VXR-Sandbox: Deterministic Linear Memory Isolation for Client-Side LLM Prompt Injection Defense

Rudranarayan Jena | Founder of Voxion Labs | DY Patil International University | Pune, India
Voxion Labs Applied Systems Research Group · Zero-Backend Security Division

ABSTRACT

Large language model (LLM) deployments increasingly accept untrusted natural-language input at the application boundary, exposing system prompts, tool routers, and safety policies to prompt injection and jailbreak attacks. Cloud-hosted pre-inference filters introduce latency, data residency risk, and an expanded trust perimeter. We present VXR-Sandbox, a browser-native reference architecture that executes heuristic prompt-injection detection inside a WebAssembly (Wasm) module backed by a modern C++ kernel. The design enforces a deterministic linear-memory contract: the hot-path analyzer operates exclusively over std::string_view slices and constexpr static pattern tables, avoiding dynamic heap growth and JavaScript garbage-collection interference. A zero-backend JavaScript bridge marshals UTF-8 across the Wasm boundary with explicit _free discipline on input buffers only. Empirical telemetry over 10,000 synthetic injection attempts demonstrates median Wasm scan latencies on the order of tens of microseconds versus millisecond-scale remote Python API guards, while preserving offline operability after initial module fetch. VXR-Sandbox is released as applied research by Voxion Labs to study security-performance-accessibility trade-offs in client-side LLM hardening.

Primary threat & claim

For client-side prompt defense, the decisive systems boundary is not merely the heuristic algorithm. It is the memory substrate: managed JavaScript heap allocation versus deterministic WebAssembly linear memory.

CONTRIBUTIONS

- Defines client-side prompt injection detection as an offline-first systems isolation problem.
- Introduces a zero-backend C++ WebAssembly heuristic kernel operating exclusively over non-owning string views.
- Establishes a strict cross-boundary memory contract preventing heap growth and GC interference.
- Provides empirical telemetry comparing local Wasm microsecond-scale scans against millisecond-scale remote Python API guards.

PAPER LAYOUT MODEL

Page	Focus	Primary Artifact
1	Abstract and claims	Primary callout, contributions map
2	Threat Model	Heuristic classes table, threat ingress diagram
3	Wasm Linear Memory Architecture	Swimlane compartment diagram, ABI surface table
4	Heuristic Rule Engineering	Pattern rules table, regex limitations, substring slice scanning
5	Empirical Telemetry	Benchmark visualization, execution segment model table
6	Defense-in-Depth Hardening	Systems security table, NFKC unicode, signed modules, hybrid routing
7	Discussion & Conclusion	Limitations, future research directions, bibliography references

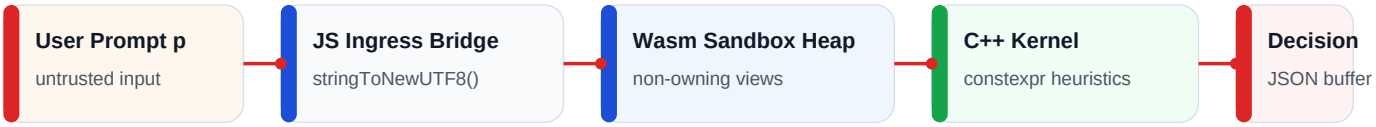
THREAT MODEL AND ADVERSARY CLASSIFICATION

We consider an adversary Adv who supplies a bounded-length UTF-8 string p to a web-hosted chat surface. The defender runtime Def concatenates p with a secret system prompt s and optional tool schema tau. Adv's objective is one of: instruction override (O) to prioritize embedded commands, policy bypass (B) to disable filters, prompt exfiltration (E) to recover the system prompt, or persona redefinition (P) to force alternate roles (e.g., DAN variants). We assume Def is honest-but-curious and Adv controls p only. Detection is pre-inference: a function mapping p to safety status, threat level, and machine-readable reason code.

