VRBDecode: Proof-Carrying Receipts for Accountable Stochastic Decoding

Anonymous Submission

Abstract

Stochastic decoding policies (temperature scaling, top-k, top-p, and randomized sampling) critically shape text generation behavior but are typically treated as non-verifiable implementation details. This makes it difficult for clients and auditors to verify whether an AI service adhered to an agreed decoding policy, randomness procedure, and output transcript.

We present VRBDecode, a proof-carrying receipt protocol for accountable stochastic decoding over fixed-size candidate sets. VRBDecode defines a deterministic fixed-point decoding relation suitable for arithmetic circuits, binds per-step pseudo-randomness to public commitments, and chains step-level evidence into a tamper-evident receipt hash. For long generations, VRBDecode supports streaming proofs by folding step circuits into a single incrementally verifiable argument.

Our prototype implementation uses Poseidon commitments and Nova folding. In an evaluation across candidate sizes $K \in \{16, 32, 64\}$ and sequence lengths $N \in \{32, 64, 128, 256\}$ (5 repetitions), we measure per-step constraint costs, end-to-end proving time scaling, verifier time, proof size, and peak memory. Verification is sub-second in our experiments, while proof artifacts are currently large (13–84 MB depending on K).

Keywords: Stochastic decoding, verifiable computation, zero-knowledge proofs, receipts, folding

1. Introduction

Large language model (LLM) deployments increasingly rely on stochastic decoding to control quality, diversity, and safety. Common policies—temperature scaling, top-k, and nucleus (top-p) sampling—are widely used to mitigate degeneration and to improve long-form behavior [1]. Yet in production systems these policies are typically enforced only by implementation convention: a client receives an output transcript but cannot externally verify which policy was used, how randomness was derived, or whether any post-processing biased the sampling procedure.

This accountability gap matters for settings where decoding behavior is part of a service-level agreement (SLA) or audit requirement: e.g., regulated summarization, content moderation pipelines, or multi-party workflows where downstream decisions depend on reproducible generation behavior. In such settings, a provider that deviates from the stated policy can (intentionally or accidentally) alter outcomes without leaving verifiable evidence. While anchoring receipts on-chain is a natural deployment option, our primary focus is audit-grade verification that can be performed off-chain by clients or independent auditors.

We address this problem with VRBDecode, a proof-carrying receipt scheme for stochastic decoding. The key idea is to verify *policy compliance conditional on a fixed candidate set* at each decoding step: given a size-K set of candidate token IDs and logits, VRBDecode proves that the emitted token matches a deterministic, fully specified fixed-point decoding relation and that per-step pseudo-randomness is bound to public commitments. This does not prevent a provider from choosing an arbitrary candidate set; rather, it provides verifiable evidence of correct policy execution given whatever candidate set was used. The system produces:

- a succinct proof attesting to compliance for all steps; and
- a hash-chained receipt that commits to the policy, randomness commitment, candidate set digest, and outputs.

VRBDecode complements (rather than replaces) zkML systems that prove model inference: a prover may compose a forward-pass proof (to justify candidate logits) with VRBDecode (to justify decoding). Our focus is the decoding layer because it is both security-critical and comparatively lightweight to prove.

Our contributions are:

- We formalize a deterministic fixed-point decoding relation for temperature scaling, sorting with a tie-break rule, top-k, top-p, and unbiased sampling over a candidate set.
- We design a proof-carrying receipt protocol that binds decoding policy parameters and per-step pseudo-randomness into a tamper-evident transcript.
- We implement streaming proofs for long generations using Nova folding and evaluate constraint costs, prover time, verifier time, proof size, and peak memory across realistic K and N ranges.

2. Related Work

VRBDecode is motivated by accountability gaps in deployed ML services. Prior cryptographic ML systems primarily prove model evaluation [2]. Recent work improves end-to-end performance for verifiable inference by optimizing compilation and

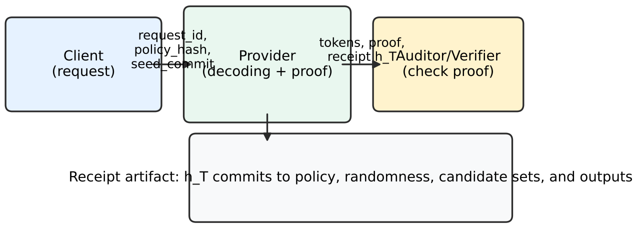

Figure 1: System architecture and receipt flow for VRBDecode.

proof generation [3, 4, 5]. These systems target the forward pass; VRBDecode targets a distinct surface: the stochastic decoding policy and its randomness procedure. The approaches are complementary and can be composed.

