

THE INTERNATIONAL YOUTH ENGINEERING SOCIETY

How Editing changes AI Detection in writing

Detection mathematics, public code and a paired rewrite experiment

By Rajdeep R Nair

25 November 2026

RESEARCH PAPER

Abstract

An edit can change how an AI-writing detector classifies a draft even though the draft's origin has not changed. This paper examines why, reviews evidence from prior studies, and proposes a paired experiment for testing current detectors. A synthetic study illustrates the method. It used a bigram classifier and 180 prompt groups, distributed equally across training, calibration, and test sets. The classifier flagged 270 of 360 original test drafts. Changing paragraph breaks left every score unchanged. Restructuring sentences removed 155 flags and introduced 23, leaving 138 flagged drafts even though the mean score rose slightly. None of the drafts produced by three alternative templates was flagged. Those templates resembled examples in the reference class, so the result describes the behavior of the synthetic classifier under known conditions. The paper also derives published detection statistics, compares their implementations, and proposes a study of current commercial detectors.

Research question

Which edits make a flagged, AI-generated draft fall below the decision threshold while preserving its argument and evidence? Do the same edits have an effect on detectors that played no part in selecting the rewrite?

Contributions

The study treats the two directions of a classification change separately and checks whether the rewrite preserves the original claims. The PDF includes the Python scripts needed to reproduce the demonstration, the complete synthetic output, and an evaluator for independently collected results.

Contents

Section	Pages
Foundations and prior research	3–6
Mathematical and computational methods	7–16
Experimental design and measures	17–20
Synthetic results and interpretation	21–30
Reproduction and AI rewrite protocol	31–35
References and supplementary material	36–40

1 What a Detector Can Observe

A text-only detector sees the finished passage. It does not observe the prompt that preceded it, the writer’s keystrokes, or the revisions that led to the final wording. Two drafts with different histories can end up identical. A deterministic detector must score them identically, and a randomized detector must give them the same distribution of scores. Increasing the size of the classifier cannot recover a difference absent from its input. [1, 12]

Four questions with different evidence

Question	Observable	Interpretation
Is this wording predictable?	Token probabilities from a reference model	A property of the wording and the model scoring it
Does this resemble labeled AI material?	A learned score or predicted class	Resemblance to patterns learned during training
Is a known generation signal present?	A watermark statistic under a specified scheme	Evidence concerning that scheme and the scored text
How was this document actually produced?	Confirmed source draft, revision history, or generation record	Evidence about provenance within the record’s coverage

Why format can matter

A paragraph break can change what a detector receives. A PDF extractor may join two paragraphs, a tokenizer may encode the break, or a classifier may assign the affected sentence to a different block of context. Which of these occurs depends on the system. The synthetic experiment provides a useful control because its tokenizer discards whitespace. A change in paragraph breaks therefore leaves the detector’s input unchanged.

What a flag means

In this study, an evasion event occurs when a draft with independently verified AI provenance is flagged before editing and falls below the same decision threshold afterward. Loss of factual fidelity is recorded separately. A rewrite that removes a flag by changing what the passage says fails the fidelity test.

2 Statistical Detection Mechanisms

Probability and rank

A reference language model assigns a probability to the observed token given the text that precedes it. Its negative logarithm is called surprisal. The mean of this quantity is the per-token negative log-likelihood, and its exponential is perplexity. Token rank measures something different. It records where the observed token sits among the model's possible continuations. GLTR displays bands of these ranks across a passage so readers can examine where predictability changes. [2]

$s(t) = -\log p(x(t) \mid \text{prefix}(t))$; $\text{PPL} = \exp(\text{mean}(s(t)))$

Consider "The panel approved the plan after the review" and "After reviewing the plan, the panel approved it." The review still precedes approval, but the reader encounters the information in a different order. The model also receives a different prefix when predicting each continuation. Rank discards some information about those predictions. A first-ranked token with probability 0.70 and one with probability 0.04 both have rank 1, yet their contributions to mean surprisal differ.

Perturbation and local preference

DetectGPT asks whether a language model assigns unusually high likelihood to the original passage compared with nearby rewrites produced by another model. The score scales that preference by the spread of the alternatives' scores. Fast-DetectGPT performs the comparison at the token level. It keeps each prefix fixed and uses a reference model's conditional distribution to estimate the expected log score and its variance at that position. This avoids generating a collection of complete rewrites. [3, 4]

DetectGPT = (original log score − average alternative score) / alternative SD

Two-model comparison

Binoculars scores the observed tokens under one model, then divides that loss by a cross-entropy term based on the predictions of two related models. The denominator supplies a reference for how the models behave on that passage. The released implementation flags scores below its cutoff. Any analysis that standardizes score direction across tools must account for that convention. [5]

Experimental prediction

Even at similar lengths and levels of word overlap, rewrites can produce different local contexts. Perplexity and token rank help describe those differences, while provenance must be established independently.

3 Learned Classifiers and Watermarks

Supervised detection

A classifier can learn patterns in vocabulary, syntax, punctuation, or the organization of an argument whenever those patterns help predict the training labels. Some associations may arise from the generator. Others may arise because the human examples came from different genres or prompts. Ghostbuster makes one approach visible. It combines features derived from the token probabilities of reference models and trains a classifier on selected combinations. [6]

GPTZero reports that it stopped using explicit perplexity and burstiness features in autumn 2023. Its 2026 technical report describes a hierarchical classifier trained to make predictions at both document and sentence level. Public documentation leaves parts of the implementation proprietary. It therefore gives no basis for attributing an individual highlight to a specific punctuation rule. [7, 8]

The products also report different quantities. Turnitin describes a share of qualifying prose that its system flags. GPTZero’s API describes class probabilities. Each should be recorded on its native scale and interpreted using its own definition. [7, 9]

Generation-time watermarking

A watermark can be introduced at generation time through a keyed sampling rule. In a green-list scheme, the generator increases the logits of a context-dependent subset of the vocabulary. The detector reconstructs these subsets and counts how often the observed token belongs to them. A rewrite can change both the token being tested and the preceding context that determines its subset. This makes rewriting a test of how much of the watermark survives. [10, 11]

z = (observed green count − expected green count) / null standard deviation

With 200 scored positions and a green fraction of 0.5, an independent null model predicts 100 matches with a standard deviation of about 7.07. An observed count of 130 gives $z \approx 4.24$. Repeated contexts can violate independence, so the accuracy of that calculation depends on the watermark rule and on how repeated positions are handled.

Mechanism comparison

A provider with access to logged generations can search for a passage with similar meaning even after a rewrite changes its wording. Such a system draws on a record of earlier generations. A classifier receiving only the submitted passage has no access to that evidence. [13]

4 Evidence from Prior Studies

Study	Main finding relevant here	Boundary of the result
Krishna et al. 2023 [13]	Contextual paraphrasing reduced one DetectGPT detection rate from 70.3% to 4.6% at a fixed 1% false-positive rate.	Specific generators, paraphraser, detector and evaluation conditions
Sadasivan et al. 2023 [12]	Repeated paraphrasing reduced detection rates for multiple approaches in the tested settings. The paper connects distinguishability to the distance between text distributions.	Performance depends on access to auxiliary evidence and the distributions being compared
Dugan et al. 2024 [14]	RAID includes over six million generations, 11 generators, eight domains and four decoding strategies. Performance changed with domain, generator, threshold and sampling policy.	A broad benchmark still reflects its model versions, prompts and human comparison corpora
Liang et al. 2023 [15]	On the study's human TOEFL essays, seven tested detectors averaged about 61% false positives.	Historical versions and a specific corpus. It does not estimate a current rate for all multilingual writers
Wang et al. 2024 [16]	SemEval evaluated detection, generator attribution and localization of a human-to-machine boundary as different tasks.	A single switch point simplifies many real mixed-authorship workflows

Evidence for specific edits

Published experiments provide stronger evidence for contextual paraphrasing than for any universal formatting rule. RAID also found that detection performance varied with decoding strategy. A paired rewrite study should therefore hold the original prompt, generator version, and sampling policy fixed. Changes to sampling belong in a separate experimental condition, because they can affect the score before any editing takes place. [13, 14]

Questions the literature leaves open

One unresolved question is whether rearranging an explanation from a fact outline changes detection more than paraphrasing it in place, given similar length and factual fidelity. Another is whether a second edit improves on the first across detectors held out from selection. An answer depends on the versions and writing populations tested. It also depends on whether an apparent gain survives a check for factual errors and omissions.

