Accurate, Costless, and Invisible AI Text Watermarking for Self-Hosted AI Inference

ARBI CITY*

Technical report, August 2026

Abstract

Statistical text watermarking — embedding a verifiable signal in a language model's word choices by deriving the sampler's randomness from a secret key — is now deployed by the major AI providers, but has so far been available only inside their proprietary serving stacks. This report describes a complete implementation of the keyed-Gumbel watermarking scheme in an open-source inference engine, arbi-serve, together with an end-to-end measurement of the three properties on which the practical value of such a watermark rests. First, *accuracy*: detection is an exact statistical test whose false-positive rate is guaranteed by arithmetic rather than by a trained classifier; we measure detection power as a function of text length, editing, and key mismatch. Second, the *absence of a quality penalty*: the construction provably leaves every token's conditional distribution unchanged, a property that requires careful statement — we explain exactly what is and is not preserved, and verify the neutral cases empirically. Third, *speed*: we describe the engineering required to make the watermark's cost unmeasurable in practice — fused GPU kernels for the keyed sampler and its context state, and an extension of the scheme through speculative decoding in which the deterministic keyed choice turns draft verification into an exact-match test, so that watermarked speculative output is bit-identical to non-speculative watermarked output at full watermark density. We report throughput overheads of 0.1–0.2% at standard decoding and discuss the measured behaviour of speculative acceptance under the watermark. The implementation, its kernels, tests, and a model-free detector are part of the arbi-serve engine.

1 Introduction

In August 2026 the transparency obligations of the European Union's AI Act came into force for providers of general-purpose AI models, accompanied by a Code of Practice whose signatories undertake to mark machine-generated content so that it can be identified [1]. For images, established mechanisms existed; for text, the industry converged on a family of statistical watermarks that hide the mark in the model's own word choices. Google DeepMind's SynthID-Text has run in production in Gemini since 2024 [2], and Anthropic announced in August 2026 that Claude's output is watermarked with a licensed variant of the same approach [3]. Both descend from a construction described by Aaronson in 2022 [4].[1]

These schemes operate entirely inside the serving stack: the model's weights are untouched, and the mark is applied at the moment the sampler chooses among candidate tokens. This locality has a consequence that has received less attention than it deserves. Any operator of an inference engine can, in principle, watermark the output of any open-weights model — and organisations that self-host models arguably have stronger reasons to do so than the original developers, since the EU's obligations attach to systems placed on the market rather than only to their developers, and since an operator may want to establish, with evidence rather than recollection, whether a given document was produced by its own systems. In practice, however, no widely used open-source serving engine offers a sampler-level text watermark, and the generic extension points those engines do offer are not compatible with their speculative-decoding fast paths.[2] To our knowledge, the implementation described here is the first in an open serving engine, though the underlying science is published and the engineering is, as we hope to show, within reach of any serious engine.

This report is organised around the three properties that determine whether such a watermark is useful in operation. Section 2 recalls the construction. Section 3 treats *accuracy*: what a detection result does and does not establish, and with what error guarantees. Section 4 treats the *quality question*: the sense in which the watermark provably does not change what the model writes, the senses in which it does change the sampling process, and the measurements that bound the residual effects. Section 5 treats *speed*: the kernel engineering that reduces the standard-decoding cost to noise, and the extension of the scheme through speculative decoding, where a 2024

impossibility result [7] appears to forbid exactly the combination we want; we show how the keyed scheme's determinism yields an exact-match verification whose output is bit-identical to non-speculative watermarked decoding, and we report its measured acceptance behaviour. Section 6 summarises the implementation, naming each mechanism and its source files, and Section 7 states limitations.

