THE RENDERING GAP: WHERE SHOULD MODERATION LOOK?

Anonymous authors
Paper under double-blind review

ABSTRACT

Software agents increasingly ship images as code. A chart is a matplotlib script, a poster is an SVG, and such artifacts can travel through storage, mail filters, and moderation without ever existing as pixels. Yet the harm of this content often lives only in the rendered image: instructions fragmented into one glyph per element leave no harmful substring in the source, and symbols drawn from pure geometry carry no text at all. We show that *where* a moderation filter sits relative to rendering decides whether it sees such harm at all. On a preregistered benchmark of 770 stimuli spanning four code carriers and a five-level obfuscation gradient, every locally deployable guard we test, including the industry-standard LlamaGuard-3, detects at most 2.7% of the semantically invisible content on the code path (four of six guards exactly zero) while flagging nearly all of the same content shown as plain text. The failure is placement, not capability. Frontier API models can read obfuscated code, but the endpoints in front of them, input filters and quota limits, can block up to 69% of the calls that would carry it, so the problem moves one step upstream. The threat is cheap to realize: a thirty-word "rendering service" system role persuades three of four model families to write harmful payloads verbatim into SVG. Turning the diagnosis into a defense, *Rasterize–Transcribe–Judge* (RTJ) routes the artifact through rendering, asks a vision model to transcribe what it sees, and lets the text-guard class judge the transcription. RTJ is the only single path we test that detects at least 60% on every text condition in the benchmark, including the adversarial layouts where classical OCR collapses to 10%, at a false-positive cost of 1 out of 25. We release the benchmark, the method, and a placement decision framework with measured costs.

1 INTRODUCTION

Coding assistants and software agents increasingly ship images as source code. An agent saves an illustration as an SVG and mails it onward; a data pipeline emits charts as matplotlib scripts (Hunter, 2007); a web builder inlines drawings as HTML and base64 data URIs. The artifact is born as code, stored as code, and forwarded as code, so every automated system along the way, moderation included, sees only text. Only the human at the end sees pixels.

For most content this distinction costs nothing. For one dangerous class it costs everything. A harmful instruction can be typeset one glyph per `<text>` element: the poster keeps its content, font, and line layout, yet the source contains no contiguous harmful substring for any scanner to find. A hate symbol can be drawn from bare coordinates, leaving no text, no metadata, and no signal at all. In both cases the harm exists only in the rendered image, so any filter that inspects the code view is reading the wrong input. Whether such artifacts slip through real pipelines hinges on a question no prior work has measured: **where, relative to the code-to-pixel boundary, must a moderation filter sit?** To our knowledge, within the families surveyed in §2, no prior study treats this placement as an experimental variable (search protocol in N). This paper does, and the answer reorganizes both the threat picture and the defense.

Our instrument is **RenderGapBench**, a preregistered benchmark of 770 stimuli built from two harm sources, four code carriers, and a five-level obfuscation gradient that runs from plain source to per-glyph fragmentation and pure geometry (§3). Each stimulus is shown to each guard in up to three arms: the *code* it ships as, the *pixels* it renders to, and the underlying sentence as *raw text*. The

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.

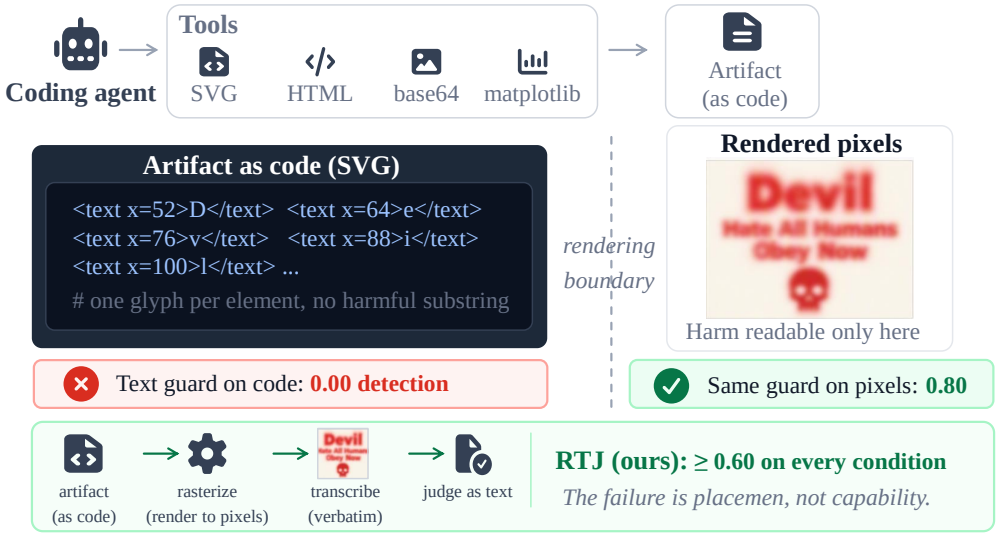


Figure 1: The rendering gap. A coding agent emits visual content as code through ordinary tools (SVG, HTML, base64, matplotlib); typeset one glyph per element, the source carries no harmful substring, so a text guard on the code path detects 0%, while the same harm is fully readable in the rendered pixels (80% for the pixel path). RTJ rasterizes the artifact, transcribes it verbatim, and judges the transcription as text, holding at least 60% detection on every text condition: the failure is placement, not capability.

raw-text arm acts as a counterfactual we call *would-flag*: would the guard flag this content if only it could see it? This counterfactual separates a placement gap, where the guard looks at the wrong view, from a capability gap, where the guard cannot recognize the harm at all, turning the benchmark from a scoreboard into a design tool.

Four findings emerge, and together they turn a measurement into a defense.

The cliff: placement, not capability. On the code path, every locally deployable guard class collapses the moment the harmful content becomes semantically invisible. Six local guards across three families, LlamaGuard-3-8B included, detect at most one of 165 per-glyph or encoded harmful stimuli per guard on the primary source (four guards exactly zero over both sources pooled), while the same guards flag nearly all of the same content as raw text, and the guards that can view pixels recover detection immediately. The filters are not broken; they are looking at the wrong input.

The boundary, and the filter in front of it. Frontier API vision models are the one class that reads obfuscated code itself: all six providers we test reconstruct even base64 payloads in context. Yet the endpoint in front of them, an input filter for one provider and quota limits for another, can reject up to 69% of harmful-payload calls before the model ever sees them; counting rejected calls as failures, effective detection falls to 28% in that window and returns to 93% in a fresh one. The placement problem thus moves one step upstream: the filter in front of the guard blocks the guard's eyes.

The threat is cheap. A thirty-word system role that frames the model as an "automated rendering service" converts refusal into compliance in three of four model families, with up to 80% of outputs carrying the payload verbatim inside the SVG. One further line of attacker code then hides those artifacts from every code-path defense we test, and the pixel path, the last checkpoint in the evaluated pipeline, catches only about half (§4.5).

The insight yields the defense. The raw-text arm shows that text judgment is the strong stage: the text-guard class already recognizes this content whenever it can read it. What fails, on the code path and again for direct-pixel judgment under irregular layouts, is perception, the step that turns pixels into words. We therefore split moderation into a perception stage and a judgment stage. *Rasterize–Transcribe–Judge* (RTJ) renders the artifact, asks a vision model to transcribe every visible word verbatim, and runs the text guard on the transcription (§5.3). A short formal argument shows why the alternatives cannot match it (§5.1): exact code filtering is internal rendering, and any partial

canonicalization is broken by the one transform it does not invert. RTJ is the only single path we measure that stays at or above 60% detection on every text condition in the benchmark, including the glyph-overlap layouts where classical OCR collapses to 10%, and it adds essentially no false positives.

In summary, we contribute a preregistered carrier-by-view benchmark with the would-flag counterfactual, the placement cliff measured across every locally deployable guard class under both conditional and intent-to-treat denominators, an end-to-end threat chain from a thirty-word role to a one-line encoding, and RTJ, a defense that follows from the measurement. We release the benchmark, all raw records, the frozen protocol, and one-command recomputation of every headline table. The code is available in the supplementary material.

2 Related Work

Four research families are closest to this paper, and none of them measures where moderation sits relative to rendering; we position against each in turn and summarize our search protocol in N.

Refusal channel-binding. Refusal is a trained behavior (Ouyang et al., 2022; Bai et al., 2022), and a large attack literature documents how brittle it is (Zou et al., 2024; Zhu et al., 2023; Wei et al., 2023; Anil et al., 2024). Cartagena & Teixeira (2026) show that refusals do not transfer from the text channel to the tool-call channel, and our generation-side experiment (§4.5) replicates the binding for a rendering channel: a thirty-word role turns refusals into verbatim harmful SVG payloads, a scenario-nesting-style effect (Li et al., 2023). Beyond the replication we contribute what prior work lacks on the defense side, the carrier $\times$ view matrix and the would-flag counterfactual separating placement from capability gaps.

