arXiv:submit/8366445 [cs.CR] 12 Nov 2026

STEALTH: Steganographic Text Encoding Agnostic to Low-Level Tensor Hardware

William Fleshman
Army Cyber Institute
United States Military Academy
West Point, New York 10996
william.fleshman@westpoint.edu

***Abstract*—Provably secure linguistic steganography hides a message inside text sampled from a language model, so that without the key, a stegotext is computationally indistinguishable from an honest model sample. Nearly every such scheme assumes the receiver can recompute the sender's token probabilities exactly bit for bit. We show this assumption is fragile: across three model families, bit-exact schemes decode almost none of our messages once the receiver's floating-point precision, quantization, or GPU architecture differs from the sender's, or even when the receiver merely scores a text in parallel versus one token at a time, with the same model, precision, and hardware held fixed. We present STEALTH, a scheme built for exactly this setting. The sender draws each token with an exponential race driven by keyed pseudorandom noise, so a receiver's probability mismatch flips a token only when the race is close; a rateless spinal code, decoded by beam search, absorbs the resulting flips without the sender and receiver ever agreeing on a single distribution. We prove stegotexts remain indistinguishable from honest samples under this construction. Empirically, STEALTH decodes every message we sent for every receiver in the sender's model family, at every precision and quantization level, on smaller models, and on a different GPU architecture. STEALTH is even robust across different model families with separate vocabularies and tokenizers, decoding every message we tested. Our approach carries far more payload per token than prior robust schemes, and trained steganalyzers and a targeted statistical test both fail to distinguish its stegotexts from honest model output.**

1. Introduction

State and corporate surveillance of digital communication is pervasive [1], and censorship regimes increasingly flag or block traffic that looks encrypted or otherwise anomalous [2]. While encrypted communications maintain privacy, they are easily identified. A useful alternative is covert channels: hiding messages inside communication that looks benign, so there is nothing anomalous for a censor or surveillant to flag. Steganography is the general technique, embedding secret messages into ordinary media such as images or text so the resulting cover object appears unremarkable to anyone but the intended recipient (Section 2). Ideally, a steganographic technique should be *provably secure*, indistinguishable from the cover channel even to a warden who can run the embedding algorithm itself, rather than merely undetected by today's tools [3].

Achieving this for images can be difficult since the true statistics of natural images are hard to model exactly [4]. Modern language models (LMs) make text more tractable because a good LM provides an explicit probability distribution over realistic text continuations. Linguistic steganography uses this distribution to hide messages: the sender uses the message bits to choose each sampled token, and the receiver, recomputing the same distribution, reads the bits back from the tokens that are observed. Several such schemes are provably secure [5–7], but only if the receiver reproduces the sender's distribution bit for bit, since the message is encoded in exactly where each sampled token falls within it (Section 2). This bit-exact requirement is harder to meet than it sounds: floating-point reductions are not associative, so different tensor hardware, kernels, precision, or quantization yield different results for the same model and text, and even identical hardware can disagree between generating text token by token and scoring it in a single pass [8–10]. In practice, sender and receiver probabilities almost always differ, and because decoding replays the sender's computation step by step, a single token whose probability differs in the last few bits can send every subsequent step down the wrong path. A recent survey of linguistic steganography lists this computational indeterminism as an open challenge [11].

We design STEALTH to tolerate this mismatch while remaining provably secure (Figure 1). The sender picks each token by an *exponential race* over the token distribution p, choosing the token that minimizes E_i/p_i, where E_i is keyed pseudorandom exponential noise. With truly random noise, this samples exactly from p; with keyed noise, it is indistinguishable from doing so [12, 13]. If the receiver's distribution p' differs from p, the same noise picks a different token with probability bounded by a simple function of the total variation distance between the two distributions. The receiver scores the cover text one position at a time using the true preceding tokens as context (teacher forcing), so a flip affects only its own position and never the context of later tokens. We design the noise at each position to depend on the message through a *spinal code* [14], a hash chain over

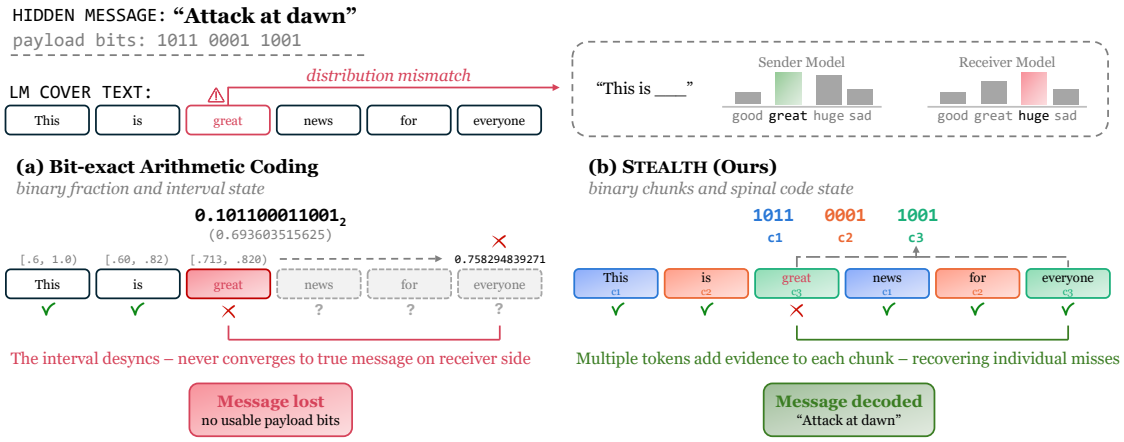


Figure 1. Differences in sender and receiver token distributions cause the receiver to lose sync at a single token, desyncing bit-exact decoding schemes and losing the message entirely; STEALTH's spinal-code state lets later tokens supply evidence that recovers the payload despite the mismatch.

message chunks that the receiver decodes by an efficient beam search with a soft likelihood treating a narrowly lost race as a probable flip rather than a contradiction. Because the code is rateless, every additional token the model writes contributes more evidence toward the payload, so the sender never needs to truncate the text to fit a fixed message length. Positions in the text are measured in byte offsets, and the noise at each position can be keyed by the token's bytes rather than its vocabulary index, so nothing the receiver computes depends on the sender's tokenizer. This lets the receiver decode with an LM from an entirely different model family, with its own vocabulary and tokenizer.

Like any steganographic technique, STEALTH is dual-use: the same mechanism that lets a dissident evade censorship can let an adversary run a covert channel undetected. Because STEALTH removes the deployment-matching requirement that made earlier schemes fragile, it also lowers the bar for using steganography as a reliable, hard-to-detect channel between two parties with varying infrastructure. Of note, any defense that relies on sender and receiver running mismatched deployments to disrupt steganographic coordination will not hold against a scheme like STEALTH. A sender on a datacenter GPU and a receiver on a consumer GPU running a 4-bit quantized model of the same family can still communicate, with messages provably indistinguishable from honest model output. In this paper, we:

- Demonstrate that schemes relying on bit-exact computation fail under ordinary mismatches in hardware, precision, or sequence processing procedures;
- Propose STEALTH, which tolerates these mismatches, even enabling senders with speculative decoding, or receivers with different models altogether;
- Prove STEALTH is secure and produces text indistinguishable from benign model output; and
- Evaluate STEALTH across three model families, five receiver configurations per family, smaller same-family receivers, receivers of other model families, and two GPU architectures, against bit-exact and robust baselines, with statistical detection tests.

2. Background and Related Work

We first place STEALTH within the steganography and covert-channel literature, then define sampling from a language model and what it means for a stegosystem built on one to be provably secure. We then explain why a receiver cannot count on recomputing the sender's distribution exactly, and cover the sampler STEALTH uses to tolerate this, the alternatives other schemes take, the coding scheme that spreads a message across tokens, and the steganalysis literature that complements provable security.

Steganography and covert channels. Censors need not break encryption to block it: deep packet inspection can flag a TLS handshake or tunnel without decrypting a single byte [15], so censorship-resistant communication increasingly turns to covert channels that hide messages inside traffic that looks benign [16]. These channels face their own challenge, since merely looking benign under casual inspection does not rule out being statistically distinguishable from genuinely benign traffic under closer scrutiny [17]. For example, in image steganography a message can be converted to a bitstream and each bit used to replace the least significant bit of a pixel's color value [18]. While imperceptible to a human eye, it is still statistically detectable, since altering the least significant bits distorts an image's natural statistics in ways statistical steganalysis can detect, motivating decades of research into more covert embedding methods [19]. This also motivates provable security [3], which we formalize in Section 6.

Provably secure linguistic steganography. An LM maps a context x (the prompt and the text so far) to a distribution $p(x)$ over a vocabulary of tokens V that includes an end-of-text outcome END; text is produced by sampling a token from $p(x)$, appending it to x, and repeating until END is sampled or a length cap is hit. Following Hopper et al. [3], a stegosystem is secure if no efficient warden who chooses the hidden messages can distinguish stegotexts from samples of the cover channel; for an LM, the channel is this sampling process. LM schemes have sender and receiver indepen-

dently recompute $p(x)$ and encode the message in exactly where the sampled token falls within it. Arithmetic coding embeds the message as a point in the token's cumulative interval [20], but reuses randomness across tokens and is not provably secure [5]. Three schemes reach this bar: Meteor masks the message with a keyed pseudorandom generator and uses the result as the sampling randomness [5], iMEC embeds through a minimum-entropy coupling between the message and $p(x)$ [6], and Discop partitions $p(x)$ into rotated "copies" and encodes the message in which copy is sampled from [7]. All four require the receiver to reconstruct the sender's $p(x)$ bit for bit to invert the embedding.

Why probabilities differ. In practice, this bit-exact requirement is difficult to meet. Matrix multiplications and softmax normalizations sum many floating-point terms, and the result depends on the order they are summed in; GPUs of different architectures, different kernels (e.g. attention implementations chosen by sequence length), and different batch sizes sum in different orders, giving different results for the same model and input [8–10]. This variation is largest for mid-range probabilities [21], which arise at exactly the high-entropy steps where a stegosystem embeds most of its message. Lower-precision deployments (bf16, fp16) and weight quantization (8-bit, 4-bit NF4 [22]) add larger, systematic differences on top of this. Generation with a key-value cache (one forward pass per token) and teacher-forced scoring (one forward pass over the whole text) compute the same function along different paths, and can disagree even on identical hardware running identical weights [10]. Section 4 measures how badly these break schemes relying on bit-exact probability distributions.

Robust steganography. Other schemes respond to probability mismatch by giving up on recomputing $p(x)$ exactly. One approach is to never let the receiver run the model at all, embedding the message in side information decoded without access to $p(x)$ [23–25]. This is robust by construction, but at a much lower payload rate than a scheme that samples from the model's own distribution. Another method hardens the exact schemes, like arithmetic coding, by running sender and receiver in float64, which removes most numerical mismatch between two instances of the same model, at substantial compute cost and without addressing precision, quantization, model changes, or detectability [26]. We compare STEALTH against both approaches in Section 7.