2 The keyed-Gumbel construction

A language model produces, at each step, a probability distribution over its vocabulary, and the serving engine samples a token from it. Efficient samplers realise the draw with the Gumbel-max identity: independent Gumbel noise is added to each candidate's log-probability and the maximiser is taken,

```
token = argmax_v ( log p(v) + g_v ),      g_v = -log(-log u_v),      u_v ~ Uniform(0,1),
```

which yields an exact sample from `p`. The watermark of [4] changes a single aspect of this procedure: the uniforms `u_v` are no longer drawn from a random-number generator but are computed by a keyed pseudorandom function of the visible context,

```
seed = Hash(key, last H tokens),          u_v = PRF(seed, v),
```

with a sliding window of $H = 4$ tokens in the published configuration and in ours. Because the keyed values are computationally indistinguishable from fair uniforms, each token's probability of being selected is unchanged; what changes is that a holder of the key can recompute, for any position in any text, the uniform that the sampler would have used there, and ask whether the observed token looks like the winner of that particular draw.

Detection is a one-sided test built directly on this replay. For each position t, the detector recomputes `u_t = PRF(Hash(key, window before t), token_t)`. If the text was not generated with this key, each `u_t` is an independent uniform — the hash of unrelated text is noise — and the statistic `S = Σ_t -log(1 - u_t)` follows a Gamma distribution exactly, giving an exact p-value. If the text was generated with the key, selected tokens carry systematically larger uniforms and S grows linearly beyond its null expectation. Two refinements matter in practice. Repeated context windows are masked: reusing a window's seed would force the same continuation and create repetition artefacts, so the generator keeps a record of used windows and falls back to ordinary randomness on recurrence, and the detector applies the same skip rule. And the detector operates on tokens alone — it never consults the model or its logits — so numerical nondeterminism in serving (batch-dependent kernels, reduction orders) cannot affect it.

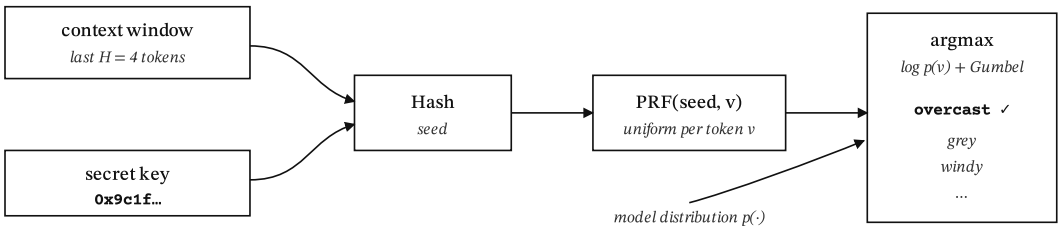

Figure 1: The keyed sampler. The only change relative to standard Gumbel-max sampling is the origin of the per-candidate uniforms: a keyed hash of the visible context window replaces the random-number generator. Per-token selection probabilities are unchanged; the key-holder can replay the noise from the text alone.

3 Accuracy: what detection establishes, and how reliably

The practical worth of a text watermark is decided at the detector, so we begin there. The central property is that the false-positive guarantee is *arithmetic*. Under the null hypothesis — the text was not produced with this key — the replayed uniforms are independent Uniform(0,1) variables regardless of who wrote the text, in what dialect, or with what style, and the p-value computed from the Gamma tail is exact. An operator who declares text watermarked only below $p = 10^{-6}$ will falsely implicate approximately one unmarked text per million tested, indefinitely, with no training data, no classifier drift, and no differential behaviour across writers. This is a

materially different foundation from the stylometric "AI detectors" in commercial use, whose error rates are empirical, unstable, and have been documented to disadvantage non-native writers. A keyed detector's positive verdict is evidence in the ordinary scientific sense: the probability of the observation under the alternative explanation is stated, and small.

The corresponding cost is on the other side of the test, and it should be stated equally plainly. A negative result establishes nothing. Text from another provider's key, from any unwatermarked open-weights deployment, from a paraphrasing tool, or from a human being produces the same silence. The watermark supports one-directional inference — positives with quantified confidence, negatives without content — and any policy or accusation built on "no watermark, therefore human" is unsound by construction.

Within these boundaries, the measured behaviour of our implementation is summarised in Table 1. Generations use Qwen3.8-27B (exl3 4-bit) served with multi-token-prediction speculation (K=5) on a single RTX 4090 at temperature 1.0, with the detection threshold fixed at $p < 10^{-6}$ throughout.

