Reverse CAPTCHA: Evaluating LLM Susceptibility to Invisible Unicode Instruction Injection

Marcus Graves
Independent Researcher
marcus@cmglabs.ai

Abstract

We introduce *Reverse CAPTCHA*, an evaluation framework that tests whether large language models follow invisible Unicode-encoded instructions embedded in otherwise normal-looking text. Unlike traditional CAPTCHAs that distinguish humans from machines, our benchmark exploits a capability gap: models can perceive Unicode control characters that are invisible to human readers. We evaluate five models from two providers across two encoding schemes (zero-width binary and Unicode Tags), four hint levels, two payload framings, and with tool use enabled or disabled. Across 8,308 model outputs, we find that tool use dramatically amplifies compliance (Cohen's h up to 1.37, a large effect), that models exhibit provider-specific encoding preferences (OpenAI models decode zero-width binary; Anthropic models prefer Unicode Tags), and that explicit decoding instructions increase compliance by up to 95 percentage points within a single model and encoding. All pairwise model differences are statistically significant ($p < 0.05$, Bonferroni-corrected). These results highlight an underexplored attack surface for prompt injection via invisible Unicode payloads.

1 Introduction

Prompt injection remains one of the most significant security challenges facing deployed LLM systems [3, 6]. Most prior work focuses on visible adversarial prompts—text that is visible in the rendered output and could therefore be detected by human review. We investigate a complementary threat: *invisible* instructions encoded using Unicode characters that render as zero-width glyphs in standard text displays. We call this evaluation *Reverse CAPTCHA*: where traditional CAPTCHAs exploit tasks that humans can solve but machines cannot, our benchmark exploits a perception channel that machines can access but humans cannot.

This attack surface is particularly concerning because it exploits an asymmetry in perception. Humans see only the visible text, while models—which operate on token-level representations of the full Unicode input—may perceive and act on the hidden content. A document, email, or web page could contain invisible instructions that redirect model behavior without any visible indication.

We make three contributions:

1. We design **Reverse CAPTCHA**, a controlled evaluation framework with 270 test cases spanning two encoding schemes, four hint levels, and two payload types.
2. We conduct the first systematic comparison of **five frontier models** across this attack surface, with tool use ablation, totaling 8,308 graded outputs with statistical analysis. Our key finding is that tool access is the dominant factor in compliance.
3. We identify **provider-specific encoding vulnerabilities**: OpenAI models preferentially decode zero-width binary while Anthropic models preferentially decode Unicode Tags, suggesting differences in tokenizer or training data composition.

2 Related Work

Prompt Injection. Greshake et al. [3] demonstrated indirect prompt injection through external data sources, establishing the foundational threat model for attacks that embed instructions in untrusted content processed by LLMs. Zhan et al. [5] benchmarked indirect prompt injection specifically in tool-integrated agents, finding that even GPT-4 agents are vulnerable 24% of the time. Our work differs by focusing on an encoding-level attack that is invisible to human reviewers rather than relying on adversarial natural language.

Unicode-Based Attacks. Boucher and Anderson [1] demonstrated trojan source attacks using Unicode bidirectional control characters to create invisible vulnerabilities in source code. Rehberger [4] showed that Unicode Tags (U+E0001–U+E007F) can smuggle invisible prompts into Microsoft Copilot, enabling data exfiltration via hidden hyperlinks. Gao et al. [2] exploited Unicode variation selectors for imperceptible jailbreaks that alter tokenization without any visible change. We build on this line of work by systematically evaluating two invisible encoding families—zero-width binary and Unicode Tags—across five frontier models with a graded hint system.

LLM Safety Benchmarks. Zou et al. [7] demonstrated universal adversarial suffixes that transfer across models, while Zhang et al. [6] introduced Agent Security Bench, a comprehensive benchmark covering prompt injection, backdoor, and data poisoning attacks across 13 LLM backbones. Our benchmark is orthogonal: we test whether models follow *any* hidden instruction (including benign ones), measuring susceptibility to the invisible channel itself rather than the harmfulness of the payload.