4.1 From Text to Model Inputs

A passage first has to become a sequence the detector can process. For a PDF, this begins with extraction. A word split at the end of a printed line might be reconstructed, retained with a hyphen, or separated into two fragments. A scanner may omit a table or treat its cells as prose. The text passed to the tokenizer can consequently differ from what a reader sees. During an experiment, the relevant artifact is the exact character sequence fed into the scoring model.

Tokens are units of the model

A tokenizer maps pieces of the input into integer IDs. A short word may occupy one token while a name or a technical expression may occupy several. Spaces, line breaks and punctuation can be represented in their own right or attached to neighboring fragments. The conditions of an editing experiment therefore need to distinguish character edits, word edits and changes in token count. A replacement that preserves the number of words can still alter the number and distribution of prediction targets.

The causal shift

An autoregressive model outputs a vocabulary-sized row for each input position. The row beneath a token predicts what follows that token. If the input is four IDs, the first three rows can be paired with the second, third and fourth IDs. The fourth row predicts an unobserved continuation. The first input ID can be scored if a beginning-of-sequence context is explicitly supplied. Otherwise the usual shifted loss leaves it out.

Sequence	First	Second	Third	Fourth
Observed input	the	bridge	bends	.
Target for that row	bridge	bends	.	unobserved

Long passages

A model with a fixed context window cannot score every token of a long document from the entire preceding document. Score successive windows, using overlap as context and masking already scored targets. Without that mask, overlapping positions appear more than once in the average. In a batched implementation, padding needs a separate loss mask. Document the maximum length, stride, beginning-of-sequence convention and whether rejected passages include insufficiently long submissions. The same byte sequence can otherwise yield slightly different scores in two ostensibly identical implementations.

4.2 Likelihood and Perplexity

Let $x_1,\ldots,x_n$ be observed tokens, and let $p_t(x_t)$ be the reference model's conditional probability of the token at position t. The chain rule gives the joint likelihood as the product of these conditional probabilities. Taking logarithms turns the product into a sum. This is the useful computational form because multiplying many small floating-point numbers can underflow.

$$\log P(x) = \Sigma \log p_t(x_t)$$

$$H(x) = -(1/n) \Sigma \log p_t(x_t);\ PPL(x) = \exp(H(x))$$

Perplexity is the inverse geometric mean of the observed conditional probabilities. It reflects how well the reference model predicted the text. With a uniform distribution over V possible next tokens at every position, each observed token has probability 1/V and the perplexity is exactly V. Actual language models produce nonuniform distributions that change with context.

Worked calculation

Consider three positions whose observed probabilities are 0.70, 0.30 and 0.35. Their surprisals are about 0.357, 1.204 and 1.050 nats. Averaging gives 0.870 nats per token and exponentiating gives 2.387. Using the arithmetic mean of the observed probabilities would instead give 0.450 and its inverse 2.222. That is a different calculation because it gives less weight to particularly unlikely observations.

Position	Observed probability	Surprisal in nats
First	0.70	0.357
Second	0.30	1.204
Third	0.35	1.050

Comparability

Use the same tokenizer, scoring model and context rules when comparing these values. The same passage can be easy for a model trained on engineering material and difficult for a model unfamiliar with its vocabulary. Unusual names, equations and domain terminology may raise the score for reasons unrelated to how it was written. Simple, constrained factual writing can have low perplexity whether a human or model produced it. [1, 2]

4.3 Token Rank and Stable Computation

To convert logits z into log probabilities, subtract the logarithm of the sum of exponentiated logits from every element. If a large positive logit is present, evaluating the exponentials directly may overflow. Subtracting the maximum logit first leaves the probabilities unchanged, because adding or subtracting the same constant from all logits cancels in their normalization.

$$m = \max(z);\ \log p_j = (z_j - m) - \log \Sigma \exp(z_v - m)$$

Rank can be computed without evaluating the softmax. The softmax preserves the ordering of its inputs, so the count of logits strictly larger than the observed logit, plus one, gives the observed token’s rank. The strict comparison gives tied logits the same rank. Libraries that sort and assign unique positions may handle ties differently, a difference that becomes relevant only at a bin boundary.

A GLTR-style diagnostic

GLTR visualizes where the observed token falls among the alternatives available to the reference model. Its bins distinguish ranks within 10, 100 and 1,000 from ranks outside those limits. A passage with many highlighted high-ranking tokens could reflect an exact technical phrase repeated throughout a report. Compare matched genres before attributing the pattern to generation. [2, 17]

```
def rank(row, target):
    return 1 + sum(value > row[target] for value in row)
```

>

```
def rank_band(value):
    return next((k for k in (10, 100, 1000) if value <= k),
                "outside top 1000")
```

Information discarded by ranks

Suppose the observed token has rank 1 in two contexts. The model assigns it 0.70 in the first and 0.04 in the second, where probability is spread over many alternatives. Rank treats the observations equally. Surprisal assigns them about 0.357 and 3.219 nats respectively. A detector using rank largely asks whether the model preferred the observed choice. A detector using likelihood also considers how strongly it preferred it.

4.4 What Sampling Changes

A language model normally produces logits; a decoding policy converts those scores into an output. Greedy decoding chooses the largest logit at each step. Random sampling instead chooses from a normalized distribution. Temperature divides logits by a positive constant before normalization. Lower temperatures widen relative probability differences, while higher temperatures flatten them. Restricting the available vocabulary changes the distribution again.

A three-token example

For logits (2, 1, 0), ordinary softmax produces probabilities approximately (0.665, 0.245, 0.090). A temperature of 0.5 doubles the logit gaps and yields approximately (0.867, 0.117, 0.016). At temperature 2, the same scores yield approximately (0.506, 0.307, 0.186). The scoring model in an evaluation can be kept fixed while the generated text changes solely because the decoding policy changed.

Generation rule	How it changes the next choice
Greedy	Uses the top-ranked token
Temperature sampling	Changes relative probability ratios
Top-k	Limits sampling to the k highest-scoring tokens
Top-p	Limits sampling to the smallest high-probability set reaching a target cumulative mass
Repetition control	Adjusts selection in response to previously used tokens or spans

Implication for detection studies

If one corpus was generated greedily and another was sampled, a metric that emphasizes high-ranked observed tokens may separate those corpora even with the same source model. A repetition setting can change both word reuse and the future contexts on which all later probabilities depend. RAID varied decoding strategies and measured material changes in detection rates. This is why generation controls should remain fixed in the paired editing experiment and be varied explicitly in a separate sampling experiment. [14]

4.5 DetectGPT and Passage Perturbations

DetectGPT posits that samples from a particular model tend to lie in high-scoring regions relative to small text perturbations. To test this, it scores the passage and a collection of modified versions under a scoring language model. A separate model can fill masked spans to generate the modifications. The discrepancy expresses how far the original score lies above the mean of those alternatives. [3, 18]

D = (ℓ(original) − mean(ℓ(alternatives))) / SD(ℓ(alternatives))

Worked calculation

Take an original mean log score of −1.85 and four perturbation scores of −2.00, −2.10, −1.90 and −2.20. Their mean is −2.05. Their population standard deviation is approximately 0.1118. The normalized discrepancy is therefore about 1.789. A version of the measurement that uses sample standard deviation would give a different value for the same four alternatives, especially with a small perturbation set. The implementation convention belongs in a replication report.

```
from detector_math import detectgpt
print(detectgpt(-1.85, [-2.0, -2.1, -1.9, -2.2]))
```

Why a rewrite affects the comparison

The original scorer and the model producing perturbations define the neighborhood. If a small rewrite changes which wording is unusually favored by the scorer, the original-to-perturbation discrepancy can shrink or reverse. A perturbation can also change meaning or register. That effect would mix the proposed local-probability signal with behavior introduced by the perturbation generator.