Efficient in-circuit hashing is central to receipt chains. Poseidon [6] is designed for proof systems over prime fields and is widely used for constraint-efficient commitments.

For long transcripts, VRBDecode relies on incremental verification. Halo [7] demonstrates practical recursive proof composition without a trusted setup. Nova [8] introduces folding schemes enabling efficient incremental verifiable computation (IVC), while SuperNova [9] and HyperNova [10] extend these ideas to more general machine models and constraint systems. VRBDecode uses Nova to fold token-by-token decoding steps while maintaining a receipt hash chain.

Finally, verifiable off-chain computation and authenticated transcripts also arise in blockchain and dispute resolution settings [11, 12, 13]. VRBDecode focuses on the decoding policy surface and produces auditable artifacts that can be stored off-chain or anchored on-chain.

3. Methodology

3.1. System model and threat model

VRBDecode involves a client, a provider that generates tokens and produces proofs, and a verifier (which may be a client, auditor, or a smart contract). We consider a malicious or faulty provider that may deviate from a stated decoding policy, bias sampling, or alter reported outputs. The verifier checks a proof that links reported outputs to public commitments and to the candidate set used at each step.

Our scope is decoding over a fixed candidate set of size K. We do not prove that the candidate set was derived from the full vocabulary, nor do we prove the model forward pass. This design choice isolates a widely used and security-relevant layer (sampling policy compliance) that can be composed with separate mechanisms for producing or validating candidate sets.

3.2. Commitments, public inputs, and receipt chaining

At request time, the client and provider agree on: (i) a unique request identifier (`request_id`); (ii) a commitment to the decoding policy parameters (`policy_hash`); and (iii) a commitment used to bind per-step pseudo-randomness (`seed_commit`).

The provider then generates tokens step by step. At step t, the provider forms a candidate set of size K (token IDs and logits), deterministically computes the policy-specified distribution over the candidate set, and samples the emitted token y_t. VRBDecode maintains a receipt hash h_t that commits to the entire transcript. A deployment may log candidate sets privately and reveal them selectively for audit; in that case, the receipt chain provides a compact commitment that auditors can re-check against the revealed transcript.

Informally, each step updates:

$$h_t = \text{Poseidon}(\text{domain} \,\|\, h_{t-1} \,\|\, \texttt{request\_id} \,\|\, \texttt{policy\_hash} \,\|\, \texttt{seed\_commi} \tag{1}$$

Here cand_hash_t is a Poseidon digest of the sorted candidate set for step t, W_s is the summed weight of the top-p prefix, and R is the sampling threshold used in the unbiased selection rule.

Figure 1 summarizes the roles and artifacts.

3.3. Per-step pseudo-randomness binding

VRBDecode binds per-step pseudo-randomness to public commitments to prevent substitution of randomness after the fact. In our implementation, a per-step 64-bit value U_t is derived inside the circuit via a domain-separated Poseidon hash over `request_id`, `policy_hash`, `seed_commit`, and the step index t. The proof enforces that the sampled token is consistent with the derived U_t and the deterministic decoding relation.

3.4. Security discussion (informal)

We summarize VRBDecode's security goal and assumptions informally. Let the public inputs include `policy_hash`, `seed_commit`, and the final receipt h_T. The prover's witness includes the per-step candidate sets, intermediate decoding values, and the emitted tokens y_t.

Guarantee (soundness-based). Assuming the soundness of the underlying proof system (Nova folding and its commitment scheme) and collision resistance of Poseidon for receipt hashing, if the verifier accepts then there exists a sequence of candidate sets and per-step states such that: (i) each emitted token y_t equals the output of the deterministic DecodingSpec pipeline applied to the step's candidate set and the committed policy parameters; (ii) the per-step pseudo-random values U_t are derived as specified from the committed context; and (iii) the receipt chain updates are computed consistently, so h_T commits to the full transcript.

