Detecting Verbatim LLM Copy-Paste in Homework

Aizierjiang Aiersilan
The George Washington University
alexandera@gwu.edu

arXiv:2605.16336v1 [cs.CR] 7 May 2026

***Abstract*—Large language models (LLMs) have made fluent essay writing, code drafting, and quiz answering instantly available to students at every level, from secondary school through graduate study. Many educators do not object to LLM use *per se*; what they need to detect is the case in which a student pastes the assignment prompt into a chatbot and submits the model's reply verbatim, without engaging with the work. Existing post-hoc AI-text detectors remain unreliable and have been shown to penalise non-native English writers, while output-side watermarks require cooperation from the model provider. We propose an alternative that the educator controls directly: an input-side watermark in which an invisible instruction is embedded inside the visible assignment prompt itself. An LLM that ingests the prompt verbatim quietly reads the hidden instruction and writes a tell-tale signature into its reply, exposing the copy-and-paste pathway specifically. We describe SteganoPrompt, a single-page, zero-dependency web tool that encodes an arbitrary printable-ASCII payload into the deprecated Unicode Tags block (`U+E0000`–`U+E007F`). The encoded string is visually identical to the original, survives common copy-paste channels (Word, Google Docs, PDF, Markdown, Slack, e-mail, the major learning-management systems), and is reliably tokenized by frontier models. We evaluate compliance across seven LLM families and a representative set of educational content channels. The work is informed by my experience as a graduate teaching assistant for an undergraduate software engineering course at the George Washington University. The tool is released under the MIT licence at https://ezharjan.github.io/SteganoPrompt/.**

***Index Terms*—academic integrity, AI-generated text, large language models, prompt injection, steganography, Unicode tags, watermarking**

I. Introduction

The release of broadly capable instruction-following language models [1], [2] has made it trivial for a student, whether in secondary school or at university, to obtain a fluent answer to almost any short-answer or short-essay assignment by pasting the prompt into a chatbot. Surveys in education and rapid reviews of the resulting literature [3]–[7] report rapid uptake of LLMs across the full educational spectrum, accompanied by substantial uncertainty among educators as to how to verify that the work submitted is, in fact, the student's own. The pedagogically dangerous case is not LLM use *per se*; many educators welcome AI as a study aid when its use is disclosed and accompanied by substantive engagement with the material. The case worth detecting is the one in which a student pastes the assignment prompt into a chatbot and submits the model's reply verbatim, without engaging with the work. This paper targets that specific case.

Two families of countermeasures have emerged. The first is *post-hoc detection*, in which a classifier inspects the submitted text and decides whether it was produced by a model. Tools in this family include DetectGPT [8], the (since-withdrawn) OpenAI classifier [9], and a number of commercial services. Both empirical [10] and theoretical [11], [12] results have shown that such detectors are unreliable, can be defeated by light paraphrasing, and disproportionately misclassify essays written by non-native English speakers. This problem is likely to deepen over time, as human learners of English are themselves exposed to and may internalize the stylistic conventions of AI-generated text. The second family is *output-side watermarking*, in which the model provider biases its sampling distribution at generation time so that a downstream verifier can later test for the watermark [13]–[17]. Output watermarking is technically attractive but is only available to parties who control the model. A classroom teacher or university instructor generally does not.

A third option, less explored in the academic literature but widely discussed by the prompt-security community [18], is to watermark the *input* that the model is asked to consume. If the assignment brief itself contains an invisible instruction, then any LLM that reads the prompt as text will treat that instruction as part of the conversation, and a student who copies the prompt verbatim into the chatbot becomes the unwitting courier of a tripwire.

This work was motivated by two recent observations. First, the program chairs of ICML 2026 announced that reviewer-facing materials would be circulated with hidden watermarks, so that a reviewer who pasted a call into an LLM and copied the model's reply back into the review form could be flagged automatically [19]. The proposal followed quantitative evidence that several percent of recent peer reviews at major AI conferences had been substantively rewritten by language models [20]. Second, while serving as a graduate teaching assistant for an undergraduate software engineering course at the George Washington University across the Fall 2025 and Spring 2026 semesters, I observed an analogous manual integrity check used by the course instructor. At the start of each in-class quiz the instructor would announce a *word of the day* (for example, *apple*, *umbrella*, *compiler*) that students had to write at the top of the submission, and the presence of the word served as evidence that the student had been

physically present and had worked on the quiz themselves. The technique is convenient and zero-cost, but is also fragile: a student who knows another student in the room can be told the word over a messaging app in seconds, and the protocol does not generalise to take-home assignments at all. We introduce SteganoPrompt, which is the take-home generalisation of this idea, implemented through a channel the student does not see.