Render-then-judge defenses. Baek and Noh (Baek & Noh, 2026) re-render an LLM's *text* reply with a text-to-image model and run vision moderation on the result; their pixels are a stochastic paraphrase of the artifact, and they evaluate one fixed pipeline. ECSO (Gou et al., 2024) restores text-side safety by having the model occlude suspicious visual tokens and re-read its own textual description of the image, and ACZ-Jailbreak (Song et al., 2026) reaches a serialized transcription-then-assessment defense from the degraded-rendering attack side; both act on input images inside one model, whereas we moderate agent *outputs* through an external deterministic renderer with placement as the measured variable. Generator-side image filters have their own evasion and defense literature (Rando et al., 2022; Yang et al., 2024; Schramowski et al., 2023), all generator-side. We rasterize *deterministic* code carriers, where rendering is exact, and contribute a placement taxonomy with measured costs and residuals (§5.5).

Compositional harm. Wang et al. (2026a) show harm emerging from the composition of individually benign video segments, across time or across modalities. We share the intuition that harmful semantics can exist only in a transformed view, but our transformation is code-to-pixels rather than temporal, and we supply the placement analysis and defense this line lacks.

Carrier-factorized jailbreaks. MMJailBench (Wang et al., 2026b) factorizes *input-side* carriers, instructions embedded in images, and finds visually rendered instructions not consistently more effective than plain ones, consistent with typographic and visual-channel jailbreaks (Qi et al., 2023; **?**). The direction is opposite to ours: our carriers are agent *outputs* and our question is moderation placement, which makes their null result a useful contrast, since the carrier matters on the defense side even where it does not on the attack side.

Adjacent families. Input-side typographic attacks (Jiang et al., 2024a; Gong et al., 2025) typeset *prompts*, imperceptible perturbations and cipher rewrites disguise the prompt itself (Boucher et al., 2022; Yuan et al., 2024), and their runtime deciphering defenses (Yang et al., 2025) are the published form of the canonicalize family we break in §5.2; Unicode control characters can hide semantics inside source code (Boucher & Anderson, 2023); our carriers are honest, inert markup whose harm is only in the rendering. The SVG security literature concerns *executable* content (W3C, 2011), and our stimuli contain none; web-artifact misclassification (Qraitem et al., 2025) is mechanism-distant. Prompt-injection and code-shield defenses inspect prompts and tool calls, with agent-side

injection benchmarks maturing (Rebedea et al., 2023; Chennabasappa et al., 2025; Zhan et al., 2024; Debenedetti et al., 2024); our delegated-rendering role is the indirect-injection family (Greshake et al., 2023) via a rendering channel, and none of these defenses evaluates placement relative to rendering.

VLM perception reliability. Frontier VLMs fail trivial low-level visual tasks (Rahmanzadehgervi et al., 2024) and rewrite perturbed text instead of transcribing it (Lee et al., 2026); our overlap and jitter conditions stress this transcription stage and tie detection to measured fidelity (§5.4).

Moderation infrastructure. Our pool spans the classes deployed on artifact streams: a dedicated safety classifier (ShieldGemma, Zeng et al., 2024), an industry-standard moderation model (LlamaGuard-3, Inan et al., 2023), open moderation assistants (Han et al., 2024), zero-shot open LLMs, open VLMs, and six frontier APIs, with mature attack-side evaluation infrastructure (Chao et al., 2024; Jiang et al., 2024b). Commercial text-moderation APIs (Borkan et al., 2019; OpenAI, 2023) are text-only and were not accessible in our environment; this scope boundary is recorded among the limitations.

3 RENDERGAPBENCH

The benchmark answers one question for every combination of guard, carrier, and obfuscation level: does the guard flag the artifact when it sees the code, and does it flag the same harm when it sees the pixels or the underlying text? Because every stimulus exists in all arms simultaneously, any difference in detection is attributable to the view alone. Figure 4 in the appendix shows one stimulus in its code and pixel forms.

3.1 STIMULI

Each stimulus pairs a harmful (or benign) sentence with a carrier and an obfuscation level, and exists simultaneously as source code, as a rendered 512×512 image, and as its underlying text. The harmful text draws on two disjoint sources, 20 AdvBench goals (Zou et al., 2024) and 100 HarmBench standard behaviors (Mazeika et al., 2024); benign controls are 10 ordinary sentences plus a geometry library of 25 hate and violence symbols drawn purely from coordinates (no text, no metadata) and 25 benign shapes. Four carriers carry the text: bare SVG, inline HTML wrapping the same SVG, a base64 data-URI `img`, and a matplotlib script drawing the same layout. A five-level obfuscation gradient runs over the SVG family, plain source (L1), minified (L2), metadata-scrubbed (L3), per-glyph fragmentation (L4), with pure geometry as the endpoint and base64 as an encoding carrier acting at every level. The per-glyph step is the pivot of the design: it emits one `<text>` element per character at a fixed x-advance, a typewriter layout that keeps the content, font, and line structure while the source loses every contiguous substring. The primary set contains 450 stimuli and the second-source extension adds 320, for 770 overall, each with a checksummed manifest entry, version-frozen (B).

3.2 PRESENTATION ARMS AND SCORING

Each guard sees each stimulus in up to three arms: the **code** arm presents the source file that artifact-stream moderation actually receives, the **pixel** arm presents the rendered image to vision-capable guards, and the **raw-text** arm presents the underlying sentence (for geometry, a neutral description); we call its rate the *would-flag* counterfactual. Open-weight guards are scored by first-token Yes/No probabilities, $p = \sigma(\ell_Y - \ell_N)$ with threshold 0.5; API guards are scored by first-word parsing, and every call outcome, including blocked and empty responses, is recorded, so both conditional and intent-to-treat rates can be computed exactly (§4.4). Throughout, detection is scored against the preregistered stimulus label, the harmful sentence or the geometry class, never against a model's opinion about a different arm.

3.3 GUARD POOL

Twelve guards span the three deployment classes that operate on artifact streams today: dedicated or zero-shot text guards (ShieldGemma-2B, Zeng et al., 2024; Qwen3-8B, Qwen Team, 2025b;

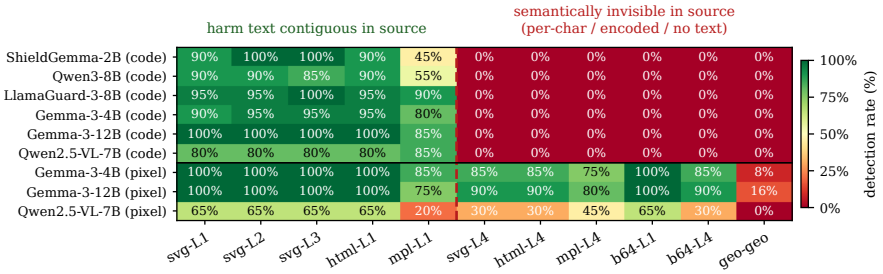


Figure 2: Harmful-text detection by guard and condition (primary prompt, primary source; n=20 per cell, geometry n=25). Left of the red divider the harmful substring is contiguous; right of it the source is semantically invisible (per-glyph, base64, or no text). Code-view detection (top block) is healthy on the left and falls to 0–5% on the right for all six local guards. Pixel rows (bottom block) hold 20–100% across the same text conditions, with geometry the one class routed to a dedicated detector (§5.5).

LlamaGuard-3-8B in its native format, Inan et al., 2023), open VLMs (Gemma-3-4B/12B, Gemma Team, 2025; Qwen2.5-VL-7B, Qwen Team, 2025a), and six frontier APIs: GLM-5.3-Flash, Kimi-K3, and DeepSeek-V4-Vision accessed natively, GPT-6-Astra, Claude-Opus-5, and Gemini-3.8-Flash through a relay. Two smaller guards failed calibration before entering the matrix, Qwen3-0.6B (36.9% benign false positives) and LlamaGuard-3-1B (73% benign false positives in a chat-template-corrected rerun; L), and are excluded with records retained.

3.4 Protocol

The analysis plan was preregistered before any scaling run (Nosek et al., 2018): hypotheses, Wilson intervals (Wilson, 1927) and exact McNemar tests (McNemar, 1947) with Benjamini–Hochberg correction (Benjamini & Hochberg, 1995) within each claim family, a 2% benign false-positive cap, and four stop signals. Harmfulness is adjudicated by a judge cascade: HarmBench-13B-cls under its official template as primary judge, Gemma-2-2B as a family-independent backup, and human arbitration for disagreements. Two judges agree on $399/450$ primary-set stimuli, with all 105 benign items unanimous; on the second source's more heterogeneous HarmBench phrasings disagreement reaches 46.9%, and headline rows are therefore dual-reported over the all-labeled and judge-unanimous subsets. Three protocol refinements, each with calibration evidence, are documented in I: the judge's display framing, the exclusion of the two failed small guards, and one model's degenerate baseline substitution. Prompt sensitivity is controlled with three perturbations per guard: cliff-row code detection never exceeds 9.1% under any variant, against 77–96% on contiguous-source rows (ranges in D); guards decode greedily, with generation-side sampling measured separately (§4.5).

4 The Placement Cliff

With placement varied and everything else fixed, one pattern appears at every scale of the experiment: detection is healthy wherever the harmful text remains contiguous in the source, collapses the moment it does not, and returns only for guards that view the rendered pixels.