The exponential race. STEALTH tolerates the mismatch starting with a different sampler, one whose errors stay local rather than cascading. Let $E_i \sim \mathrm{Exp}(1)$ be drawn independently for each $i \in V$. Then $\arg\min_i E_i/p_i$ equals i with probability exactly p_i. Taking logarithms, the winner minimizes $\ln E_i - \ln p_i$, the Gumbel-max trick in another form [27, 28]. Unlike inverse-CDF sampling, where a probability difference at one token can shift the intervals used to sample every other token, a mismatch here changes at most the one token whose race it affects. Aaronson's LM watermark and its successors already use this sampler, but with a fixed key and no payload [13, 29].

Spinal codes. What remains is spreading a message across many tokens to absorb individual differences between sender and receiver. A spinal code [14] splits a message into k-bit chunks $c_1, \ldots, c_L$ and computes a *spine* $s_j = H(s_{j-1}, c_j)$ with a hash function H. Each spine value seeds the pseudorandom noise transmitted in place of the chunk itself. Because s_j depends on every chunk up to j, a decoder that guesses a chunk wrong will see a mismatch from that point on, which lets it test guesses in order and keep only the best B partial guesses (beam search). The code is rateless: the sender can keep producing symbols for as long as it takes the receiver to succeed.

Detecting steganography. A separate line of work tries to catch steganography empirically rather than prove its absence. Linguistic steganalyzers train word-level neural networks [30–32] or fine-tuned transformers [33] to tell stegotext from cover text by its statistics alone, without access to the embedding scheme. Cao et al.'s missing-tail test instead targets a specific artifact of fixed-precision decoding: interval-based schemes systematically under-sample a model's most improbable tokens [34]. A provable-security proof rules out detection by any efficient warden, but Section 7.5 checks STEALTH against these detectors directly.

3. Threat Model and Goals

As a running example, we will consider a comment section on a public news article as our covert channel. The article serves as context for the prompt, and the sender's model writes a comment on the article to pass information. If the message needs more text, the conversation continues with further comments, each prompted by the article and everything posted before it. The receiver reads the article and the comments off the page, in the order they were posted. The same structure fits any multi-turn channel where two parties exchange LM-generated text in public view, such as a chat, a forum thread, or messages passed between agents.

Before sending any message, the sender and receiver agree on a public *prompt policy*, describing how each prompt is built from the article and the texts posted so far, and on a payload length. They also share a uniformly random key K. The sender runs a model M on its own hardware and software, while the receiver runs a model $M' \approx M$ that may differ from M in precision, quantization, GPU or software stack, model size within the same family, or model family entirely, each with its own tokenizer (Section 5). The receiver sees only the posted strings, not the sender's probabilities, and does not know M's exact configuration.

Standing between them is a passive warden, who reads every posted text and knows everything about the scheme except the key: the model and its exact configuration, the prompt policy, and how to sample honest text from M itself. We allow the warden to adaptively choose the article used as context and the hidden messages being sent. The warden wins by correctly telling whether a given conversation carries a hidden message at all.

STEALTH aims for two properties against this warden. *Security* requires that the posted texts be computationally indistinguishable from honest samples of M, which we formalize in Section 6. *Robustness* requires that a receiver

TABLE 1. BIT-EXACT SCHEMES FAIL UNDER MISMATCH. PERCENT OF MESSAGES DECODED. *: RESULTS ACROSS BF16, FP16 AND FP32.

Receiver	Gemma	Llama	Qwen
Arithmetic coding, 7–9B models			
Same model, replay	100	100	100
Different GPU, replay	0	0	0
Same model, teacher forced*	0	0	0
int8, NF4, smaller model	0	0	0
Discop, 7–9B models			
Same model, replay	100	100	100
Different GPU, replay	0	0	0
Same model, teacher forced*	0.5–1.5	1.5	0.5–1
int8, NF4, smaller model	0	0	0
Arithmetic coding, w/ float64 senders, 1.5–3B models			
fp64, replay	100	100	100
Different GPU, fp64 replay	100	100	100
fp64, teacher forced	100	100	100
fp32	0	0	0.5
bf16*, int8, NF4	0	0	0

running $M' \neq M$ recover the correct payload from as little text as possible, and almost never output a wrong one.

4. Mismatch Breaks Bit-Exact Schemes

To demonstrate how ordinary mismatch breaks the bit-exact schemes of Section 2, we test both arithmetic coding [20] and Discop [7], and the float64 hardening proposed by Bauer et al. [26]. Each scheme generates comments on the same set of news articles from CNN/DailyMail [35] used throughout our evaluation (Section 7), with the same three LM senders in bf16 (Gemma-2-9B-it [36], Llama-3.1-8B-Instruct [37], and Qwen2.5-7B-Instruct [38]), so that every scheme faces the same mismatch as STEALTH. A receiver can decode either by *replaying* the sender's exact computation, which requires matching its weights, precision, and hardware exactly, or by *teacher forcing* the posted text in one pass on the receiver's own hardware. First, we keep sender and receiver on the same hardware (best-case scenario), and vary only the processing mode, precision, quantization, or model size, dropping to the smaller sibling in each family (Gemma-2-2b-it, Llama-3.2-3B-Instruct, Qwen2.5-1.5B-Instruct) [36, 38, 39]. Then, we keep configurations identical, but vary the hardware between sender and receiver. The results described below are summarized in Table 1.

Arithmetic coding. We implemented arithmetic-coding steganography [20] over the full vocabulary, with tokens sorted by probability and 32-bit interval precision. With the three 7–9B senders in bf16, replaying the sender's own generation path decodes every message for all three families, as expected. Teacher forcing the same cover text decodes none. No other receiver configuration does better: decoding in fp16 or fp32, quantizing to int8 or 4-bit NF4, or substituting a smaller model from the same family fails on every message. Decoding derailed after only 5–7 correct bits with the same model, and 1–4 with other configurations.

Discop. For Discop [7], replay again decodes all messages per family. Teacher forcing decodes $\sim$ 1%, and switching precision, quantization, or model size decodes none. At every step Discop builds a Huffman tree over the whole vocabulary, so a small probability change in even a rare token can change how the tree merges, which changes every later token's path through it.

TABLE 2. TOTAL VARIATION PER TOKEN ON SAME GPU W/ REPLAY VS. TEACHER-FORCING (TF) AND ON DIFFERENT (DIFF) GPUS W/ REPLAY.

Receiver	Same GPU (replay vs. TF)	Diff GPU (replay)
Gemma 9B	0.86%	0.84%
Gemma 2B	1.11%	1.11%
Llama 8B	0.95%	0.94%
Llama 3B	0.94%	0.91%
Qwen 7B	1.24%	1.23%
Qwen 1.5B	1.42%	1.38%

Hardware. Next, we pivot from best-case to a more practical scenario where the sender and receiver have different hardware. We generated each family's comments on an NVIDIA H100 and decoded them by exact replay on an NVIDIA A6000. While both methods decode all same-GPU messages this way, across the two architectures they decode none. Table 2 shows why: the total variation (TV) between the two architectures' token distributions runs 0.8–1.4% per token, on par with the TV between generating a text step-by-step and scoring it with teacher forcing on the same GPU. Not one of the positions in any message we measured was bit-identical across different architectures.

Hardening with float64. Bauer et al. propose running both sides in float64, reasoning that the added precision shrinks numerical differences below what the coder can detect, though they stop short of claiming this holds for every device combination [26]. We found a pitfall in replicating this with Hugging Face models: several normalization and attention operations (RMSNorm in Qwen2 and Llama, Gemma 2's eager-attention softmax) silently cast to float32 regardless of the requested dtype, so float32-sized differences survive and decoding still fails after 20–24 bits. After patching these casts, float64 senders decoded all messages by teacher forcing, and float64 text generated on an H100 decodes correctly on an A6000, for every family, including inside batches with other texts. On the hardware we tested, float64 hardening removes the mismatch it targets, though as Bauer et al. themselves note, nothing guarantees this holds on configurations we did not test.

What float64 does not fix. The hardening keeps only one, carefully matched configuration in sync; it is not robust to anything else. A receiver at fp32 decoded at most 0.5% of messages tested, and lower precision or quantization always failed. It is also expensive: a 7–9B model needs 61–74 GB in float64, more than a 48 GB GPU holds, so we evaluated this baseline on smaller 1.5–3B models instead. On the hardware we tested, float64 generation is 2.5–5.5$\times$ slower per token than our approach, and decoding a message takes 5.8–16.5 s against 55–71 ms for STEALTH on the same 1.5–3B models (Section 7.4). Most importantly to our threat model, a warden that holds both a bf16 and a float64 copy of the model can also tell float64 comments from bf16 ones,

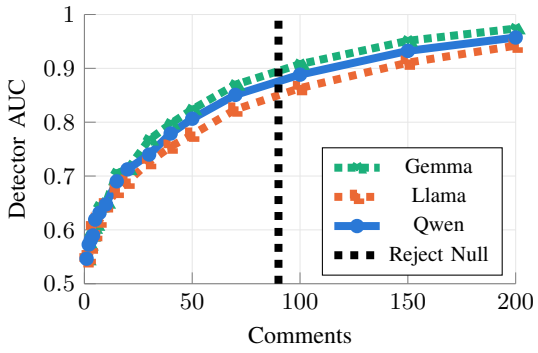


Figure 2. A warden can distinguish float64 comments from bf16 (AUC 0.5 is chance). The vertical line marks the median, across the three families, of the number of comments at which the null hypothesis that the sender is using bf16 becomes rejectable at the 5% significance level.

by comparing how likely each comment's tokens are under the two models and accumulating that evidence across many comments (Figure 2).

Across both coders, any mismatch collapses decoding, and the mitigation using float64 on both sides is slow, rarely deployed, and detectable given enough text. Our approach (Section 5) instead assumes sender and receiver will disagree, and asks only that the receiver recover the message despite that disagreement while the posted text stays indistinguishable from an honest sample.

5. STEALTH

We now describe STEALTH in full. Section 5.1 gives the exponential race the sender uses to sample each token, and why a mismatch at one position can flip only that position's token rather than cascading into every token that follows. Section 5.2 shows how a spinal code spreads the payload across many such tokens, and Section 5.3 gives the receiver's beam search over partial guesses. Section 5.4 covers how a text ends and how re-tokenization is handled, and Section 5.5 removes the sender's tokenizer from the noise entirely, letting a receiver on a different model family decode the same text. Figure 3 walks through each step on the running example from Figure 1.

5.1. Exponential Race

Each message gets its own key K_m, derived from the shared secret K and a fresh nonce ν by a standard keyed hash function (Section 6); ν need not be secret, since it only has to tell the receiver which posted texts belong to this message, not protect anything by itself. At each token position, the sender draws noise $E_i \sim \mathrm{Exp}(1)$ for every $i \in V$ from a pseudorandom function keyed by K_m and the token's position in the cover text the sender is writing, and outputs the token $y = \arg\min_{i \in V} E_i / p_i(h)$, the exponential race introduced in Section 2. Repeating the race on the receiver's own distribution $p'(h)$ with the same noise reproduces y whenever $p(h)$ and $p'(h)$ agree, and disagrees with probability at most $2D/(1+D)$, where D is their total variation distance [40]. Section 5.3 explains how STEALTH recovers the payload despite these disagreements. Section 6 formalizes that the race samples exactly from $p(h)$; the disagreement bound is proved by Daliri et al. [40].