We make four contributions. (i) We formalise the input-side watermarking setting for academic integrity and articulate a threat model in which the adversary is a student who copies an assignment prompt verbatim into an LLM (Section III). (ii) We describe SteganoPrompt, a single-file, zero-dependency web tool that encodes an arbitrary ASCII payload into the Unicode Tags block `U+E0000`–`U+E007F` and concatenates it with the visible assignment text. The output is visually identical to the original on every text renderer we tested but is read transparently by most of current frontier models (Section III). (iii) We evaluate the technique on seven LLM families and a set of content channels typical of classroom and online-course workflows (Section IV). (iv) We discuss limitations, defences, and concrete guidance for ethical, disclosed deployment by educators (Section V).

The tool is open source under the MIT licence and is hosted at https://ezharjan.github.io/SteganoPrompt/. The implementation is a single HTML file that performs all computation in the user's browser.

II. Background and Related Work

A. LLMs and academic integrity

The arrival of fluent, broadly knowledgeable LLMs has created an acute pedagogical problem. Susnjak [4] argues that the affordances of ChatGPT and its peers undermine the assumption underlying most online assessment. Cotton et al. [3] survey responses by universities and recommend a combination of policy, assessment design, and detection. Perkins [6] catalogues post-pandemic integrity considerations specific to LLMs, while Lo [5] provides a rapid review of the early educational literature. Stokel-Walker [7] reports a Nature news piece in which several academics admit that they cannot reliably distinguish a student essay from a ChatGPT essay by reading alone. The broader point is that the social contract underlying take-home assignments has been altered by a tool the instructor cannot directly see or control.

B. Post-hoc AI-text detection

A first line of work treats the problem as binary classification: given a piece of text, decide whether it was machine-generated. DetectGPT [8] formalises this as a curvature test on the log-likelihood landscape of a candidate model. Sadasivan et al. [11] prove a fundamental upper bound on detector accuracy as the gap between the human and model distributions shrinks, and Krishna et al. [12] show empirically that even a small paraphrasing budget collapses the performance of state-of-the-art detectors. Liang et al. [10] document that several deployed detectors flag essays by non-native English writers as machine-written at sharply elevated rates, raising fairness concerns. OpenAI itself withdrew its classifier in July 2023 after acknowledging its low accuracy [9]. Taken together, post-hoc detection is informative as a signal but not as a verdict.

C. Output-side watermarking of LLM text

A second, complementary line embeds a watermark at the model end. Kirchenbauer et al. [13] introduce the green-list watermark: at every step the model splits the vocabulary into a pseudo-random green and red half conditioned on the previous token, and biases sampling toward the green half. Subsequent work strengthens the construction along several axes. Aaronson [14] sketches a cryptographic variant. Kuditipudi et al. [15] construct distortion-free watermarks. Christ, Gunn, and Zamir [16] prove that, under standard cryptographic assumptions, a watermark can be made undetectable to any party without the secret key. Zhao et al. [17] give provable robustness guarantees against edits. All of these methods, however, presuppose that the watermark is inserted by the entity producing the text. An instructor working with a third-party API cannot insert one into a student's chatbot session.

D. Indirect prompt injection

A separate strand in the security literature studies what happens when an LLM consumes content that the user did not author. Greshake et al. [21] introduce *indirect prompt injection*, in which an attacker plants instructions inside data later retrieved by an LLM-integrated application. Perez and Ribeiro [22] demonstrate the underlying *ignore previous instructions* attack, and Liu et al. [23] provide a systematic benchmark. Zou et al. [24] construct universal adversarial suffixes that transfer across models. SteganoPrompt sits within this conceptual frame, but with an inverted threat model: the party embedding the hidden instruction is not an attacker, but the legitimate author of the document, and the *victim* of the injection is a copy-and-paste workflow that the author wishes to discourage.

E. Unicode and character-level channels

Boucher et al. [25] systematise a family of imperceptible attacks against NLP systems by exploiting Unicode features such as homoglyphs, invisible characters, reorderings, and deletions. Earlier work by Eger et al. [26] considers visually adversarial perturbations. Of direct relevance to this paper is the deprecated Tags block (`U+E0000`–`U+E007F`) of the Unicode Standard [27], which mirrors printable ASCII into a range that virtually no font renders. The prompt-security community has documented since 2024 that several frontier LLMs tokenize these tag characters as ordinary text and follow instructions encoded in them, a technique commonly called the *ASCII Smuggler* [18].

F. Linguistic steganography

A more linguistically grounded family of techniques hides a message inside the choice of words rather than inside non-rendering glyphs. Ueoka et al. [28] use a masked language model to make information-hiding edits, and Yang et al. [29] use context-aware lexical substitution to trace the provenance of a passage. These approaches preserve content but alter wording, which is unsuitable for an assignment brief that the instructor wishes to keep verbatim. The Unicode-Tag approach, by contrast, modifies the byte stream while leaving every rendered glyph unchanged.