Condition	Median p-value	Detected
First 25 tokens of each completion	4.6×10^{-3}	6%
First 50 tokens	5.7×10^{-5}	25%
First 100 tokens	1.6×10^{-7}	69%
Full completion	8.4×10^{-23}	100%
Full completion, 10% of tokens edited	6.9×10^{-11}	88%
Full completion, 20% edited	2.2×10^{-4}	6%
Watermarked, tested with a wrong key	0.56	0%

Table 1: Detection at a fixed 10^{-6} false-positive threshold (Qwen3.8-27B, MTP K=5, temperature 1.0, 16 completions per condition). Evidence accumulates with length; full completions detect without exception, editing degrades the signal gracefully because a changed token corrupts only the handful of context windows overlapping it, and the wrong-key null sits at chance. The early-prefix rows are weaker than on a small model: the 27B is more confident, so its opening tokens — often predictable essay scaffolding — carry little per-token surprise, and the evidence concentrates in the free-flowing body.

Three aspects of this table deserve comment. Evidence accumulates per token, and only at positions where the model had genuine freedom: roughly twenty-five tokens of ordinary prose already provide odds of billions to one, while constrained material contributes almost nothing. Editing degrades the signal gracefully rather than catastrophically, because the seed depends on a four-token window: a substituted word corrupts the few windows that overlap it and the scheme re-synchronises immediately after. And the null rows — including watermarked text tested with the wrong key — sit exactly at chance, as the arithmetic requires.

The freedom-dependence has a further consequence worth isolating, because it bears on what a positive result means. We asked the watermarked model to reproduce a 116-token human-written paragraph verbatim. It complied, and the copy scored a mean luck of 0.535 against a chance level of 0.500 — orders of magnitude short of the detection threshold, where freely composed text of the same length scores beyond 10^{-25}. Where the model merely transmits text it was given, every "choice" is forced and no evidence accrues. The watermark therefore testifies to *composition* by the keyed sampler, not to mere passage of text through the model; quoted material inside an AI-generated answer carries no mark, and the detector cannot be used to claim that a model echoed a particular passage.

Finally, the detector itself is deliberately small. It requires the key, a tokenizer, and roughly one hundred lines of arithmetic — no GPU, no model weights, no access to logits — and our public demonstration runs it client-side in a browser, bit-identical to the engine's implementation. Auditability of the detector matters as much as its guarantees: a party affected by a detection claim can re-run the entire test themselves.

4 The absence of a quality penalty, stated carefully

The providers' claim that watermarking "does not affect quality" invites scepticism, and it deserves a more careful statement than it usually receives, because it is simultaneously stronger and narrower than it sounds.

The strong part is that the core property is a theorem, not a tuning outcome. Conditional on the context, the keyed uniforms are (computationally) exactly the fair uniforms the sampler would otherwise have used, so the probability that any given token is selected is unchanged: the watermarked sampler draws from precisely the same conditional distribution as the unwatermarked one, at every temperature and under every top-k or nucleus filter. There is no bias toward "watermark-friendly" words, no restriction of the vocabulary, and no additional content in the text — the mark consists entirely of *which* of the equally legitimate outcomes occurred. In this respect the scheme differs fundamentally from distortionary watermarks that tilt the distribution toward a keyed subset of the vocabulary and pay a measurable quality cost at fixed strength.[3]

The narrow part is that this guarantee is *per-response*. Across responses, the sampling process is visibly different: with a fixed key, the randomness at a given context is fixed, so an identical prompt regenerates an identical completion where an unwatermarked sampler would produce a fresh variation. The per-response quality is intact; what is spent is diversity across retries of the same prompt. Whether this matters depends entirely on the deployment. For a consumer chat product with a regenerate button and best-of-n features, it is a real cost, and it is part of why the deployed proprietary schemes chose a construction that retains per-query randomness at some expense in mechanism complexity.[4] For self-hosted and enterprise deployments the calculus is different: deterministic, reproducible output conditioned on input is routinely a feature — the analogue of seeded sampling, which every major engine offers deliberately — and is congenial to auditing and caching. Our implementation also softens the effect structurally: because repeated context windows fall back to per-request randomness (Section 2), two identical requests share an opening stretch and then diverge at the first recurring four-token window rather than remaining identical throughout. An operator who wants full per-query diversity can additionally mix a per-conversation nonce into the key, accepting that the detector must then be given the nonce alongside the text.