A flip at one position never impacts the other tokens. The position t is just the token's byte offset in the already-written text, so both sender and receiver read t directly off the string. The noise at that position depends only on the message key, that offset, and the spine value its word carries (Section 5.2), never on which token was chosen there or at any earlier position, or on either party's probabilities. So, a mismatch that flips y_t leaves every other position's race untouched. The bit-exact schemes of Section 2 behave differently: their decoders carry state from one step to the next, such as an arithmetic coder's current interval, so a single wrong step corrupts every bit decoded after it. STEALTH's races are independent trials, each succeeding or failing on its own.

5.2. Spinal Code

The sender frames the n-bit payload m with a 16-bit cyclic redundancy check (CRC), which bounds how often a failed decode returns a wrong payload rather than reporting failure (Section 7), and splits the result into L chunks of $k = 4$ bits each. A *spine* [14] ties these chunks to the exponential race: a fixed s_0 starts the chain, and each later value $s_j = H(s_{j-1}, c_j)$ hashes the previous spine value together with chunk c_j, so s_j depends on every chunk up to j and changes completely if any of them does. Every token position is assigned one of the L spine values when its noise is derived, so the chunk a position carries evidence for is fixed before any text is written.

Positions are assigned to spine values by counting words rather than tokens or bytes, where a word begins at the first non-whitespace byte following a whitespace byte (or at the start of the text): a token carries spine value $s_{1+(w \bmod L)}$, where w is the number of words begun before the token's first byte. The token that begins the w-th word (0-indexed) therefore carries $s_{1+(w \bmod L)}$, and the rest of that word's tokens carry the next spine value; the values cycle back to s_1 after s_L. Counting words rather than tokens keeps every chunk supplied with roughly the same number of positions regardless of how the sender's tokenizer happens to split the text, and like the position itself, a word count does not depend on which tokenizer computes it, which matters once sender and receiver use different ones (Section 5.5).

The spinal code is rateless. Reaching s_L does not end the payload's encoding, it only starts another pass over the same L chunks, so a longer cover text keeps adding independent evidence for every chunk rather than running out of positions to carry them. A sender facing more mismatch, or wanting a safer margin before stopping, simply writes more text. The sender never needs to shorten, pad, or otherwise fit the payload to a fixed cover text length. This also lets the model end a text naturally, rather than cutting it off wherever the payload happens to run out, which is itself something

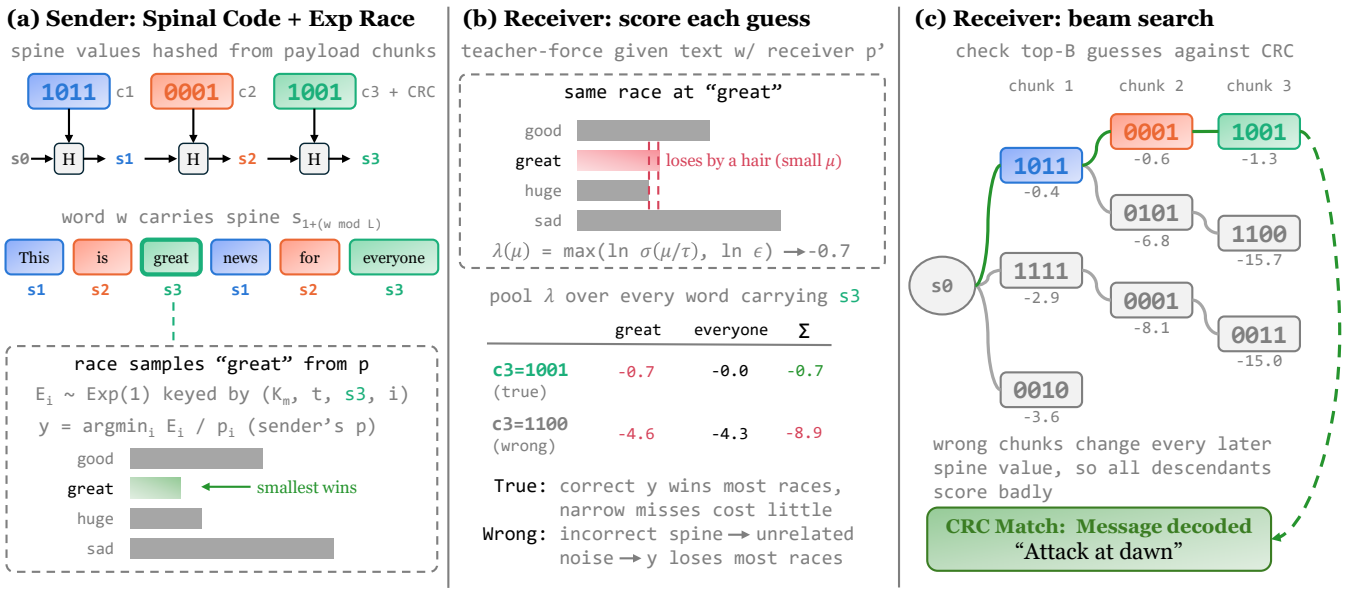


Figure 3. How STEALTH encodes and decodes the payload from Figure 1. (a) Payload chunks are hashed into a chain of spine values, each keying the noise for its words' tokens. (b) The receiver replays each race under its own p': a narrow loss costs little, but a wrong chunk guess changes the noise and loses most races. (c) Beam search keeps the B best partial guesses and outputs the best one that passes the CRC.

Algorithm 1 Sender: encode payload m into a cover text

Require: shared key K, nonce ν, payload m, prompt, model M

$K_m \leftarrow \mathrm{KeyedHash}(K, \nu)$
$c_1, \ldots, c_L \leftarrow$ chunks of $m \,\|\, \mathrm{CRC}(m)$
$s_0 \leftarrow 0$; $s_j \leftarrow H(s_{j-1}, c_j)$ for $j = 1, \ldots, L$
$t, w \leftarrow 0$; text $\leftarrow ()$
repeat
  $h \leftarrow$ prompt $\|$ text
  $p \leftarrow p_M(h)$
  $s \leftarrow s_{1+(w \bmod L)}$
  draw $E_i \sim \mathrm{Exp}(1)$ keyed by (K_m, t, s, i), $i \in V$
  $y \leftarrow \arg\min_{i \in V} E_i / p_i$
  append y to text
  $t \leftarrow t + \mathrm{len}_{\mathrm{bytes}}(y)$
  $w \leftarrow w + \mathrm{newwords}(y)$
until $y =$ END
post text (without END)

a warden would pick up on. The full STEALTH encoding procedure is given in Algorithm 1.

5.3. Decoding

To test a guess for chunk c_j, the receiver teacher forces every posted text once under its own model M', obtaining log-probabilities $\ell'(t)$ at each position, and recomputes the race's noise at every position assigned the guessed spine value. At such a position the observed token y_t wins the race by some margin μ_t, the gap between its race score and the best of the rest. A correct guess should see y_t win almost every one of these positions, and lose only the narrow ones that Section 5.1's disagreement bound predicts will flip; a wrong guess sees y_t win only with probability $p'_{y_t}(h)$ at each position, and while this can be high at a single low-entropy position, it is rarely high at every position a chunk is assigned, so a wrong guess mostly loses once its evidence is pooled across all of the chunk's positions.

The receiver turns each margin into a score $\lambda(\mu_t) = \max(\ln \mathrm{sigmoid}(\mu_t/\tau), \ln \epsilon)$, where the temperature τ sets how much a narrow loss is discounted rather than treated as a contradiction, and the floor ϵ caps how much even a maximally wrong guess can cost at any one position. A correct guess that narrowly lost a race is charged little, while a wrong guess pays close to the floor at every position that carries real evidence. Summing λ over the positions assigned to chunk j turns this into a score for how well a guessed value of that chunk explains the text. Neither τ nor ϵ has to be fixed in advance, since the receiver already holds the whole posted text. If decoding with one setting leaves no guess passing the CRC, it can simply decode again with a different τ or ϵ, for instance a larger τ if it suspects heavier mismatch than usual (Section 7).

The receiver searches for the whole payload with a beam of partial guesses, extending each by every possible value of the next chunk and keeping only the best-scoring extensions. Because the spine's hash chain makes every later spine value depend on every earlier chunk, a wrong chunk corrupts the scores of everything built on top of it, and the correct sequence of chunks overtakes these corrupted guesses within the beam. Once all L chunks have been guessed, the receiver outputs the highest-scoring surviving guess whose CRC passes, or reports failure if none does.

Algorithm 2 Receiver: decode a payload from posted text

Require: shared key K, nonce ν, payload length n, posted text, model M'

$K_m \leftarrow \mathrm{KeyedHash}(K, \nu)$
$L \leftarrow (n+16)/4$ ▷ CRC: 16 bits, Chunk width: 4 bits
tokenize the text; append END if it ended naturally
compute (t, w) for each token
$\ell' \leftarrow$ log-probabilities of M' over the prompt and text
beam $\leftarrow \{(0, s_0, ())\}$ ▷ score, spine value, chunks
for $j = 1, \ldots, L$ **do**
  $P \leftarrow$ positions with $w \bmod L = j - 1$
  cand $\leftarrow \emptyset$ ▷ candidates for this round
  for $(sc, s, cs) \in$ beam, $c \in \{0, \ldots, 2^k{-}1\}$ **do**
    $s' \leftarrow H(s, c)$ ▷ spine hash function
    $sc' \leftarrow sc + \sum_{t \in P} \lambda(\mu_t(K_m, t, s'))$
    add $(sc', s', cs \,\|\, c)$ to cand
  end for
  beam $\leftarrow$ the highest-scoring entries of cand
end for
return the highest-scoring cs in beam whose CRC checks, else FAIL

Section 5.2's CRC is what bounds how often this outputs a wrong payload instead of failing outright. The full decoding procedure is given in Algorithm 2.

5.4. Additional Details

Natural endings. The END token takes part in the exponential race alongside every other vocabulary entry, so a text ends exactly where an honest sample from M would end it, rather than wherever the payload happens to run out. While writing, the sender checks whether the text so far is already decodable by running the receiver's own algorithm on its own model. Once the text ends, the sender stops there if that check passes. Tokens written after the text first becomes decodable are not wasted, since the rateless code (Section 5.2) turns them into further passes over the same L chunks, giving a safety margin for receivers with more mismatch than the sender assumed.

Multiple texts. If a model ends the text before the check passes, the message continues in another text, built from a prompt that includes every text posted so far. Position t and word count w keep advancing across the whole stream rather than resetting at each text's start, so a chunk's positions can span several texts exactly as they would span one.

Re-tokenization. The receiver works from the posted text, not the sender's tokens directly, and re-tokenizing that text does not always reproduce the sender's token boundaries. The sender samples one token at a time, so it may write "␣hel" followed by "lo", which the receiver's tokenizer reads back as the single token "␣hello". Because position is a byte offset rather than a token index (Section 5.1), the damage stays local. The merged token sits at the first part's offset and is scored as an ordinary flip; the second part's offset is never visited, an *erasure* that contributes no evidence rather than wrong evidence. Every later token keeps the sender's offsets, so the rest of the stream decodes normally, and the rateless code's redundancy (Section 5.2) absorbs the flip and erasure like any other.