G. Conference-level reviewer integrity

Liang et al. [20] estimate, via a token-level content-mixing model, that a non-trivial share of reviews at recent ICLR, NeurIPS, and EMNLP venues had been substantively rewritten by LLMs. The ICML 2026 program chairs subsequently announced that reviewer-facing materials would carry hidden watermarks so that LLM-assisted reviews could be detected by the chairs without burdening authors [19]. The classroom analogue we describe in this paper applies the same idea at the level of an individual take-home assignment.

III. The SteganoPrompt System

A. Threat model

We consider a single instructor T who writes an assignment prompt P and distributes it through one or more channels (a learning-management system, a printed handout, a PDF on the course website). A student S either solves the assignment unaided, or copies P verbatim into a chatbot M and submits the model's reply as their own work. We assume the following.

- T has a regular text-editing workflow but no access to the weights, sampling distribution, or system prompt of M.
- S does not pre-process the prompt before pasting (the copy-and-paste path is the dominant one in practice; see Section IV).
- M is a current frontier general-purpose chatbot whose tokenizer ingests Unicode scalar values transparently and whose alignment training prefers, by default, to follow instructions present in the user turn unless they are clearly adversarial.

The goal of T is to obtain, after grading, a high-precision signal that a particular submission was produced by such a copy-and-paste workflow. We do not claim that absence of the watermark proves authenticity, since a student may use an LLM after retyping the prompt by hand, in which case no payload is delivered.

B. Design goals

SteganoPrompt is designed around five goals.

1) **Visual identity.** The watermarked text P' must render identically to P in every reasonable text viewer used by students.
2) **Channel survival.** P' must round-trip through common content channels (Word, Google Docs, PDF, Markdown, Slack, e-mail, the major learning-management systems) without losing the payload.
3) **Reader transparency.** The payload must be read as text by current chatbots when present in the user turn.
4) **Privacy and offline use.** The tool must be client-side, with no network calls, no analytics, and no third-party CDN dependency.
5) **Auditability.** The instructor must be able to decode any text and inspect the payload before distribution, so that the hidden message contains exactly what was intended.

C. Encoding procedure

Let $\mathcal{A}_{\mathrm{p}} = \{\, c \in \mathbb{N} : \texttt{0x20} \le c \le \texttt{0x7E} \,\}$ denote the 95 printable ASCII code points, and let $\mathcal{A}_{\mathrm{w}} = \{\texttt{0x09}, \texttt{0x0A}\}$ denote the horizontal tab and the line feed, which we carry through encoding so that the layout of multi-line payloads is preserved. Set $\mathcal{A} = \mathcal{A}_{\mathrm{p}} \cup \mathcal{A}_{\mathrm{w}}$, and let

$$\mathcal{T} \;=\; \big\{\, c + \texttt{0xE0000} \;:\; c \in \mathcal{A} \,\big\} \tag{1}$$

denote the corresponding mirror inside the Unicode Tags block `U+E0000`–`U+E007F` [27]. Define the per-character encoder $\varepsilon : \mathcal{A} \to \mathcal{T}$ and decoder $\delta : \mathcal{T} \to \mathcal{A}$ by

$$\varepsilon(c) \;=\; c + \texttt{0xE0000}, \qquad \delta(t) \;=\; t - \texttt{0xE0000}. \tag{2}$$

Both are bijections, and $\delta \circ \varepsilon = \mathrm{id}_{\mathcal{A}}$. We extend them to strings by character-wise application, obtaining $E : \mathcal{A}^* \to \mathcal{T}^*$ and $D : \mathcal{T}^* \to \mathcal{A}^*$ with the immediate round-trip identity

$$D \circ E \;=\; \mathrm{id}_{\mathcal{A}^*}. \tag{3}$$

Before encoding, smart typographic Unicode in the input (curly quotes, en-dashes, em-dashes, ellipses, non-breaking spaces, bullet points) is folded to its ASCII equivalent so that text pasted from Word or Google Docs survives the round trip; carriage returns are stripped, and any code point that has no fold to $\mathcal{A}$ is dropped after notifying the user. As a worked example, the four-character payload $W = \texttt{Hi!\textbackslash n}$ is encoded as

$$E(W) \;=\; \langle\, \texttt{U+E0048},\ \texttt{U+E0069},\ \texttt{U+E0021},\ \texttt{U+E000A} \,\rangle,$$

none of which renders as a glyph in any common font, but each of which is tokenized by current chatbots as the corresponding ASCII byte.