Between the theorem and the process-level differences sits an empirical question: does the implementation realise the theorem, and do the second-order effects (repeated-context masking, the interaction with filtering) leave any measurable trace in the text? Table 2 summarises the checks. The sampler-level test draws two hundred thousand tokens from identical menus through the keyed and unkeyed paths and finds the distributions identical within Monte-Carlo error. Greedy (temperature-zero) decoding, where the theorem promises exact equality, is bit-identical with the flag on and off. On free generation with the real model, the per-token negative log-likelihood of watermarked completions is statistically indistinguishable from the control arm, and standard degeneration statistics (distinct-n, longest repeated run) show no repetition artefacts — the check that the repeated-context masking is doing its job.

Check	Method	Result
Conditional distribution	200,000 keyed vs. unkeyed draws, identical menus	equal within MC error
Greedy decoding	temperature-0 output, watermark on vs. off	bit-identical
Model-level quality	per-token NLL, watermarked vs. control arm	2.500 vs. 2.441 nats (Δ +0.059 $\pm$ 0.255)
Degeneration	distinct-2; longest repeated run	0.95 vs. 0.93; 3 vs. 2
Speculative decoding	committed stream vs. sequential keyed sampling	bit-identical (Section 5.2)

Table 2: Quality-neutrality checks (Qwen3.5-0.8B, temperature 1.0 unless stated). The first two rows verify the distribution-preservation theorem directly at the sampler; the model-level rows bound any residual effect of the masking and filtering interactions; the last row is the speculative-decoding guarantee of Section 5.

One further consequence of the construction is worth making explicit because it is often asked as a question about privacy. The signal has no payload. Its only inputs are the provider-wide key and the tokens visible on the page; there is mathematically no channel in which a user identity, account, timestamp, or conversation identifier

could be embedded. The strongest statement any detector can produce is that *this key's sampler composed these words*. Whether that property is read as a limitation (no traitor-tracing) or as a safeguard (no covert user marking) depends on the application, but it is a structural fact of the scheme rather than a policy choice.

5 Speed: reaching performance parity

The remaining question is cost. The arithmetic added by the watermark is trivial — a hash and a pseudorandom draw per candidate — but serving engines are pipelined systems in which fixed per-step work translates directly into latency, and a naive implementation is not free. This section describes what was required to make the watermark's cost unmeasurable at standard decoding, and then the more interesting problem: keeping it free under speculative decoding, where an impossibility result appears to stand in the way.

5.1 Standard decoding

arbi-serve's sampler already drew its Gumbel noise from a counter-based generator (Philox [9]) keyed on a seed, so the watermark reduces conceptually to changing where the seed comes from. The engineering lies in keeping that change off the critical path. Each request's four-token context window lives in a device-resident ring buffer, so seed computation never synchronises with the host; the per-request record of used windows is a small Bloom filter, also GPU-resident; and the entire per-step watermark work — window hash, Bloom probe, seed selection, and after commitment the ring advance and Bloom insertion — is fused into two small kernels. The distinction matters in practice: our first, straightforward implementation cost roughly 10% of throughput on a deliberately adversarial configuration (a 0.8B-parameter model stepping every ~2 ms, where some twenty small kernel launches per step are material), and the fused version reduces the isolated cost to ~35 µs per step, which the engine's pipelined execution then hides almost entirely. Table 3 shows the end-to-end result under locked GPU clocks with interleaved arms and length-matched requests: 0.1–0.2%, within the noise of the harness. On production-scale models with 10–40 ms steps the same fixed cost is an order of magnitude further below measurement noise.

