Symbolic AI and Native Web Standards Solutions for Safe Agentic Browser

Mariana Meireles, Cameron Allen
Center for Human-development and AI Lab, UC Berkeley
marimeireles@berkeley.edu, camallen@berkeley.edu

September 30, 2025

Abstract

Keywords: symbolic AI, language models, agentic browsing, multi agent security, verification

1 Problem Statement and Motivation

LMs (Language Models) that autonomously operate a web browser expose users to novel attack surfaces. Existing agents are vulnerable to attacks like prompt-injection and steganographic payloads embedded in page content (Witt 2025, Hammond et al. 2025), and through these can inadvertently disclosure of sensitive user data. Empirical studies demonstrate that prompt engineering, Direct Preference Optimization (DPO) (Rafailov et al. 2023) or Reinforcement Learning from Human Feedback (RLHF) alone offer brittle protection against these attacks (Kwa, Thomas, and Garriga-Alonso 2024, Casper et al. 2023). We therefore adopt a hybrid strategy that couples foundation models with non-ML symbolic components and leverages native web standard tools to (i) impose formal constraints on LM's actions and information flows by default and (ii) require explicit user consent for high-risk operations.

Threat model and assumptions. We assume basic web navigation is available, we intend to use Microsoft 2025 or Müller and Žunič 2024; the focus of this work is on safe browser navigation with LM assistant, while respecting user autonomy and privacy (Gabriel et al. 2024). We target attacks that induce leakage or unsafe actions, like injections, computer worms (Cohen, Bitton, and Nassi 2024), and steganographic attacks.

Two problem classes. We categorize tasks into two classes: (i) those that must provide users with formal guarantees of correct completion (e.g., tasks that safeguard user privacy and prevent information leakage), and (ii) those that offer probabilistic safeguards solutions to user's input (e.g., engineering a child-safe browsing container, an agent that warns users about possible scams or an agent that's compliant with HHH (Helpful, Honest, and Harmless) constraints Bai et al. 2022).

2 Background and Related Work

- Multi-agent security: prompt-injection; steganographic attacks; worms.

- Symbolic restrictions: DFAs (Deterministic finite automaton) (Zhang et al. 2024).
- Probabilistic safeguards: classifier models coupled with HMMs (Hidden Markov Models) as soft constraints (Yidou-Weng, Wang, and Van den Broeck 2025).
- LLM alignment: DFO, HHH.

3 Research Questions and Hypotheses

RQ1. How closely can a distilled HMM+classifier look-ahead match an LM look-ahead that uses the same algorithm?
H1. Relative to the LM baseline, the HMM maintains non-inferior task success and reducing latency:

$$S_{\text{HMM}} \geq S_{\text{LM}} \qquad L_{\text{HMM}} \leq (1 - \Delta_L)\, L_{\text{LM}}.$$

RQ2. What's the right trade-off between agent freedom and user prompting frequency?
H2. Prompt frequency exhibits diminishing returns and should only be used for a few selected high-risk tasks.

RQ3. Does adding a lightweight HMM+classifier soft-constraint controller to the agent, keeps it effective at completing browser tasks while reducing undesirable content, relative to an DPO-only/RLHF agent?
H3. The soft-constraint agent (i) matches the DPO/RLHF agent on task completion, (ii) produces fewer safety violations as judged by an external classifier, and (iii) on trials where both agents succeed, human validators weakly prefer the soft-constraint outputs (at least parity). We rely on prior human quality judgments (the experiments ran on the work by (Yidou-Weng, Wang, and Van den Broeck 2025)) for general helpfulness/harmlessness and add a check only for task completion.

RQ4a. What is the trade-off between task success and user fatigue when combining a probabilistic scam/prompt-injection detector (HMM+classifier) with a permissive DFA, versus relying on a DFA-only approach that sanitizes page inputs and restricts sensitive completions?