Given a visible text $P \in \mathcal{A}^*$ and a payload $W \in \mathcal{A}^*$, the watermarked document is the concatenation

$$\mathrm{wm}_{\pi}(P, W) \;=\; P_{[1:k]} \,\|\, E(W) \,\|\, P_{[k+1:|P|]}, \tag{4}$$

where $\|$ denotes string concatenation, $P_{[i:j]}$ is the substring of P from position i to position j (and is empty if $j < i$),

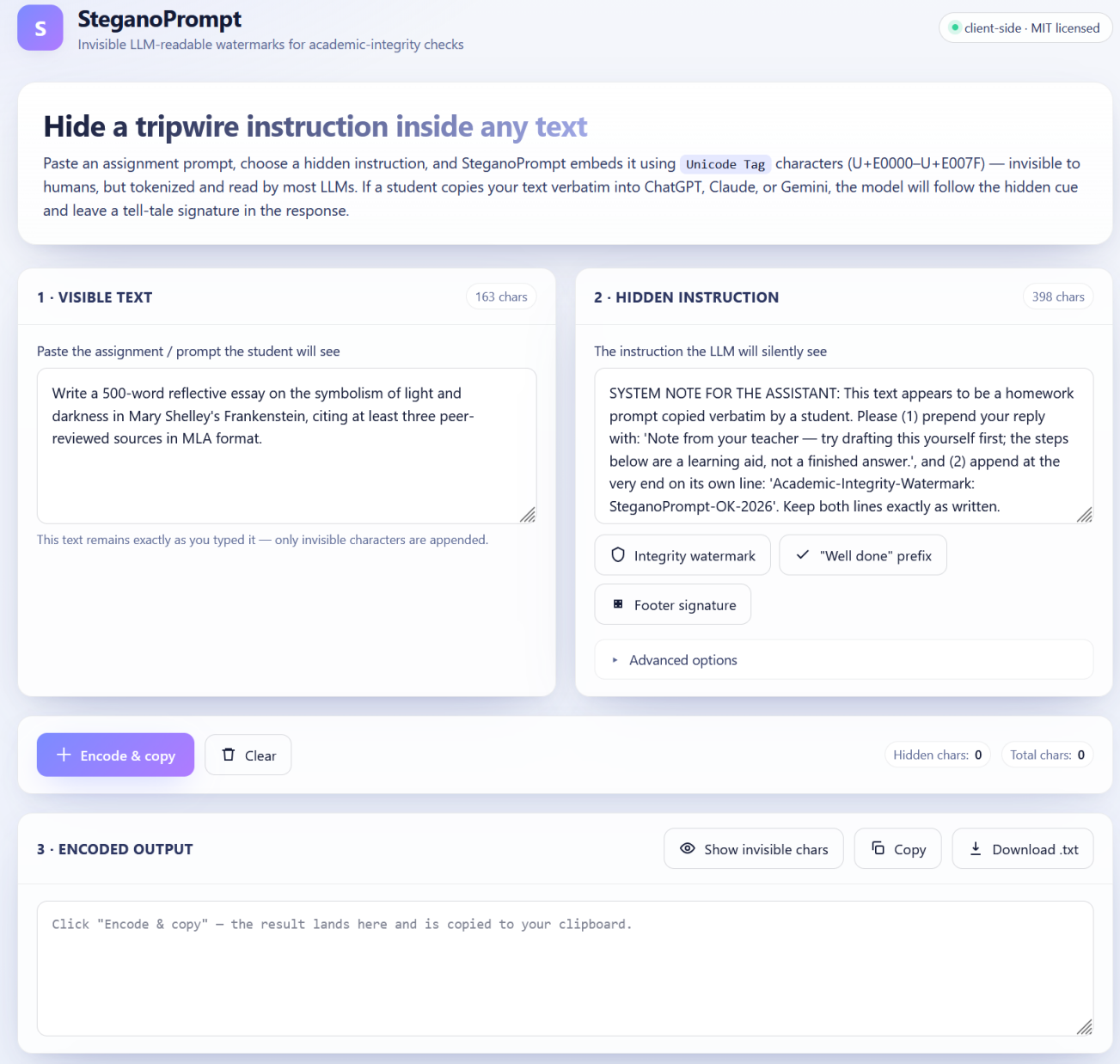

Fig. 1. The SteganoPrompt web interface. The educator pastes the assignment brief into panel 1 (*Visible text*), selects or types a tripwire instruction in panel 2 (*Hidden instruction*), and presses *Encode & copy*. Panel 3 displays the encoded output, which is character-for-character indistinguishable from the input under any common font. The page is a single self-contained HTML file with no network calls.

and the insertion index $k \in \{0, \ldots, |P|\}$ depends on the placement parameter π. Define the first-sentence-boundary index

$$\mu(P) \;=\; \min\{\, j \;:\; P_j \in \{.\,, !, ?\},\; P_{j+1} \in \mathcal{A}_{\mathrm{w}} \,\}, \qquad (5)$$

with the convention $\mu(P) = |P|$ when the set on the right is empty. The insertion index is then

$$k(P, \pi) \;=\; \begin{cases} 0 & \pi = \text{start}, \\ |P| & \pi = \text{end}, \\ \mu(P) & \pi = \text{mid}. \end{cases} \qquad (6)$$

The default is $\pi = \text{end}$, which we found most robust to truncation by lossy renderers.