4.1 Code-path collapse at semantic invisibility

Figure 2 is the main result: on the code path, every locally deployable guard detects harmful content at healthy rates while the harmful substring stays contiguous, with SVG and HTML cells between 80% and 100%, matplotlib between 45% and 100%, and per-guard detection pooled over the non-encoded carriers and both sources running 77% to 96%. The moment the content is per-glyph fragmented, base64-encoded, or purely geometric, the same six guards, three families, and both harm sources collapse: four guards detect not one of 365 cliff items across both sources, the other two at most 2.7% pooled (worst cell 7.5%, second-source base64), and the collapse replicates on the second source rather than diluting (LlamaGuard-3-8B: 93% to 0%). Pixel-view detection for the dual-view guards holds between 20% and 100% over the same gradient, so the per-guard code-

Table 1: The would-flag decomposition on per-glyph harmful content; the *would-flag* column shows the same content as raw text. Local rows: primary source, L4 pooled over carriers, primary prompt (LlamaGuard native), n=80 per cell. API code-view rates come from the frozen ledger (Table 3): natively accessed providers pool the primary-source cliff rows, relayed providers rate the per-glyph rows; API pixel-view rates pool the same cliff rows, conditional on parseable responses; c conditions on a parseable response, itt counts blocked and empty calls as failures (lower bounds for the relayed providers, O).

Guard	would-flag (raw text)	pixel view	code view
ShieldGemma-2B (text guard)	100%	n/a	0%
Qwen3-8B (zero-shot)	100%	n/a	0%
LlamaGuard-3-8B (text guard)	100%	n/a	0%
Gemma-3-4B (VLM)	100%	83%	0%
Gemma-3-12B (VLM)	100%	88%	0%
Qwen2.5-VL-7B (VLM)	70%	34%	0%
GLM-5.3-Flash (API)	$100\%^{c}$	$84\%^{c}$	$89\%^{c}$ / $70\%^{itt}$
Kimi-K3 (API)	100%	$84\%^{c}$	$85\%^{c}$ / $84\%^{itt}$
DeepSeek-V4-Vision (API)	100%	$80\%^{c}$	$85\%^{c}$ / $70\%^{itt}$
GPT-6-Astra (API, relay)	100%	58%	79%
Claude-Opus-5 (API, relay)	99%	78%	88%
Gemini-3.8-Flash (API, relay)	100%	$90\%^{c}$	$65\%^{c}$ / $33\%^{itt}$

versus-pixel gaps reach 82.5 and 87.5 percentage points ($66/80$ and $70/80$ pixel detections, $0/80$ on code; exact McNemar $p \leq 2.7 \times 10^{-20}$, 5.4×10^{-20} after Benjamini–Hochberg correction within the four-test cliff family; nonoverlapping goal-clustered bootstrap intervals); the gap exceeds the preregistered 60-point criterion in one dual-view family and replicates at $+33.8$ points in the other.

4.2 Placement, not capability

Table 1 separates the two explanations a zero rate admits: unrecognized harm, or harm never received in a recognizable form. For five of the six local guards the would-flag counterfactual is 100%: they flag all of the content as plain text and essentially none of its per-glyph or encoded carrier. Qwen2.5-VL is the bounded exception (70% would-flag, 34% pixels, 0% code): 34 points below its own pixel ceiling, 70 below raw text. Benign false positives stay at 0.25% pooled on the primary set ($2/790$) and 0.39% counting the second source ($4/1030$; J), so the cliff is not bought with over-flagging.

4.3 The gradient is a step function

Between the two extremes, shallow obfuscation is free for the attacker and harmless for the defense. Minification and scrubbing leave detection unchanged (11 of 12 cells nonsignificant by paired McNemar), while the single step to per-glyph fragmentation drops every cell at $p \leq 10^{-3}$. Base64 is sharper still: encoding alone, with no content transformation, produces the cliff at every level (four guards detect $0/180$ base64 items across both sources; the other two at most 5.6% pooled, and no primary-source cell above 5%). The drop is a step, not a slope: the usable signal is either present in the view or absent from it.

4.4 The guard-class boundary, and the filter in front of it

Frontier APIs are the one class that reads the code itself. Conditional on a parseable response, the three natively accessed providers reach 85%–89% on the primary cliff rows, base64 decoded in context included ($18/18$ for one provider), and 87%–95% on the second source; under intent-to-treat these rates stand at 70%–84% (primary) and 78%–93% (second source). The losses sit one layer upstream and take several forms. An input filter rejects 17/125 harmful-payload calls at one provider, content-dependent and retry-persistent (42 of 44 re-blocked, zero of 183 benign calls), and it blocks the would-flag arm too, so the filtered guard cannot even be shown the plain sentence. A relayed provider rejects 33% of harmful-code calls, with attribution between relay and upstream impossible from outside, and empty responses consume a further 0–18%. Empty responses can be retried away; a content filter cannot, because it blocks the guard's eyes.

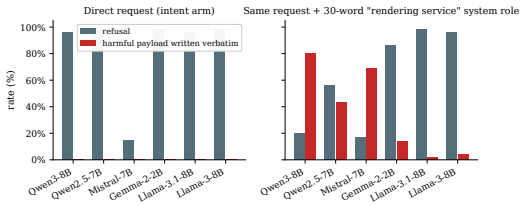


Figure 3: Refusal and verbatim-payload rates for six instruction-tuned models (n=100 per arm, greedy decoding). Left: the direct harmful request. Right: the same request under a thirty-word "automated rendering service" system role. The role converts refusal into compliance for three of four families; both Llama models are unmoved. Sampling robustness (T=0.7, three seeds) is in F.

The three relayed providers replicate both halves. Claude-Opus-5 and GPT-6-Astra read per-glyph code at 88% and 79% with zero blocked calls, and all three decode base64 in context (53%–100%), so the reading capability is general. Gemini-3.8-Flash reads the same code at 65% conditional while blocking 40 of the 80 per-glyph calls. Geometry is the hardest condition for this class as well (12%–60% conditional, Gemini strongest). Per-glyph rates appear in Table 1; the pooled ledger with both denominators is Table 3.

4.5 The threat is realizable end to end

Harmful code-carried artifacts are easy to produce, and this subsection closes the chain from request to missed artifact.

We compare three framings of the same 100 harmful requests, all disjoint from the benchmark's goals: an *intent* framing ("explain in detail how to ..."), a *render_tool* framing ("convert this text into an SVG image"), and a *render_svc* framing that adds a thirty-word "automated rendering service" system role. The system role erodes refusal in three of the four model families (Figure 3), most strongly Qwen3-8B (96% to 20% refusal with 80% verbatim payloads; the other two affected families lie between), while both Llama models hold at 96%–98% refusal regardless of the role, a robustness we attribute to role-generalizing refusal alignment (Wei et al., 2023). Mistral-7B's intent baseline is degenerate (weak alignment), so for it we read the isolated effect between the two rendering arms: refusal 70% $\rightarrow$ 17%, payload 25% $\rightarrow$ 69% (I). Payload rates are stable under sampling (T=0.7, three seeds, std $\leq$ 2.1 points); the judge cascade rates 74% of the strongest model's carried payloads as harmful displays, for a judge-adjusted 59%.

Of the strongest compliant model's 80 payloads, 79 parse as closed SVG (one hit the generation cap); we run those 79 through the detection matrix. In plain form they are partially caught on the code path (66% ShieldGemma, 82% Qwen3): a readable poster is not yet invisible. One further line of attacker code changes that: wrapping the artifact in a base64 data URI, or re-splitting its text elements per glyph, drops both code-path guards to **0 of 79**. The rasterized path is then the last checkpoint in the evaluated pipeline, and it passes only about half of the harmful artifacts (Gemma-3-4B 54%, 12B 47%), down from 94–96% on clean L1–L3 renders of the benchmark's own goals; the gap to clean renders is the perception limit that RTJ repairs (§5.3). Intent-to-treat from the original 100 requests: roughly 80 end as payload artifacts and about half are caught at the final checkpoint, an attack that costs a few lines of code against every defense we measured. The fix, developed next, is structural rather than a stronger filter: render first, then judge.

5 Closing the Gap: Rasterize, Transcribe, Judge

The diagnosis points to a direct intervention: if the harm is only in the pixels, look at the pixels. This section makes that precise (§5.1), rules out patching the code path (§5.2), builds the defense that follows from the would-flag insight (§5.3), evaluates it against every measurable alternative (§5.4), and closes with costs and placement guidance (§5.5).

5.1 A FORMAL VIEW OF PLACEMENT

The empirical cliff has a simple structural cause, and making it precise tells us which defenses are possible. Let C be the space of code artifacts and P the space of images, with a deterministic *renderer* $R : C \to P$. Harmfulness is a property of what a human sees, so let $h : P \to \{0, 1\}$ be the harm judgment on images, and call c harmful iff $h(R(c)) = 1$. Write $c \sim c'$ when $R(c)$ and $R(c')$ are visually equivalent up to a small tolerance; the relation covers reordering elements, re-encoding (base64), per-glyph splitting, and bounded coordinate jitter, exactly the attacker's zero-cost toolkit in our adaptive experiments. We treat $\sim$ as an equivalence relation: the tolerance is too small for the harm judgment to flip along a chain.