5.5. Decoding Across Model Families

We alter our encoding and decoding procedures slightly for scenarios where the sender and receiver are using different model families. Position t and word count w are defined on the posted text itself, so nothing about them ties the receiver to the sender's tokenizer, and Section 5.4 already covers a receiver that re-tokenizes the same text differently. What still does tie the two together is the noise's fourth input, the vocabulary index i in (K_m, t, s, i). Index i names a token in the sender's vocabulary, and a model of another family has its own, unrelated vocabulary, with its own index for the same string. A receiver on another model family has no way to know which of its indices corresponds to which of the sender's.

The fix is to key the noise by the token's own bytes instead of its index. Writing $\kappa(i)$ for a fixed hash of token i's byte string, the sender and receiver both draw noise from $(K_m, t, s, \kappa(i))$ rather than (K_m, t, s, i); only this one input to the keyed pseudorandom function changes, in both Algorithm 1 and Algorithm 2. Because κ depends only on a token's spelling, the sender's and receiver's vocabularies agree on $\kappa(i)$ for every string they share, even though they disagree on i itself. This requires κ to be injective on the sender's vocabulary, so that two of its tokens never draw the same noise. Some vocabularies contain several tokens with the same spelling, such as Gemma's byte-fallback tokens, which duplicate ordinary single-byte tokens. Among such a group, the ordinary token, the one any tokenizer produces for that string, is keyed by its bytes, and each of the others by a hash of its index under a separate domain tag, with END given its own reserved key.

The receiver tokenizes the posted text with its own tokenizer and its own vocabulary, and otherwise decodes exactly as in Section 5.3. Wherever the two tokenizers split the text the same way, $\kappa(i)$ lines the receiver's token up with the sender's and the race scores it normally; wherever they split it differently, the mismatch is just a re-tokenization, handled the same flip-and-erasure way as Section 5.4.

6. Security Analysis

A warden watching the conversation sees only the sender's posted texts, never whether or how a receiver decodes them. What has to be shown is that those texts are indistinguishable from an honest conversation with the sender's own model, down to where each text ends.

6.1. Security Definition

Write $\hat{p}(h)$ for the distribution the sender's model actually assigns a context h, computed from its own float32 log-probabilities rather than some idealized real-valued one.

An *honest text* is one sampled from $\hat{p}$, token by token, until END; an *honest conversation* chains such texts together under the public prompt policy, each one's prompt built from everything posted before it. STEALTH's stegotext is secure if no warden can tell it apart from an honest conversation, even when it gets to choose the payloads and context adaptively, the chosen-hiddentext notion of security introduced by Hopper et al. [3] and instantiated here for our setting.

Definition 1 (Security)**.** *A warden A queries an oracle with a payload m, a nonce ν, and a starting context h_0, and gets back either the sender's posted texts for m (the stego oracle O_S), or an honest conversation under the same stopping rule (O_H). A chooses every query adaptively, knowing the outcome of every earlier one, but never reuses a nonce. Its advantage is*

$$\mathrm{Adv}(A) = \left|\Pr[A^{O_S} = 1] - \Pr[A^{O_H} = 1]\right|.$$

STEALTH *is secure against A to the extent $\mathrm{Adv}(A)$ is small; Theorem 1 bounds it concretely in terms of the warden's number of queries q and total tokens T.*

For our experiments we use HMAC-SHA256 [41] for Section 5.1's keyed hash and the ChaCha20 block function [42] for its keyed pseudorandom function (PRF). We assume both are PRFs with advantage $\epsilon_{\mathrm{HMAC}}(q)$ against q queries and $\epsilon_{\mathrm{ChaCha}}(r)$ against r block evaluations.

6.2. Security Theorem

Lemma 1 (Exponential race [27, 28])**.** *For a distribution $p(h)$ and independent $E_i \sim \mathrm{Exp}(1)$ for every i with $p_i(h) > 0$, the race winner $W = \arg\min_i E_i/p_i(h)$ satisfies $\Pr[W = i] = p_i(h)$, and the winning score $\min_i E_i/p_i(h)$ is itself $\mathrm{Exp}(1)$, independent of W. Proof in Appendix A.*

This is the exact-sampling half of the claim from Section 5.1; its other half, the bound on how often a mismatched receiver's race disagrees, is proved by Daliri et al. [40].

Lemma 1 needs independent noise at every position, but the inputs STEALTH feeds its noise function depend on the sender's own earlier tokens. The next lemma shows that, with the keyed PRF replaced by a truly random function, this never matters: the noise read at each position is fresh given everything before it. Theorem 1 then returns to the real scheme, paying for the PRF assumption.

Lemma 2 (Freshness)**.** *Let $\mathcal{H}_t$ be everything the sender has computed before choosing the token at position t. Replacing Section 5.1's keyed pseudorandom function by a truly random function, the noise it reads to choose the token at position t is independent uniform, independent of $\mathcal{H}_t$ as well. Proof in Appendix A.*

Nothing in the proof needs the payload to be secret, so the warden may choose it. Because t is itself part of the noise's input, two positions sharing a spine value still draw independent noise from each other. The argument also survives Section 5.5's byte-keyed variant unchanged, since κ is injective on the sender's vocabulary and distinct tokens still read distinct inputs there too.

Corollary 1 (Ideal-world texts are honest)**.** *With the keyed PRF replaced by a truly random function and the race computed exactly on continuous uniforms, let the sender keep generating texts indefinitely under the public prompt policy, ignoring its own stopping rule. The resulting sequence of texts has the same law as an honest conversation. Proof in Appendix A.*

Theorem 1 (Security)**.** *If the number of texts per hidden message is fixed in advance, or set by any rule that does not use the key, and the honest oracle uses the same rule, then for q queries, T tokens in total, and at most r ChaCha20 evaluations per message,*

$$\mathrm{Adv}(A) \le \epsilon_{\mathrm{HMAC}}(q) + q\,\epsilon_{\mathrm{ChaCha}}(r) + T\delta,$$

where δ is the per-token disagreement probability between the implemented race and the ideal one (Section 6.3). Proof in Appendix A.

Every text ends where the sender's model would have ended an honest sample, so the only quantity in a message that can depend on the key at all is how many texts it took. Other distribution-preserving schemes end a text where the message ends rather than where the model would. Arithmetic coding, Meteor, and Discop stop once the message's bits are exhausted, and iMEC stops once the receiver's posterior over the message is confident enough [5–7, 20]; none of these coincide with where an honest sample would end. Each could pad with honest, message-independent text until END, but that padding carries nothing; in STEALTH, every token up to the natural ending also adds evidence toward the payload.

Theorem 1 covers a sender that posts a fixed number of texts per query, independent of the key, and simply fails the message if that was not enough. We will see in Section 7.2, that a fixed count of two would have failed none of the 600 messages we evaluated: 33 Gemma, 3 Llama, and 1 Qwen message needed a second comment, and none needed a third. A sender that instead stops as soon as the decodability check of Section 5.4 passes posts less text, but its number of texts N can depend on the key. By coupling it with a sender fixed at k texts, a warden gains at most $\Pr[N \neq k]$, which for $k = 1$ is the 6% of messages that needed a second comment.

6.3. Finite-Precision Error

Theorem 1's remaining term, δ, bounds how often Section 5.1's implemented race disagrees with the mathematically ideal one it is supposed to compute. Write $\mathrm{Race}(\ell, W)$ for the token the sender's implementation outputs from log-probabilities ℓ and raw 64-bit noise words W, with every score computed in float64, and $\mathrm{IdealRace}(\ell, U)$ for the same race run on exactly continuous uniforms U, each drawn inside its word's 2^{-64}-wide cell. The implementation departs from the ideal in two ways: rounding, since each computed score can be off by up to some η from its exact value, and

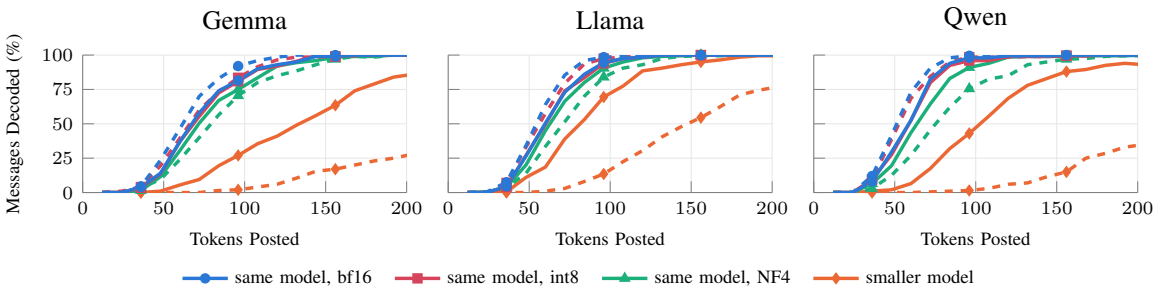


Figure 4. Messages decoded against tokens posted. Solid lines: $\tau = 1.0$; dashed: $\tau = 0.1$. Smaller models: Gemma 2B, Llama 3B, Qwen 1.5B. The axis stops at 200 tokens; Table 3 gives the full counts.

discretization, since each uniform is read from the midpoint of its 2^{-64}-wide cell rather than sampled continuously.

Lemma 3 (Floating-point race)**.** *Let $\omega = 2^{-64}$, W_i, $i \in V$, be independent uniform 64-bit words, Z_i independent uniforms on $[0, 1)$, and $U_i = \omega(W_i + Z_i)$, which are themselves independent uniforms on $[0, 1)$. Suppose the implementation reads noise $\hat{E}_i = -\ln(1-(W_i+\frac{1}{2})\omega)$ from the midpoint of W_i's cell and computes every log score within η of $\ln \hat{E}_i - \ell_i$, and let $\rho = e^{2\eta} - 1$. Then for every ℓ and every $a > 0$ with $\rho(a + \frac{1}{2}) \le \frac{1}{4}$,*

$$\Pr[\mathrm{Race}(\ell, W) \neq \mathrm{IdealRace}(\ell, U)] \le (6 + 2e)\, a\, |V|\omega + 8|V|\omega + e^{-a} + \tfrac{3}{2}\rho.$$

Proof in Appendix A.

Our kernel's rounding error is $\eta \approx 3 \times 10^{-14}$. With $|V| \le 2^{18}$, true of every vocabulary across all three model families, and $a = 30$, the bound gives $\delta \le 5.2 \times 10^{-12}$ per token, about 1.6×10^{-9} over a 300-token message. By Theorem 1, finite precision adds at most $T\delta$ to a warden's advantage, so even a warden that observes a million such messages gains less than 2×10^{-3} from it.