Tamper evidence. Any attempt to claim a different policy, randomness commitment, step index, or output token while keeping the same receipt value would require either breaking proof soundness or finding a Poseidon collision across receipt updates.

Out-of-scope. VRBDecode does not prevent a provider from choosing an arbitrary candidate set; it provides policy-compliance evidence conditional on that set. Proving that the candidate set matches the model's full-vocabulary distribution is complementary work.

3.5. Deterministic decoding relation over a candidate set

The core technical challenge in proving decoding is eliminating ambiguity: floating-point operations, sorting ties, and sampling modulo bias can lead to implementation differences. VRBDecode therefore specifies a deterministic fixed-point decoding relation.

Each step consumes a candidate set of size K consisting of token identifiers and logits. Logits are represented as signed Q16.16 integers. Temperature scaling uses a fixed rounding rule. Candidates are sorted by scaled logit in descending order, with ties broken by ascending token identifier. The relation applies top-k filtering by restricting to the first k candidates after sorting.

To implement nucleus sampling within top-k, the relation computes approximate exponent weights in fixed-point, forms a cumulative sum, and selects the minimal prefix reaching a fixed-point threshold corresponding to top-p. Sampling uses an unbiased multiply-high rule: letting U_t be a 64-bit pseudorandom value and W_s be the total weight of the retained prefix, the sampler computes $R = \text{high64}(U_t \cdot W_s)$ and selects the first prefix whose cumulative weight exceeds R.

3.6. Fixed-point exponentiation approximation

The decoding relation requires exponentiation to compute unnormalized weights. We use a deterministic approximation designed to be efficient in circuits and consistent across implementations. Let z_i be the stabilized logit difference after temperature scaling, where $z_i \leq 0$. We clamp z_i to a fixed minimum (-12.0) and decompose z_i into an integer part and a remainder: $z_i = -(n) + r$, where $n \in \{0, \ldots, 12\}$ and $r \in [-1, 0]$ in Q16.16.

We compute $\exp(z_i)$ as a product of: (i) a precomputed lookup constant $E[n] \approx \exp(-n)$ in Q30; and (ii) a 5th-order polynomial approximation to $\exp(r)$ on $[-1, 0]$: $P(r) = 1 + r + r^2/2 + r^3/6 + r^4/24 + r^5/120$, evaluated in fixed-point with flooring rules.

The per-step weight is $w_i = \lfloor E[n] \cdot P(r)/2^{30} \rfloor$ (Q30), and weights are summed to form W_k and the nucleus threshold $\lfloor \text{top\_p} \cdot W_k \rfloor$. This method is fully specified and testable, trading floating-point fidelity for deterministic reproducibility.

3.7. Correctness and tamper-detection tests

To support reproducibility and to reduce specification ambiguity, VRBDecode is accompanied by a compliance test suite for the deterministic decoding relation. We use:

- a golden vector set (50 cases) covering edge conditions (ties, extreme temperatures, and boundary values for top-k and top-p);
- a randomized equivalence suite (1000 cases) comparing against a reference implementation; and
- negative tamper tests demonstrating that changes to policy parameters, per-step randomness, or the claimed output token lead to mismatches in the derived decoding outputs (and therefore would be rejected by the proof/receipt verification).

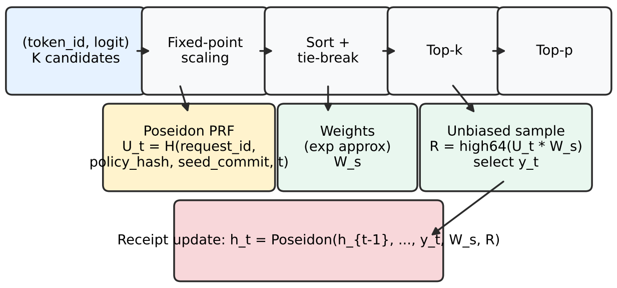

Figure 2: Deterministic decoding pipeline (fixed-point scaling, sorting, top-k, top-p, unbiased sampling) and where the proof binds.

3.8. Streaming proof construction via folding

Long generations require amortization across many steps. VRBDecode uses Nova [8] to fold a sequence of per-step relations into a single IVC proof. We implement two relations: `StepCircuit` is a per-step R1CS circuit that enforces decoding, pseudo-randomness derivation, and receipt updates; `StepFCircuit` is a Nova-compatible step relation that supports folding while maintaining a compact folded state whose primary evolving component is the receipt hash.