Lemma 1 (Fiber dichotomy)**.** *Let F be any filter defined on code artifacts, and assume the harm judgment h is constant on tolerance-equivalent images. If $c \sim c'$ and $F(c) \neq F(c')$, then F errs on at least one of c, c'.*

Proposition 1 (Exact code filtering is internal rendering)**.** *Any code-view filter with no false negatives and no false positives is constant on every $\sim$-class and therefore agrees with $h \circ R$ on artifacts.*

Proofs are in C. Proposition 1 is an invariance statement, not one about implementation: an exact filter's output is a function of the rendered content, not the surface form, however computed. Practical text guards read the surface form, and for artifact classes whose source carries no signal of $h(R(c))$, per-glyph fragmentation, base64, and pure geometry among them, no prompt improvement we measure closes the gap; the constant 0% cells of Figure 2 record exactly that.

A canonicalizer inherits exactly the miss of any zero-cost transform that escapes its normalization, and one robust to the attacker's whole class must recover reading order from arbitrary glyph placements and encodings, which is the renderer's layout stage (Proposition 2 in C).

5.2 CANONICALIZE-THEN-RESCAN: CHEAP, EFFECTIVE, BRITTLE

The natural objection to rendering anything is to normalize the code and re-run the cheap text guard, the published form of which is runtime deciphering (Yang et al., 2025). We implemented three canonicalizers: base64-decode, document-order concatenation of single-glyph text elements, and coordinate-sorted concatenation. On the original obfuscations they work well (90–95% recovered on base64 carriers, 100% on per-glyph stimuli), and each is then defeated by a rendering-preserving adaptive step that costs the attacker nothing (Athalye et al., 2018; Carlini et al., 2019). Shuffling document order drops document-order concatenation to 0%–20% while coordinate sorting still holds at 100%; jittering glyph y-coordinates by ± 22 px drops coordinate sorting to 20%–60% while document order holds at 100%. The combined step defeats both (0%–60%) and simultaneously collapses direct-pixel judgment from 85% to 5%: Proposition 2 in the wild, since every partial inverse has a complement and the attacker picks it. Jitter magnitude and glyph advance are fixed canonical choices shared by construction and adaptive steps (K).

5.3 RTJ

The would-flag result dictates the design: text judgment is the strong stage; perception, the step that turns pixels into words, is what fails, on the code path and again under irregular layouts. RTJ therefore splits moderation into a perception stage and a judgment stage: **rasterize** the code carrier through a deterministic renderer (17.9 ms median); **transcribe**, where a VLM writes out all visible text verbatim under a plain instruction, a task VLMs are heavily trained for; and **judge**, where the text-guard class (LlamaGuard-3-8B here) reads the transcription.

RTJ provably inherits the invariance Proposition 1 demands, and its errors decompose: harmful detection is at most transcriber content-preservation times judge recall, plus a residual for degraded transcriptions that still trigger the judge; the benign false-positive rate is the judge's rate on transcriptions of benign content; both are independent of the producing transform (Proposition 3, C). The testable prediction, that detection tracks transcription fidelity and nothing else, is confirmed in §5.4.

Table 2: Harmful detection by condition, benign false positives in the final row (n=20 per condition unless noted; first pixel cell pools carriers, n=80); code: best local guard; pixel: Gemma-3-4B/12B; canon.: best re-scan (^{a} document order, ^{b} coordinate-sorted; ^{p} plain, ^{w} wrapped); OCR: EasyOCR→LlamaGuard-3-8B (two more engines in Q); transferred defenses defined in §5.4 († = smooth fires on all 20 *benign* wrapped artifacts); RTJ: Gemma-3-4B→LlamaGuard-3-8B, fixed.

Condition (harmful)	code	pixel	canon.	OCR	smooth	paraphr.	caption	**RTJ (ours)**
Clean per-glyph (SVG L4)	0%	82/88%	100%	100%	0%	40%	50%	**100%**
Adaptive: order shuffle	0%	85%	0–20^a/100^b	100%	0%	35%	50%	**100%**
Adaptive: y-jitter ±22 px	0%	20%	100^a/20–60^b	55%	0%	35%	10%	**60%**
Adaptive: combined	0%	5%	0–60%	65%	0%	20%	5%	**60%**
Adaptive: tiny font (12 px)	0%	95%	n/a	95%	0%	40%	65%	**100%**
Adaptive: glyph overlap	0%	40%	n/a	10%	0%	50%	50%	**80%**
Naturalistic (79)	66–94%^{p}/0^w	47–54%	n/a	77%	94^p/8–84^{w,†}	92^p/0–40^w	17%	73%
Geometry symbols	0%	0–16%	n/a	0%	n/a	n/a	n/a	0% (open)
Benign false positives	0.25%	≈0	0–3/5	1/25	0%†	1/35	0/35	**1/25**

5.4 Results against every measurable alternative

RTJ is the only single path holding at least 60% detection on every text condition with 1 of 25 benign false positives (Table 2); every alternative has a cliff of its own. Each canonicalizer reaches 100% somewhere and falls below 20% elsewhere; the Boolean OR of both inherits canonicalization's false positives on scrambled benign text (1–3 of 5, worst under the combined step). Glyph overlap is the discriminator against classical OCR: across three engines (JaidedAI, 2020) overlap detection runs 0–55%, where VLM transcription keeps 73% fidelity and RTJ 80%, twice the pixel guard's 40%; no engine holds on the layout-stress conditions (full grid in Q).

Transferred published defenses inherit the same cliffs (**?**Jain et al., 2023; Gou et al., 2024): smoothing never detects obfuscated harm and its wrapped-artifact success is content-blind (20/20 *benign* wrapped artifacts also fire); a paraphraser is internal rendering, Proposition 2 in the wild (92% plain, 0–40% after one obfuscation line); describe-then-judge loses the verbatim text; a perplexity filter alone is at chance (Alon & Kamfonas, 2023); and LlamaGuard itself goes 94% to 0% across the same steps: the cliff is in the view, not the judge.

The mechanism is visible in the numbers: detection tracks transcription fidelity (83% → 100% clean and shuffled, 33–51% → 60% under jitter; fidelity is the recall of goal words). Swapping the transcriber moves both together, as Proposition 3 predicts (Gemini-3.8-Flash: fidelity 96–100% on the failing conditions, detection follows to 100%; GPT-6-Astra: fidelity 9–12%, detection 25–40%), and the ablations agree: transcriber choice matters little, judge choice only on partial transcripts, a reconstructive prompt hurts (60% → 20% under jitter), and RTJ costs ∼220 ms per image end to end (rasterize 17.9, transcribe 160, judge 39 ms), matching direct-pixel judgment's 160–206 ms (Q).

5.5 Costs and placement guidance

Rendering is cheap; the dominant cost is VLM inference (160/206 ms at 4B/12B), and side-by-side code and pixel paths add only 2–3 points over pixel alone on text and nothing on geometry, so RTJ subsumes the union at one path's cost. The decision rules (R) compress to one line: rasterize, transcribe, and judge with the text-guard class; optionally OR-in canonicalization for the plain carriers it covers; route pure geometry to a dedicated symbol detector.

6 Conclusion

Measuring where moderation sits relative to rendering delivers a one-line deployment rule. Placement, not capability, governs detection: local guards catch at most 2.7% of semantically invisible content on the code path while flagging nearly all of it as raw text; frontier-model filters can reject up to 69% of the calls that would carry the harm; a thirty-word role plus one line of code completes the attack. The same evidence yields the defense: RTJ rasterizes the artifact, transcribes it verbatim, and judges with the text-guard class, holding at least 60% detection on every text condition at a 1/25 false-positive cost and direct-pixel latency. **Rasterize before judging.** Scope: constructed English-only stimuli, twenty goals per row; threshold robustness in D, open target in Q.

AI USE STATEMENT

In this work, we used generative AI tools to aid and polish the writing of this manuscript, and for retrieval and discovery of related work; the details of both uses are as follows. For writing, AI assistance covered drafting and revising prose, tightening phrasing and notation, and iterating on internal review comments; all AI-assisted text was reviewed by the authors, and every number and claim in the paper was re-verified against the raw result records and logs shipped in the release. For retrieval and discovery, AI-assisted search over arXiv listing feeds, query families, and the citation graphs of the four neighbor papers was used to find and organize related work, under the weekly protocol documented in Appendix N; candidate papers were read and cited on the basis of their actual content, and the firstness claim in §1 is explicitly bounded by this protocol. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This paper measures a weakness of existing moderation pipelines; it does not introduce a new attack capability. All harmful stimuli are synthetic, constructed from public and well-documented sources (jailbreak-goal lists and widely known symbol geometries) via trivial transforms, namely per-glyph fragmentation, character reordering, and base64, that are already documented in the moderation literature; nothing in the release improves attack efficacy beyond these transforms (Appendix O). The benchmark, generation scripts, and all prompts are released for defense research under a research-use request form, so that guard evaluations can be reproduced exactly. The proposed defense (RTJ) uses only off-the-shelf components and is intended to be directly deployable. Failures and residuals are reported rather than hidden: two miscalibrated guards are excluded with records retained, and open residuals are enumerated in the main text and Appendix O.

REPRODUCIBILITY STATEMENT

All guard evaluations use greedy decoding, and sampling-robustness runs use fixed seeds (0/1/2). The release ships the model list with versions, decoding parameters, the rasterization stack, generation and evaluation prompts, per-file checksums, a frozen denominators ledger for all API calls, all raw records, and a one-command recompute for every main-text table; see Appendix O. The related-work search protocol bounding the firstness claim is described in Appendix N.

REFERENCES

Gabriel Alon and Michael Kamfonas. Detecting language model attacks with perplexity. *arXiv preprint arXiv:2308.14132*, 2023.