Reproduction details

Report the scoring model, perturbation model, masking rate, span length, perturbation count and handling of failed perturbations. Match the score normalization across original and alternative passages, especially when rewrites differ in length. If all perturbations receive the same score, the denominator vanishes. The attached reference function raises an error for that case so it can be logged rather than silently classified.

4.6 Fast DetectGPT and Conditional Alternatives

Fast-DetectGPT replaces full-passage perturbations with alternatives at individual token positions. For fixed observed prefixes, let $q_t(v)$ be the reference probability of vocabulary token v, and let $L_t(v)$ be the scoring model's log probability for the same token. These two vectors must refer to the same vocabulary entries. [4, 19]

$$\mu_t = \Sigma_v \, q_t(v) L_t(v)$$

$$\sigma_t^2 = \Sigma_v \, q_t(v)(L_t(v) - \mu_t)^2$$

$$D = \Sigma_t \, (L_t(x_t) - \mu_t) \, / \, \sqrt{(\Sigma_t \, \sigma_t^2)}$$

The expectation is the typical log score across the alternatives at each position, weighted by their reference probabilities. The numerator sums the observed tokens' excess over that baseline. The denominator follows from treating the conditional draws across fixed positions independently in the reference calculation. This assumption refers to the simulated alternatives. The tokens of an actual document still depend on one another.

A one-position calculation

Use a reference distribution that gives each of two tokens probability 0.5. Suppose their scoring-model probabilities are 0.8 and 0.2. The two log scores are about −0.223 and −1.609. Their weighted mean is −0.916, and their standard deviation is 0.693. Observing the first token gives +1 after standardization. Observing the second gives −1. Neither value is a class probability.

```
from math import log
from detector_math import fast_detectgpt
print(fast_detectgpt([[0., 0.]], [[log(.8), log(.2)]], [0]))
```

Reproduction details

The analytic computation sums over the vocabulary instead of sampling many alternatives. Implementations must align target rows, vocabulary IDs, attention masks and input truncation. If the variance is zero, standardization is undefined. When comparing two models whose token IDs differ, equal vocabulary dimensions provide insufficient evidence of alignment. Use compatible tokenizers or an explicitly justified mapping.

4.7 Binoculars and Cross Entropy

Binoculars uses two related language models, an observer and a performer. In the public mean-mode implementation, the numerator is the performer's mean negative log probability on observed next tokens. The denominator averages a cross-entropy term whose distribution comes from the observer and whose log probabilities come from the performer. A low ratio receives the machine-generated label at the repository's selected operating point. [5, 20, 23]

$$H(P,Q) = -\Sigma_v P(v) \log Q(v)$$

$$B = \text{mean observed-token NLL} / \text{mean observer-to-performer cross-entropy}$$

Cross-entropy is asymmetric. With P = (0.5, 0.5) and Q = (0.8, 0.2), H(P,Q) ≈ 0.9163. Reversing the roles yields H(Q,P) ≈ 0.6931. If the observed token is the performer's 0.8 option, the numerator is −log 0.8 ≈ 0.2231. With those same probabilities at every included position, the illustrative ratio is about 0.2435.

A detail that affects numerical parity

The released numerator shifts logits and labels for next-token prediction and masks shifted targets. The denominator uses a separate padding mask over the input positions. For a single unpadded input, this includes the terminal prediction row, even though that row has no observed next token for the numerator. The attached example mirrors that convention for unpadded mean-mode input. Thresholds from a release should be used only with its corresponding model pair and scoring conventions.

```
from detector_math import binoculars
# The prediction rows include a terminal row.
raw_score = binoculars(observer_rows, performer_rows, target_ids)
```

Interpretation

The ratio is a comparison between observed predictability and a model-to-model baseline. It is not calibrated by construction and need not be bounded by 1. Exponentiating both quantities before taking their ratio produces a different statistic, even if both are informally called perplexities. When an analysis expects larger values to indicate stronger AI evidence, it must reverse both the score direction and the threshold inequality consistently.

4.8 How a Trained Classifier Learns a Decision

A supervised detector has a labeled training set. An encoder maps each document to contextual vectors, a pooling rule combines them, and a classification head maps the pooled vector into one or more logits. For a binary decision with logit a, the sigmoid maps a to a number between zero and one. Cross-entropy penalizes disagreement between that number and the observed training label.

$$q = 1 / (1 + \exp(-a))$$

$$\text{loss} = -y \log q - (1 - y) \log(1 - q)$$

For a positive training example, a prediction of 0.90 contributes about 0.105 to the loss. A prediction of 0.10 contributes about 2.303. The same probability in a deployed system may describe the training environment poorly if the observed topic, genre or generator shifts. A calibration check asks whether cases assigned a given probability show a corresponding event frequency on comparable held-out data.

Adding sentence-level supervision

A shared encoder can support a document head and a sentence head. The document head may distinguish human, AI and mixed writing, while the sentence head learns from local labels where they are available. A weighted joint loss penalizes errors at both scales. Missing sentence labels are masked, because absence of a label is not evidence for either class. GPTZero’s technical report describes a joint hierarchical design, while details of the released production system remain proprietary. [7, 8]

$$L(\text{total}) = L(\text{document}) + \lambda\, L(\text{labeled sentences})$$

A public point of comparison

OpenAI’s earlier GPT-2 detector publicly describes a RoBERTa model fine-tuned to distinguish GPT-2 outputs from human text. The source provides a concrete route for studying a trained detector. Its checkpoint and training task are historically specific. Applying it to recent models tests transfer to a new distribution and does not reproduce the performance claimed for current commercial classifiers. [24]

4.9 Watermark Counting and Statistical Evidence

In a green-list watermark, the generator assigns a subset of candidate next tokens to a keyed group for each context. Increasing the logits of the group makes its tokens more likely to be selected. The detector regenerates the group assignments and counts the observed tokens that match. Its null model describes what would happen without the intended bias. [10, 11, 25]

$$E[G] = n\gamma; \operatorname{Var}(G) = n\gamma(1 - \gamma)$$

$$z = (G - n\gamma) / \sqrt{(n\gamma(1 - \gamma))}$$

The variance follows by adding the variances of n independent Bernoulli indicators with success probability γ. In real text, repeated contexts can cause dependence between indicators. A repeated string might repeatedly visit the same keyed choices. A detector must account for such behavior before using the reference distribution to support a false-positive claim.

Worked example and assumptions

For n = 200, γ = 0.5 and G = 130, the standardized value is approximately 4.243. The arithmetic alone verifies the count statistic. A probability statement requires an appropriate null, prespecified key and analysis procedure, and a decision rule. In the opposite direction, an unremarkable z score cannot distinguish among unwatermarked generation, weak watermark evidence, and a human-written passage.

```
from detector_math import watermark_z
print(watermark_z(green=130, total=200, gamma=.5))
```

Sensitivity to revision

Replacing a token can change its own membership and change subsequent contexts. A test of a watermark after editing therefore needs genuine watermarked baseline outputs, a fixed key and a comparable set of unwatermarked controls. A generic style classifier and a keyed watermark detector answer distinct questions. Their results can disagree without either algorithm malfunctioning.