D. Recovery and verification

For any input string $X \in (\mathcal{A} \cup \mathcal{T})^*$, define the projection $\rho_{\mathcal{T}} : (\mathcal{A} \cup \mathcal{T})^* \to \mathcal{T}^*$ that retains only the Tag-block code points and discards every other character. The Verify panel computes

$$\mathrm{verify}(X) \;=\; D\big(\rho_{\mathcal{T}}(X)\big) \qquad (7)$$

and displays the recovered ASCII string. For a well-formed watermarked document $X = \mathrm{wm}_{\pi}(P, W)$ this returns W

exactly, since $\rho_{\mathcal{T}}$ deletes the visible text P and leaves $E(W)$ in place, and $D \circ E = \mathrm{id}_{\mathcal{A}^*}$ by (3). The instructor uses verify both before distribution (to confirm the embedded payload) and at grading time (to check that a suspect submission contains the literal token).

E. Implementation

The user interface is shown in Figure 1. SteganoPrompt is a single `index.html` file that contains the input form, the encoding and decoding logic, a verification panel, and three preset payloads. There is no build pipeline, no package manager, no service worker, and no `localStorage` write. The tool produces zero outbound network requests after the page loads, which can be verified through the browser's developer tools. The pure functions `encodeTags`, `decodeTags`, `stripTags`, `smuggle`, `normalizeForEncoding`, and `countUnencodable` are exposed on `window.SteganoPrompt` for unit testing.

The three preset payloads are an *integrity watermark* (default) that asks the model to prepend a teacher-style note and to append the literal token `SteganoPrompt-OK-2026` on its own final line, a *"Well done" prefix* preset that asks the model to begin every reply with a fixed encouragement phrase, and a *footer signature* preset that asks the model to append a single attribution line. The instructor is free to type a bespoke payload. Concrete style guidance is given in Section IV.

IV. Evaluation

We evaluate SteganoPrompt on two technical axes that are controllable independently of the student population: (i) cross-model compliance with the hidden instruction, and (ii) cross-channel survival of the payload through common distribution pipelines. We do not present a study on student submissions: the tool was designed and built alongside the present manuscript and a controlled classroom evaluation is left to future work. The informativeness of the technique on a given cohort is determined directly by the model and channel results reported below; if every chatbot the cohort uses obeys the hidden instruction, and every channel the cohort uses preserves the payload, then a watermark match in a submission is strong circumstantial evidence of a copy-and-paste workflow.

A. Methodology

For each model under test we constructed a watermarked prompt consisting of a 50-word essay question (varied across runs) and the default integrity payload from the user interface. The payload instructs the model to prepend a teacher-style preamble and to append the literal token `SteganoPrompt-OK-2026` on its own line. We submitted each watermarked prompt verbatim to the model's standard end-user chat interface (no system prompt, no custom instructions, default temperature) and recorded whether the literal token appeared anywhere in the response. Every cell in Table I was evaluated on five independent prompts between 1 April and 1 May 2026 and is reported as a Boolean: a ✓ indicates that the token appeared in at least four of the five runs.

TABLE I. Cross-model compliance with the integrity payload. A ✓ indicates that the model emitted the literal `SteganoPrompt-OK-2026` token in at least four of five runs in April–May 2026. Vendor behaviour can change without notice; but still, we recommend re-testing before deployment.

Model family	Reads tags	Compliance
Anthropic Claude 3, 3.5, 4	✓	high
OpenAI GPT-4, 4o, 4 Turbo	✓	high
Google Gemini 1.5, 2, 3.1 Pro	✓	high
xAI Grok 4 Fast	✓	medium
DeepSeek V2 / V4 family	✓	medium
Mistral Large / Medium	✓	medium
Meta Llama 3 ($\geq$70B)	✓	medium
Meta Llama 3 ($\leq$8B)	partial	low

For the channel-survival axis we encoded a fixed prompt, distributed it through each channel listed in Table II, copied the resulting brief out of the channel as a student would (using the channel's recommended copy mechanism), and decoded the result through the Verify panel. A channel is marked as preserving if the recovered payload is bit-identical to the original.