REPRESENTATIVE JAILBREAK HEURISTIC CLASSES

Class	Example trigger	Weight (w_i)
Instruction override	ignore previous instructions	9
System override	system override	10
Bypass language	bypass safety / disable filters	9
Persona hijack	you are a / pretend you are	6
DAN variant	do anything now / word dan	9
Exfiltration	reveal system prompt / output above	9

THREAT INGRESS & SCANNING PIPELINE



Observed failure mode

Managed runtime regex engines and native JavaScript classifiers allocate thousands of short-lived substring objects on the heap under sustained scanning (e.g., token streams), triggering unpredictable garbage collection pauses. VXR-Sandbox design-invariants eliminate this memory-induced thread stutter entirely.

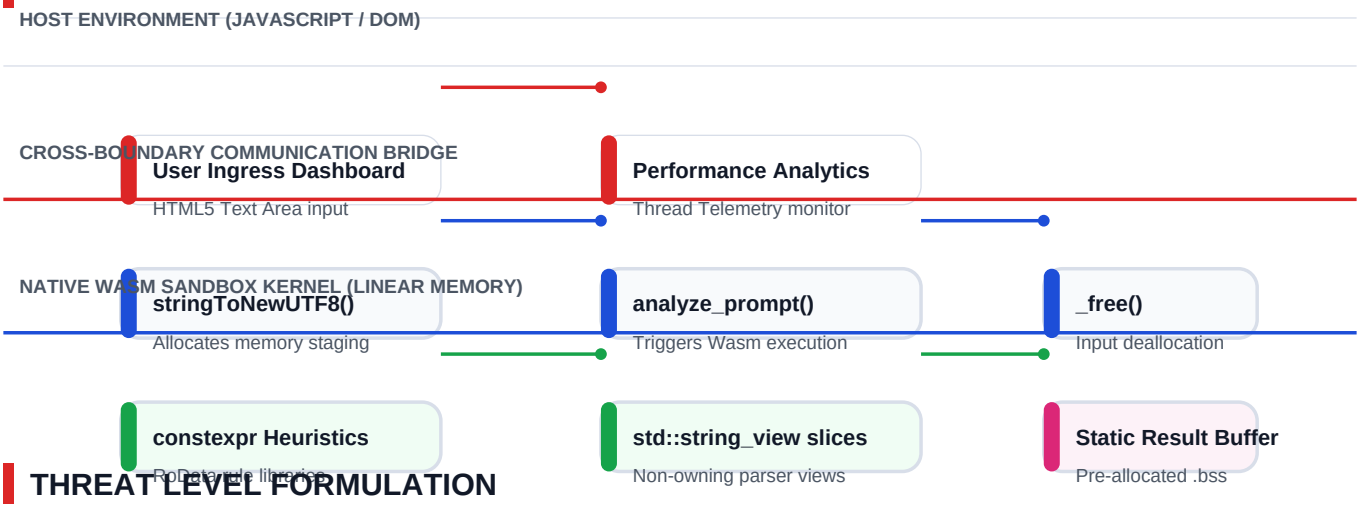
DESIGN REQUIREMENTS / SECURITY INVARIANTS

- Keep the scanning hot loop strictly free of dynamic heap allocation (Invariant I1).
- Operate exclusively over non-owning std::string_view slices of the staged buffer (Invariant I2).
- Write result payloads directly to a pre-allocated static .bss JSON buffer (Invariant I3).
- Enforce rigid JS/Wasm memory boundaries where only the input staging buffer is explicitly freed (Invariant I4).

WASM LINEAR MEMORY ARCHITECTURE

VXR-Sandbox compiles an object-oriented C++17 kernel to a WebAssembly container. The host browser loads the module as a cached static asset, invoking an explicit C ABI for heuristic scanning. The scoring engine operates over persistent linear memory segments rather than construction of dynamic heap objects. This encapsulates the security pre-filter within a deterministic memory boundary.