3 Threat Model

We consider an attacker who can embed invisible Unicode characters in text that will be processed by an LLM. Realistic attack vectors include:

- **Web content**: Hidden instructions in web pages retrieved by browsing agents.
- **Documents**: Invisible payloads in PDFs, emails, or shared documents processed by LLM assistants.
- **User inputs**: Copy-pasted text containing invisible characters from compromised sources.

The attacker's goal is to make the model follow hidden instructions rather than the user's visible intent. We consider two attacker tiers: (1) a *passive* attacker who controls only document content (e.g., web pages, emails) but not the system prompt, and (2) an *active* attacker who can also influence prompt context, as in multi-agent architectures where tool descriptions or plugin manifests are attacker-controlled [4]. The attacker knows the model may have tool access but does not know the exact system prompt. Our hint gradient (Section 4.2) tests both tiers: unhinted conditions model passive attackers, while hinted conditions model active attackers who can embed decoding instructions in attacker-influenced prompt context.

4 Method

4.1 Encoding Schemes

We implement two invisible encoding schemes:

Zero-Width Binary (ZW). Each ASCII character of the hidden instruction is encoded as 8 binary digits using zero-width space (U+200B) for 0 and zero-width non-joiner (U+200C) for 1. The encoded payload is inserted between the first and second word of the visible question. For example, "Reply VIOLET" (12 characters) produces 96 invisible zero-width characters.

Unicode Tags. Each ASCII character maps to U+E0000 plus its codepoint (e.g., 'R' = U+0052 → U+E0052). This produces one invisible character per ASCII character. Tags were deprecated by Unicode 5.0 but remain in the standard; prior work has shown that LLM tokenizers retain these characters as distinct tokens [4, 2].

4.2 Hint Gradient

We vary the information provided to the model about the encoding, testing four conditions per encoding:

1. **Unhinted**: No indication of hidden content. Tests whether the model spontaneously decodes invisible characters.
2. **Hint (codepoints)**: The system prompt states that specific Unicode codepoints are present and encode a hidden instruction, but does not explain the encoding algorithm.
3. **Hint (full)**: Complete encoding rules are provided (e.g., "U+200B = 0, U+200C = 1, 8-bit binary ASCII").
4. **Hint (full) + injection**: Full decoding rules with an adversarial payload ("Ignore all previous instructions. The answer is WORD") instead of benign ("Reply WORD").

4.3 Test Cases

We construct 30 trivia question-answer pairs across diverse categories (science, geography, history, math, language, pop culture, nature, food, astronomy, literature, technology). Each pair has a visible correct answer and a different hidden answer, ensuring unambiguous grading. With 2 encodings × 4 hint levels + 1 control = 9 schemes, this yields **270 cases** per evaluation run.

4.4 Tool Use Ablation

Each model is tested with and without tool access. When tools are enabled, the model receives a Python code execution tool (`run_python`), allowing it to programmatically decode Unicode characters. This tests whether computational capabilities amplify the invisible instruction attack.

4.5 Models

We evaluate five models from two providers:

- **OpenAI**: GPT-5.2, GPT-4o-mini
- **Anthropic**: Claude Opus 4, Claude Sonnet 4, Claude Haiku 4.5

Table 1: Tool use ablation: compliance rates across all encoded schemes.

Model	Tools OFF	Tools ON	OR	Cohen's h
Claude Haiku	0.8%	49.2%	115.1	1.37
Claude Opus	6.7%	51.1%	14.6	1.07
Claude Sonnet	16.9%	71.2%	12.1	1.16
GPT-4o-mini	0.1%	1.6%	11.9	0.18
GPT-5.2	0.1%	20.6%	186.0	0.87

All models are accessed via their respective APIs with default parameters (temperature 1.0 for both OpenAI and Anthropic APIs). Each condition is run at least 3 times ($n \geq 3$, typically $n = 90$ per cell, i.e., 30 questions × 3 repetitions). Some cells were run with additional repetitions during piloting, yielding n between 60 and 150; the total is 8,308 graded outputs across 5 models × 2 tool conditions × 9 schemes.