4.10 Turnitin AI Writing Detection

Turnitin publishes a separate AI Writing Report and Similarity Report. A low similarity percentage does not imply a low AI-writing percentage. The AI-writing percentage estimates how much qualifying prose the detector flagged. It is neither the model's probability that a document used AI nor a count of all words in the file. [9]

Historical scoring design

Turnitin's early FAQ describes roughly five-to-ten-sentence segments. The segments overlap so the same sentence can be evaluated in more than one context. The FAQ describes contextual sentence scoring followed by document-level aggregation. This gives a plausible mechanism for why identical wording might receive different treatment inside different paragraphs. The disclosure is historical. The current weights, aggregation details and decision rules are proprietary. [27]

Eligibility and reporting

The current guide requires at least 300 words of long-form prose. Tables, bullet points and other non-prose elements may fall outside the qualifying text. Reports in the 1% to 19% range display an asterisk instead of an exact percentage and individual highlights. Turnitin also describes detection of some AI-paraphrased and bypasser-modified text in its English model. This is a vendor disclosure about coverage. It does not quantify success for the rewrites in this paper. [9, 28]

Report case	How to record it
Exact percentage	Keep the original value and any available highlights
*%	Record the suppressed band without imputing an exact value
Missing or unsupported	Record as unscorable, with the reason if available

Turnitin-specific test

Use authorized reports on drafts that satisfy the prose requirement before and after editing. Hold the file type and detector version fixed. Compare both the report category and the qualifying text span where available. A transition from an exact percentage to an asterisk is a reporting-band change. A missing report cannot establish a bypass. The synthetic scores in this paper supply no measured Turnitin result.

5 Experimental Design

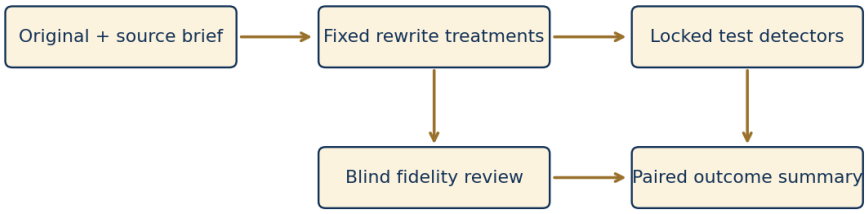


Figure 1. Each original draft yields a set of paired variants. Detector decisions and independent fidelity reviews are joined for analysis.

Corpus and unit of analysis

A pilot could generate 60 drafts from 20 prompts and three models, balancing the prompts across at least two distinct genres. At larger scale, adding independent prompts is more informative than generating many near-duplicates of each one. Give each prompt a source brief specifying the required claims, figures, units, qualifications, and citations where applicable. Retain the generator version, date, settings, input, and output. The document is the unit scored by a detector. The prompt family is the unit used to account for shared content.

Estimate false positives on independently verified human texts comparable in language, genre, task, and length. Keep human-only writing separate from human drafts edited with AI when the study aims to distinguish editing from drafting. Human samples written for unrelated prompts may introduce differences that the classifier can recognize independently of authorship.

Separation of data

Split	Purpose	Boundary
Development	Set editing instructions, identify software failures, explore candidate selection	Prompt families and templates here stay out of confirmation
Calibration	Choose a threshold or interpret vendor defaults	Use human-authored data independent from development and final test
Confirmation	Evaluate locked treatments at the locked threshold	Freeze documents, settings and fidelity rules before inspecting results

Archive the exact text seen by the scanner, or retain an immutable copy and its hash. Use the same file format across treatments unless the format itself is under test. Record refusals and truncation separately. A scanner that declines a document has supplied no classification, while one that truncates a passage has classified only the portion it processed.

6 Rewrite Treatments and Examples

Treatment	Permitted changes	What the comparison tests
Original	None	Baseline coverage
Layout only	Paragraph breaks and whitespace. Preserve punctuation and wording.	Whether layout changes the text representation
Sentence structure	Change clause boundaries and sentence grouping, retaining wording where possible.	Changes to context and syntax with limited lexical change
Contextual paraphrase	Rebuild sentences with neighboring context. Preserve claim order.	Sensitivity to joint changes in wording and syntax
Outline rewrite	Extract facts and relationships, then rebuild in a new explanatory order.	Sensitivity to a different explanatory order
Second paraphrase	Repeat the contextual paraphrase from the previous output.	Marginal gain and risk of accumulating factual drift

Worked example

Source facts. A footbridge inspection lasted 18 days. Four engineers took part. They documented surface wear. The source gives no severity rating and makes no claim about structural safety.

Original. The footbridge inspection lasted 18 days. The inspection involved four engineers. The inspection documented surface wear.

Sentence restructure. The footbridge inspection lasted 18 days and involved four engineers. It documented surface wear.

Contextual paraphrase. Over 18 days, four engineers inspected the footbridge and recorded surface wear.

Outline rewrite. Surface wear was recorded during an 18-day inspection of the footbridge by four engineers.

Fidelity failure. Four engineers completed an 18-day safety audit and found that the footbridge was structurally sound. The source describes an inspection and gives no safety conclusion. The revised passage therefore fails fidelity review whatever score it receives.

Prompt constraint

The editor receives the same factual constraints in each condition. The output must retain the claims, figures, units, quotations, citations, and qualifications, while staying close to the original length and addressing the same audience. The assigned treatment governs which parts of the wording or structure may change.

7 Detector Feedback and Transfer

Detector feedback belongs in a separate condition. Give both groups of rewrites the same candidate budget. In one group, select the candidate using prespecified rules for fidelity and readability. In the other, use the development detector to rank the candidates that pass the same factual review. Freeze the selections before scoring them on held-out tools. Prespecify one primary comparison and report the remaining comparisons as exploratory.

How to read the outcomes

Observed pattern	Interpretation to test
Lower development score only	Candidate selection exploited preferences specific to the development detector or measurement noise
Lower scores on held-out detectors	Evidence of transfer in the tested conditions, subject to common training signals and selection bias
Lower flags with worse fidelity	The treatment trades detection against accuracy or completeness
Higher abstention	Coverage changed. Report abstention separately from false negatives
Fewer flags and higher mean score	The population moved unevenly around the threshold, so the average and thresholded result answer different questions

Decisions to record before scoring

Before scoring, record the primary detector and decision rule, the prompts and sample size, and the budget and order of candidate edits. Specify how ties and reviewer disagreements will be resolved. A detector version or configuration change should be recorded as a separate run.

Quality annotations

Where possible, two reviewers should assess each randomly ordered passage against its source brief without seeing either the treatment or the score. They should mark each claim as preserved, contradicted, omitted, or indeterminate, then identify unsupported additions and rate readability separately. Report agreement before resolving disagreements. Similarity measures can help flag passages for review, but they can miss a reversed causal claim, an altered unit, or a negation.

8 Statistical Analysis

Primary outcomes

Quantity	Calculation
Baseline detection	Originals flagged divided by scorable originals
After-edit detection	Variants flagged divided by scorable originals and variants in the paired set
Conditional evasion	Flag removals divided by originally flagged, paired scorable cases
Fidelity-preserving evasion	Flag removals passing the rubric divided by originally flagged, paired scorable cases
New flags	Originally unflagged cases flagged after editing
Net detection change	New flags minus flag removals, divided by paired scorable cases
Human false-positive rate	Verified human controls flagged divided by scorable human controls

A paired comparison requires a scorable result for both the original and the variant. Report how many pairs were excluded and why. To show how exclusions might affect the conclusion, calculate bounds by assigning the missing decisions to each possible class in turn. Keep cases with unknown fidelity visible.

Uncertainty and multiple comparisons

Resample whole prompt groups with replacement, carrying each group’s originals and rewrites together. The resulting bootstrap interval reflects variability across the sampled prompts. It cannot account for a future model update or a mismatch between the sample and the intended population. McNemar’s test compares the two directions of flag changes when pairs are independent. With several documents per prompt, use a cluster-aware interval or test. Prespecify primary comparisons, and mark wider searches across tools, treatments, and genres as exploratory unless an appropriate adjustment is applied.