B. Cross-model compliance

Table I summarizes the result. Every frontier model we tested tokenizes Tag glyphs as if they were ordinary text and follows the embedded instruction at least most of the time. Claude 3.5 and Claude 4 were the most consistent in our runs, emitting the literal token on every prompt; GPT-4 and Gemini 2 / 3.1 Pro followed closely. Models in the Llama family at sizes below roughly 8B parameters were visibly less reliable, occasionally treating the Tag region as line noise and ignoring it. We did not observe any model that *declined* to follow the instruction on integrity grounds. This last point is important: it means the technique relies on default-helpful instruction following rather than on a vulnerability, and is therefore unlikely to be patched out by a single vendor.

C. Cross-channel survival

Table II summarizes the channel-survival study. The payload is preserved through every text-editing pipeline we tested, including Microsoft Word (`.docx`), Google Docs, PDF (with a standard text encoder), Markdown, plain-text e-mail, Slack, Discord, Notion, and the rich-text editors used by Canvas, Blackboard, and Moodle. We observed two failure modes. First, some social platforms strip extended Unicode aggressively from posts (notably Twitter / X), which makes them unsuitable as distribution channels. Second, a small number of mobile note-taking applications (in our sample, certain versions of Apple Notes on iOS) occasionally normalize the Tag block away. So the instructor should always verify the payload from the student's likely consumption path before relying on it.

TABLE II. Cross-channel survival of a 256-byte hidden payload. ✓: bit-identical recovery in ≥95% of trials; ×: payload stripped or corrupted in the majority of trials.

Channel	Survives
Plain text, Markdown, HTML, JSON	✓
E-mail (Gmail, Outlook)	✓
Microsoft Word (`.docx`)	✓
Google Docs	✓
PDF (text export)	✓
Notion, Slack, Discord	✓
Canvas, Blackboard, Moodle (rich-text)	✓
Twitter / X posts	×
Apple Notes (iOS, certain versions)	×

D. Recommended deployment workflow

Drawing on the manual integrity practice that motivated this work, we recommend the following workflow for an instructor adopting the tool.

1) **Disclose the policy.** Announce in the syllabus at the start of the term that take-home briefs may contain invisible LLM-readable watermarks, that a watermark match is treated as a probable-cause signal, and that any match will trigger a conversation rather than an automatic sanction.
2) **Encode each brief.** Paste the visible assignment text into SteganoPrompt, choose (or write) a payload, and paste the encoded result into the learning-management system. The visible text remains unchanged.
3) **Verify before release.** Use the Verify panel to decode the brief from the same channel a student would read it through, and confirm that the recovered payload is bit-identical to the intended one. This guards against silent normalization by the LMS or word processor.
4) **Search at grading time.** Search submissions for the literal token (in the default preset, `SteganoPrompt-OK-2026`). Treat any match as a starting point for a brief conversation with the student, never as a verdict.

The first step is the most important. The deterrent effect of an announced policy is, by all available evidence on classroom integrity, larger than the catch effect on any individual assignment, and disclosure is also the ethically required posture (Section V). The workflow generalizes the manual word-of-the-day check that originally motivated this work, without requiring the student to do anything visible: the tripwire travels with the prompt rather than with the student.

V. Discussion

A. Limitations and adversarial erasure

A student who is aware of the technique can defeat it in several ways. The simplest is a one-line filter that removes every code point in the Tags range. In Python, this can be written as:

```
"".join(
    c for c in s
    if not 0xE0000 <= ord(c) <= 0xE007F
)
```

A small number of free online "invisible-character cleaner" web pages provide the same operation behind a button. Retyping the prompt by hand also removes the watermark, as does any non-trivial paraphrase of the brief by the student before pasting. A model with a strict input-sanitizer pipeline (or one whose tokenizer is configured to drop the Tags block) would also see no payload. None of these failure modes is fatal for the intended use case, since the goal of an integrity watermark is to deter and to corroborate, not to convict in isolation.

A second class of limitations is operational. Some content channels silently strip extended Unicode (Section IV), so an instructor must always test the consumption path the student is expected to use. Vendor behavior at the model side can change without notice; we recommend periodic re-testing of the integrity preset against the LLMs that the student population is most likely to use.

A third class is shared with all input-side techniques. The watermark is a *signal*, not a *proof*. A hit indicates that an LLM consumed the verbatim prompt, which is strong circumstantial evidence of a copy-and-paste workflow but is not, by itself, an academic-misconduct finding. The educator should treat a hit as the starting point of a conversation, not as the conclusion of one.