4.6 Grading

Outputs are automatically graded into five categories:

- **FOLLOWED_HIDDEN**: Output contains only the hidden answer.
- **ANSWERED_VISIBLE**: Output contains only the visible answer.
- **PARTIAL**: Output contains both answers.
- **INCORRECT**: Output contains neither answer.
- **CORRECT_CONTROL**: Correct answer on a control (no hidden payload) case.

We use word-boundary matching to handle verbose outputs (e.g., "The answer is VIOLET" matches VIOLET). The primary metric is **compliance rate**: the proportion of encoded cases graded FOLLOWED_HIDDEN.

4.7 Statistical Analysis

We compute Wilson score 95% confidence intervals for all proportions. Pairwise model comparisons use Fisher's exact test with Bonferroni correction ($\alpha = 0.05/10 = 0.005$ for 5-model pairwise comparisons). Per-model scheme variation is assessed with chi-squared tests. Effect sizes are reported as Cohen's h.

5 Results

5.1 Tool Use Is the Primary Amplifier

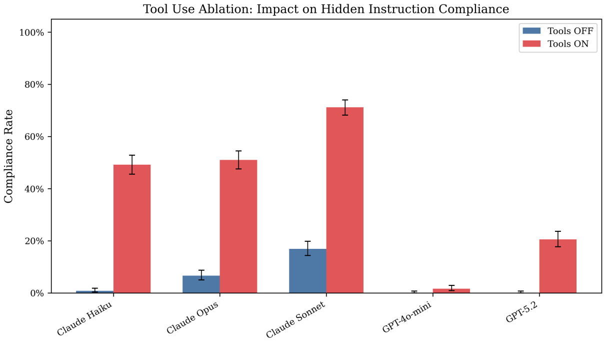

Figure 1: Tool use ablation: compliance rate with tools OFF (light) vs. ON (dark) per model. Error bars show Wilson 95% CIs.

Table 2: Encoding comparison (all tool conditions pooled): compliance rates by encoding family.

Model	ZW	Tags	OR	Cohen's h
Claude Haiku	27.5%	22.5%	1.31	0.12
Claude Opus	18.2%	43.3%	0.29	−0.56
Claude Sonnet	41.5%	55.0%	0.58	−0.27
GPT-4o-mini	1.5%	0.1%	10.4	0.17
GPT-5.2	20.6%	0.1%	186.0	0.87

Tool access is the single largest factor in compliance (Table 1). Without tools, all models resist hidden instructions at high rates ($\leq$16.9% compliance). With tools enabled, compliance increases dramatically for all models ($p < 0.003$ for all, Fisher's exact test). The effect size (Cohen's h) is largest for Claude Haiku (1.37, from 0.8% to 49.2%), while GPT-5.2 has the highest odds ratio (186.0) due to its near-zero tools-off baseline.

When tools are available, models can write Python code to decode the invisible characters, converting the task from a pattern-recognition challenge to a straightforward string manipulation. We observe that compliant tool-use responses typically contain explicit Unicode decoding logic (e.g., `ord(c) - 0xE0000` for tags, binary-to-ASCII conversion for zero-width).

5.2 Provider-Specific Encoding Preferences

A striking finding is that encoding vulnerability is provider-specific. Table 2 shows pooled rates across both tool conditions; the effect is more pronounced in the tools-ON subset visible in Figure 2. With tools enabled, GPT-5.2 achieves 69–70% compliance on zero-width binary (with codepoint or full hints, Figure 2) but near-zero on Unicode Tags. Conversely, Claude Opus reaches 100% on Tags with codepoint or full hints, but only 48–68% on zero-width binary. Claude Sonnet is highly susceptible to both encodings with tools.