Why prevalence matters

Suppose 100 of 10,000 submissions belong to the AI class. A detector with 80% recall and a 1% false-positive rate would flag about 80 of those and 99 of the remaining 9,900. On those assumptions, 80 of 179 flags, or about 44.7%, would correspond to the target class. The example shows why a flag’s predictive value depends on prevalence and why recall needs to be reported alongside the false-positive rate and its denominator.

9 Synthetic Experiment

Implementation

The synthetic study used three disjoint sets of 60 prompt groups. A group supplied a site, an inspection duration, and a team size and yielded six examples per class. The machine-template class relied mainly on one recurring form. The reference class used several alternate forms and retained some examples in the recurring form. Deterministic Python templates generated both classes, providing two known distributions for studying the classifier's behavior.

Two additively smoothed bigram models were fitted on the training set, one per class. Each estimates a token's probability from the preceding token and the transitions observed in training. A positive detection score means the passage was more predictable under the machine-template model. The vocabulary is fixed after training, with unfamiliar tokens mapped to an unknown symbol.

S(x) = mean surprisal under reference model − mean surprisal under machine-template model

Controls

The tokenizer retains punctuation and discards whitespace. As a result, replacing the space after a sentence with a paragraph break leaves the token sequence unchanged. The script checks that every layout-only score equals its original score. The alternative templates preserve the site, duration, team size, and finding. In this demonstration, the second pass is a preset wording. The proposed real-model condition would instead edit the actual output of the first pass.

Threshold and execution

The cutoff was set using reference-class calibration data alone, at a nominal false-positive target of 15%. The implementation moves one floating-point step above the relevant order statistic, so a tie at that boundary can put the measured rate below target. With the resulting threshold of 0.490270, the calibration reference rate was 53/360 (14.7%). The held-out reference rate was 54/360 (15.0%). The prompt-group bootstrap used 2,000 resamples and a fixed seed.

10 Results from the Synthetic Lab

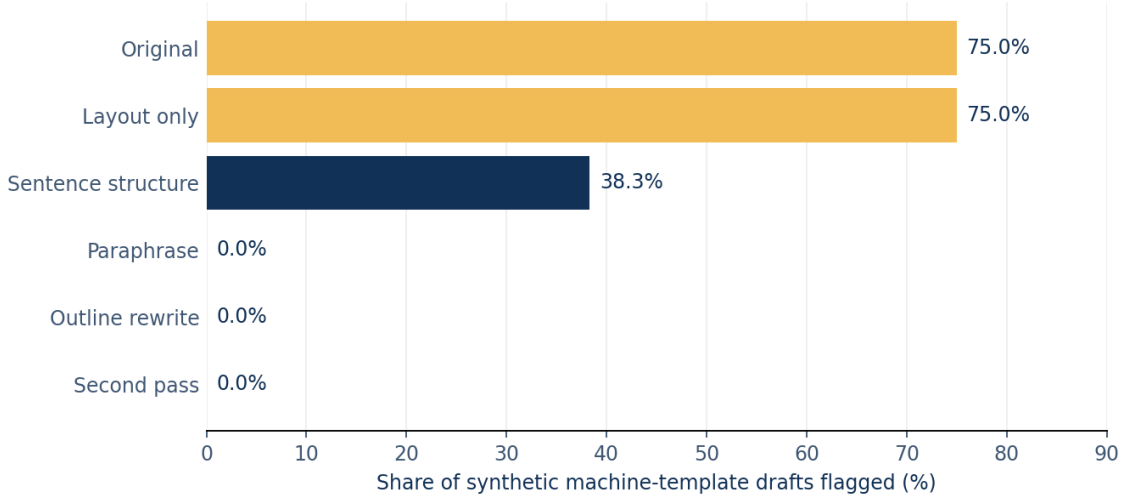


Figure 2. Share of the 360 synthetic machine-template drafts flagged under each treatment. The alternate rewrites use structures similar to those in the reference class.

Treatment	Total flags	Flags removed	New flags	Mean score change
Original	270	0	0	0.000
Layout only	270	0	0	0.000
Sentence structure	138	155	23	+0.024
Paraphrase	0	270	0	−0.453
Outline rewrite	0	270	0	−3.520
Second pass	0	270	0	−2.779

Changes in classification

Sentence restructuring removed 155 of the 270 original flags (57.4%) and caused 23 previously unflagged drafts to be flagged. The net change was therefore −132/360, or −36.7 percentage points. The prompt-group bootstrap interval ran from approximately −48.6 to −24.2 percentage points. The 360 examples include shared prompt content and recurring wording, which is why the interval resamples groups.

11 Why Scores and Flags Disagree

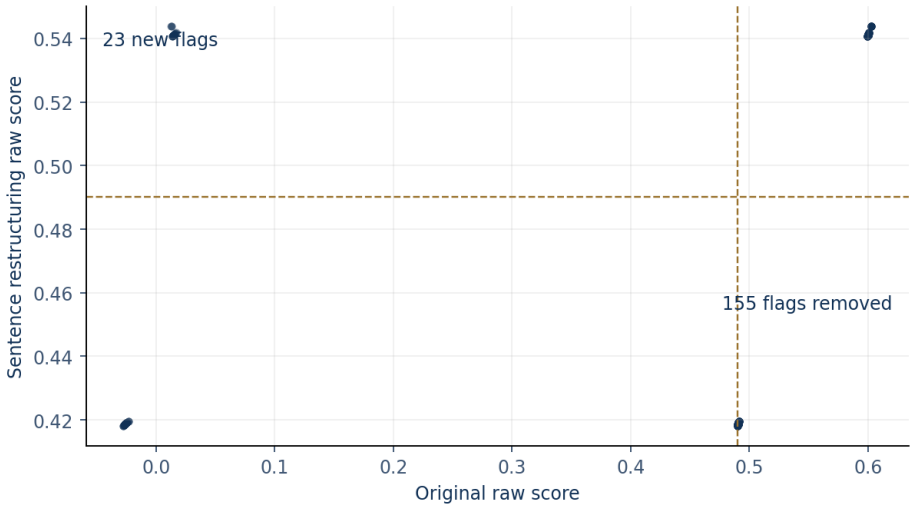


Figure 3. Original and restructured scores for each paired example. The dashed lines show the same decision threshold. Some points overlap because templates and fact combinations recur.

A mean score is sensitive to distance from the cutoff

The positive mean score change may look at odds with the decline in flags. The score rose enough in some cases to outweigh the decreases when averaged. At the cutoff, the pattern was different. More drafts crossed into the unflagged region than crossed back into the flagged region. The mean reflects the size and direction of movements on the score scale. The flag count reflects crossings at one selected threshold.

Influence of the reference templates

The alternate rewrites resemble the reference templates by construction. Their local word transitions therefore give the reference model an advantage in predicting them. The three data splits contain disjoint prompt IDs, but they reuse some fact combinations and the same writing templates. The reported interval describes variation across the sampled synthetic groups. It cannot establish an error rate for a population of independent writers or for a different detector.

Applicability to current products

A test of a current commercial scanner would need independently sourced material, generated drafts, verified human writing, and a selection procedure fixed before evaluation. It would compare the paraphrase and outline conditions directly, record abstentions, and check factual fidelity. The synthetic rates cannot substitute for those observations.

11.1 Sensitivity to the Decision Threshold

A model produces a ranking or score before a reporting rule turns it into a decision. A comparison at one cutoff cannot establish how a treatment moves the entire distribution. The synthetic scores were therefore swept across five cutoffs without refitting the classifier. This is a sensitivity analysis on held-out synthetic scores. The alternative cutoffs were not used to select or calibrate a new detector.

Cutoff	Original	Sentence structure	Paraphrase	Outline rewrite
0.300000	300	360	0	0
0.400000	300	360	0	0
0.490270	270	138	0	0
0.550000	115	0	0	0
0.650000	0	0	0	0

At 0.40, restructuring adds 60 flags in aggregate. At the main cutoff, it removes 132 net flags. At 0.55 the original still yields 115 flags and the restructured set yields none. The treatment therefore cannot be summarized as a uniform score decrease. The original and edited distributions put different masses near each cutoff.