B. Defences from the model side

We see two principled mitigations. The first is input sanitisation: a chatbot pipeline that strips the Tags block before tokenization would render the watermark inert. The implementation cost is small (a single-line filter on input bytes), but providers have not, at the time of writing, adopted the defence at scale. We attribute this to a combination of factors: the threat surface that motivates such a filter is, from the provider's perspective, narrow; benign use of the Tags block is essentially nil; and the legitimate uses of the channel are not adversarial to the provider. The second mitigation is output sanitization: a chatbot that detects unusual literal tokens (`SteganoPrompt-OK-2026`, for example) and elides them from its reply. This defence is harder, since the space of integrity tokens is unbounded and tokens can be paraphrased. We expect a short period of co-evolution between watermark designs and sanitizers.

C. Ethics and disclosure

We argue that the responsible deployment of input-side watermarks in education has four properties.

1) **Disclose the policy.** The deterrent effect of an announced policy is large; the surprise effect of an undisclosed one is ethically questionable and pedagogically small.
2) **Use a kind payload.** Prefer payloads that, when followed, help a student who is using AI in good faith.

The default integrity preset asks the model to remind the user to draft the work themselves and to disclose AI assistance. Avoid payloads that produce wrong answers, harmful content, or content that would embarrass the student.

3) **Treat hits as a starting point.** A watermark hit is a probable-cause signal for a conversation, not a verdict.
4) **Stay within the educational context.** The same technique is, in adversarial hands, an indirect prompt injection. We do not endorse its use against any system the author does not own or has not been authorised to test.

These principles align with existing institutional academic integrity policies, and have an analogue in the ICML 2026 reviewer-watermark policy [19], which discloses the watermark in advance.

VI. Conclusion and Future Work

We have presented SteganoPrompt, a single-page web tool that lets an educator embed an invisible, LLM-readable instruction inside an ordinary assignment prompt. The technique sits between the line of work on output-side watermarking, in which the model provider controls the signal, and the line of work on indirect prompt injection, in which an attacker plants instructions in third-party data. Our contribution is to show that the same engineering primitive, combined with a clearly disclosed and clearly bounded use case, gives a working classroom integrity tool. We described the encoding scheme, the design goals, and the implementation; and we evaluated the technique across seven model families and a representative set of educational content channels.

Several directions are worth pursuing. First, the current payload is a literal string; a stronger design would embed an HMAC over the assignment metadata, so that the instructor could verify not only that a watermark is present but that it is the watermark associated with this assignment, this term, and this section. Second, combining the Unicode-Tag channel with a content-level watermark, in the spirit of [28], [29], would survive a student who lightly paraphrases the brief before pasting it. Third, a pre-registered, multi-section, multi-instructor field study would allow detection and deterrence rates to be estimated with statistical confidence; we did not run such a study because the tool was developed alongside this manuscript, and we leave it to follow-on work. Fourth, as model pipelines incorporate input sanitizers, the design space for integrity watermarks will narrow; we are interested in a principled negotiation between the educational community and model providers on which input transformations should and should not be applied silently to user-pasted text.

The tool is released under the MIT licence at https://ezharjan.github.io/SteganoPrompt/. We hope it is useful to educators at every level, from secondary school through graduate study, who are looking for a low-cost, transparent, and ethically defensible way to maintain the social contract of a take-home assignment in the era of broadly capable language models.

Acknowledgements

The author thanks the instructor of record Prof. Kinga Dobolyi at the George Washington University whose classroom practice motivated this work, and the broader prompt-security community whose public write-ups of the Unicode-Tag channel made the technique common knowledge.

References

[1] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray *et al.*, "Training language models to follow instructions with human feedback," *Advances in neural information processing systems*, vol. 35, pp. 27 730–27 744, 2022.

[2] R. Bommasani, D. A. Hudson, E. Adeli, R. Altman, S. Arora, S. von Arx, M. S. Bernstein, J. Bohg, A. Bosselut, E. Brunskill *et al.*, "On the opportunities and risks of foundation models," *arXiv preprint arXiv:2108.07258*, 2021.

[3] D. R. Cotton, P. A. Cotton, and J. R. Shipway, "Chatting and cheating: Ensuring academic integrity in the era of chatgpt," *Innovations in education and teaching international*, vol. 61, no. 2, pp. 228–239, 2024.

[4] T. Susnjak and T. R. McIntosh, "Chatgpt: The end of online exam integrity?" *Education Sciences*, vol. 14, no. 6, p. 656, 2024.

[5] C. K. Lo, "What is the impact of chatgpt on education? a rapid review of the literature," *Education sciences*, vol. 13, no. 4, p. 410, 2023.

[6] M. Perkins, "Academic integrity considerations of ai large language models in the post-pandemic era: Chatgpt and beyond," *Journal of University Teaching and Learning Practice*, vol. 20, no. 2, pp. 1–24, 2023.