Concurrency	Baseline (tok/s per request)	Watermarked	Δ
1	458.0	457.4	−0.1%
8	339.8	339.3	−0.2%
64	237.6	237.6	−0.0%

Table 3: Per-request generation throughput, watermark off vs. on (Qwen3.5-0.8B, RTX 4090, GPU clocks locked at 2.7 GHz, arms interleaved, decode lengths pinned equal). A small model is deliberately the *worst case* for this measurement: its ~2 ms steps make the watermark's fixed per-step cost maximally visible, and it is still within harness noise.

On a production-scale model the same fixed cost is a far smaller fraction of each step. On Qwen3.8-27B (exl3 4-bit, MTP K=5) on a single RTX 4090, base decode runs at 54.6 tok/s and speculation raises it to 73.4 tok/s (single-stream, temperature 0) — a 34% gain from MTP at a per-step time of ~14 ms, against which the watermark's ~35 µs is 0.25% at most, and the watermark-on served throughput (70–74 tok/s) is indistinguishable from watermark-off within run-to-run variance. The watermark cost does not grow with model size; it is a constant the larger step simply amortises.

5.2 Speculative decoding: verification as an equality test

Modern engines rarely decode one token per model step. In speculative decoding [8], a cheap drafter proposes several tokens, the full model scores them in a single batched pass, and a rejection rule accepts a prefix of the proposals while preserving the output distribution exactly. Combining this with watermarking is known to be genuinely constrained: Hu and Huang [7] prove that, over all model and drafter distributions, no procedure can simultaneously retain the full watermark strength of non-speculative watermarked decoding and the full acceptance rate of unwatermarked speculation. The deployed proprietary scheme reflects the same tension in its two published speculative variants, one preserving speed at reduced detectability and one the reverse [2].

The keyed-Gumbel scheme, however, has a property that changes the character of the problem: once the key and context are fixed, the watermarked "draw" is not a distribution to be sampled but a single reproducible token, `t* = argmax(log p + keyed noise)`. Verification of a drafted token therefore does not require accept coins or residual distributions at all; it collapses to the question *is the draft equal to t*?* Our verifier computes t* for every speculation depth in one batched pass — each depth's seed derives from its own context window, namely the committed text extended by the drafted prefix — accepts the longest matching prefix, and commits t* itself at the first mismatch (Figure 2). Three properties follow directly. The committed stream is *bit-identical* to what non-speculative keyed sampling would have produced, a stronger statement than the distributional equality that speculative sampling normally targets, and one we pin with a golden test rather than argue informally. Every committed token carries the key, so the watermark runs at full density with no reduced-strength variant. And the whole procedure adds three small kernel launches per speculative step — the seed grid, the batched keyed draw, and the commit that advances the ring and records only the context windows actually consumed — the same cost class as standard keyed decoding.

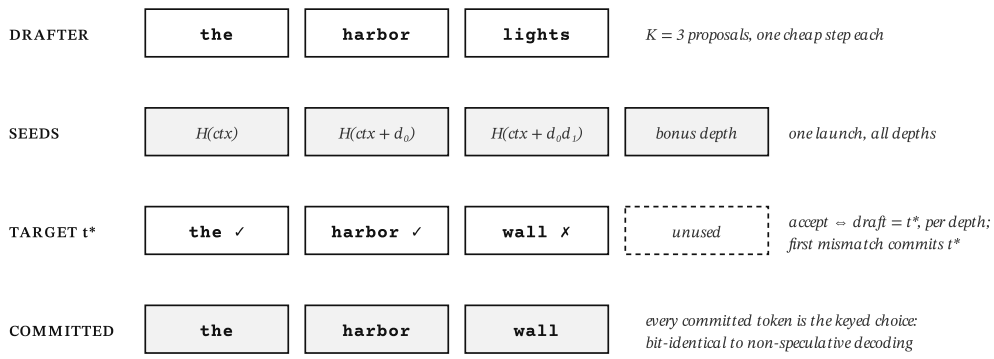

Figure 2: Exact-match verification under multi-token prediction. Each speculation depth receives the keyed seed of its own context window; the model's keyed choice t* is deterministic at every depth, so acceptance is an equality test and the committed stream equals the non-speculative watermarked stream token for token.