Why the table does not select a better cutoff

A detector cutoff must reflect the false flags it produces on a relevant human comparison set, including uncertainty in that estimate. The table contains results for the machine-template class. It cannot establish an acceptable operating point on human documents. Picking the most dramatic cutoff from this table would turn a descriptive sensitivity analysis into a post hoc selection.

Reproduction

Run the attached **threshold_sweep.py** beside **synthetic_scores.csv**. The script uses the full stored cutoff for the main analysis and rounds it only when printing. A displayed value of 0.490270 is insufficient for bit-for-bit replication when observations are close to the threshold. The original and edited texts are scored on the same scale throughout.

11.2 Paired Effects and Score Change

The same source draft appears in several conditions. Treating all of those rows as independent would ignore their shared content. For one treatment, the most compact representation is a two-by-two table of original and revised decisions. The off-diagonal cells describe the event of interest. The diagonal cells establish how much of the population stayed in the same state.

Original decision	Revised flagged	Revised unflagged
Flagged	115	155
Unflagged	23	67

These counts describe sentence restructuring in the synthetic study. The unconditional probability of removing a flag is 155/360. The conditional probability among originally flagged documents is 155/270. The net change is (23 − 155)/360. All three quantities are useful because they answer different questions about a treatment.

Δflag = (new flags − flags removed) / scorable pairs

Raw scores add a different view

For each pair, subtract the original score from the revised score. The mean gives an estimate on a single model’s scale, and the median can show whether a few large movements are dominating it. Plot original and revised scores jointly to see whether changes concentrate near the threshold. A paired ranking plot can reveal cases in which the broad movement is small but enough texts cross to alter the decision rate.

Resampling clustered prompts

In the supplied bootstrap, the resampling unit is a prompt group. When a group is drawn, its entire vector of variant results travels with it. This retains the correlation induced by a shared source. A comparison pooling several tools should also account for the repeated use of each document. Separate per-tool results are usually easier to interpret than a single average that hides which detector contributed the changes.

11.3 Operating Points and Calibration

An evaluation can compare detectors at their own defaults or at a prespecified common false-positive target. Defaults describe the experience of a typical user. A common target answers how much recall each detector achieves under a specified tolerance for falsely flagging human writers. The two comparisons may produce different rankings. RAID explicitly examined performance at controlled false-positive rates. [14]

TPR(τ) = P(score ≥ τ given generated)

FPR(τ) = P(score ≥ τ given human)

The receiver operating characteristic plots true-positive rate against false-positive rate across possible cutoffs. AUROC summarizes ranking over the curve. With ties counted as half, it is the probability that a random positive scores higher than a random negative. A model can rank well on average and still have poor recall at the low false-positive rates that matter in practice.

How many negatives are enough

If zero of n independent human samples are flagged, the probability of that outcome under false-positive rate p is $(1 - p)^n$. Setting this probability to 0.05 gives a one-sided 95% upper bound of 1 − 0.05 raised to the power 1/n. For large n, this is approximately 3/n. Zero flags out of 100 allow a rate near 3%. Zero out of 1,000 allow a rate near 0.3%. The calculation assumes independent, representative samples, so many near-duplicate passages cannot be counted as independent evidence.

Probability needs separate checking

If a system reports a probability, bin comparable test cases by predicted probability and compare the observed frequency with the average prediction in each bin. A well-ranked model may still be overconfident. The Brier score, which averages the squared difference between a predicted probability and the true binary label, offers another measure. A raw curvature score or commercial percentage of prose requires its own interpretation before such a calculation is applicable.

11.4 Fidelity as a Measurable Outcome

A document can preserve familiar words and still reverse its meaning. “The intervention lowered pressure by 5 kPa” and “the intervention raised pressure by 5 kPa” differ by one verb and imply opposite outcomes. Similarly, replacing “associated with” with “caused” changes the strength of an inference even if the rest of the sentence is unchanged. The fidelity review must examine claims and their relationships rather than relying on lexical overlap.

Build a claim ledger

Assign each claim an identifier before writing variants. Record the event or relationship, its subjects, magnitude and unit, limiting conditions, source and stated uncertainty. Preserve direct quotations separately. During review, inventory claims added by the rewrite as well as claims inherited from the original. A variant passes only after the prespecified material claims are supported and no consequential unsupported addition remains.

Claim dimension	Example failure
Direction	Increased becomes decreased
Magnitude or unit	5 kPa becomes 5 Pa
Scope	Under dry conditions becomes under all conditions
Evidence strength	Association becomes causation
Provenance	A finding is attributed to a different study

Two ways to summarize quality

Report a strict document-level pass rate to answer how often a rewrite is usable without substantive correction. A separate claim-level error rate helps locate what went wrong. A document with one altered numerical result may matter more than a document with three minor omissions. Decide how to treat ambiguous evidence before seeing the detector results, and report initial reviewer agreement before adjudication.

11.5 Error Analysis Across Populations

The patterns a detector sees depend on its training corpus and the population it is asked to evaluate. Coursework, product reviews and technical abstracts differ in their allowable vocabulary, regularity and dependence on templates. A study can produce an aggregate score that looks strong while performing poorly in a particular genre or language group.

Match more than length

A useful human reference set shares the prompt type, intended audience and constraints of the generated samples. If humans were asked to write timed exam responses while AI systems were asked to produce polished essays, the detector can separate the resulting registers. That does not tell us how well it would separate humans and models writing under comparable conditions. The same logic applies to differences in editing support and translation.

What an observed disparity supports

Liang and colleagues reported a large disparity on the human TOEFL essays in their evaluation. The result supports testing for unequal false-positive rates. It does not identify which proprietary feature caused the disparities or supply today’s error rate for every multilingual group. Compare performance on matched contemporary corpora, retain sample sizes and uncertainty by subgroup, and document differences in prompt and collection method. [15]

Mixed text is its own task

A binary label is inadequate when paragraphs have different authorship histories or when a model adjusts grammar throughout the whole text. Marking a single transition boundary is measurable on constructed continuations, as the SemEval task illustrates. Iterative co-editing may lack a clean token boundary. A robust benchmark should distinguish insertion, generation followed by human revision, and human drafting with machine editing. It should document how those processes are mapped into the categories each detector supports. [16]

11.6 Benchmark Design and Provenance

The central comparison should be fixed before outcome data are examined. A benchmark can support several questions, but a primary endpoint determines what evidence would count as a successful replication. For example, the outcome might be the paired change in recall for outline rewrites versus contextual paraphrases on a held-out model family, counting only variants that pass factual review.

Document the sampling frame

List domains, generators, decoding settings and the source of human examples. Include prompt identifiers and shared factual briefs. Keep related texts together when splitting data. If a benchmark draws multiple near-identical documents from the same template, both training and test can reward recognition of that template. A stronger transfer test holds out a generator, a family of prompts, or a genre, depending on the claim under study. [14, 16, 26]

Keep provenance independent of the score

A generated document should be labeled from a confirmed generation record rather than the judgment of another detector. A human-origin document should have provenance appropriate to the study, such as a documented writing session or authenticated revision history. For hybrid examples, record what operation the model performed and where its contribution appears. Weak labels can otherwise make a competitor’s recognition of a similar style look like agreement with ground truth.

Collect the complete record

Archive the native detector response, cutoff, version, date and treatment. Preserve adjudicated fidelity judgments and the pre-adjudication labels used to assess agreement. Report the number of pairs that were generated, scored, refused, truncated and excluded. If an outside detector changes during data collection, stratify analyses by version to avoid interpreting product updates as editing effects.

11.7 Interpretation and Replication

The calculations in this report separate three kinds of result. Published studies establish performance within their own test conditions. The synthetic experiment establishes what happened to an inspectable bigram classifier under controlled template changes. The external evaluation protocol identifies what still needs to be observed on current detectors. These results differ in the evidence they supply.