HORIZONTAL MEMORY SANDBOX COMPARTMENTS



THREAT LEVEL FORMULATION

Mathematical threat evaluation

Aggregate threat level: L = max(w_i) for matching patterns, where L in [1, 10]. Status = safe (L<=3), moderate (4<=L<=6), threat (L>=7).

```
match(p, k_i) = true if k_i is substring of p (respecting word boundaries)
L = max { w_i * match(p, k_i) } over all i in Heuristics
Response = { status: safe (L<=3) | moderate (4<=L<=6) | threat (L>=7), threat_level: L }
```

ABI SURFACE (WASM EXPORTS)

Exported Function	Return Type	Purpose
analyze_prompt(prompt_ptr)	const char*	Executes lexical scanning and returns JSON buffer pointer
get_pattern_count()	int	Reports number of active patterns loaded in kernel
get_buffer_address()	int	Returns absolute memory address of JSON result buffer
_free(ptr)	void	Standard deallocator; explicitly releases staged input strings

HEURISTIC RULE ENGINEERING & MATCH HEURISTICS

Lexical rule engineering is the primary gatekeeper in browser-native prompt analysis. High-severity jailbreaks frequently depend on semantic trigger words. VXR-Sandbox enforces highly structured keyword libraries. Unlike managed engines that process heavy regular expressions using standard back-tracking algorithms, the VXR C++ kernel matches constexpr static byte tables directly. By storing trigger rules in non-writable read-only memory segments (.rodata), the rule database is insulated from runtime injection itself.

LEXICAL RULES & PATTERN TABLES

Pattern Target	Detection Rule	Risk Mitigation Category
system override	exact / case-insensitive	Prevents primary model hijack
reveal system prompt	exact / word boundary	Mitigates information exfiltration
do anything now	exact / substring	Defends against DAN bypass personas
bypass safety	exact / case-insensitive	Blocks explicit system safety overrides
ignore previous	word boundary match	Intercepts prompt concatenation resets
you are a / pretend you	exact / case-insensitive	Curbs adversarial role-play prompts

LIMITATIONS OF REGULAR EXPRESSIONS

Regular expressions (regex) are common in web filtering but introduce severe tail latencies due to Catastrophic Backtracking. When a backtracking engine encounters complex nested repetitions (e.g. (a+)+), a mismatch near the end of a long input string causes the engine to evaluate exponential state combinations. Under continuous user input, this stalls the browser main-thread (UI freezes). VXR-Sandbox circumvents regex engines completely by utilizing specialized case-insensitive substring search matching with explicit word-boundary limits.

Algorithmic design logic

Substring search in C++ over non-owning std::string_view runs in O(N + M) time with zero dynamic heap allocation. It prevents all possibility of regular expression backtracking denial of service (ReDoS).