3.9. Implementation

Our prototype is implemented in Rust over the BN254 scalar field using arkworks. Poseidon [6] is instantiated over this field for the per-step PRF and receipt updates. The Nova construction is instantiated using the Sonobe folding-schemes library.

4. Results and Discussion

4.1. Experimental setup

We evaluate across candidate set sizes $K \in \{16, 32, 64\}$ and sequence lengths $N \in \{32, 64, 128, 256\}$. All results report 5 repetitions. We measure constraint counts for per-step circuits, circuit generation time, Nova preprocessing time, average proving time per step, end-to-end folding time, verifier time, proof size, and peak resident set size (RSS).

Benchmarks were run on Linux (Ubuntu 22.04) on an Intel Core i9-10940X CPU with 125 GiB RAM using optimized release builds.

For constraint benchmarking (Figure 3 and the first part of Table 1), we instantiate a synthetic candidate set and choose a simple "worst-case" policy configuration with top_k $= K$, top_p $= 1.0$, and $T = 1.0$ to exercise all pipeline components.

For Nova folding benchmarks (Figures 4 and 5 and the second part of Table 1), we use a pre-generated test vector suite (golden plus randomized vectors) that provides candidate sets and policy parameters per step; we then fold the first N vectors for each K. To isolate proving costs, we set request and commitment fields to fixed values in the benchmark harness; this does not change the circuit structure and still exercises the in-circuit hashing for per-step pseudo-randomness and receipt updates.

Table 1: Constraint counts and performance for verifiable decoding and folding.

K	StepCircuit (constraints)	Gen(s)	StepFCircuit (constraints)	G
16	122,452	0.0829	54,236	0.
32	306,288	0.2208	169,752	0.
64	880,232	0.4640	607,120	0.

K	Preprocess(s)
16	4.2594
32	7.0391
64	23.7664

K	Steps	Avg step(s)	Total(s)	Verify(s)	Size(bytes)
16	32	2.5065	80.2075	0.1678	13,515,928
16	64	2.5480	163.0694	0.1410	13,515,928
16	128	2.5575	327.3587	0.1647	13,515,928
16	256	2.5631	656.1574	0.1588	13,515,928
32	32	4.5892	146.8557	0.2621	28,297,880
32	64	4.5883	293.6528	0.2581	28,297,880
32	128	4.6693	597.6750	0.2104	28,297,880
32	256	4.6530	1191.1676	0.2342	28,297,880
64	32	16.3128	522.0112	0.6782	84,272,792
64	64	16.4933	1055.5734	0.5752	84,272,792
64	128	16.7118	2139.1090	0.6445	84,272,792
64	256	16.8266	4307.6000	0.6047	84,272,792

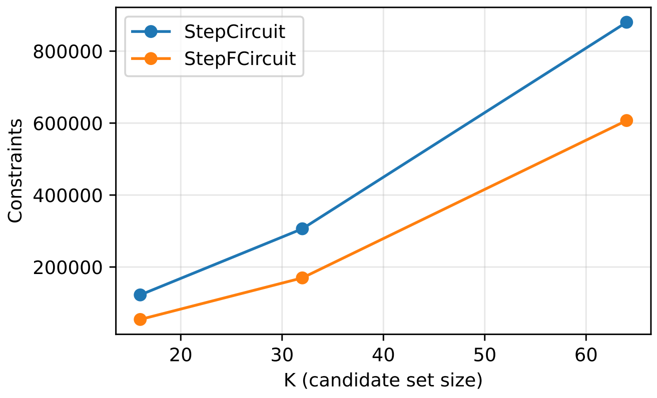

Figure 3: Constraint scaling with candidate set size K.