From mechanisms to hypotheses

Changes in local token context motivate a prediction about likelihood and rank. Reordering claims affects both local context and longer passages processed by a classifier. A keyed watermark has a separate dependence on tokens and preceding contexts. If several metrics change together after rewriting, this suggests possible mechanisms. It cannot identify what a proprietary model relied on inside its decision unless further evidence or controlled experiments support that inference.

A useful replication sequence

First, run likelihood and rank on fixed texts with a declared scoring model. Check the returned token count and the effect of changing only layout. Next, compare detector decisions using frozen drafts and thresholds. Finally, inspect both directions of the paired transitions and check the original claims in every apparent improvement. The procedure can be repeated on unseen generators or genres. A result that follows the source topic across conditions is different from a result that generalizes to unrelated topics.

Limits of the implementation

The attached mathematical reference uses controlled logits so each calculation can be checked by hand. The causal-model example supplies real logits when compatible dependencies and weights are available. None of the runnable examples requires a commercial scanner. Production-level performance requires a separate dataset, a versioned model or service, and a defensible operating point, regardless of whether the mathematical core is public.

12 Reproducibility and Failure Checks

Run the attached lab

A PDF reader with attachment support can extract the scripts and data. The lab uses only the standard library in Python 3.10 or later and writes a JSON summary and a CSV containing all 2,160 scored variants.

```
python detector_lab.py --output lab_output
python audit_scores.py example_audit.csv
```

The evaluator reads one row per detector, draft, and treatment. Required columns are **detector_version**, **draft_id**, **treatment**, **decision**, and **fidelity**. Each detector–draft pair needs a row named **original**. The decision must be **flagged**, **unflagged**, or **unscorable**. Fidelity may be **pass**, **fail**, or **unknown**. The script counts paired transitions and detects duplicate rows or missing originals. Record a missing planned scan as unscorable, with a separate column for the reason, so it remains visible in the denominator.

Audit checks

Failure mode	Check
Prompt or author leakage	Group splits before training or previewing held-out outcomes
Threshold leakage	Apply a versioned cutoff established using only calibration data
Score sign error	Verify the native score direction before comparing numeric outputs
Changed ground truth	Retain provenance labels after edits, with fidelity recorded separately
Misleading average	Show score changes, both transition directions, and net flag change
Scanner truncation	Record length and actual analyzed span when available
Version drift	Archive timestamp, model or endpoint version, and vendor rules
Synthetic overreach	Label synthetic outputs and report actual external results only after running the protocol

Answer to the research question

Published experiments show that contextual rewriting can substantially reduce detection under some conditions. The synthetic study illustrates one mechanism and shows why a lower flag rate can coexist with a higher average score. Whether a specific rewrite has the same effect on new prompts or current detectors remains a question for the paired experiment, with each apparent gain checked against the original facts.

Appendix A Code Walkthrough

Tokenization and class score

The scorer compares the passage’s mean negative log-likelihood under two local transition models. Additive smoothing lets both models assign a nonzero probability to an unseen transition.

```
def score(text):
    return reference.surprisal(text) - machine.surprisal(text)
```

Calibrating a threshold with ties

The calibration scores are sorted from largest to smallest. The cutoff sits one floating-point step above the next score after the permitted number of flags. Observations tied at the boundary therefore stay below the cutoff.

```
quota = math.floor(target * len(scores))
boundary = sorted(scores, reverse=True)[quota]
threshold = math.nextafter(boundary, math.inf)
```

Paired record

Every variant is linked to the original draft and its flag. This record makes it possible to count transitions in opposite directions and to resample the corresponding prompt groups.

```
baseline_flagged = baseline >= threshold
variant_flagged = current >= threshold
flag_change = int(variant_flagged) - int(baseline_flagged)
```

Scoring a real detector

Save the detector’s raw output before aggregating it. If a product reports categories or suppresses some values, retain the original labels and apply the decision mapping defined before scoring. The evaluator then compares resulting decisions without treating the products’ numerical scales as equivalent.

Attachment manifest

File	Purpose
detector_lab.py	Generates data, fits the classifier, calibrates and bootstraps results
audit_scores.py	Aggregates explicitly labeled paired observations from real or synthetic scans
example_audit.csv	Four-draft evaluator example with fidelity failure and abstention
synthetic_scores.csv	All synthetic held-out detector scores and text variants
synthetic_summary.json	Exact run results and calibration details

Appendix B Expanded Audit Sheet

Source brief

Field	Example
Prompt ID and genre	BR-047, engineering inspection summary
Required claims	Inspection took 18 days, employed four engineers, documented surface wear
Scope limitations	No severity rating. No structural safety conclusion.
Intended register and length	General technical audience, target length set before generation
Review rule	All three claims preserved with no contradictory or consequential extra claim

Draft record

Field	What to capture
Generation	Provider, model ID, settings, timestamp, prompt and input hash
Edit history	Source draft ID, treatment, number of edits, candidate index, editor model and settings
Detector	Tool and version, rules, run date, raw output, accepted span, native decision
Human review	Reviewer identifier, order randomized, claim verdicts, unsupported additions, readability
Analysis	Baseline decision, edited decision, fidelity outcome, abstention reason, prompt cluster

Interpreting an illustrative pilot

Suppose the detector flags 40 of 60 original texts. If rewriting removes 18 flags and adds three, 25 texts remain flagged (41.7%, down from 66.7%). Among the 40 originally flagged texts, 45% lost the flag. If 13 of those removals also passed the fidelity review, the fidelity-preserving removal rate is 32.5%. The net change describes the entire set of 60, while the removal rates describe only the 40 originally flagged drafts.

Exploratory measurements

Word count, sentence length, character edit ratio, n-gram overlap, and surprisal can help explain why some scores moved. A correlation with one of these measurements does not isolate it as a cause when a rewrite changes several features at once. A follow-up condition that alters only paragraph breaks or sentence boundaries offers a narrower test.

Appendix C Copy and Paste AI Prompt

Copy the block into an AI chat and replace the bracketed input fields. The prompt applies the paper’s four rewrite treatments and checks them against the source before using detector feedback. Use a brief that lists the claims and evidence the rewrite must retain.

```
Use the text in the DRAFT field and the facts in the BRIEF
field. Create four rewrites without adding any claims.
1. Restructure clauses while preserving the order of the ideas.
2. Paraphrase each idea in context, keeping the same order.
3. Extract a fact outline, then rebuild the explanation in a
   different order while preserving the logical relationships.
4. If version 3 passes review, paraphrase it one more time.
Check every version against the draft and brief. Reject versions
that lose or alter facts, numbers, units, quotations, citations,
or qualifications. Keep the audience and length comparable.
If you have access to an actual detector and its fixed cutoff,
score the original and the candidates that passed review. Select
the lowest AI-likeness score only if it improves on the original.
Report the baseline, final score and whether the tool's decision
actually changed. Otherwise return the most faithful, readable
candidate and label detector performance untested. Return the
revised passage under a separate heading for easy copying.

BRIEF: [Paste the source facts and constraints here.]
DRAFT: [Paste the original prose here.]
```

Verify the reported result

A prompt can generate a candidate but cannot reveal how Turnitin or another service scores it without a real report. A measured bypass is a flag that disappears under the same detector version and predeclared decision rule. When the selection detector is available, test the selected rewrite separately on a held-out detector to measure transfer.

Appendix D Runnable Rewrite Selection

The attached **rewrite_protocol.py** accepts three functions: an AI editor, a reviewer that checks factual fidelity, and a detector that returns a score. The script scores the original and proposes up to four rewrites. It scores only those that pass review and retains the lowest-scoring candidate if it improves on the baseline. Ties follow the treatment order in Appendix C.