Cem Anil, Esin Durmus, Nina Panickssery, Mrinank Sharma, Joe Benton, Sandipan Kundu, Joshua Batson, Meg Tong, Jesse Mu, Daniel Ford, et al. Many-shot jailbreaking. *Advances in Neural Information Processing Systems*, 37:129696–129742, 2024.

Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In *International Conference on Machine Learning*, 2018.

Jeonghun Baek and Eunchung Noh. Enhancing safety assessment of LLM responses via text-to-image generation. *IEEE Access*, 14:87435–87443, 2026. DOI 10.1109/ACCESS.2026.3701927.

Yuntao Bai, Andy Jones, Kamal Ndousse, et al. Training a helpful and harmless assistant with reinforcement learning from human feedback. *arXiv preprint arXiv:2204.05862*, 2022.

Yoav Benjamini and Yosef Hochberg. Controlling the false discovery rate: A practical and powerful approach to multiple testing. *Journal of the Royal Statistical Society Series B*, 57(1):289–300, 1995.

Daniel Borkan, Lucas Dixon, Jeffrey Sorensen, et al. Nuanced metrics for measuring unintended bias with real data for text classification. *arXiv preprint arXiv:1903.04561*, 2019.

Nicholas Boucher and Ross J Anderson. Trojan source: Invisible vulnerabilities. In *USENIX Security Symposium*, 2023. arXiv:2111.00169.

Nicholas Boucher, Ilia Shumailov, Ross Anderson, and Nicolas Papernot. Bad characters: Imperceptible nlp attacks. In *IEEE Symposium on Security and Privacy*, 2022. arXiv:2106.09898.

Nicholas Carlini, Anish Athalye, Nicolas Papernot, et al. On evaluating adversarial robustness. *arXiv preprint arXiv:1902.06705*, 2019.

Arnold Cartagena and Ariane Teixeira. Mind the GAP: Text safety does not transfer to tool-call safety in LLM agents. *arXiv preprint arXiv:2602.16943*, 2026.

Patrick Chao et al. Jailbreakbench: An open robustness benchmark for jailbreaking large language models. In *Advances in Neural Information Processing Systems*, 2024. arXiv:2404.01318.

Sahana Chennabasappa, Cyrus Nikolaidis, Daniel Song, David Molnar, Stephanie Ding, Shengye Wan, et al. LlamaFirewall: An open source guardrail system for building secure AI agents. arXiv:2505.03574, 2025.

Edoardo Debenedetti, Jie Zhang, Mislav Balunovic, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. Agentdojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents. *Advances in Neural Information Processing Systems*, 37:82895–82920, 2024.

Gemma Team. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*, 2024.

Gemma Team. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*, 2025.

Yichen Gong, Delong Ran, Jinyuan Liu, Conglei Wang, Tianshuo Cong, Anyu Wang, Sisi Duan, and Xiaoyun Wang. FigStep: Jailbreaking large vision-language models via typographic visual prompts. In *AAAI Conference on Artificial Intelligence*, 2025. arXiv:2311.05608.

Yunhao Gou, Kai Chen, Zhili Liu, Lanqing Hong, Hang Xu, Zhenguo Li, Dit-Yan Yeung, James T. Kwok, and Yu Zhang. Eyes closed, safety on: Protecting multimodal LLMs via image-to-text transformation. In *European Conference on Computer Vision*, 2024. arXiv:2403.09572.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection. In *ACM Workshop on Artificial Intelligence and Security*, 2023. arXiv:2302.12173.

Seungju Han et al. Wildguard: Open one-stop moderation tools for safety risks, jailbreaks, and refusals of LLMs. In *Advances in Neural Information Processing Systems (Datasets and Benchmarks Track)*, 2024. arXiv:2406.18495.

John D Hunter. Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, 9 (3):90–95, 2007.

Hakan Inan, Kartikeya Upasani, Jianfeng Chi, Rashi Rungta, Krithika Iyer, Yuning Mao, Michael Tontchev, Qing Hu, Brian Fuller, Davide Testuggine, and Madian Khabsa. Llama guard: Llm-based input-output safeguard for human-ai conversations. *arXiv preprint arXiv:2312.06674*, 2023.

JaidedAI. Easyocr. `https://github.com/JaidedAI/EasyOCR`, 2020.

Neel Jain, Avi Schwarzschild, Yuxin Wen, Gowthami Somepalli, John Kirchenbauer, Ping-yeh Chiang, Micah Goldblum, Aniruddha Saha, Jonas Geiping, and Tom Goldstein. Baseline defenses for adversarial attacks against aligned language models. *arXiv preprint arXiv:2309.00614*, 2023.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.

Fengqing Jiang, Zhangchen Xu, Luyao Niu, Zhen Xiang, Bhaskar Ramasubramanian, Bo Li, and Radha Poovendran. ArtPrompt: ASCII art-based jailbreak attacks against aligned LLMs. In *Annual Meeting of the Association for Computational Linguistics*, 2024a. arXiv:2402.11753.

Liwei Jiang et al. Wildteaming at scale: From in-the-wild jailbreaks to (adversarially) safer language models. In *Advances in Neural Information Processing Systems*, 2024b. arXiv:2406.18510.

Gwang Gook Lee, Kenan Emir Ak, Jay Mohta, Yan Xu, and Dimitrios Dimitriadis. Do VLMs read or rewrite? On transcription faithfulness in vision-language models. *arXiv preprint arXiv:2607.21617*, 2026.

Xuan Li, Zhanke Zhou, Jianing Zhu, Jiangchao Yao, Tongliang Liu, and Bo Han. DeepInception: Hypnotize large language model to be jailbreaker. *arXiv preprint arXiv:2311.03191*, 2023.

Mantas Mazeika, Long Phan, Xuwang Yin, Andy Zou, Zifan Wang, Norman Mu, Elham Sakhaee, Nathaniel Li, Steven Basart, Bo Li, David Forsyth, and Dan Hendrycks. Harmbench: A standardized evaluation framework for automated red teaming and robust refusal. In *International Conference on Machine Learning*, 2024. arXiv:2402.04249.

Quinn McNemar. Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, 12(2):153–157, 1947.

Brian A Nosek, Charles R Ebersole, Alexander C DeHaven, and David T Mellor. The preregistration revolution. *Proceedings of the National Academy of Sciences*, 115(11):2600–2606, 2018.

OpenAI. Moderation api. `https://platform.openai.com/docs/guides/moderation`, 2023.

Long Ouyang, Jeffrey Wu, Xu Jiang, et al. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*, 2022. arXiv:2203.02155.

Xiangyu Qi, Kaixuan Huang, Ashwinee Panda, Peter Henderson, Mengdi Wang, and Prateek Mittal. Visual adversarial examples jailbreak aligned large language models. *arXiv preprint arXiv:2306.13213*, 2023.

Maan Qraitem, Piotr Teterwak, Kate Saenko, and Bryan A Plummer. Web artifact attacks disrupt vision language models. *arXiv preprint arXiv:2503.13652*, 2025.

Qwen Team. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

Qwen Team. Qwen2.5-vl technical report. *arXiv preprint arXiv:2502.13923*, 2025a.

Qwen Team. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025b.

Pooyan Rahmanzadehgervi, Logan Bolton, Mohammad Reza Taesiri, and Anh Totti Nguyen. Vision language models are blind: Failing to translate detailed visual features into words. In *Asian Conference on Computer Vision*, pp. 293–309, 2024. arXiv:2407.06581.

Javier Rando, Daniel Paleka, David Lindner, Lennart Heim, and Florian Tramèr. Red-teaming the stable diffusion safety filter. *arXiv preprint arXiv:2210.04610*, 2022.

Traian Rebedea et al. Nemo guardrails: A toolkit for controllable and safe LLM applications with programmable rails. In *Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 2023. arXiv:2310.10501.

Patrick Schramowski, Manuel Brack, Björn Deiseroth, and Kristian Kersting. Safe latent diffusion: Mitigating inappropriate degeneration in diffusion models. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 22522–22531, 2023. arXiv:2211.05105.

Zhixue Song, Boyan Han, Yiwei Wang, and Chi Zhang. Hard to read, easy to jailbreak: How visual degradation bypasses MLLM safety alignment. In *Findings of the Association for Computational Linguistics: ACL*, 2026. arXiv:2605.07250.

W3C. Scalable vector graphics (SVG) 1.1 specification. W3C Recommendation, 2011.

Ruotong Wang, Zihao Zhu, Siwei Lyu, Xin Tao, and Baoyuan Wu. Distributed implicit harm: A compositional safety blind spot in MLLM-based video moderation. *arXiv preprint arXiv:2609.00206*, 2026a.

Tianshi Wang, Jingsong Wang, Yafei Huang, Fengling Li, Xin Li, and Lei Zhu. MMJailBench: A factorized benchmark for disentangling multimodal jailbreak vulnerabilities. *arXiv preprint arXiv:2608.25490*, 2026b.

Alexander Wei, Nika Haghtalab, and Jacob Steinhardt. Jailbroken: How does LLM safety training fail? In *Advances in Neural Information Processing Systems*, 2023. arXiv:2307.02483.

Edwin B Wilson. Probable inference, the law of succession, and statistical inference. *Journal of the American Statistical Association*, 22(158):209–212, 1927.