No missing tail. Cao et al. [34] detect arithmetic coding and Meteor from the probability mass their finite-precision intervals can never reach. The exponential race has no such gap: it never builds a CDF, since each token draws its own noise, and scores are compared in log space, so a token with probability 10^{-30} keeps its log-probability at full precision rather than being rounded to a zero-width interval. The race loses probability only where a token would need noise below the smallest representable value, 2^{-65}, to win. On Qwen2.5 this unreachable mass totals 1.4×10^{-17} per token, against 1.7×10^{-6} for 32-bit arithmetic coding on the same model. This mass is part of the disagreement δ bounded by Lemma 3, so Theorem 1 already accounts for it. We directly run their detection test in Section 7.5.

7. Evaluation

7.1. Setup

Models. Senders use the same 7–9B models from Section 4, each in bf16, run both plainly and with speculative decoding using the smallest model of the same family as the draft (Gemma 2B, Llama 3.2 1B, Qwen 0.5B). Receivers cover the sender's own weights at bf16, fp16, fp32, int8, and 4-bit NF4, the smaller models of the same family, and for Section 7.3, the large models of the other two families reading a byte-keyed sender. Because bf16, fp16, and fp32 gave nearly identical results throughout, the tables report bf16 only.

Hardware. Experiments are run on two NVIDIA architectures, Ampere (RTX A6000) and Hopper (H100). To isolate mismatch across hardware from mismatch across precision or model size, we also decode each architecture's saved texts on the other.

Channel. The sender's prompt is a request to comment on one of 200 CNN/DailyMail articles [35], with further comments posted if the first does not carry enough evidence to decode. Texts are sampled at temperature 1 for all models.

Code parameters. Messages are 32-bit payloads protected by a 16-bit CRC, spun with chunk width $k = 4$ (so $L = 12$ chunks) and beam width $B = 16$, and decoded with the decoder's score floor set to $\epsilon = 10^{-3}$. We report both of the decoder's operating points, $\tau = 0.1$ and $\tau = 1.0$, and a cascade that falls back to $\tau = 1.0$ only if $\tau = 0.1$ passes no guess through the CRC. The sender's own stopping check always uses $\tau = 0.1$; it runs every 12 tokens, and once it passes, the sender writes 12 more tokens as a margin before stopping at the end of the current comment. Our sweeps decode prefixes at the same 12-token granularity.

Metrics. *Success* is whether the decoded payload matches the sent one, over the 200 messages per condition. *Flip rate* is the share of tokens where the race winner under the receiver's distribution differs from the winner under the sender's, and *TV* is the mean total variation distance between their per-token distributions. Both measure how far the receiver's distribution is from the sender's, which explains why more mismatch needs more tokens. *Tokens to $N\%$* is the shortest prefix length at which that fraction of messages decode (Figure 4 plots the full curves).

7.2. Robustness Under Mismatch

With STEALTH, every receiver we tested decodes every message given enough text, across every precision, quantization level, and same-family model size; 200 of 200 bounds

TABLE 3. TOKENS POSTED TO REACH 90% / 99% / 100% OF MESSAGES DECODED, USING THE CASCADE DECODER.

Receiver	90%	99%	100%
Gemma 9B bf16	96	132	132
Gemma 9B int8	108	132	144
Gemma 9B NF4	120	156	192
Gemma 2B	228	372	384
Llama 8B bf16	84	108	132
Llama 8B int8	84	120	132
Llama 8B NF4	96	132	144
Llama 3B	132	192	252
Qwen 7B bf16	72	96	156
Qwen 7B int8	84	96	180
Qwen 7B NF4	96	132	204
Qwen 1.5B	180	300	300

each configuration's failure rate below 1.9% (95% Wilson interval). Figure 4 sweeps the decoder over prefixes of long streams written and decoded on the A6000, and Table 3 reports how much text each configuration needs. The worst case was with Gemma 2B, where no message needed more than 384 tokens.

What drives the cost. Within each family, the token cost in Table 3 tracks flip rate (Table 4) rather than precision or quantization: int8 receivers, whose flips stay under 5%, need little more than a same-configuration receiver, while NF4 and the smaller models, whose flips run from 8% to 37%, need more text, increasingly so as flips rise. Each flip costs evidence rather than adding it, so the spinal code's beam search needs a larger margin of correct tokens to recover.

Decoding at the sender's stopping point. A sender cannot see its receiver's flips; it knows only when its own model can decode. Table 4 evaluates the simplest stopping rule built on that knowledge: the sender stops once its same-model stopping check succeeds, plus 12 more tokens, and we measure how often each receiver decodes on the sender's GPU at that point. A receiver matching the sender's configuration decodes every message in every family, since it repeats the sender's own check. Receivers close to that configuration nearly always do too: with the cascade, fp16 and fp32 lose at most one message per family, and int8 at most four. Receivers farther away do not: NF4 decodes 93.5–99.5% of messages, and the smaller same-family models only 42–87.5%. The sender's check alone is therefore a reliable signal only for receivers near its own configuration; to reach one farther away, the sender can post additional passes beyond the one its check calls for or, if it knows the receiver's model, run that model for its check.

Neither operating point wins everywhere: $\tau = 1.0$ decodes up to five fewer messages than $\tau = 0.1$ on the same receiver, while $\tau = 0.1$ collapses on the smaller same-family receivers, decoding 8.5–54% of messages where $\tau = 1.0$ decodes 42–87.5%. The cascade matches or beats the better setting on every row, and beats both on Gemma and Llama NF4, as the two settings tend to fail on different messages.

Across GPU architectures. Table 3 and Figure 4 decode texts on the same A6000 that wrote them. To isolate the hardware, we took the H100's own sweep texts and decoded them on both GPUs, with the cascade decoder and every receiver configuration; Table 5 gives the A6000 side. Every configuration, including NF4 and the smaller same-family models, still reaches 100% of messages decoded. Moving the receiver from the H100 to the A6000 changes no configuration's token counts by more than two 12-token steps, in either direction: the 90% column is unchanged for 10 of 12 receivers, and 5 of 12 are unchanged in every column, so the hardware mismatch adds no systematic token cost on top of precision and quantization. The new GPU changes about 1% of race winners at bf16 and 4–6% at int8 relative to the same receiver on the H100, yet the flip rate against the sender stays within 0.3 points of Table 4's for every receiver, and fp32 picks the same winner at every position on both architectures. Bit-exact schemes have no such slack: across the same two GPUs, arithmetic coding and Discop decode none of their messages, even by exact replay (Section 4).

TABLE 4. PERCENT OF MESSAGES DECODED AT THE SENDER'S STOPPING POINT (ITS OWN CHECK PLUS 12 TOKENS), WITH FLIP RATE.

Family	Receiver	Flips (%)	$\tau = 0.1$	$\tau = 1.0$	Cascade
Gemma	9B bf16	0.9	100	97.5	100
	9B int8	3.8	98	96	98
	9B NF4	8.4	87.5	91	93.5
	2B bf16	31.5	8.5	42	42
Llama	8B bf16	1.0	100	99.5	100
	8B int8	4.9	100	99.5	100
	8B NF4	10.2	93.5	97.5	98
	3B bf16	23.5	54	87.5	87.5
Qwen	7B bf16	1.2	100	100	100
	7B int8	4.2	100	99.5	100
	7B NF4	12.1	96.5	99.5	99.5
	1.5B bf16	36.9	32.5	84	84

TABLE 5. TOKENS POSTED TO REACH 90% / 99% / 100% OF MESSAGES DECODED WITH THE CASCADE DECODER, READING H100-WRITTEN STEGOTEXT ON AN A6000.

Receiver	90%	99%	100%
Gemma 9B bf16	96	120	180
Gemma 9B int8	96	144	180
Gemma 9B NF4	120	156	180
Gemma 2B	228	300	324
Llama 8B bf16	84	96	132
Llama 8B int8	84	108	132
Llama 8B NF4	96	132	156
Llama 3B	144	180	180
Qwen 7B bf16	84	96	96
Qwen 7B int8	84	96	96
Qwen 7B NF4	96	108	168
Qwen 1.5B	168	252	264

Efficient samplers. Because STEALTH tolerates mismatch, the sender is free to generate however is most efficient on its own hardware, even along computation paths the receiver never reproduces. Speculative decoding [43, 44] is one example. Each round, a small draft model runs the same keyed race on its own distribution for a few tokens; the sender's model then scores all of them in one forward

TABLE 6. CROSS-FAMILY DECODING WITH BYTE-KEYED NOISE. ALIGNED: SHARE OF THE RECEIVER'S TOKENS MATCHING A SENDER TOKEN'S BYTE SPAN. FLIPS: SHARE OF ALIGNED TOKENS WHOSE RACE WINNER DIFFERS FROM THE POSTED TOKEN.

				At stopping point	Whole stream, $\tau = 1.0$
Sender	Receiver	Aligned (%)	Flips (%)	$\tau = 0.1$ / $\tau = 1.0$ / cascade (%)	Tokens to 90% / 99% / 100%
Gemma 9B	Gemma 9B bf16	99.9	1.0	99.5 / 95.5 / 100	108 / 156 / 204
	Gemma 9B NF4	99.9	8.4	88.5 / 91.5 / 94	120 / 168 / 240
	Llama 8B bf16	88.2	35.5	1 / 19 / 19	336 / 528 / 636
	Llama 8B NF4	88.2	36.7	0.5 / 15.5 / 15.5	348 / 516 / 648
	Qwen 7B bf16	88.5	37.4	0 / 14 / 14	396 / 612 / 756
	Qwen 7B NF4	88.5	37.7	0 / 12.5 / 12.5	384 / 552 / 756
Llama 8B	Llama 8B bf16	99.9	1.0	100 / 99.5 / 100	96 / 120 / 180
	Llama 8B NF4	99.9	10.7	93 / 99 / 99	96 / 144 / 180
	Gemma 9B bf16	88.9	35.6	7.5 / 60.5 / 60.5	216 / 288 / 372
	Gemma 9B NF4	88.9	35.9	6 / 58.5 / 58.5	228 / 276 / 324
	Qwen 7B bf16	99.5	36.6	13 / 71 / 71	180 / 300 / 324
	Qwen 7B NF4	99.5	36.5	12.5 / 71 / 71	192 / 300 / 384
Qwen 7B	Qwen 7B bf16	99.9	1.2	100 / 100 / 100	84 / 108 / 180
	Qwen 7B NF4	99.9	10.5	99.5 / 100 / 100	96 / 120 / 180
	Gemma 9B bf16	91.0	36.0	12.5 / 79.5 / 79.5	252 / 528 / 612
	Gemma 9B NF4	91.0	36.2	14.5 / 81 / 81	240 / 516 / 588
	Llama 8B bf16	97.5	34.2	27.5 / 90.5 / 90.5	180 / 432 / 468
	Llama 8B NF4	97.5	34.9	27.5 / 90 / 90	168 / 432 / 444

pass, runs its own race at each position with the same noise, and keeps the drafted tokens up to the first position where its race picks a different token, emitting its own winner there, as in drafter-invariant speculative decoding [40]. Every posted token is still the winner of the sender's race, and no acceptance test or residual sampling is needed. A bit-exact scheme could not do this unless the receiver reproduced every verification pass exactly, including which drafted tokens were accepted, since a batched pass computes the sender's probabilities along yet another path (Section 2). For STEALTH, it is just one more small mismatch. We repeated the runs of Table 4 with four drafted tokens per round, of which 26–42% were accepted, and receivers see no difference (Appendix B, Table 10): flip rates match the plain sender's within 0.5 points for every receiver, and every same-size receiver decodes within three messages of its plain-sender count, in either direction. Smaller same-family receivers vary more, by 3–17 messages, mostly in the speculative sender's favor but not consistently (Llama 3B gains five messages on the H100 and loses five on the A6000). One Gemma message passed the sender's own check yet was missed by the same-configuration bf16 receiver: no tokens flipped, but its score sat at the edge of the beam, and a wider beam does decode it successfully.