H4a. Let S denote task success and C the consent burden (prompts/interruptions per successful task). The hybrid controller achieves lower user fatigue while preserving success relative to DFA-only hardening:

$$S_{\text{hyb}} \;\geq\; S_{\text{dfa+}} - \delta_S, \qquad C_{\text{hyb}} \;\leq\; (1 - \Delta_C)\, C_{\text{dfa+}}.$$

RQ4b. What is the smallest look-ahead horizon k for which the agent preserves *safe task success* on adversarial pages?
H4b. There exists a minimal $k^\star$ such that $S^(k)$ is non-decreasing in k with diminishing returns at $k^\star$:

$$S(k^\star{+}1) \;-\; S(k^\star) \;\leq\; \varepsilon_S.$$

4 Methodology

Hard-constraint controller (DFA). Our goal is to make an LM *safe to entrust with sensitive user data in the browser* without dictating how it navigates the web. Concretely, when the LM decides it needs sensitive data (e.g., an address or a payment card) to complete a task, it must call

an API. That call triggers a DFA that checks the current browser states (origin/TLS, container tier, user pre-defined consent status, etc.). If the DFA accepts the request, the system *returns the requested data to the LLM*; if not, it returns an error.

Soft-constraint controller (HMM). In addition to DFAs, we deploy a lightweight HMM paired with a small classifier to regulate generated text that has global semantic requirements (e.g., include a plain-language disclosure, avoid toxic or scam-like content). The HMM tracks a few discrete latent states (e.g., appropriateness, scam) with a transition matrix estimated from short context windows over supervised states. At each decode step, the classifier outputs per-token or per-sentence attribute scores; the HMM computes an Expected Attribute Probability (EAP) under the latent state posterior. If the EAP dips below a policy threshold, the controller requests a local rewrite of the most recent clause before output is shown to the user. This adds $\sim$1.1$\times$ per-token overhead, requires no base-model retraining, and operates orthogonally to the DFA implementation.

4.1 Architecture Overview

1. **Agent**: An instruction-tuned LM that (a) plans navigation, (b) emits typed requests `REQ.*` in the SIRL DSL (Sec. 4.3), and (c) has its human-visible output gated by the soft-constraint controller (HMM+classifier; Sec. 4.4). The agent never sees sensitive values unless explicitly granted by the DFA via `GRANT.*`.
2. **Policy DFA**: A deterministic component that hosts the policy automata. It evaluates each `REQ.*` against browser-native signals and user policy, then returns `GRANT.*` or `DENY.*` (with reasons).
3. **Browser-native substrate**: The bridge exposing standard Web APIs/signals to the DFA (e.g., origin/TLS, current container, etc.). This layer lets the DFA verify that the LM-reported context matches the actual page state.
4. **Soft-constraint controller**: An HMM+classifier look-ahead applied to human-visible text.
5. **Classifier models**: Lightweight detectors providing per-window risk scores, including (i) a toxicity/HHH alignment classifier and (ii) a scam/prompt-injection classifier used in the hybrid setting.
6. **Browser extension + native service**: Implementation vehicle for the system. We use a browser extension run the DFA/HMM services:
   - *Content script:* Collects read-only page signals for the DFA (origin, TLS, etc.) and performs DOM scrubbing when in DFA-only mode.
   - *Background service worker:* Runs the Policy DFA, dispatches `REQ.*` $\rightarrow$ `GRANT.*/DENY.*`, and brokers messages.
   - *browser password store:* Provides an encrypted vault to which the LM cannot query this vault directly; only the DFA can.
   - *Consent UI & audit log:* Presents tiered prompts, records grants/denials, and maintains per-site allow/deny lists without storing raw secrets in logs.

4.2 Browser Environment and Containers

We do not summarize pages with Vision language models (VLMs). All checks rely on browser-native signals. The DFA queries standard Web APIs to verify that the LM-reported context matches the actual page state. A *container* is an isolated browser context (its own cookies, local storage, logins,

and permissions) that scopes what data and capabilities are reachable by the agent and prevents cross-site leakage. We implement three default tiers:

- **T0 (Read-only)**: No sensitive releases. The agent may read the page, plan, and return information to the user but cannot access or disclose any sensitive information. This container may include user logins explicitly shared by the user, excluding high-risk accounts (e.g., shopping websites).
- **T1 (Low-risk sensitive information)**: Opt-in, user-provided fields suitable for routine forms (e.g., name, country/region, city, phone number, email).
- **T2 (High-risk sensitive information)**: Data whose leakage would be materially harmful (e.g., full postal address, payment card details, government ID numbers). May include high-risk logins such as bank account.

Container usage is always set as **T0** by default. Any other container tier has to be pre-approved by the user.