How does this coexist with the impossibility result? The theorem's premises concern procedures that must preserve a watermarked *distribution* for every proposal distribution the drafter might use. Two features of the deployment move it outside the binding region. First, production drafters overwhelmingly propose greedily — a point-mass proposal — and for a point mass at token d the unwatermarked acceptance probability is p(d), while the probability that the keyed choice equals d is, marginally over the key's pseudorandomness, also p(d): at this operating point the two quantities the theorem separates coincide. Second, when the drafter genuinely samples from a distribution q — a mode engines use because it raises acceptance on high-entropy text — the acceptance loss comes from the independence of the draft's randomness, and can be removed rather than suffered: we draw the draft with the *same* keyed noise the verifier will use at that depth, so that draft and target agree whenever their filtered distributions do, with probability one when q = p. The committed stream is unchanged in either case; only the acceptance rate, and hence speed, is at stake.

Table 4 reports the measured acceptance of the production configuration: Qwen3.8-27B with its native multi-token-prediction head at K=5, watermark on. The head commits about 2.5 tokens per model step, the source of the throughput gain of §5.1. The measurement carries two lessons. First, acceptance is preserved under the watermark by construction: exact-match verification commits the same token non-speculative keyed sampling would emit, so a draft is accepted exactly when it equals that token — a condition independent of whether the noise came from the key or an RNG. There is no watermark-induced acceptance penalty on this greedy-drafted path; the coupling of §5.9 is needed only when the drafter itself samples. Second, acceptance is temperature-stable, slightly favouring temperature 1.0, because a native MTP head is trained to predict the model's sampled continuations rather than its greedy argmax — so it aligns best with the sampling regime it was trained on, not with temperature-0 decoding. A methodological caveat we learned the hard way: acceptance must be measured over controlled request batches, not read from a server's cumulative counters, which mix prompt lengths and regimes and badly understate the rate.

The coupling applies wherever drafting is sequential: the engine wires it through the bundled MTP head's chain, the DSpark drafter's left-to-right block realisation, and the Gemma-4 shared-KV assistant chain. A purely parallel block drafter (plain DFlash) cannot be coupled — each position's keyed seed depends on the realised predecessors, which a single parallel pass does not have — and simply accepts at the independent rate, with output correctness unaffected. This is a general observation about watermarked speculation: verification-side correctness is drafter-agnostic, while acceptance-side coupling requires the drafter to realise its proposals in sequence.

Configuration	Accept rate	Tokens/step
Native MTP head (K=5), temperature 0	0.290	2.45
Native MTP head (K=5), temperature 1.0	0.333	2.67

Table 4: Speculative acceptance with the watermark on (Qwen3.8-27B, native MTP head, K=5 drafts per step, essay prompts, controlled batches). Because exact-match verification commits the same token the non-speculative watermarked sampler would emit, the watermark-on acceptance *is* the drafter's hit rate on that token — a watermark-off run of the same greedy-drafted slate accepts the identical prefix, so these rates are the watermark-neutral rates too (the bit-identity of §5.2). Two points worth noting: acceptance is temperature-*stable* here — indeed slightly higher at temperature 1.0 — because the native head is trained to predict the model's *sampled* tokens rather than its argmax, so it aligns with temperature-1.0 sampling; and ~2.5 committed tokens per model step is the source of the 34% throughput gain in §5.1. Keyed draft-side coupling (§5.9) applies when the drafter itself samples; the native head drafts greedily, where the exact-match equality already gives watermark-neutral acceptance without coupling. We also confirmed the neutrality head-to-head rather than only by construction: against a watermark-off twin of the same K=5 server driven by an identical batch of essay prompts at temperature 1.0, the watermark-on acceptance rate landed within run-to-run noise of the watermark-off rate — two repeats gave differences of −1.8% and +1.1%, bracketing zero — so the acceptance-neutrality is observed empirically and not merely argued.