7.3. Cross-Family Decoding

We re-ran the sweep of Section 7.2 on the A6000 with each large sender's noise keyed by token bytes rather than vocabulary index (Section 5.5), so receivers of the other two model families can decode from the posted strings alone. Table 6 gives the results, with same-family bf16 and NF4 receivers kept as a control. The byte-keyed variant costs nothing within a family: its control rows decode within one percentage point of Table 4's at the sender's stopping point with the cascade, and within three points at either τ alone, as expected for an independent sample of 200 messages. They also need within two 12-token steps as much text as Table 3's, even though Table 6's token counts use $\tau = 1.0$.

Across families, the receiver's tokenizer splits the text differently from the sender's: 88–99.5% of its tokens line up with a sender token at the same byte offset and length, lowest whenever Gemma's tokenizer is on either side. Of the aligned tokens, 34–38% flip, comparable to the smaller same-family receivers of Table 4 (24–37%), and quantizing the receiver to NF4 adds almost nothing on top, since the difference between families already dominates. Given the whole stream, every cross-family receiver decodes every message, 90% of them within 168–252 tokens for Llama and Qwen senders and 336–396 for Gemma's.

The sender's same-model stopping check is not calibrated for these receivers: at the sender's stopping point they decode only 12.5–90.5% of messages, almost entirely through $\tau = 1.0$, since $\tau = 0.1$ decodes at most 27.5%. A sender that knows its receiver's model can instead run that model for its stopping check. Replaying the sweeps under that rule, stopping at the end of the first comment after the receiver's $\tau = 0.1$ check has decoded plus 12 more tokens, the sender posts more text, and 11–88% of messages need a second comment.

Across roughly 450,000 cross-family sweep decodes, a wrong payload was output 35 times. Each decode checks at most $B = 16$ final guesses against a 16-bit CRC, which bounds false accepts at $16 \cdot 2^{-16}$ per decode, about 110 over these decodes; the 35 observed sit well within that bound.

7.4. Comparison With Other Methods

Section 4 showed what mismatch does to bit-exact schemes: arithmetic coding and Discop decode every mes-

TABLE 7. PAYLOAD BITS PER POSTED TOKEN. ROBUST: DECODES DESPITE SENDER–RECEIVER MISMATCH. EXACT: STEGOTEXT FOLLOWS THE SENDER'S MODEL DISTRIBUTION. –: IMPRACTICAL AT 256 BITS. †RATE REPORTED BY THE AUTHORS.

Method	Robust	Exact	32 bits	256 bits
Arithmetic coding [20]	No	Yes	0.19–0.28	0.95–0.99
Discop [7]	No	Yes	0.19–0.29	0.81–0.92
Zamir [23]	Yes	Yes	0.002–0.024	–
Bai et al. [24]†	Yes	Yes	0.002–0.044	–
STEALTH (ours)	**Yes**	**Yes**	**0.19–0.26**	**0.37–0.43**

sage when the receiver replays the sender's exact computation (Table 1), and almost none under any mismatch, including a smaller model of the same family. Table 7 sets their rates against ours for context. Under replay, arithmetic coding posts 0.19–0.28 bits per token at 32-bit payloads and 0.95–0.99 at 256 bits, against our 0.19–0.26 and 0.37–0.43, and Discop is similar. Those rates hold only for a receiver that reconstructs the sender's exact distribution, which Section 4 found does not happen in practice.

Among receiver-free schemes, we reimplemented Zamir's [23], following its reference code, on the same news channel with a budget of 32 comments, tuning its parameters on held-out messages. It delivers 0.017, 0.024, and 0.002 payload bits per posted token on Gemma, Llama, and Qwen respectively, and decodes 92%, 99.5%, and 60% of messages from its own tokens within that budget. Being receiver-free does not protect it from re-tokenization: decoding from the posted strings instead of the sender's tokens drops Llama from 99.5% to 78.5%, since one changed token corrupts its binary choices and every symbol's score after it. We deliver 9–15× more bits per posted token on Gemma and Llama, and nearly 100× more on Qwen, whose low-entropy text starves Zamir's per-choice evidence. We did not reimplement Bai et al. [24]. Table 7 instead reports their own measured rate as a rough point of comparison.

Rate vs. payload size. Our rate rises with payload size because a comment has to end where the model ends it: a short message is often decodable well before the comment it rides in is over, so most of that comment's tokens count against the rate for nothing, while a longer message uses more of the same comment before it ends. Figure 5 traces this for the same-model bf16 receiver from 32 to 256 bits. The rate rises by 20–90% from 32 to 64 bits, then flattens into the 0.35–0.45 range by 128 bits, as the unused comment tail shrinks relative to a growing message instead of staying a fixed cost. It does not keep climbing toward the replay-mode rates of arithmetic coding or Discop, which spend nothing on a receiver that reproduces the sender's distribution bit for bit, while ours turns only about 40% of the text's entropy into payload, the rest being the redundancy the beam search needs to absorb flips.

Computational cost. Hiding a message adds little time on either side. We timed 50 messages per sender on an A6000, one job per GPU at batch size 1. The exponential race runs within 3–4% of the speed of honest multinomial sampling, at 25–52 ms per token for the 7–9B models,

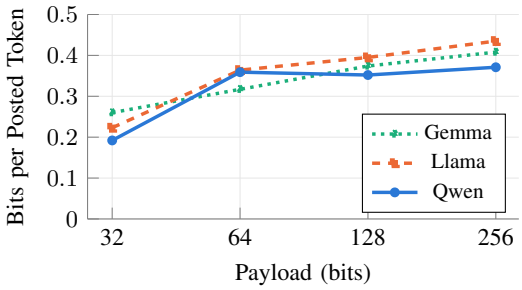


Figure 5. Payload bits per posted token rises with payload size and then flattens, as the unused tail of the last comment shrinks relative to the whole message (median over 200 messages).

and the sender's decodability checks add another 6–7%, for 4.2–7.0 s per message. Our receiver scores the posted text in one teacher-forced pass (44–69 ms) and runs the beam search (24–27 ms), for 70–95 ms per message on the 7–9B models. Arithmetic coding sends at the same speed but pays at the receiver: in bf16 it decodes only by regenerating the sender's text token by token, at 3.7–5.5 s per message, 50–60× ours. The float64 hardening of Section 4 costs more still: 2.5–5.5× our time per token on the sender, and 5.8–16.5 s per message on the receiver against our 55–71 ms on the same 1.5–3B models. Zamir's scheme also generates at the model's speed, but its low rate means a 32-bit message takes a median of 1,200–7,200 tokens, or 33–196 s to send.

7.5. Detection Tests

The proof of Section 6 covers every efficient warden; here we check that real detectors agree. Each test compares the first comment posted on each article, 2,000 per family and kind: *stego*, our scheme's output; *race*, an honest sample from the same exponential race run on fresh rather than keyed noise; and *mult*, an honest sample from ordinary multinomial sampling, Hugging Face's default. Stego-vs-race and stego-vs-mult are the tests that matter. Race-vs-mult compares two honest samplers and shows what chance looks like. As a positive control, *ac12* is arithmetic coding run deliberately at a coarse 12-bit interval precision, a scheme with known statistical artifacts.

Trained steganalyzers. We re-implemented three word-level linguistic steganalyzers from their published descriptions, with embeddings trained from scratch: FCN [30], R-BiLSTM-C [31], and a densely connected LSTM with feature pyramid [32]. Alongside them we fine-tune RoBERTa [33] end to end as a transformer-based baseline. Each is trained and evaluated with 5-fold cross-validation grouped by article, so no article's texts appear in both training and test, and we report out-of-fold AUC with a 95% bootstrap interval. Table 8 gives RoBERTa, the strongest of the four on the positive control. On every stego-vs-race, stego-vs-mult, and race-vs-mult pair, RoBERTa's AUC is within 0.007 of chance and the other three are within 0.015, none significant ($|z| \leq 1.3$). The control shows how little power these text-only classifiers have on distribution-

TABLE 8. RoBERTa steganalyzer AUC (0.5 = chance). *: $p < 0.05$ against chance.

Pair	Gemma	Llama	Qwen
stego vs. mult	0.503	0.498	0.507
stego vs. race	0.501	0.500	0.499
race vs. mult (control)	0.499	0.500	0.496
ac12 vs. mult (positive control)	0.515	0.585*	0.570*

preserving schemes: on only 200 texts per class, RoBERTa detects ac12 on Llama and Qwen ($z = 2.93, 2.43$) but not on Gemma, the densely connected LSTM detects it on Llama alone ($z = 2.40$), and FCN and R-BiLSTM-C never do.

A detector that knows the model. Our warden knows the sender's model (Section 3), so the stronger test scores each comment with that model. For every token it computes the token's log-probability, rank, the distribution's entropy, and a randomized probability integral transform (PIT), which is uniform on $[0, 1]$ under honest sampling. Each comment becomes a vector of histograms and moments of these, tail counts, PIT autocorrelation at lags 1–12 (the spine value repeats every 12 words), and length. Logistic regression, gradient boosting, and an MLP are trained on these features with the same grouped cross-validation. Across the three families and classifiers, stego-vs-mult and stego-vs-race AUCs lie in 0.48–0.52, inside the 0.48–0.54 spread of the honest race-vs-mult pair. The same detector finds ac12 in every family, at AUC up to 0.67 ($z = 6.0$).

The missing-tail test. Cao et al.'s test [34] targets schemes that can never emit some tokens, the failure Section 6.3's unreachable-mass bound rules out for STEALTH. It counts how many tokens land in the extreme tail of the sender model's distribution, below a threshold probability, and compares that count with the number honest sampling predicts, pooled into a z-score over a corpus. Positive z is a deficit of tail tokens, the signature Cao et al. test for; negative z is an excess, which we also count as detection. Table 9 pools all three families, about 880,000 tokens per honest or stego kind, at three thresholds. Stego and race stay within $|z| < 1.7$ at every threshold. The honest mult sampler shows a real deficit at 2^{-16} and 2^{-24} ($z = 3.7$, 6.4): `torch.multinomial` draws through a float32 cumulative sum, which under-samples the most extreme tail. The test is correctly flagging a known flaw in that particular honest sampler. Our race reads probabilities directly and has no such flaw. Arithmetic coding at 32-bit precision, the exact baseline of Section 4, is borderline at 2^{-16} and 2^{-24} ($z \approx -2$). At 16 bits it is caught at 2^{-32} ($z = -18.5$), and at 12 bits at every threshold, down to $z = -385$.