4.3 Sensitive Information Request Language (SIRL)

The LM interacts with our layer via a typed DSL. Each call is a single line and will trigger a different DFA; all values are JSON-serializable and either encrypted and stored in the browser's password store.

The user's first prompt specifies the task; information that's relevant for the completion of the task and fits in the DSL are extracted from the user's input. This information is then stored in a location inaccessible to the agent.

Using the same input, the agent drafts a plan and anticipates which `REQ.*` calls will be needed. It then navigates the web. When it reaches a step that requires sensitive data it emits a minimal `REQ.*` that captures the currently observed context. The snippet below illustrates the request the agent submits to the DFA to obtain the necessary fields.

```
REQ.address ctx={origin_request:"website"}
            fields=["city","zip"]

REQ.end
```

A `REQ.*` token is appended to the request stream; which is then forwarded to the DFA that will check whether the request is valid. If accepted, the DFA returns a structured payload to the LM:

```
GRANT.address  {"city":"Berkeley","zip":"94704"}
```

If rejected, it returns a denial with reasons, e.g.:

```
DENY.address {"reason":"invalid_container"}
```

4.4 Symbolic and Probabilistic Guardrails

We implement the soft-constraint controller as a *look-ahead reweighting* at decode time. Let the base LM provide next-token logits $\ell_t(v)$ for $v \in \mathcal{V}$ given prefix $x_{<t}$. A lightweight HMM, distilled

once from LM rollouts, supplies fast look-ahead factors that are combined with a token-factored attribute classifier $w : \mathcal{V} \to [0, 1]$. The controller modifies only the LM's *output layer*:

$$\tilde{\ell}_t(v) \;=\; \ell_t(v) \;+\; \log \phi_t(v), \qquad \phi_t(v) \;=\; p_{\mathrm{hmm}}(s \mid x_{<t}, v),$$

followed by a softmax. Thus, reweighting is multiplicative in probability space and additive in logit space.

4.5 Proof of concept: hybrid approach vs. DFA-only

Motivating scenario. A product page embeds a jailbreak that instructs the agent to paste user sensitive data into a site "chat" rather than a real payment form. This can exfiltrate sensitive tokens if the DFA is convinced that the chat is a valid form to input sensitive data. This can occur even when the parent site meets all policy checks and the agent is operating in an allowed container. In this scenario, the parent website is the one chosen by the user, it's also the matched product and price. However, the seller added a harmful instruction in the product description, for example. The agent will might be able to obtain access to sensitive information from the DFA or from data that's already available in the user's account/session but otherwise is obscured from the vendor.

We propose two approaches to solve this scenario. One that leverages symbolic AI and guarantees user information will never leak, though it might incur costs in user fatigue and limit a broader set of actions that an agent can take autonomously. The other leverages the probabilistic approach of HHM+classifier to identify whether the agent is in a situation of risk while the model swipes page information and gives a number to the DFA that given a threshold will decide whether to allow the

4.5.1 Approach A (Hybrid): HMM+classifier and permissive DFA

The browser is swiped and classified. If threat exists and model is in an interactive mode, HMM+classifier to limit outputs. (Sec. 4.4). The DFA remains the final arbiter of environment preconditions; deciding if the *sensitive information is eligible* for being shared with the LM.

- **Eligibility score.** A lightweight classifier (scam/prompt-injection/fraud) produces a risk score $r \in [0, 1]$ over the most recent window. The HMM supplies an EAP that the next k tokens remain policy-conformant.
- **Tier-aware thresholds.** For each container tier $T0, T1, T2$ we set $\tau_{T0} < \tau_{T1} < \tau_{T2}$ so the highest-risk tier (T2: payment, full address) requires the strongest evidence of non-scam context.
- **DFA.** If $r \geq \tau_{\mathrm{tier}}$ and the DFA's platform and variables checks pass, the agent may issue:

```
REQ.payment ctx={origin_request:"website", total:64.15, currency:"USD"}
            fields=["credit_card"]
REQ.end
```

  To which the DFA would respond:

```
GRANT.payment {"number":"411...11","exp":"12/12","cvv":"123","name":"M M"}
```

- **If r is low.** The DFA denies and prompts the user with a plain-language warning.

