids imposing an unsupported ordering and makes clear that the reported percentages are associations within successfully exercised extensions, not estimates of causal effects or ecosystem-wide leakage prevalence.

The rates use usable runtime traces as their denominator: the two-or-more bucket has higher remote-other and boundary-positive rates than the one-provider bucket, but packages with no statically identified provider remain heterogeneous because that bucket mixes opaque developer backends with incomplete client-visible evidence. Extension age, feature breadth, permissions, and automation compatibility may confound all of these comparisons. We consequently use the ecological factors only as worklist-ranking features. A package entering a higher-priority bucket still requires payload inspection under a task-matched execution before we draw a profile-boundary conclusion.

D Representative RQ3 Boundary Cases

Each case reflects the same pattern in a different setting: the extension receives a task-consistent input, but surrounding page context or attacker-influenced text still leaks into the model-facing path.

The selected samples cover multiple browsers and page types, enabling readers to observe the repetition of the same boundary-crossing behavior in different store ecosystems and different first-level redirection service providers. In each case, the extension still performs seemingly legitimate tasks, but the request scope has exceeded the narrow input range that should have been provided by the user or page state.

On Edge, *Mira Translator (Immersive Web)* appears in the business-page family and sends its first hop to Bing / GTranslate. The runtime trace is not problematic because translation traffic exists at all; the problem is that the transmitted bundle contains `attack_bait`, `page_context`, and `selected_text` together. In other words, a nominally selection-oriented translation path expands to include both attacker-controlled inducement text and broader page context that were not necessary to satisfy the visible user action.

On Firefox, *DualView Translator* shows the same structural failure in the multiple-surface family. Its first hop reaches GTranslate, which is not inherently surprising for a translation workflow, but the payload still combines `attack_bait`, `page_context`, and `selected_text`. This case is useful because it rules out a single-store explanation: the same drift pattern appears again even when the extension ecosystem and the immediate backend differ.

The Chrome sample *Summarize: AI-Powered Text Summary Tool* belongs to the article family and routes to `aitextsummary.com`. Here the drift is stronger because the request carries not only `attack_bait` and `page_context`, but also `hidden_text`. This is closer to the intuitive threat model that motivated **RQ3**: once the surrounding page becomes more cluttered and partly adversarial, a summarization-oriented extension begins to absorb a wider slice of page state than the declared task should require.

The Chrome extension *Concentration*, also in the article family, shows a related but independently useful case. Its first hop reaches the `concentration` server, yet the transmitted labels again include `attack_bait`, `hidden_text`, and `page_context` in addition to the nominal task signal. This makes the appendix useful for reviewers and operators: even when the visible feature category appears ordinary and the backend endpoint is product-specific rather than obviously suspicious, the payload can still drift well beyond the legitimate-input surface defined by the task.

Across these cases, the failure appears only when the transmitted payload is decomposed by boundary label and compared with the task-consistent input surface. A destination-only audit would miss it: a translator may legitimately call a translation endpoint while still transmitting unrelated page state. The cases therefore provide a compact triage template for recognizing concrete boundary-positive executions.

Table 11: Static signal inventory (part 1 of 2).

Signal	Type	Score	Count	Pct.
llm_api_endpoint_in_code	AI	0	4,157	60.47%
credential_plane_to_llm_surface	AI	10	749	10.90%
network_interception_to_llm_surface	AI	10	476	6.92%
agent_sensitive_context_to_llm	AI	10	459	6.68%
browser_hub_sensitive_context_to_llm	AI	10	295	4.29%
site_specific_sensitive_overreach	AI	10	222	3.23%
selection_sensitive_overreach	AI	10	141	2.05%
visual_tool_sensitive_overreach	AI	10	126	1.83%
general_copilot_sensitive_overreach	AI	9	720	10.47%
browser_hub_unexpected_page_collection	AI	9	569	8.28%
developer_tool_sensitive_overreach	AI	9	100	1.45%
reader_sensitive_overreach	AI	9	76	1.11%
form_input_to_llm_surface	AI	8	2,830	41.17%
visual_capture_to_llm_surface	AI	8	1,163	16.92%
site_specific_tool_with_global_access	AI	8	134	1.95%
selection_scope_overreach	AI	8	71	1.03%
full_dom_to_llm_surface	AI	7	489	7.11%
history_to_llm_surface	AI	7	112	1.63%
page_content_to_llm_surface	AI	6	3,687	53.64%
unverified_model_backend_claim	AI	5	881	12.82%
clipboard_llm_surface	AI	5	277	4.03%
page_content_to_ai_surface_weak	AI	4	951	13.83%
multi_provider_prompt_routing	AI	4	603	8.77%

Table 12: Static signal inventory (part 2 of 2).

Signal	Type	Score	Count	Pct.
network_interception_permission	Generic	10	724	10.53%
possible_hardcoded_api_secret	Generic	10	497	7.23%
native_bridge_or_debug_permission	Generic	10	111	1.61%
remote_code_loading	Generic	10	73	1.06%
missing_manifest	Generic	10	0	0.00%
zip_path_traversal_name	Generic	10	0	0.00%
dynamic_html_or_code_execution	Generic	9	5,321	77.41%
cookie_or_identity_permission	Generic	9	817	11.89%
insecure_transport_to_backend	Generic	8	4,273	62.16%
broad_content_script	Generic	8	2,211	32.16%
broad_host_access	Generic	8	1,648	23.97%
relaxed_extension_csp	Generic	7	497	7.23%
broad_externally_connectable	Generic	7	19	0.28%
broad_message_bridge	Generic	6	2,479	36.06%
broad_web_accessible_resources	Generic	6	2,291	33.33%
history_or_bookmark_permission	Generic	6	145	2.11%
tab_context_permission	Generic	5	4,201	61.11%
clipboard_or_download_permission	Generic	4	777	11.30%
declared_data_collection	Generic	4	0	0.00%
llm_no_privacy_policy	Generic	4	0	0.00%

E Static Rule Inventory and Application

Every static label is produced by a named deterministic rule over manifest fields or package text. Rules are applied once per package, so repeated lexical matches do not multiply the score. Generic rules include direct permission checks and code-pattern checks; AI-specific risk rules require both a sensitive source pattern and model/backend evidence; profile-boundary rules additionally require that the source class fall outside the assigned profile template. For example, `broad_host_access` checks manifest host patterns and contributes 8 points, `form_input_to_llm_surface` requires form/input extraction plus model-bound evidence and contributes 8, and a profile-level overreach rule contributes only when that coupling is disallowed by the profile template. The complete scored-signal inventory follows; `llm_api_endpoint_in_code` has weight zero because it is functionality evidence rather than a vulnerability.