What these tests show. A score near chance means only that a test with this much power found nothing, so the controls matter as much as the stego rows. Both model-aware tests catch ac12 in every family, and the missing-tail test also catches 16-bit arithmetic coding and the float32 flaw in an honest sampler. That test's power falls with its threshold. At 2^{-24} honest sampling expects about 74 tail tokens across the corpus, so a sampler that never reached that tail would score $z \approx 8.6$; at 2^{-32} it expects fewer than two, so even a fully missing tail would score only $z = 1.3$, as mult does. STEALTH's unreachable mass, 1.4×10^{-17} per token (Section 6.3), lies far below what any such test can resolve. No finite set of tests rules out every warden; that is the job of the reduction in Section 6. What these results add is that detectors able to find small artifacts in other samplers find none in ours.

TABLE 9. Cao et al.'s test pooled over the three families, by tail threshold. Positive: fewer tail tokens than honest sampling predicts; negative: more.

Kind	2^{-16}	2^{-24}	2^{-32}
stego	0.3	1.6	-0.2
race (control)	-1.1	1.1	-1.6
mult (control)	3.7	6.4	1.3
ac, 32-bit	-1.9	-2.1	0.4
ac, 16-bit	-1.0	-1.7	-18.5
ac, 12-bit	-8.3	-56.4	-385.2

8. Conclusion

Provably secure linguistic steganography has assumed that the receiver reproduces the sender's token distribution bit for bit. Section 4 shows that ordinary deployments break this assumption: a different precision, quantization level, model size, or GPU leaves arithmetic coding and Discop decoding almost no messages. STEALTH drops the assumption. Its sender picks each token by a keyed exponential race, so a receiver whose distribution differs slightly still picks the same token almost everywhere, and a disagreement costs only that one token's evidence. Because the spinal code is rateless, every further token adds evidence for the whole payload, so a receiver that disagrees more often simply needs more text.

Across three model families, this held for every receiver we tested. Given enough text, every same-family receiver, quantized or smaller and on either GPU, decodes every message, and with the noise keyed by token bytes, so does every receiver of another family. The sender is also free to generate with speculative decoding, with no measurable effect on its receivers. Robustness costs no security: Section 6 proves the posted texts indistinguishable from honest samples of the sender's model, and no detector we tried, including one that scores text with that model, tells them apart (Section 7.5). Nor does it cost practicality: STEALTH delivers 0.19–0.26 payload bits per posted token on 32-bit messages, an order of magnitude more than the receiver-free scheme of Zamir [23], and its race runs within a few percent of honest sampling.

Sender and receiver therefore no longer need matching deployments to communicate covertly. They can run different hardware, different precisions, or different model families, and the texts they exchange remain provably indistinguishable from honest model output.

References

[1] S. Farrell and H. Tschofenig, "Pervasive monitoring is an attack," RFC 7258 (BCP 188), IETF, 2014.

[2] M. Wu, J. Sippe, D. Sivakumar, J. Burg, P. Anderson, X. Wang, K. Bock, A. Houmansadr, D. Levin, and E. Wustrow, "How the great firewall of China detects and blocks fully encrypted traffic," in *32nd USENIX Security Symposium*, 2023, pp. 2653–2670.

[3] N. J. Hopper, J. Langford, and L. von Ahn, "Provably secure steganography," in *Advances in Cryptology – CRYPTO 2002*, ser. LNCS, vol. 2442, 2002, pp. 77–92.

[4] A. D. Ker, P. Bas, R. Böhme, R. Cogranne, S. Craver, T. Filler, J. Fridrich, and T. Pevný, "Moving steganography and steganalysis from the laboratory into the real world," in *ACM Workshop on Information Hiding and Multimedia Security (IH&MMSec)*, 2013, pp. 45–58.

[5] G. Kaptchuk, T. M. Jois, M. Green, and A. D. Rubin, "Meteor: Cryptographically secure steganography for realistic distributions," in *ACM Conference on Computer and Communications Security (CCS)*, 2021, iACR ePrint 2021/686.

[6] C. S. de Witt, S. Sokota, J. Z. Kolter, J. Foerster, and M. Strohmeier, "Perfectly secure steganography using minimum entropy coupling," in *International Conference on Learning Representations (ICLR)*, 2023, arXiv:2210.14889.

[7] J. Ding, K. Chen, Y. Wang, N. Zhao, W. Zhang, and N. Yu, "Discop: Provably secure steganography in practice based on "distribution copies"," in *IEEE Symposium on Security and Privacy (S&P)*, 2023.

[8] B. Atil, S. Aykent, A. Chittams, L. Fu, R. J. Passonneau, E. Radcliffe, G. R. Rajagopal, A. Sloan, T. Tudrej, F. Ture, Z. Wu, L. Xu, and B. Baldwin, "Non-determinism of "deterministic" LLM settings," 2024, arXiv:2408.04667.

[9] J. Yuan, H. Li, X. Ding, W. Xie, Y.-J. Li, W. Zhao, K. Wan, J. Shi, X. Hu, and Z. Liu, "Give me FP32 or give me death? challenges and solutions for reproducible reasoning," 2025, arXiv:2506.09501.

[10] Thinking Machines Lab, "Defeating nondeterminism in LLM inference," https://thinkingmachines.ai/blog/defeating-nondeterminism-in-llm-inference/, 2025.

[11] R. Yan, C. Chu, Z. Yang, and Y. Murawaki, "A comprehensive survey on linguistic steganography: Methods, countermeasures, evaluation, and challenges," in *Proceedings of EMNLP*, 2026.

[12] M. Christ, S. Gunn, and O. Zamir, "Undetectable watermarks for language models," in *37th Conference on Learning Theory (COLT)*, ser. PMLR, vol. 247, 2024, pp. 1125–1139, arXiv:2306.09194.

[13] R. Kuditipudi, J. Thickstun, T. Hashimoto, and P. Liang, "Robust distortion-free watermarks for language models," *Transactions on Machine Learning Research*, 2024, arXiv:2307.15593.

[14] J. Perry, P. A. Iannucci, K. E. Fleming, H. Balakrishnan, and D. Shah, "Spinal codes," in *ACM SIGCOMM*, 2012.

[15] D. Xue, M. Kallitsis, A. Houmansadr, and R. Ensafi, "Fingerprinting obfuscated proxy traffic with encapsulated TLS handshakes," in *33rd USENIX Security Symposium*, 2024, pp. 2689–2706.

[16] L. Dixon, T. Ristenpart, and T. Shrimpton, "Network traffic obfuscation and automated internet censorship," *IEEE Security & Privacy*, vol. 14, no. 6, pp. 43–53, 2016.

[17] C. Cachin, "An information-theoretic model for steganography," in *2nd Workshop on Information Hiding (IH)*, ser. LNCS, vol. 1525, 1998, pp. 306–318.

[18] N. F. Johnson and S. Jajodia, "Exploring steganography: Seeing the unseen," *IEEE Computer*, vol. 31, no. 2, pp. 26–34, 1998.

[19] J. Fridrich, M. Goljan, and R. Du, "Reliable detection of LSB steganography in color and grayscale images," in *Proceedings of the ACM Workshop on Multimedia and Security (MM&Sec)*, Ottawa, Canada, 2001, p. 27.

[20] Z. M. Ziegler, Y. Deng, and A. M. Rush, "Neural linguistic steganography," in *Proceedings of EMNLP-IJCNLP*, 2019.

[21] T. Fu, G. Martínez, J. Conde, C. Arriaga, X. Qi, S. Liu, and P. Reviriego, "Beyond reproducibility: Token probabilities expose large language model nondeterminism," *IEEE Transactions on Computers*, 2026.

[22] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "QLoRA: Efficient finetuning of quantized LLMs," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2023, arXiv:2305.14314.

[23] O. Zamir, "Excuse me, sir? your language model is leaking (information)," *Transactions on Machine Learning Research (TMLR)*, 2024.

[24] M. Bai, J. Yang, K. Pang, X. Xu, Z. Yang, and Y. Huang, "Provably robust and secure steganography in asymmetric resource scenario," in *IEEE Symposium on Security and Privacy (S&P)*, 2025, pp. 1438–1456.

[25] S. Guo, S. Vithana, A. Gilani, L. Sankar, O. Kosut, and F. P. Calmon, "Feedback coding enables inference-time covert agentic communication," 2026, arXiv:2609.24994.

[26] L. A. Bauer, J. K. H. IV, S. A. Markelon, V. Bindschaedler, and T. Shrimpton, "Leveraging generative models for covert messaging: Challenges and tradeoffs for "dead-drop" deployments," in *ACM Conference on Data and Application Security and Privacy (CODASPY)*, 2024, pp. 67–78, arXiv:2110.07009.

[27] J. I. Yellott, Jr., "The relationship between luce's choice axiom, thurstone's theory of comparative judgment, and the double exponential distribution," *Journal of Mathematical Psychology*, vol. 15, no. 2, pp. 109–144, 1977.

[28] C. J. Maddison, D. Tarlow, and T. Minka, "A* sampling," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2014.

[29] S. Aaronson, "My AI safety lecture for UT effective altruism," https://scottaaronson.blog/?p=6823, 2022.

[30] Z. Yang, Y. Huang, and Y.-J. Zhang, "A fast and

efficient text steganalysis method," *IEEE Signal Processing Letters*, vol. 26, no. 4, pp. 627–631, 2019.
[31] Y. Niu, J. Wen, P. Zhong, and Y. Xue, "A hybrid R-BiLSTM-C neural network based text steganalysis," *IEEE Signal Processing Letters*, vol. 26, no. 12, pp. 1907–1911, 2019.
[32] H. Yang, Y. Bao, Z. Yang, S. Liu, Y. Huang, and S. Jiao, "Linguistic steganalysis via densely connected LSTM with feature pyramid," in *Proceedings of the ACM Workshop on Information Hiding and Multimedia Security (IH&MMSec)*, 2020, pp. 5–10.
[33] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "RoBERTa: A robustly optimized BERT pretraining approach," arXiv:1907.11692, 2019.
[34] W. Cao, Y. Wang, and D. Hu, "Breaking the "provable security": Detecting finite-precision artifacts in LLM-based steganography via low-probability vanishing," in *Findings of the Association for Computational Linguistics (ACL)*, 2026, pp. 20 262–20 274.
[35] A. See, P. J. Liu, and C. D. Manning, "Get to the point: Summarization with pointer-generator networks," in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2017, pp. 1073–1083.
[36] Gemma Team, "Gemma 2: Improving open language models at a practical size," 2024, arXiv:2408.00118.
[37] Llama Team, AI @ Meta, "The llama 3 herd of models," 2024, arXiv:2407.21783.
[38] Qwen Team, "Qwen2.5 technical report," 2024, arXiv:2412.15115.
[39] Meta, "Llama 3.2 model card," https://github.com/meta-llama/llama-models/blob/main/models/llama3_2/MODEL_CARD.md, 2024, release date: October 24, 2024.
[40] M. Daliri, C. Musco, and A. T. Suresh, "Coupling without communication and drafter-invariant speculative decoding," in *Proceedings of the IEEE International Symposium on Information Theory (ISIT)*, 2025, pp. 1–6.
[41] M. Bellare, "New proofs for NMAC and HMAC: Security without collision-resistance," in *Advances in Cryptology – CRYPTO 2006*, ser. LNCS, vol. 4117, 2006, iACR ePrint 2006/043.
[42] D. J. Bernstein, "ChaCha, a variant of Salsa20," https://cr.yp.to/chacha/chacha-20080128.pdf, 2008, document ID 4027b5256e17b9796842e6d0f68b0b5e.
[43] Y. Leviathan, M. Kalman, and Y. Matias, "Fast inference from transformers via speculative decoding," in *International Conference on Machine Learning (ICML)*, 2023, pp. 19 274–19 286.
[44] C. Chen, S. Borgeaud, G. Irving, J.-B. Lespiau, L. Sifre, and J. Jumper, "Accelerating large language model decoding with speculative sampling," 2023, arXiv:2302.01318.