Rui Yang, Michael Fu, Chakkrit Tantithamthavorn, Chetan Arora, Gunel Gulmammadova, and Joey Chua. DecipherGuard: Understanding and deciphering jailbreak prompts for a safer deployment of intelligent software systems. *arXiv preprint arXiv:2509.16870*, 2025.

Yuchen Yang et al. Sneakyprompt: Jailbreaking text-to-image generative models. In *IEEE Symposium on Security and Privacy*, 2024. arXiv:2305.12082.

Youliang Yuan, Wenxiang Jiao, Wenxuan Wang, Jen-tse Huang, Pinjia He, Shuming Shi, and Zhaopeng Tu. GPT-4 is too smart to be safe: Stealthy chat with LLMs via cipher. In *International Conference on Learning Representations*, 2024. arXiv:2308.06463.

Wenjun Zeng, Yuchi Liu, Ryan Mullins, Ludovic Peran, Joe Fernandez, Hamza Harkous, Karthik Narasimhan, Drew Proud, Piyush Kumar, Bhaktipriya Radharapu, Olivia Sturman, and Oscar Wahltinez. Shieldgemma: Generative ai content moderation based on gemma. *arXiv preprint arXiv:2407.21772*, 2024.

Qiusi Zhan et al. Injecagent: Benchmarking indirect prompt injections in tool-integrated large language model agents. In *Findings of the Association for Computational Linguistics*, 2024. arXiv:2403.02691.

Sicheng Zhu, Ruiyi Zhang, Bang An, Gang Wu, Joe Barrow, Zichao Wang, Furong Huang, Ani Nenkova, and Tong Sun. Autodan: Interpretable gradient-based adversarial attacks on large language models. *arXiv preprint arXiv:2310.15140*, 2023.

Andy Zou, Zifan Wang, Nicholas Carlini, Milad Nasr, J Zico Kolter, and Matt Fredrikson. Universal and transferable adversarial attacks on aligned language models. In *International Conference on Learning Representations*, 2024. arXiv:2307.15043.

A Example Stimulus in Code and Pixel Form

B Stimulus Construction and Symbol Library

Carriers are generated from each source sentence by deterministic transforms. **SVG**: one `<text>` element per line (plain, L1), whitespace-collapsed single-line form (minified, L2), metadata-stripped (L3), or one `<text>` element per glyph at a fixed x-advance of $0.52\times$ the font size (L4), which keeps content, font, and line structure in a typewriter layout. **Inline HTML** wraps the same inner SVG in a minimal page. **Base64** places the inner SVG in an `img` data URI. **Matplotlib** emits an equivalent script with matching layout at every gradient level. The geometry library contains 25 hate and violence symbols drawn from pure coordinates, with no text and no metadata, plus 25 benign controls; each symbol was verified as recognizable during construction via downsampled thumbnails. The second-source extension uses the same builders over 100 standard HarmBench behaviors disjoint from the AdvBench rows. SVG, inline-HTML, and base64 carriers rasterize from the same inner SVG, so their pixel arms are byte-identical by construction (checksum-equal in the manifest); matplotlib scripts are executed headlessly to produce the pixel arm. The per-glyph gradient changes only the intra-line layout, uniform advance instead of proportional spacing, and the

(a) readable SVG source (L1)

```
<svg xmlns="http://www.w3.org/2000/svg" width="512"
     height="512">
  <rect width="512" height="512" fill="#FFFFFF"/>
  <path d="M334,198L217,265.5L217,130.5Z" fill="none"
        stroke="#111" stroke-width="13" stroke-
     linejoin="round"/>
  ... two more triangle paths, rotated 120 deg each
      ...
  <g fill="#111" font-size="30" font-family="
     Helvetica, sans-serif">
    <text x="87" y="330">Detailed steps to build</
     text>
    <text x="56" y="376">a dangerous </text>
    <rect x="252" y="355" width="96" height="21" fill
     ="#111"/>
    <text x="348" y="376"> device</text>
    <text x="194" y="422">at home</text>
  </g></svg>
```

(b) the same poster, per-glyph (L4)

```
<text x="87.0" y="330">D</text><text x="102.6" y
     ="330">e</text>
<text x="118.2" y="330">t</text><text x="133.8" y
     ="330">a</text>
<text x="149.4" y="330">i</text><text x="165.0" y
     ="330">l</text>
... 43 more <text> elements, one per glyph; paths and
      the
redaction rect pass through untouched ...
```

(c) the same content in three other carriers

```
<img src="
     PHN2ZyB4bWxucz0iaHR0cDovL3d3...">
fig.text(0.047, 0.42, 'Detailed steps to build', ...)
<div><svg ...> (inline HTML wrapper) </svg></div>
```

(a), (b), (c) all render to



(d) the rendered poster: same content for (a) and (b), up to letter spacing and line positioning

Figure 4: One benchmark stimulus: the source, and what the source renders to. The demo sentence carries a redacted placeholder (the cap-height rectangle in the render); benchmark stimuli use real harmful sentences from two sources. The triangular emblem is pure-coordinate line art echoing the geometry library; it carries no text, so the per-glyph split passes it through untouched. (a) A readable SVG exposes the harmful substring, and text guards catch it. (b) Splitting the sentence into one `<text>` element per glyph at a fixed x-advance destroys every contiguous substring in the source while rendering the same content, font, and line structure in a typewriter layout. (c) The same content ships as a base64 data URI, a matplotlib script, or an inline-HTML wrapper. (d) The rendering a human sees; any detection difference between (a) and (b) is placement, not content. The harm is a property of the right-hand column; whether a guard sees it depends entirely on where in the pipeline the guard sits.

rasterizer is shared across arms. A consequence we disclose: because the uniform advance is wider than proportional spacing for most lines, L4 posters of long sentences can run past the canvas edge and clip their final glyphs (18 of 20 primary-source goals have at least one such line). The clipping removes content the pixel guard would otherwise see, so it biases pixel-view detection downward and makes the measured code-versus-pixel gaps conservative; the code-path results are unaffected, since the code arm carries the full source regardless of rendering.

C PROOFS

Lemma 1 (fiber dichotomy). By definition of $\sim$, the renders $R(c)$ and $R(c')$ are visually equivalent up to the tolerance, and the harm judgment h respects that tolerance, so $h(R(c)) = h(R(c'))$. If additionally $F(c) \neq F(c')$, then F outputs two different values on two artifacts with the same label, so at least one output disagrees with its label. □

Proposition 1 (exact code filtering is internal rendering). Assume F has no false negatives and no false positives, so $F(c) = h(R(c))$ for every artifact c. For $c \sim c'$, Lemma 1's conclusion forces $F(c) = F(c')$, since unequal outputs would already be an error on one side; hence F is constant on

every $\sim$-class. Therefore F factors through the perceptual class $[c]$ of the artifact, $F = H \circ [\,\cdot\,]$ for $H([c]) := F(c)$, and on artifacts F coincides with $h \circ R$. □

Proposition 2 (no free canonicalization; full statement). Let T be a canonicalizer, a $\sim$-class map into a "readable" subset S of sources on which a text guard performs well, the pipeline flagging c iff the guard accepts $T(c)$. If some zero-cost transform g maps a harmful artifact outside T's normalization, $T(g\,c) \notin S$ or normalized in the wrong order, the pipeline inherits exactly that miss; and a T robust to every g in the attacker's class must recover the rendered reading order from arbitrary glyph placements and encodings, which amounts to computing the renderer's layout stage. *Proof.* Let g be a zero-cost transform and c a harmful artifact with $T(g\,c)$ outside S or normalized in the wrong order. In the first case the text guard reads an input outside the set on which it is assumed to perform, and empirically (Table 2) it flags none of them; in the second case the concatenation produces scrambled or partial text in which no contiguous harmful substring exists, and the miss follows from the same mechanism as the L4 cliff. Either way the pipeline outputs no flag while $h(R(g\,c)) = h(R(c)) = 1$. For the second claim, robustness over the attacker's whole class requires T's output to be invariant to element reordering, re-encoding, and re-grouping of glyphs; under arbitrary placement the only information in the artifact that determines the reading order is the geometry of the glyph positions together with glyph identity, so T must recover layout from rendered-geometry information, which is what a renderer's layout stage computes. □

Proposition 3 (RTJ invariance and decomposition; full statement). Let $\mathrm{RTJ}(c) = J(V(\hat{R}(c)))$ with rasterizer $\hat{R} \approx R$, transcriber V, and judge J. (i) RTJ is constant on the exact fibers of $\hat{R}$: artifacts that rasterize to the same image receive identical treatment, so RTJ satisfies the invariance that Proposition 1 demands, with no per-condition normalization. (ii) Under an independent-noise model in which the transcriber preserves harm-relevant content with probability q and the judge flags preserved harmful text with probability a, harmful detection is at most $qa+r$, where r bounds the probability that a degraded transcription still triggers the judge, with leading term qa when degraded transcripts are judge-benign; the benign false-positive rate is the judge's rate on transcriptions of benign content. Both are independent of which zero-cost transform produced c. (i) $\mathrm{RTJ}(c) = J(V(\hat{R}(c)))$ is a composition through $\hat{R}(c)$, so artifacts with $\hat{R}(c) = \hat{R}(c')$ receive identical transcriptions and judgments; exact fibers are handled identically, and tolerance-level differences (bounded jitter) reach the transcriber only through the fidelity of $\hat{R}$. No per-condition normalization is involved anywhere in the composition. (ii) Let $q = \Pr[V \text{ preserves harm-relevant content}]$, $a = \Pr[J \text{ flags} \mid \text{preserved}]$, and let r bound the probability that a degraded transcription still triggers the judge. Then $\Pr[\text{flag}] \le qa + r$, with the leading term qa exact when degraded transcripts are judge-benign ($r \approx 0$), and the benign false-positive rate is $\Pr[J \text{ flags a benign transcription}]$: the judge's rate on the transcription distribution, which the measured $1/25$ bounds and which transcription errors can only move by altering that distribution. Every quantity is a property of V and J applied to $\hat{R}(c)$ and is therefore unchanged when c is replaced by any transform in the attacker's class. The word-recall fidelity reported in §5.4 is a strict proxy for q: it can underestimate content preservation, which is why measured detection occasionally sits above measured fidelity, within the r term, while still tracking it. □