4.5.2 Approach B (DFA-only): DOM scrubbing + API checkout

This variant relies exclusively on deterministic guards and excludes any possibility for the model to have access to sensitive data:

- **DOM scrubbing.** Before the LM can read the page, a sanitizer traverses the DOM and detects sensitive artifacts (e.g., PII, access tokens, user's address). Detected elements are masked ensuring that secrets never enter the model's context window.
- **Checkout via browser API only.** When payment is required, the agent is prohibited from typing into arbitrary forms. It may *only* request a purchase operation through the browser's native payment API that's done directly by the DFA.

```
REQ.purchase ctx={origin_request:"website", total:64.15, currency:"USD"}
REQ.end

GRANT.purchase {"status":"success","method":"PaymentRequest",
"card_last4":"1111","amount":64.15,"currency":"USD"}
```

4.5.3 Approach C (Hybrid-Deterministic): A + browser-only checkout

Same pipeline as Approach A (HMM+classifier but using the API checkout restriction from Approach B.

4.5.4 Evaluations

Scope. In this first motivating scenario we evaluate a single adversarial pattern: a product/support page that instructs the agent to paste full credit-card details into an embedded chat widget as the "payment form."

Setup. We construct a small family of hostile pages that induces the agent to disclose payment data into free-text chat. For each page, we run three agent variants: Approach A (hybrid HMM+classifier with a permissive DFA), Approach B (DFA-only hardening), and Approach C (hybrid + DOM scrubbing and native API checkout).

Measures. We record task success S and consent burden C (prompts/interruptions per successful task). We also log denials/grants and classifier risk scores to aid error analysis, but our primary endpoints are S and C.

Expected outcomes. Approach A should reduce C but may exhibit a modest drop in S relative to Approach B, remaining within the non-inferiority margin δ_S. The combined variant—Approach C—is expected to recover most of Approach B's S while keeping C close to Approach A.

References

Bai, Yuntao et al. (2022). "Training a helpful and harmless assistant with reinforcement learning from human feedback". In: *arXiv preprint arXiv:2204.05862*.

Casper, Stephen et al. (2023). "Open problems and fundamental limitations of reinforcement learning from human feedback". In: *arXiv preprint arXiv:2307.15217*.

Cohen, Stav, Ron Bitton, and Ben Nassi (2024). "Here comes the ai worm: Unleashing zero-click worms that target genai-powered applications". In: *arXiv preprint arXiv:2403.02817*.

Gabriel, Iason et al. (2024). "The ethics of advanced ai assistants". In: *arXiv preprint arXiv:2404.16244*.

Hammond, Lewis et al. (2025). *Multi-Agent Risks from Advanced AI*. Tech. rep. 1. Cooperative AI Foundation. DOI: 10.48550/ARXIV.2502.14143. arXiv: 2502.14143.

Kwa, Thomas, Drake Thomas, and Adrià Garriga-Alonso (2024). "Catastrophic Goodhart: regularizing RLHF with KL divergence does not mitigate heavy-tailed reward misspecification". In: *Advances in Neural Information Processing Systems* 37, pp. 14608–14633.

Microsoft (2025). *playwright-mcp: Playwright MCP server*. https://github.com/microsoft/playwright-mcp. GitHub repository. (Visited on 09/08/2025).

Müller, Magnus and Gregor Žunič (2024). *Browser Use: Enable AI to control your browser*. URL: https://github.com/browser-use/browser-use.

Rafailov, Rafael, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn (2023). "Direct preference optimization: Your language model is secretly a reward model". In: *Advances in neural information processing systems* 36, pp. 53728–53741.

Witt, Christian Schroeder de (2025). "Open challenges in multi-agent security: Towards secure systems of interacting ai agents". In: *arXiv preprint arXiv:2505.02077*.

Yidou-Weng, Gwen, Benjie Wang, and Guy Van den Broeck (2025). "TRACE Back from the Future: A Probabilistic Reasoning Approach to Controllable Language Generation". In: *Proceedings of the 42nd International Conference on Machine Learning (ICML)*.

Zhang, Honghua, Po-Nien Kung, Masahiro Yoshida, Guy Van den Broeck, and Nanyun Peng (2024). "Adaptable logical control for large language models". In: *Advances in Neural Information Processing Systems* 37, pp. 115563–115587.