Appendix A. Proofs

Proof of Lemma 1. Scaling an $\mathrm{Exp}(1)$ variable by $1/p_i(h)$ gives $E_i/p_i(h) \sim \mathrm{Exp}(p_i(h))$, so the race scores are independent exponentials with rates $p_i(h)$ summing to 1. For independent exponentials, the index achieving the minimum wins with probability its own rate over the sum of all rates, here $p_i(h)$, and the minimum itself is exponential with rate equal to that sum, here $\mathrm{Exp}(1)$, independent of the index. □

Proof of Lemma 2. A truly random function's values at distinct inputs are independent and uniform; querying it at an input not yet read returns a value independent of every value read so far (lazy sampling). Positions strictly increase, so every noise value read before position t, whether to sample a token or to run a decodability check, has a first component $t' < t$: the decodability check only replays decoding over tokens already written, so it never queries position t or later. No input at t has therefore been queried before, so by lazy sampling its value is a fresh independent uniform, regardless of which spine value selects it. □

Proof of Corollary 1. By Lemmas 1 and 2, each token, given everything chosen before it, is distributed exactly as $\hat{p}(h)$ for the current context h; END and the prompt policy are those of the honest process, so the token sequence follows the honest transition law at every step. □

Proof of Theorem 1. We bound $\mathrm{Adv}(A)$ through a sequence of hybrids, starting at the real stego oracle and ending at the honest oracle.

H_0 is the real stego oracle O_S.

H_1 replaces $\mathrm{HMAC\text{-}SHA256}(K, \cdot)$ by a random function; since the warden never repeats a nonce (Definition 1), the message keys K_m it induces are independent and uniform. A warden distinguishing H_0 from H_1 yields a distinguisher for HMAC-SHA256: it runs the warden, simulates the sender, and obtains each K_m by querying its own oracle on the nonce ν, so $|\Pr_{H_0} - \Pr_{H_1}| \le \epsilon_{\mathrm{HMAC}}(q)$.

H_2 replaces the ChaCha20 block function under each of the q message keys by an independent random function; a standard hybrid over the q messages gives $|\Pr_{H_1} - \Pr_{H_2}| \le q\,\epsilon_{\mathrm{ChaCha}}(r)$.

H_3 replaces the sender's implemented race by the ideal race on exact uniforms. H_2 and H_3 read the same noise words and draw the same warden coins, so they agree until the first position where the two races disagree. By Lemma 2 the noise at that position is fresh given everything before it, so by Lemma 3 the races disagree there with probability at most δ; a union bound over at most T positions gives $|\Pr_{H_2} - \Pr_{H_3}| \le T\delta$. The sender's decodability checks (Section 5.4) add no term: they query only positions already sampled, and under the theorem's stopping rule their outcome does not affect what is posted.

Finally, by Corollary 1, each message in H_3 is an honest conversation cut by a rule that does not use the key; the honest oracle O_H applies the same rule to an honest

conversation, and distinct messages use independent random functions throughout H_1–H_3, so H_3 and O_H have exactly the same law. Chaining the four bounds gives the claim. □

Proof of Lemma 3. The strategy is to condition on a "good" event Γ in which the winning margin is not too close to a tie and no noise word landed in an unusually wide cell; we bound the probability the good event fails to hold, and separately bound the implementation's disagreement probability given that it does.

Write $G_i = 1 - U_i$, $X_i = E_i/\hat{p}_i$ with $E_i = -\ln G_i$, $i^\star = \arg\min_i X_i$ and $x = X_{i^\star}$. By Lemma 1, $\Pr[i^\star = i] = \hat{p}_i$, $x \sim \mathrm{Exp}(1)$ is independent of $i^\star$, and given $(i^\star, x)$ the values $u_j = \hat{p}_j(X_j - x)$, $j \neq i^\star$, are independent $\mathrm{Exp}(1)$. Let I_i be the interval of values of $-\ln(1-u)/\hat{p}_i$ over u in W_i's cell $[W_i\omega, (W_i+1)\omega)$, and Δ_i its length. Write $\hat{S}_i$ for the implementation's computed log score, within η of $\ln \hat{E}_i - \ell_i$ by assumption. Both X_i and $\hat{X}_i = \hat{E}_i/\hat{p}_i$ lie in I_i, and the computed $\tilde{X}_i = \exp(\hat{S}_i + c)$, with c the normalizing constant of ℓ, lies within a factor $e^{\pm\eta}$ of $\hat{X}_i$.

Cell widths. If $G_i \geq 2\omega$, the cell is $[g, g+\omega)$ in G with $g \geq \omega$, so $-\ln$ varies over it by $\ln(1+\omega/g) \leq \omega/g \leq 2\omega/G_i = 2\omega e^{E_i}$, and $\Delta_i \leq 2\omega e^{E_i}/\hat{p}_i$.

Disagreement needs a close competitor. If the implemented race picks $v \neq i^\star$,

$$X_v - \Delta_v \leq \hat{X}_v \leq e^{\eta}\tilde{X}_v \leq e^{\eta}\tilde{X}_{i^\star} \leq e^{2\eta}\hat{X}_{i^\star} \leq e^{2\eta}(x + \Delta_{i^\star}),$$

so $X_v - x \leq \Delta_v + \Delta_{i^\star} + \rho(x + \Delta_{i^\star})$.

Good event. Let Γ be the event that $G_i \geq 8\omega$ for all i, $x \leq a$, and $\Delta_{i^\star} \leq \frac{1}{2}$. A union bound gives $\Pr[G_i < 8\omega \text{ for some } i] \leq 8|V|\omega$, and $\Pr[x > a] = e^{-a}$. Since $E_{i^\star} = \hat{p}_{i^\star}x$, if $G_{i^\star} \geq 2\omega$ then $\Delta_{i^\star} \leq 2\omega e^{\hat{p}_{i^\star}x}/\hat{p}_{i^\star} \leq 2\omega e^{x}/\hat{p}_{i^\star}$, so $\Delta_{i^\star} > \frac{1}{2}$ requires $\hat{p}_{i^\star} < 4\omega e^{x}$. Since $i^\star$ is independent of x and $\Pr[\hat{p}_{i^\star} < \theta] = \sum_{i:\hat{p}_i<\theta} \hat{p}_i \leq |V|\theta$, this has probability at most $4|V|\omega\,\mathbb{E}[e^x; x \leq a] = 4a|V|\omega$. So $\Pr[\bar{\Gamma}] \leq 8|V|\omega + e^{-a} + 4a|V|\omega$. It remains to bound the disagreement probability assuming Γ holds.

On Γ. Multiplying the disagreement condition by $\hat{p}_v \leq 1$ and using $\hat{p}_v\Delta_v \leq 2\omega e^{E_v} \leq \frac{1}{4}$ (as $G_v \geq 8\omega$) gives $u_v \leq \frac{1}{2} + \frac{1}{4} + \rho(a + \frac{1}{2}) \leq 1$. Hence $E_v = \hat{p}_v x + u_v \leq x + 1$, so $\hat{p}_v\Delta_v \leq 2e\,\omega e^{x}$, and

$$u_v \leq c_v := 2\omega e^{x}\left(\hat{p}_v/\hat{p}_{i^\star} + e\right) + \hat{p}_v\,\rho\,(x + \tfrac{1}{2}).$$

Given $(i^\star, x)$, $\Pr[u_j \leq c_j] \leq c_j$ for each j, and $\sum_{j\neq i^\star} c_j \leq 2\omega e^{x}(1/\hat{p}_{i^\star} + e|V|) + \rho(x + \frac{1}{2})$. Averaging over $i^\star$ ($\mathbb{E}[1/\hat{p}_{i^\star}] = |\{i : \hat{p}_i > 0\}| \leq |V|$) and then over $x \leq a$ ($\mathbb{E}[e^x; x \leq a] = a$, $\mathbb{E}[x] + \frac{1}{2} = \frac{3}{2}$) bounds the disagreement probability on Γ by $2(1+e)a|V|\omega + \frac{3}{2}\rho$. Adding $\Pr[\bar{\Gamma}]$ gives the claim. □

Appendix B. Speculative Decoding Results

Table 10 gives the per-receiver results behind the speculative-decoding comparison of Section 7.2. On each GPU, the sender drafts four tokens per round with a small model of its own family (Gemma 2B, Llama 1B, Qwen 0.5B) and keeps them up to the first position where its own race disagrees. Every receiver decodes the posted text at the sender's stopping point with the cascade decoder. The H100 plain columns are the runs behind Table 4, including the fp16 and fp32 receivers it omits, given as counts of 200 rather than percentages; the A6000 columns repeat the comparison with texts written and decoded on the A6000.

TABLE 10. PLAIN AND SPECULATIVE SENDERS ON EACH GPU: FLIP RATE (%) AND MESSAGES DECODED OF 200 AT THE SENDER'S STOPPING POINT, WITH THE CASCADE DECODER.

		H100				A6000			
		Plain		Spec.		Plain		Spec.	
Family	Receiver	Flips	Dec.	Flips	Dec.	Flips	Dec.	Flips	Dec.
Gemma	9B bf16	0.9	200	1.0	199	0.9	200	0.9	199
	9B fp16	1.3	199	1.4	200	1.3	200	1.3	199
	9B fp32	1.3	199	1.4	200	1.3	200	1.3	199
	9B int8	3.8	196	4.0	199	4.2	199	3.9	198
	9B NF4	8.4	187	8.9	188	8.9	189	8.7	186
	2B bf16	31.5	84	31.2	101	31.6	87	31.4	90
Llama	8B bf16	1.0	200	1.0	200	1.1	200	1.0	200
	8B fp16	1.0	200	0.9	200	1.0	200	1.0	200
	8B fp32	1.1	200	0.9	200	1.1	200	1.0	200
	8B int8	4.9	200	4.6	200	4.7	200	4.8	200
	8B NF4	10.2	196	10.1	196	10.3	197	10.2	198
	3B bf16	23.5	175	23.6	180	23.6	185	23.6	180
Qwen	7B bf16	1.2	200	1.2	200	1.2	200	1.3	200
	7B fp16	1.3	200	1.3	200	1.4	200	1.3	200
	7B fp32	1.3	200	1.3	200	1.3	200	1.3	200
	7B int8	4.2	200	4.2	200	4.3	199	4.3	200
	7B NF4	12.1	199	12.0	200	12.3	199	12.3	198
	1.5B bf16	36.9	168	37.1	173	37.2	173	37.1	178