Connect the inputs

```
from rewrite_protocol import select_rewrite
result = select_rewrite(
    original=draft,
    brief=source_brief,
    edit=editor,
    preserves=review,
    score=detector_score,
    threshold=locked_threshold,
)
print(result["text"])
print(result["flag_removed"])
```

The **editor(instructions, brief, passage)** function returns revised prose. The **review(original, brief, revision)** function returns True when the material claims are preserved. The **detector_score(passage)** function returns a finite number that increases with estimated AI-likeness. Convert scores where necessary and use a cutoff on the same scale.

Run the bundled example

```
python rewrite_protocol.py
```

In the bundled example, fixed inputs give the original a score of 0.82 and the outline rewrite 0.27, with a threshold of 0.50. The reviewer rejects a candidate claiming there was no structural damage because that conclusion is absent from the source. The supplied scores test how the code handles a flag removal and a factual error. They come from the example, not a scan.

Preserve the audit trail

Keep the original and selected text with the treatment, fidelity verdict, score, threshold, detector version, and classification change. Held-out scores belong in a separate record. The protocol can be followed directly from the preceding page, while automatic selection requires the three functions above.

References and Source Notes

Foundations and detection methods

[1] OpenAI. 2023. New AI classifier for indicating AI-written text. Open source

[2] Gehrmann, S., Strobelt, H., and Rush, A. M. 2019. GLTR: Statistical Detection and Visualization of Generated Text. ACL System Demonstrations. Open source

[3] Mitchell, E., et al. 2023. DetectGPT: Zero-Shot Machine-Generated Text Detection Using Probability Curvature. ICML. Open source

[4] Bao, G., et al. 2024. Fast-DetectGPT: Efficient Zero-Shot Detection of Machine-Generated Text via Conditional Probability Curvature. ICLR. Open source

[5] Hans, A., et al. 2024. Spotting LLMs With Binoculars: Zero-Shot Detection of Machine-Generated Text. ICML. Open source

[6] Verma, V., et al. 2024. Ghostbuster: Detecting Text Ghostwritten by Large Language Models. NAACL. Open source

[7] Adam, G. A., et al. 2026. GPTZero: Robust Detection of LLM-Generated Texts. arXiv version 2. Open source

[8] GPTZero. 2026. How do I interpret burstiness or perplexity? Open source

[9] Turnitin. Using the AI Writing Report. Open source

[10] Kirchenbauer, J., et al. 2023. A Watermark for Large Language Models. ICML. Open source

[11] Dathathri, S., et al. 2024. Scalable watermarking for identifying large language model outputs. Nature. Open source

Notes on use

The likelihood and rank chapters use the GLTR paper as a source for the visualized token ranks. The DetectGPT and Fast-DetectGPT papers supply the statistical motivation. The worked numerical examples in this paper use deliberately small, invented inputs that can be checked directly. For Binoculars, consult the public implementation listed on page 38 when reproducing the exact row convention.

References on Rewriting and Evaluation

Robustness and benchmark studies

[12] Sadasivan, V. S., et al. 2023. Can AI-Generated Text be Reliably Detected? arXiv preprint. Open source

[13] Krishna, K., et al. 2023. Paraphrasing evades detectors of AI-generated text, but retrieval is an effective defense. NeurIPS. Open source

[14] Dugan, L., et al. 2024. RAID: A Shared Benchmark for Robust Evaluation of Machine-Generated Text Detectors. ACL. Open source

[15] Liang, W., et al. 2023. GPT detectors are biased against non-native English writers. Patterns, 4(7), 100779. Open source

[16] Wang, Y., et al. 2024. SemEval-2024 Task 8: Multidomain, Multimodel and Multilingual Machine-Generated Text Detection. SemEval. Open source

[26] Li, Y., et al. 2024. MAGE: Machine-generated Text Detection in the Wild. ACL. Open source

[27] Turnitin. 2023. FAQs for Turnitin’s AI writing detection capabilities. Historical vendor FAQ hosted by Siena. Open source

[28] Turnitin. Why is the AI Writing Detection report score showing as *%? Open source

Interpreting the benchmarks

The benchmark findings in this paper are tied to the versions, samples and operating points reported by the authors. The original paper introducing a detector provides evidence from its evaluation. An independent benchmark can test different populations and settings and may reveal performance changes absent from the original report. Commercial documentation establishes what the tool says its outputs mean. Those descriptions alone cannot establish accuracy on a new population.

Open Source Implementations

Published code

[17] Gehrmann, Strobelt and Rush. GLTR source and visualization. Open source

[18] Mitchell et al. DetectGPT source. Open source

[19] Bao et al. Fast-DetectGPT source. Open source

[20] Hans et al. Binoculars source. Open source

[21] Google DeepMind. SynthID Text reference implementation. Open source

[22] Krishna et al. DIPPER project materials. Open source

[23] Hans et al. Binoculars metrics.py, public scoring implementation. Open source

[24] OpenAI. GPT-2 Output Detector, public RoBERTa detector implementation. Open source

[25] Kirchenbauer et al. lm-watermarking, public watermark implementation. Open source

Reproduction scope

The linked repositories provide the authors' implementations and their associated configuration details. The scripts embedded in this PDF independently implement the equations and audit logic. Their examples do not reproduce the full training pipelines or results of the linked systems. A close numerical match requires identical weights, tokenization, context handling, masks and score conventions.

Reproduction Files and Commands

The PDF contains the files below. Extract the attachments together into one folder using a PDF reader that supports embedded files. The mathematics demo, synthetic classifier, threshold sweep, score evaluator and rewrite-selector demonstration run with the Python standard library. The optional causal-model example also requires PyTorch and Transformers, plus available model weights.

File	Purpose
detector_math.py	Reference implementations of the published scoring equations
detector_math_output.json	Exact output of the controlled numerical examples
causal_lm_example.py	Obtain live next-token logits from a compatible causal model
detector_lab.py	Fit and evaluate the synthetic bigram classifier
threshold_sweep.py	Sensitivity table computed from the held-out synthetic scores
audit_scores.py and example_audit.csv	Aggregate paired decisions and exercise refusal and fidelity cases
rewrite_protocol.py and ai_rewrite_protocol.yaml, plus copy_prompt.txt	Python selector and AI-ready protocol
synthetic_scores.csv and synthetic_summary.json	Full synthetic outputs and main run summary

Commands

```
python detector_math.py
python detector_lab.py --output lab_output
python threshold_sweep.py
python audit_scores.py example_audit.csv
python rewrite_protocol.py
python causal_lm_example.py "The bridge expands in the heat."
```

The optional final command may download a public model if its weights are not cached. The adapter was checked with local test doubles for tokenization and model output. Production-model inference requires separate execution against the chosen model weights.

Glossary and Reporting Conventions

Term	Meaning in this paper
Calibration	Choosing a cutoff or assessing whether model probabilities match event frequencies, depending on context
Cross-entropy	Expected negative log probability under a scoring distribution, with expectations taken under a reference distribution
Detector evasion event	A known generated draft moves from flagged to unflagged under a fixed rule
Fidelity	Preservation of the material claims, qualifications, evidence and scope of a passage
Held-out	Reserved from training, calibration or selection for the corresponding final comparison
Logit	An unnormalized score converted to a probability distribution using softmax
Perturbation	A modified version of the text used to probe a detector statistic
Surprisal	Negative log probability of an observed token in its context
Token	A text fragment represented by a vocabulary ID
Unscorable	A submitted passage received no usable classification in the specified run

Statistical notation

Natural logarithms are used throughout. NLL denotes negative log-likelihood, and probabilities are expressed on the unit interval unless labeled as percentages. Percentage-point changes are differences between two rates. A score delta is a difference on the tool’s own scale. Both directions of paired decision changes are counted before computing a net change.

Study record

The synthetic analysis uses the original seeds and test cases. All experimental quantities in the paper’s synthetic results and sensitivity table come from the embedded reproducible data. The score examples in the mathematical chapters are explicitly constructed examples. No new commercial-detector performance measurements are claimed.