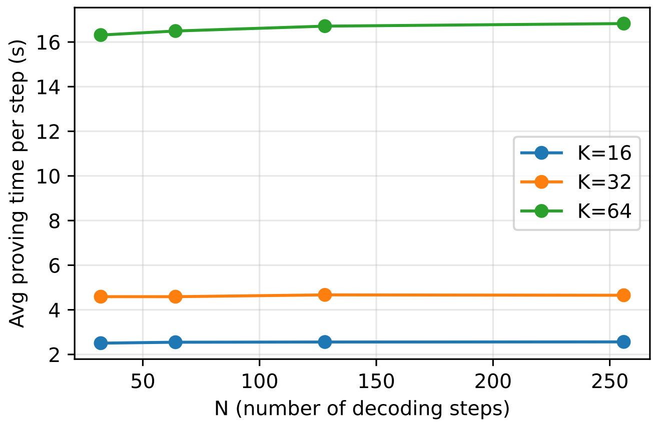

Figure 4: Average proving time per step versus number of steps N.

4.2. Constraint costs and synthesis overhead

Figure 3 shows constraint scaling with K. For $K = 64$, our per-step decoding circuit requires 880,232 constraints and the Nova-compatible step relation requires 607,120 constraints (Table 1). This growth reflects the costs of sorting with deterministic tie-breaking, hashing the candidate set, and enforcing unbiased sampling.

In addition to performance, we validated functional correctness of the deterministic decoding relation using the compliance tests described in Section 3.7, including golden vectors, randomized equivalence against a reference implementation, and negative tamper tests for policy/randomness/output changes.

4.3. End-to-end proving time, verifier time, proof size, and memory

Figures 4 and 5 summarize scaling with the number of folded steps N. For $K = 16$, average proving time is approximately 2.5 s per step across $N \in \{32, 64, 128, 256\}$, and peak RSS stays below 0.5 GiB. For $K = 32$, proving time is approximately 4.6 s per step with peak RSS below 0.75 GiB. For $K = 64$, proving time rises to roughly 16–17 s per step with peak RSS around 1.6 GiB.

Verification remains sub-second across configurations (Table 1), suggesting that proofs can be checked efficiently by clients or auditors even when proving is expensive.

Proof size scaling. In our implementation, proof size is effectively independent of N for a fixed K because Nova folding produces a constant-size IVC proof whose size does not grow with the number of folded steps. However, proof size grows substantially with K because increasing K increases the per-step circuit size, which increases the size of the committed R1CS instances and associated proof artifacts under our chosen commitment scheme and serialization.

Compression and packaging strategies. Reducing proof size is an important engineering step for deployment. Promising directions include: (i) wrapping the final folded proof in a succinct SNARK to compress proof artifacts into a small constant number of group elements; (ii) optimizing commitment parameters and proof encodings to avoid redundant elements; and (iii) treating h_T as the primary on-chain artifact while storing the full proof off-chain with content-addressed references.

4.4. Deployment discussion

The measured proving costs (2.5–17 s/step) make interactive, real-time proof generation challenging for long generations. We therefore view the most immediate deployment scenario as *batch or post-hoc auditing*: a provider serves responses normally, then generates proofs asynchronously for transcripts selected by policy, sampling, or dispute triggers. In this regime, sub-second verification still enables lightweight third-party checking, while proof generation can be provisioned on dedicated hardware.

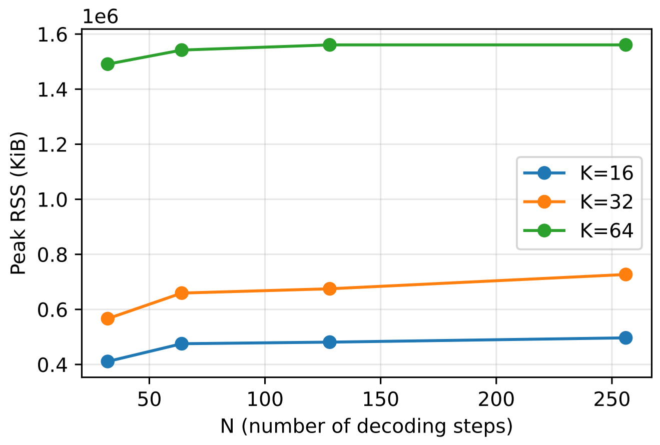

Figure 5: Peak memory usage versus number of steps N.

4.5. *Limitations*

VRBDecode provides verifiability for the decoding layer, but it does not by itself guarantee that the candidate set corresponds to the full-vocabulary model distribution. Proving the forward pass (or proving a correct top-K extraction from the full vocabulary) is complementary future work that can be composed with VRBDecode.