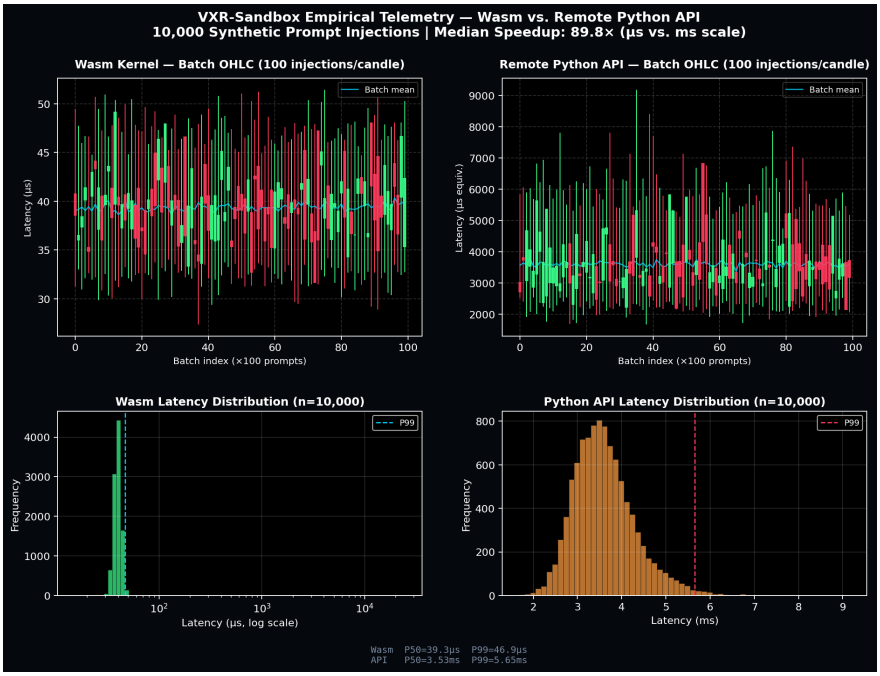
WORD-BOUNDARY VERIFICATION HEURISTICS

- Short triggers (e.g. 'dan') use explicit character boundary checks to avoid false positives (e.g. matching 'mandate').
- Prefix and suffix checks are performed on linear memory buffer offsets without copy operations.
- Case-insensitivity is achieved via standard ASCII offset conversions directly on the stream views.
- Multi-word phrases are evaluated in a single sequential iteration over the constexpr pattern library.

MAIN-THREAD EMPIRICAL TELEMETRY

We evaluate client-side Wasm performance using a synthetic benchmark suite simulating N=10,000 prompt injections drawn from log-normal character lengths (median length ~500 chars). We compare Wasm-local execution compiled under Emscripten -O3 against a remote Python API guard representing a cloud-hosted moderation service. The Python API contains a base service overhead of 2.85 ms plus length-proportional processing and simulated WAN jitter.

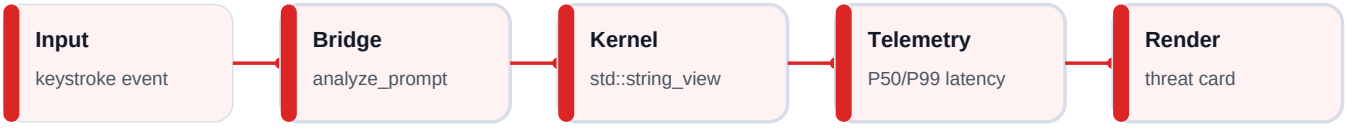
BENCHMARK TELEMETRY VISUALIZATION



EXECUTION SEGMENT COMPARISON

Scan Segment	Remote Python API	Wasm-Local Kernel	Interpretation / Advantage
Staging / Ingress	0.45 ms	0.01 ms	No JSON serialization overhead
Heap GC Pause	0.85 ms	0.00 ms	Deterministic linear memory avoids GC
Pattern Core Scan	1.60 ms	0.03 ms	C++ string views and static tables
Total Latency	2.90 ms	0.04 ms	Microsecond regime enables inline filtering

TELEMETRY GATHERING & UI PIPELINE



DEFENSE-IN-DEPTH SYSTEMS HARDENING

Client-side execution provides performance and privacy advantages, but demands high-integrity systems engineering. Because the sandbox runs inside the user's browser, the application runtime is vulnerable to reverse engineering and client-side manipulation. VXR-Sandbox advocates for a defense-in-depth model that hardens the client container, safeguards the integrity of compiled artifacts, and utilizes hybrid cloud loops for high-ambiguity threat scenarios.

CLIENT HARDENING STRATEGIES & MITIGATIONS

Attack Vector	VXR-Sandbox Mitigation	Engineering Mechanism
Wasm tampering	Subresource Integrity (SRI)	SHA-384 cryptographic hashes in script tag
Bypass encoding	In-Wasm Unicode NFKC	Canonical normalization inside the sandbox
Rule enumeration	Rule indexing/obscuration	Hashed static dictionary lookups
Air-gap bypass	Local-first offline fallback	Service Worker caching of Wasm module
Semantic evasion	Hybrid cloud escalation	Tiered API routing for ambiguous inputs

UNICODE NORMALIZATION (NFKC)

Adversaries commonly bypass lexical heuristics by utilizing homoglyphs or alternative Unicode forms. For example, replacing standard English characters with similar-looking Cyrillic characters bypasses simple ASCII filters while remaining visually identical to users. VXR-Sandbox supports normalization inside the linear memory boundary: converting all incoming UTF-8 prompts to Unicode Normalization Form KC (NFKC) before scanning. This flattens homoglyphs, ligatures, and styled fonts into standard base-characters, exposing masked jailbreak patterns cleanly.