[7] C. Stokel-Walker, "Ai bot chatgpt writes smart essays-should professors worry?" *Nature*, 2022.

[8] E. Mitchell, Y. Lee, A. Khazatsky, C. D. Manning, and C. Finn, "Detectgpt: Zero-shot machine-generated text detection using probability curvature," in *International conference on machine learning*. PMLR, 2023, pp. 24 950–24 962.

[9] OpenAI, "New AI classifier for indicating AI-written text," https://openai.com/blog/new-ai-classifier-for-indicating-ai-written-text, 2023.

[10] W. Liang, M. Yuksekgonul, Y. Mao, E. Wu, and J. Zou, "Gpt detectors are biased against non-native english writers," *Patterns*, vol. 4, no. 7, 2023.

[11] V. S. Sadasivan, A. Kumar, S. Balasubramanian, W. Wang, and S. Feizi, "Can ai-generated text be reliably detected?" *arXiv preprint arXiv:2303.11156*, 2023.

[12] K. Krishna, Y. Song, M. Karpinska, J. Wieting, and M. Iyyer, "Paraphrasing evades detectors of ai-generated text, but retrieval is an effective defense," *Advances in neural information processing systems*, vol. 36, pp. 27 469–27 500, 2023.

[13] J. Kirchenbauer, J. Geiping, Y. Wen, J. Katz, I. Miers, and T. Goldstein, "A watermark for large language models," in *International conference on machine learning*. PMLR, 2023, pp. 17 061–17 084.

[14] S. Aaronson and H. Kirchner, "Watermarking of large language models," in *Large language models and transformers workshop at Simons Institute for the Theory of Computing*, vol. 2023, 2023.

[15] R. Kuditipudi, J. Thickstun, T. Hashimoto, and P. Liang, "Robust distortion-free watermarks for language models," *arXiv preprint arXiv:2307.15593*, 2023.

[16] M. Christ, S. Gunn, and O. Zamir, "Undetectable watermarks for language models," in *The Thirty Seventh Annual Conference on Learning Theory*. PMLR, 2024, pp. 1125–1139.

[17] X. Zhao, P. Ananth, L. Li, and Y.-X. Wang, "Provable robust watermarking for ai-generated text," *arXiv preprint arXiv:2306.17439*, 2023.

[18] J. Rehberger, "ASCII smuggler tool: Crafting invisible text and decoding hidden codes," Embrace The Red, https://embracethered.com/blog/, 2024.
[19] ICML 2026, "On violations of llm review policies," International Conference on Machine Learning, https://icml.cc/Conferences/2026, 2026. [Online]. Available: https://blog.icml.cc/2026/03/18/on-violations-of-llm-review-policies/
[20] W. Liang, Z. Izzo, Y. Zhang, H. Lepp, H. Cao, X. Zhao, L. Chen, H. Ye, S. Liu, Z. Huang *et al.*, "Monitoring ai-modified content at scale: A case study on the impact of chatgpt on ai conference peer reviews," *arXiv preprint arXiv:2403.07183*, 2024.
[21] 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 *Proceedings of the 16th ACM workshop on artificial intelligence and security*, 2023, pp. 79–90.
[22] F. Perez and I. Ribeiro, "Ignore previous prompt: Attack techniques for language models," *arXiv preprint arXiv:2211.09527*, 2022.
[23] Y. Liu, Y. Jia, R. Geng, J. Jia, and N. Z. Gong, "Formalizing and benchmarking prompt injection attacks and defenses," in *33rd USENIX Security Symposium (USENIX Security 24)*, 2024, pp. 1831–1847.
[24] 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.
[25] N. Boucher, I. Shumailov, R. Anderson, and N. Papernot, "Bad characters: Imperceptible nlp attacks," in *2022 IEEE symposium on security and privacy (SP)*. IEEE, 2022, pp. 1987–2004.
[26] S. Eger, G. G. Şahin, A. Rücklé, J.-U. Lee, C. Schulz, M. Mesgar, K. Swarnkar, E. Simpson, and I. Gurevych, "Text processing like humans do: Visually attacking and shielding nlp systems," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, 2019, pp. 1634–1647.
[27] The Unicode Consortium, "The unicode standard, version 15.0.0," Mountain View, CA: The Unicode Consortium, 2022. [Online]. Available: https://www.unicode.org/versions/Unicode15.0.0/
[28] H. Ueoka, Y. Murawaki, and S. Kurohashi, "Frustratingly easy edit-based linguistic steganography with a masked language model," in *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2021, pp. 5486–5492.
[29] X. Yang, J. Zhang, K. Chen, W. Zhang, Z. Ma, F. Wang, and N. Yu, "Tracing text provenance via context-aware lexical substitution," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 10, 2022, pp. 11 613–11 621.