Tensor parallelism composes with the scheme without additional communication. Under a sharded verifier, each GPU holds a slice of the vocabulary and the ranks exchange only small per-shard candidate sets; because the candidate set provably covers the entire kept support, the candidate-restricted keyed argmax equals the full-vocabulary keyed argmax, and every rank computes the same t* from rank-agreed state with no extra exchange on the critical path. A golden test drives the two-GPU sharded path and the single-GPU path step-for-step on the same inputs and asserts bit-identical committed streams.

6 Implementation summary

The implementation totals one new sampler module, two kernel files, and hooks in the speculative-decoding verifier, all shipped in the arbi-serve engine and enabled by a single environment variable (`ARBI_WATERMARK_KEY`); with the variable unset the engine is byte-identical to its pre-watermark behaviour. Table 5 names the mechanisms and their source locations.

Mechanism	Location
Context ring, Bloom masking, host detector (`WatermarkContextCache`, `detect_tokens`)	`arbi_serve/sampler/watermark.py`
Fused seed/advance kernels; speculative seed-grid and commit kernels	`arbi_serve/sampler/_watermark_kernel.py`
Keyed Gumbel-argmax draws (shared noise layout, all consumers)	`arbi_serve/sampler/gumbel_argmax_triton.py`, `_gumbel_argmax_kernel.py`
Exact-match speculative verification	`arbi_serve/spec_decode/rejection_sampler.py`, `mtp_verify_accept.py`, `mtp_verify_async.py`
Tensor-parallel (sharded) keyed verification	`arbi_serve/spec_decode/mtp_verify_sharded.py`
Keyed draft-side coupling for sampled drafters	`arbi_serve/spec_decode/rejection_sampler_drafter.py`, `mtp_driver_draft.py`
Configuration guards (undetectable configurations refused at boot)	`arbi_serve/engine/build_phases_load.py`, `flag_contracts.py`
Golden and parity tests (bit-identity; TP2 ≡ TP1; coupling; kernel/host parity)	`tests/test_watermark*.py`, `tests/test_watermark_mtp*.py`

Table 5: Implementation map. A shared property runs through the design: the keyed uniform at a token is defined once — Philox keyed on the seed with the vocabulary index as counter — and consumed bit-identically by the standard sampler, the speculative verifier, the coupled drafter, the tensor-parallel path, and the host-side detector, with GPU/host parity pinned by tests for every kernel.

Two design rules are worth recording beyond the file map. First, correctness claims are carried by golden tests rather than by argument: the speculative stream is asserted bit-identical to the sequential keyed stream; the two-GPU and single-GPU verifiers are asserted bit-identical step-for-step, with a wrong-key mutation confirming the test's discriminative power; and the drafter coupling is asserted to accept every draft when the drafter's distribution equals the target's. Second, configurations whose draws the detector could not replay — an alternative inverse-CDF sampler backend, a rollback verification mode, and the use of the publicly disclosed demonstration key — are refused at boot rather than served silently, on the view that a watermark that silently fails to mark is worse than none.

7 Limitations

The limitations follow from the construction and should be read together with Sections 3–5. Detection is one-directional; silence certifies nothing, and in particular cannot distinguish human text from any unwatermarked model's output. The mark is only as strong as the entropy of the text: code, arithmetic, verbatim reproduction, and temperature-zero decoding are faintly marked or unmarked by design. Same-prompt determinism, discussed in Section 4, is a real property of the fixed-key scheme; deployments that require per-query diversity must spend a nonce and its bookkeeping. The key is a single point of failure in both directions — whoever holds it can detect and, in principle, forge — so key custody is an operational requirement, not an afterthought. The coupled drafting paths currently maintain their watermark state on a single GPU, falling back to uncoupled (correct but slower-accepting) drafting under tensor parallelism; and acceptance benchmarking under a deterministic watermark requires prompt-diverse workloads, since each (key, prompt) pair contributes exactly one trajectory (Section 5.2). Finally, all measurements here are from one implementation on one model family and one GPU class; the numbers are offered as an existence proof and a methodology, not as universal constants.