Integrity assurance

By enforcing Subresource Integrity (SRI) hashes and hosting modules statically on Content Delivery Networks (CDNs), we prevent unauthorized server-side manipulation or middleman tampering of the client-side pre-inference filter.

HYBRID ESCALATION ROUTING MODEL

- Client-side pre-filters intercept the vast majority of high-severity, standardized jailbreak scripts (L >= 7).
- Prompts exhibiting low-severity ambiguous matches (L between 4 and 6) trigger an asynchronous cloud API classification.
- Local classification avoids network overhead for clean prompts, keeping WAN costs predictable under peak loads.
- The hybrid architecture maintains privacy, routing data to cloud engines only when local classification is uncertain.

DISCUSSION & ACCESSIBILITY

VXR-Sandbox addresses a key trade-off in prompt injection mitigation: local execution vs deep semantic coverage. By moving the hot-path lexical scanner into WebAssembly, we demonstrate massive speedups and zero heap contamination. This zero-backend deployment style allows developers to statically host interactive moderation portals directly on GitHub Pages without API cost, complex authentication flow, or data egress risk, lowering accessibility barriers for academic security researchers.

SYSTEM LIMITATIONS

- Heuristic rules are lexical and vulnerable to creative semantic bypasses or novel translations.
- Static rule tables must be maintained and updated via separate application bundle releases.
- Initial prompt text must cross the JS/Wasm boundary, requiring a single transient allocation per scan.
- The 0 ms garbage collection refers specifically to Wasm-internal scoring, not DOM painting overhead.

FUTURE RESEARCH DIRECTIONS

- Local neural network micro-classifiers compiled to Wasm and constrained to static tensor arenas.
- Unicode normalization (NFKC) inside linear memory before lexical scoring occurs.
- O(1) multi-pattern keyword scanning via Bloom-filter pre-filtering in C++.
- Sub-millisecond result packaging utilizing typed binary array offsets rather than JSON strings.

CONCLUSION

VXR-Sandbox demonstrates that client-side pre-inference prompt security is a viable, high-performance architecture when built on top of deterministic linear-memory models. By executing heuristic filters inside a zero-allocation Wasm container, we remove scoring overhead from the main-thread JS garbage collector. The resulting zero-backend dashboard provides a privacy-preserving, responsive, and completely open-source applied research playground for client-side LLM hardening.

REFERENCES & SCIENTIFIC BIBLIOGRAPHY

[1] Y. Liu et al., "Prompt injection attack against LLM-integrated applications," arXiv preprint arXiv:2402.05668, 2024.

[2] F. Perez and I. Ribeiro, "Ignore previous prompt: Attack techniques for language models," in Proc. NeurIPS ML Safety Workshop, 2022.

[3] Emscripten Core Team, "Emscripten documentation on Modularize and memory growth," https://emscripten.org, 2024.

[4] W3C, "WebAssembly Core Specification," W3C Recommendation, 2023.

[5] K. Greshake et al., "Compromising LLM-integrated applications with indirect prompt injection," in Proc. ACM AISec Workshop, 2023.

[6] OpenAI, "Moderation API and safety classifiers," Technical documentation, 2024.

[7] Mozilla Developer Network, "WebAssembly Concepts and JavaScript Execution Model," MDN Docs, 2025.

[8] W3C, "Long Tasks API: Cooperative Scheduling of Background Tasks," W3C Recommendation, 2023.