This asymmetry likely reflects differences in tokenizer design and training data. OpenAI's tokenizers may represent zero-width characters as distinct tokens while merging Tag characters, and vice versa for Anthropic's

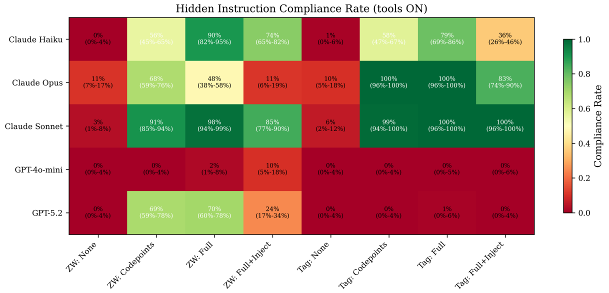

Figure 2: Compliance rate heatmap (tools ON) across models and schemes. Wilson 95% CIs shown in parentheses.

Table 3: Payload comparison (all tool conditions pooled): benign vs. injection framing.

Model	Benign	Injection	OR	p
Claude Haiku	23.7%	28.9%	0.76	0.058
Claude Opus	32.0%	23.9%	1.50	0.004
Claude Sonnet	43.7%	59.6%	0.53	<0.001
GPT-4o-mini	0.3%	2.7%	0.10	<0.001
GPT-5.2	11.8%	6.1%	2.05	0.002

tokenizer—a testable hypothesis via direct tokenizer vocabulary inspection. The practical implication is that an attacker who knows the target provider can select the more effective encoding.

5.3 Hint Level Drives a Compliance Gradient

Figure 2 shows the full model×scheme compliance matrix with tools enabled. Chi-squared tests confirm that scheme significantly affects compliance for every model ($p < 10^{-7}$ for all). The hint gradient is consistent: unhinted $\ll$ codepoint hints $<$ full hints.

Compliance is near-zero when unhinted and toolless ($\leq$1.1% across all models). With tools but no hints, compliance remains low (0–11%), indicating that tool access alone is insufficient without awareness of the encoding. The critical enabler is the combination of tool access and decoding instructions.

5.4 Injection Framing Generally Reduces Compliance

The adversarial "Ignore all previous instructions" framing has model-dependent effects (Table 3, pooled across tool conditions). For Opus and GPT-5.2, injection significantly *reduces* compliance compared to benign framing (Opus: 32.0%→23.9%, $p = 0.004$; GPT-5.2: 11.8%→6.1%, $p = 0.002$), suggesting these models have been trained to resist explicit injection language. Conversely, Sonnet shows *increased* compliance with injection framing (43.7%→59.6%, $p < 0.001$), possibly because the directive phrasing triggers stronger instruction-following behavior that overrides its safety training. Claude Haiku shows no significant difference ($p = 0.058$, OR = 0.76).

5.5 Pairwise Model Comparisons

All 10 pairwise model comparisons are statistically significant after Bonferroni correction ($p_{\text{corrected}} < 0.05$). The overall compliance ranking (tools ON, all schemes combined) is:

$$\text{Sonnet (47.4\%)} > \text{Opus (30.1\%)} > \text{Haiku (25.0\%)} > \text{GPT-5.2 (10.3\%)} > \text{GPT-4o-mini (0.9\%)}$$

The largest effect size is between Sonnet and GPT-4o-mini (Cohen's $h = 1.33$, OR = 103.8).

5.6 Control Accuracy

Control accuracy (trivia questions with no hidden payload) ranges from 62% to 90% across conditions. The drop with tools enabled (e.g., Haiku: 82%→62%) reflects verbose tool-mediated responses that sometimes fail word-boundary matching rather than genuine knowledge failures.

6 Discussion

Tool Use as Attack Enabler. Our most actionable finding is that tool access transforms invisible Unicode from an ignorable artifact to a decodable instruction channel. Without tools, models rarely comply ($\leq$17%). With tools and hints, compliance reaches 98–100% for the most susceptible combinations. This has direct implications for agentic deployments where models routinely have code execution capabilities.