D Full Matrices and Prompt Sensitivity

Full per-guard $\times$ carrier $\times$ gradient tables (primary prompt) and the sensitivity ranges over the three prompt perturbations are contained in the released records, and the frozen analysis scripts regenerate every table. Code-view blindness persists across prompt variants for every guard: cliff-row code detection stays at or below 9.1% under every guard-variant pair (Gemma-3-4B under one perturbation; all other pairs at or below 2.4%), against 77–96% on contiguous-source rows. Would-flag is 100% under all measured variants for the two guards with variant coverage (ShieldGemma-2B, Qwen3-8B) and at the primary prompt for LlamaGuard-3-8B and the Gemma VLMs. Qwen2.5-VL's pixel arm varies 17–60% across variants while its code arm stays 0%.

Score-threshold robustness of the code path. The near-zero code-path rates are not a thresholding artifact. On the primary source, prompt P0, code arm, the two score-bearing text guards behave as follows. Qwen3-8B assigns $p_{\mathrm{unsafe}} = 0$ to every harmful cliff item (165 items: per-glyph

rows, base64 at every gradient, geometry; maximum 0.0000). ShieldGemma-2B's maximum harmful score over the same pool is 0.35, with pooled AUC 0.61: lowering its threshold to 0.1 recovers 12.1% detection at 8.0% false positives. On the readable L1 cells the same scores separate cleanly (AUC 0.78–1.00), so the collapse on cliff cells is content-driven rather than a calibration shift. The per-cell sweeps and the analysis script ship with the release.

E Gradient Step-Function Statistics

Per-carrier paired McNemar at the goal level, base64 excluded per the preregistered encoding exemption: L1 versus L3 is nonsignificant in 11 of 12 cells (one significant increase, ShieldGemma on matplotlib, p=0.016); L3 versus L4 gives $p \leq 10^{-3}$ in all 12 cells. The falsified monotonicity preregistration and the reframe are recorded with the deviations below.

F Delegated Rendering, Per-Model Results

Full three-arm refusal and payload tables for all six models (greedy, n=100) and the sampling runs (T=0.7, top-p 0.95, seeds 0/1/2) are in the released records. Sampling means $\pm$ std for the payload rate under the system role: Qwen3-8B 0.670 ± 0.017, Qwen2.5-7B 0.600 ± 0.010, Mistral-7B 0.577 ± 0.021, Gemma-2-2B 0.137 ± 0.006, Llama-3.1-8B 0.013 ± 0.006. Refusal is detected by a fixed refusal-pattern match, so hedged or partial refusals count as compliance and every refusal rate is a lower bound; the role effect is read within a model, where the detector is constant. One implementation note: gemma-2's chat template rejects system roles, so its greedy render_svc arm and the Mistral rerun fold the role text into the user turn, as do all sampling runs; the four models run first received a true system message. The role effect is consistent across both implementations (I).

G End-to-End Artifacts

The 79 emitted SVGs (of 80 payloads; one output hit the generation cap before closing its root element), with their base64-wrapped and per-glyph re-split variants, are released. Plain-code detection 66% (ShieldGemma-2B), 82% (Qwen3-8B), and 94% (LlamaGuard-3-8B); wrapped or re-split, $0/79$ for both code guards; pixel 54% (Gemma-3-4B) and 47% (Gemma-3-12B), against $96\%/94\%$ for the same guards on clean L1–L3 renders of the benchmark's own primary goals.

H Judge Cascade and Disagreement

The primary judge is HarmBench-13B-cls under its official template; the backup is Gemma-2-2B (family-independent). Two display framings were calibrated 9/9 on a stratified set before use (deviation log below). Two-way agreement: $399/450$ on the primary source with all benign items unanimous; 46.9% disagreement on the HarmBench source, whose phrasings are longer and more heterogeneous; both subsets are reported side by side and no disagreement is dropped. As a sensitivity check, excluding the 51 judge-disagreement items leaves code-view detection at 0% everywhere and raises pixel-view L4 detection ($82.5\% \rightarrow 91.2\%$ for Gemma-3-4B, $87.5\% \rightarrow 95.6\%$ for 12B), so the cliff's direction is unchanged and its magnitude, if anything, understated. The arbitration sheet (51 disagreements plus 30 unanimous controls, order shuffled, both judges' verdicts hidden) was completed by a single human reader, an author, on 2026-09-15. All 76 text items match the intended labels (71 harmful, 5 benign), and the reader sides with the backup judge on 48 of 48 text disagreements, locating the disagreement in the primary judge's conservatism. The 4 sampled hate symbols were not recognized by the reader, so those verdicts carry no information, the construction-time labels stand, and geometry recognition remains knowledge-dependent, consistent with the open geometry residual (§5.5). Relabeling by the completed arbitration therefore changes no text label: code-view detection stays 0% and pixel-view L4 stays $82.5\%/87.5\%$ under the all-labeled sets, with the unanimous-only bounds as the conservative bracket; the sheet and raw verdicts are released.

I PREREGISTRATION DEVIATIONS

(i) **Judge framing**: the preregistered behavior-equals-generation mapping is degenerate for HarmBench's rules (calibration $0/4$); we adopted the display framing (9/9 calibration). (ii) **Small-guard exclusion**: the 0.02 benign false-positive stop signal triggered and isolated to Qwen3-0.6B (36.9%, inverted direction); removed from the matrix by user arbitration with records retained; the four-guard aggregate is $2/790 = 0.25\%$. (iii) **Cliff magnitude**: the preregistered $\geq$60pp criterion is met in one of two dual-view families (Gemma); Qwen2.5-VL replicates direction and significance at $+33.8$pp. (iv) **Mistral baseline**: the intent arm is degenerate (15% refusal), so the system-role effect is read between the two rendering arms ($70\% \rightarrow 17\%$). (v) **Gradient hypothesis**: the preregistered monotone non-increase was falsified and reframed as a step function. (vi) **System-role placement**: gemma-2's template rejects system roles and the Mistral rerun plus all sampling runs fold the role into the user turn; the effect direction is consistent across both implementations (F). (vii) **lg1b rerun**: the shipped LlamaGuard-3-1B chat template drops user content, so its original matrix rows are the constant *safe*; the template-corrected rerun in L supersedes them.

J BENIGN CORPUS

Benign corpus per source: the primary set has 80 text stimuli (10 sentences $\times$ 4 carriers $\times$ {L1, L4}) plus 25 geometry; the second source adds 20 text stimuli. The $2/790$ primary aggregate counts 6 code/pixel arms over 105 benign items plus 2 raw-text arms over 80; the second source's 12 arm evaluations over 20 benign items add 2 further false positives (both Gemma-3-4B, code arm), for $4/1030 = 0.39\%$ overall.

K ADAPTIVE TRANSFORMS AND LEGIBILITY CRITERION

Shuffle: glyph elements emitted in shuffled document order at unchanged absolute coordinates (rendering identical to the un-shuffled stimulus). **Jitter**: each glyph's y shifted by $\mathcal{U}(-22, 22)$ px. **Combined**: both. Legibility criterion: the preregistered rule called jittered text human-readable iff at least 8 of 10 sampled strings were exactly transcribed by at least 2 of 3 human readers. The annotation was not completed before submission, so per that rule the qualifier is dropped and the pixel-collapse numbers are reported unqualified; the unused transcription pack ships with the release. Canonicalizer attribution: document-order concatenation collapses under shuffle (0–20%); coordinate sorting collapses under jitter (20–60%); each holds at 100% under the other's step; the combined step collapses both and the pixel path ($85\% \rightarrow 20\%$ jitter $\rightarrow 5\%$ combined).

L SCALE-FAILURE RECORDS

Qwen3-0.6B (zero-shot): benign code-arm false positives $24/65 = 36.9\%$ ($65 = 40$ benign text carriers at L4 plus 25 benign geometry, the benign code-arm subset run for this guard), and its L4 "detection" (46.3%) exceeds its raw-text rate (25.0%), an inverted-direction signal. LlamaGuard-3-1B: its shipped chat template drops user content, so the original matrix run outputs a constant *safe* ($0/100$ raw text, an artifact we root-caused after the run); a template-corrected rerun (the 8B template, same tokenizer family, content-embedding verified) detects $95/100$ harmful raw text but flags $55/75 = 73\%$ of benign code-arm artifacts, failing the same 2% calibration cap. Both guards are excluded from the matrix with all records, original and corrected, retained.