8 Availability

The implementation, kernels, tests, standalone detector, and the interactive browser demonstration are part of the arbi-serve engine, currently in limited preview; access and the accompanying non-technical exposition are available through the ARBI CITY article [10]. The detector requires only the key and a tokenizer and is straightforward to reimplement from Section 2 for independent audit.

---

[1] The lineage is worth a footnote of its own. Aaronson described the keyed-Gumbel ("exponential") scheme while at OpenAI in 2022 [4]; press reporting in 2024 established that OpenAI built a working prototype and kept it unreleased over product concerns [5], and ChatGPT text remains unwatermarked as of this writing. SynthID-Text [2] is not the Gumbel scheme: it selects tokens through keyed knockout tournaments among candidates sampled from the model, a construction that preserves per-query randomness and offers a tunable strength parameter, and the Nature paper benchmarks Gumbel sampling as a separate baseline. Anthropic's 2026 deployment [3] licenses the tournament-family approach and applies it selectively, skipping low-entropy output.

[2] vLLM, SGLang, TGI, and TensorRT-LLM expose user-supplied logits processors, and a SynthID processor exists for the HuggingFace Transformers library; but processor interfaces do not receive prompt tokens uniformly, are bypassed or unsupported on speculative and multi-step paths (see e.g. vLLM issues #8182 and #17799), and in any case place per-step Python on the hot path. A watermark that is silently absent whenever the fast path is active is arguably worse than none, which motivates the in-engine approach taken here.

[3] The best-known distortionary family is the green-list scheme of Kirchenbauer et al. [11], which boosts a keyed subset of the vocabulary by a fixed logit offset; strength and distortion trade off explicitly. Distortion-free alternatives include the fixed-key-sequence construction of Kuditipudi et al. [12], which shares the determinism trade-offs discussed in Section 4.

[4] Tournament sampling redraws its candidates from the model at each query, so regenerating a prompt yields a fresh watermarked completion; the price is a more intricate mechanism and, per [2], a modestly higher serving cost than the Gumbel scheme (0.57% versus 0.26% added latency in their measurement). The choice between the families is thus largely a product decision about the value of per-query diversity; Section 5.2's argument suggests the Gumbel family's determinism is specifically advantageous where speculative decoding must be watermarked at full strength.

References

[1] European Commission. *Code of Practice on Transparency of AI-Generated Content*; transparency obligations under Regulation (EU) 2024/1689 (AI Act), in force 2 August 2026.

[2] S. Dathathri, A. See, S. Ghaisas, et al. Scalable watermarking for identifying large language model outputs. *Nature* 634:818–823, October 2024.

[3] Anthropic. How Claude's text watermark works. Technical explainer, August 2026.

[4] S. Aaronson. My AI safety lecture for UT Effective Altruism, November 2022. Describes the keyed-Gumbel watermark developed at OpenAI.

[5] D. Seetharaman and M. Barnum. There's a tool to catch students cheating with ChatGPT. OpenAI hasn't released it. *The Wall Street Journal*, August 2024.

[6] vLLM project. Issues #8182 (prompt tokens and logits processors) and #17799 (logits processors under the V1 engine). github.com/vllm-project/vllm.

[7] Z. Hu and H. Huang. Inevitable trade-off between watermark strength and speculative sampling efficiency for language models. In *Advances in Neural Information Processing Systems 37* (NeurIPS 2024). arXiv:2410.20418.

[8] Y. Leviathan, M. Kalman, and Y. Matias. Fast inference from transformers via speculative decoding. In *Proceedings of the 40th International Conference on Machine Learning* (ICML 2023).

[9] J. K. Salmon, M. A. Moraes, R. O. Dror, and D. E. Shaw. Parallel random numbers: as easy as 1, 2, 3. In *Proceedings of SC'11*.

[10] ARBI CITY. ARBI CITY advances AI text watermarking for the EU and beyond. August 2026. arbicity.com/news/ai-text-watermarking-for-self-hosted-ai/.

[11] J. Kirchenbauer, J. Geiping, Y. Wen, J. Katz, I. Miers, and T. Goldstein. A watermark for large language models. In *ICML 2023*.

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