Our implementation uses a fixed-point approximation of exponentiation to make the decoding relation circuit-friendly. While the approximation is deterministic and testable, it differs from standard floating-point implementations; thus VRBDecode is best viewed as enabling *policy compliance with respect to a public, precise specification* rather than compliance with an informal floating-point reference.

5. Conclusion

We presented VRBDecode, a proof-carrying receipt protocol for accountable stochastic decoding over fixed-size candidate sets. VRBDecode binds policy parameters and per-step pseudo-randomness into a tamper-evident receipt hash and supports streaming proofs for long generations via Nova folding. Our prototype demonstrates feasibility across candidate sizes up to $K = 64$ and sequence lengths up to 256 steps, with subsecond verification and measurable scaling in proving time and memory.

We view VRBDecode as a building block for audit-grade AI services: it makes decoding behavior externally verifiable and composable with emerging systems for verifiable inference.

References

[1] A. Holtzman, J. Buys, L. Du, M. Forbes, Y. Choi, The curious case of neural text degeneration (2020). `arXiv:1904.09751`.
URL `https://arxiv.org/abs/1904.09751`

[2] P. Mishra, R. Lehmkuhl, A. Srinivasan, W. Zheng, R. A. Popa, Delphi: A cryptographic inference service for neural networks, in: 29th USENIX Security Symposium (USENIX Security 20), 2020.
URL `https://www.usenix.org/conference/usenixsecurity20/presentation/mishra`

[3] B. Chen, S. Waiwitlikhit, I. Stoica, D. Kang, ZKML: an optimizing system for ML inference in zero-knowledge proofs, in: Proceedings of the Nineteenth European Conference on Computer Systems, EuroSys 2024, Athens, Greece, April 22-25, 2024, ACM, 2024, pp. 560–574. `doi:10.1145/3627703.3650088`.
URL `https://doi.org/10.1145/3627703.3650088`

[4] H. Sun, J. Li, H. Zhang, zkllm: Zero knowledge proofs for large language models (2024). `arXiv:2404.16109`.
URL `https://arxiv.org/abs/2404.16109`

[5] B.-J. Chen, L. Tang, D. Kang, Zktorch: Compiling ml inference to zero-knowledge proofs via parallel proof accumulation (2025). `arXiv:2507.07031`.
URL `https://arxiv.org/abs/2507.07031`

[6] L. Grassi, D. Khovratovich, C. Rechberger, A. Roy, M. Schofnegger, Poseidon: A new hash function for Zero-Knowledge proof systems, in: 30th USENIX Security Symposium (USENIX Security 21), 2021.
URL `https://www.usenix.org/conference/usenixsecurity21/presentation/grassi`

[7] S. Bowe, J. Grigg, D. Hopwood, Recursive proof composition without a trusted setup, Cryptology ePrint Archive, Paper 2019/1021 (2019).
URL `https://eprint.iacr.org/2019/1021`

[8] A. Kothapalli, S. Setty, I. Tzialla, Nova: Recursive zero-knowledge arguments from folding schemes, Cryptology ePrint Archive, Paper 2021/370 (2021).
URL `https://eprint.iacr.org/2021/370`

[9] A. Kothapalli, S. Setty, SuperNova: Proving universal machine executions without universal circuits, Cryptology ePrint Archive, Paper 2022/1758 (2022).
URL `https://eprint.iacr.org/2022/1758`

[10] A. Kothapalli, S. Setty, HyperNova: Recursive arguments for customizable constraint systems, Cryptology ePrint Archive, Paper 2023/573 (2023).
URL `https://eprint.iacr.org/2023/573`

[11] A. E. Kosba, A. Miller, E. Shi, Z. Wen, C. Papamanthou, Hawk: The blockchain model of cryptography and privacy-preserving smart contracts, IACR Cryptol. ePrint Arch. (2015) 675.
URL `http://eprint.iacr.org/2015/675`

[12] F. Zhang, E. Cecchetti, K. Croman, A. Juels, E. Shi, Town crier: An authenticated data feed for smart contracts, in: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016, ACM, 2016, pp. 270–282. `doi:10.1145/2976749.2978326`.
URL `https://doi.org/10.1145/2976749.2978326`

[13] J. Teutsch, C. Reitwießner, A scalable verification solution for blockchains (2019). `arXiv:1908.04756`.
URL `https://arxiv.org/abs/1908.04756`