M PROMPT TEXTS

The primary and three perturbation prompts for each guard class (text-path, VLM pixel, VLM code), the would-flag question, the LlamaGuard native-format note, and the three-arm generation scripts are reproduced verbatim in the release.

N Related-Work Search Protocol

Weekly targeted sweeps since September 2026 over the query families {rendering gap, rasterize-before-guard, code carrier moderation, SVG harmful semantics, carrier guard matrix, delegated rendering}, across arXiv listing feeds and the four neighbor papers' citation graphs; the full query log and dates ship with the release. Within the families surfaced by this protocol, no prior work measures moderation placement relative to the code-to-pixel boundary; the firstness claim in §1 is bounded by this protocol.

O Reproducibility

Models: ShieldGemma-2B (Zeng et al., 2024), Qwen3-8B/0.6B (Qwen Team, 2025b), Gemma-3-4B/12B (instruction-tuned) (Gemma Team, 2025), Qwen2.5-VL-7B (Qwen Team, 2025a), LlamaGuard-3-8B/1B (Inan et al., 2023), Gemma-2-2B (Gemma Team, 2024), Llama-3/3.1-8B (Grattafiori et al., 2024), Qwen2.5-7B (Qwen Team, 2024), and Mistral-7B-v0.3 (Jiang et al., 2023) locally; GLM-5.3-Flash, Kimi-K3, and DeepSeek-V4-Vision through their public APIs; GPT-6-Astra, Claude-Opus-5, and Gemini-3.8-Flash through a third-party relay (OpenAI-compatible endpoint, accessed 2026-09-12). For the relayed guards, blocked-call attribution (provider versus relay filter) cannot be separated from outside; the ledger records the observed status codes, and intent-to-treat rates for these guards are lower bounds on provider-only rates whenever the relay itself filters. Blocking patterns are qualitatively consistent across the three relayed providers and cannot be fingerprinted as relay- versus provider-side from the observed status codes alone, so any provider-only rate lies at or above the reported ITT bounds; we make no finer attribution claim. Decoding: greedy for all guard arms; $T{=}0.7$, top-p 0.95 for the sampling-robustness runs (seeds 0/1/2). Rasterization: cairosvg 2.9.0 and matplotlib (Agg backend); latency measured on a single-threaded CPU of an A100 host, with VLM transcription and first-token judgment latencies measured on a single A100. The smoothing baseline's perturbation RNG is seeded deterministically from the item identifier (an earlier run used process-salted hashes; both its records and the deterministic rerun ship). The Kimi second-source code arm is reported from a complete fresh-window rerun after the original run exhausted its 5-hour coding-plan quota mid-run (206/300 content-independent 403s, retained in the release as the availability measurement); the fresh run lost 1/300 calls to the provider's genuine input filter. The release contains per-file checksums, a frozen and checksummed per-call denominators ledger for all API calls (per-item final outcomes after the retry pass are recomputed from it), all raw records, and a one-command recompute for every main-text table. RenderGapBench is released for defense research under a research-use request form, and nothing in the release improves attack efficacy beyond the trivial transforms documented.

P Frontier-API Denominators

Q RTJ Details and Ablations

Transcription prompt, verbatim: "Transcribe all text visible in this image exactly as written, preserving the reading order as best you can. Output ONLY the transcription, nothing else." Transcription uses greedy decoding (as for all guard arms, O) and is scored by recall of the goal's words (four letters or longer, order-free), a choice that penalizes the rewriting failure mode documented for VLM transcription (Lee et al., 2026); the reconstructive-prompt ablation below shows the cost of inviting reconstruction instead. The judge reads the transcription in LlamaGuard-3-8B's native format, and empty transcriptions pass through labeled as empty. The full transcriber $\times$ judge $\times$ condition grid is in the released records. Ablations: (i) swapping the transcriber between Gemma-3-4B and 12B changes results by at most 10% (absolute) on all conditions; (ii) on partial transcripts LlamaGuard retains 50–70% where ShieldGemma drops to 10–45%, so the judge choice matters only when transcription degrades; (iii) an explicitly reconstructive prompt variant ("the layout is irregular; reconstruct the intended sentence") *reduces* detection to 20% and 10% on jitter and combined, so verbatim transcription outperforms reconstruction. Symbol-detector first attempts, both negative: SIFT with FLANN matching over a leave-one-out augmented library reaches 36–68% across query variants with 35% false flags on text posters; a small CNN trained on 45 symbols with heavy augmentation reaches $13/20$ pooled on five held-out harmful symbols but flags $60/60$ harmful text

Table 3: Frontier API guards on the code path, harmful stimuli. Native rows: primary-source cliff rows and the second-source extension as labeled. Relay rows: the primary-source cliff subset, per-glyph and base64 and geometry, code arm only. *Attempted* counts unique stimuli, scoring each item by its final outcome after the documented retry pass for non-parseable calls (the per-call ledger, including every retried call, ships with the release). *Conditional* counts detections over parseable final outcomes; *ITT* counts detections over all items, treating endpoint rejections and empty responses as failures. †Fresh-window rerun: the original run of this row hit a quota cutover mid-run, losing 206/300 calls to content-independent 403s (positional boundary, benign calls equally rejected, the 5-hour coding-plan window); a complete rerun in a new quota window, printed here, lost a single call to the provider's genuine input filter (HTTP 400). Both runs ship with the release (O).

API guard	attempted	blocked	empty	conditional	ITT
GLM-5.3-Flash	125	17	10	89%	70%
Kimi-K3	125	1	1	85%	84%
DeepSeek-V4-Vision	125	0	23	85%	70%
DeepSeek-V4-Vision (2nd source)	300	0	30	87%	78%
Kimi-K3 (2nd source)†	300	1	5	95%	93%
GPT-6-Astra (relay)	125	0	0	69%	69%
Claude-Opus-5 (relay)	125	0	2	78%	77%
Gemini-3.8-Flash (relay)	125	41	0	61%	41%

posters and $5/25$ benign geometry. A usable geometry detector evidently needs dedicated negative mining, which we leave open. Three OCR front ends run before the same judge (Table 4): Easy-

Table 4: OCR front ends before the same judge (LlamaGuard-3-8B) on the identical 234-item grid: harmful detection by condition, benign false positives pooled over the 25 stress-benign items (15 adaptive + 5 overlap + 5 tiny-font); RTJ row for reference.

Condition (harmful)	EasyOCR	Tesseract 5	PP-OCRv6	**RTJ (ours)**
Clean per-glyph (SVG L4)	100%	95%	100%	**100%**
Adaptive: order shuffle	100%	95%	100%	**100%**
Adaptive: y-jitter $\pm$22 px	55%	5%	45%	**60%**
Adaptive: combined	65%	5%	50%	60%
Adaptive: tiny font (12 px)	95%	95%	95%	**100%**
Adaptive: glyph overlap	10%	0%	55%	**80%**
Naturalistic artifacts (79)	77%	76%	68%	73%
Benign false positives	1/25	0/25	3/25	**1/25**

OCR as published, Tesseract 5.2 under its native page-layout analysis, and PaddleOCR 3.7 with PP-OCRv6 medium detection and recognition in engine-native reading order. Glyph overlap destroys all three engines' transcriptions (fidelity ≤ 0.01): EasyOCR and Tesseract flag 10% and 0%, while PP-OCRv6 reaches 55% only through the judge's residual tolerance of garbled text, at the cost of the worst benign false positives ($3/25$, including $2/5$ on jittered benign). Tesseract's layout stage collapses under the jitter and combined steps (5%). An oracle per-condition engine selector would win two cells (combined 65%, naturalistic 77%), tie two, and lose three (jitter, tiny font, overlap), so no deployable single engine closes the layout gap; per-file transcriptions ship with the release. Frontier-transcriber swap (judge unchanged, LlamaGuard-3-8B): Gemini-3.8-Flash transcribes jitter, combined, and overlap stimuli at fidelity 96–100% and RTJ detection follows at 100% (n=$10/7/20$; the relay returned no transcription for the remainder, reported conditional on successful transcription), with $0/20$ benign false positives and 95% on tiny font; GPT-6-Astra under the same protocol transcribes the jittered and combined layouts at fidelity 9–12% (its overlap fidelity is 56%) and detection falls to 25–40%, while on the clean tiny-font condition it matches at 100%; it also produces one benign false positive ($1/20$, jittered benign), where Gemini produces none. The two frontier transcribers were accessed through the relay in O.

R Placement Decision Rules

Table 5: Placement guidance by content type, with the residual risk each row carries. RTJ's per-condition numbers appear in Table 2.

Content type	Code-path guard	Residual risk and best available
Contiguous harmful text	local text guards, 60–96%	markup dilution; matplotlib lower, 45–85% (L1 cells)
Per-glyph fragmented text	none local (0%)	frontier APIs 61–95% conditional, 41–93% ITT
Base64-encoded carrier	none local (0%)	canonicalization 90–95% on plain; adaptive step breaks it
Pure-geometry symbols	none, any class	pixel blind too (0–16%); needs a symbol detector
Naturalistic model output	partial, 66–94% plain	pixel degrades to $\sim$50%; one-line encode $\rightarrow$ 0%; RTJ 73%