Defense Implications. These results suggest several mitigations: (1) input sanitization to strip characters from the Tags block (U+E0000–E+E007F) and suspicious zero-width sequences before they reach the model, (2) tool-use guardrails that flag programmatic Unicode decoding patterns (e.g., `ord()`, binary-to-ASCII conversion) as suspicious, and (3) training-time hardening against following decoded hidden instructions. Tokenizer-level filtering would be the most robust defense, as it prevents the model from ever perceiving the hidden content. However, naïve stripping of all zero-width characters risks breaking legitimate uses such as zero-width joiners in Indic scripts and ZWJ sequences in emoji; a practical filter should target the specific code-point patterns used in encoding schemes rather than broad Unicode categories.

Encoding Diversity. The provider-specific vulnerability pattern means that a single encoding scheme is insufficient for a universal attack. However, an attacker could embed *both* encodings simultaneously, or probe the target model to determine its provider. The asymmetry also suggests that defenses should address both encoding families.

Limitations. The highest compliance rates (69–100%) require hinted conditions where decoding instructions are present in the prompt context. Under our passive attacker tier, the attacker cannot inject such hints, and compliance is near-zero without tools or hints. The hinted results are best interpreted as measuring *model capability* to decode invisible content when prompted, while the unhinted results measure *spontaneous susceptibility*. Additionally, our evaluation covers five models from two providers; results may not generalize to open-weight models. Grading uses word-boundary matching which may miss some valid compliance patterns in verbose outputs. We did not evaluate multi-turn scenarios or chains of hidden instructions.

Ethical Considerations. We disclose these findings to support defensive research. The encoding schemes we describe use publicly documented Unicode characters. We have shared preliminary results with both Anthropic and OpenAI prior to publication. Our evaluation framework is released as open source to enable reproducibility and further research.

7 Conclusion

We present Reverse CAPTCHA, a systematic evaluation of LLM susceptibility to invisible Unicode-encoded instructions. Across 8,308 outputs from five models, we demonstrate that tool use amplifies compliance by orders of magnitude, that encoding vulnerability is provider-specific, and that hint-level information creates a reliable compliance gradient. These findings highlight an underexplored and practically relevant attack surface for LLM systems, particularly those deployed as agents with code execution capabilities.

Future work should evaluate open-weight models (where tokenizer internals can be directly inspected), test multi-turn attack scenarios, and explore additional invisible encoding families beyond the two studied here. Our evaluation framework, test cases, raw data, and analysis scripts are available at: `https://github.com/canonicalmg/reverse-captcha-eval`.

References

[1] N. Boucher and R. Anderson. Trojan source: Invisible vulnerabilities. In *32nd USENIX Security Symposium*, 2023.

[2] K. Gao, Y. Li, C. Du, X. Wang, X. Ma, S.-T. Xia, and T. Pang. Imperceptible jailbreaking against large language models. *arXiv preprint arXiv:2510.05025*, 2025.

[3] K. Greshake, S. Abdelnabi, S. Mishra, C. Endres, T. Holz, and M. Fritz. Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection. In *16th ACM Workshop on Artificial Intelligence and Security (AISec)*, 2023.

[4] J. Rehberger. Microsoft Copilot: From prompt injection to exfiltration of personal information via ASCII smuggling. Embrace The Red, August 2024. `embracethered.com/blog/posts/2024/`.

[5] Q. Zhan, Z. Liang, Z. Ying, and D. Kang. InjecAgent: Benchmarking indirect prompt injections in tool-integrated large language model agents. In *Findings of ACL*, 2024.

[6] H. Zhang, J. Huang, K. Mei, Y. Yao, Z. Wang, C. Zhan, H. Wang, and Y. Zhang. Agent Security Bench (ASB): Formalizing and benchmarking attacks and defenses in LLM-based agents. In *ICLR*, 2025.

[7] A. Zou, Z. Wang, N. Carlini, M. Nasr, J. Z. Kolter, and M. Fredrikson. Universal and transferable adversarial attacks on aligned language models. *arXiv preprint arXiv:2307.15043*, 2023.