

Deterministic Factoring in Time $N^{o(1)}$

Isaac Rudich

Louis-Martin Rousseau

Polytechnique Montréal

Abstract

The current best upper bound for deterministic factoring is $N^{1/5+o(1)}$. We provide a Lean certificate that there exists a deterministic Turing machine that computes the smallest prime factor of every N in $N^{o(1)}$ steps: for every $k \geq 1$ it runs within $C_k 2^{n/k}$ steps on n -bit inputs. The algorithm reduces factoring to counting divisors with prescribed leading bits, and counts them by leveraging the counting engine used to refute the Exponential Time Hypothesis (ETH). The Lean statement that a reader must check to verify the proof is only 13 lines and only uses Mathlib. The full proof is about 36,000 lines. This note was written less than 24 hours after the ETH refutations were posted. No human-readable explanation of these results exists yet, and their Lean proofs are long: Anthropic’s alone runs to about 152,000 lines. Our proof re-proves only the part of the counting engine it needs and imports nothing from those developments. We hope this application, with its short statement, serves as a stepping stone for those working to understand the engine; a follow-up paper will explain this proof in detail.

1 Introduction

Let $\text{minFac}(N)$ be the smallest prime factor of N . We prove, in Lean 4 [13] with Mathlib [38]:

Theorem 1 (`subexponentialRate`). *Let $N \in \mathbb{N}$ have an n -bit binary encoding. There is one deterministic Turing machine, with a finite alphabet, that takes in N and outputs $\text{minFac}(N)$, such that for every $k \geq 1$ there is a constant C_k for which it halts within $C_k 2^{n/k}$ steps on every input of length n . Its running time is therefore $2^{o(n)} = N^{o(1)}$.*

Theorem 1 gives the smallest prime factor of N . The following is a direct implication.

Theorem 2 (not in Lean). *There is one deterministic Turing machine that takes in $N \geq 1$ and outputs its complete factorization into primes in time $N^{o(1)}$.*

Proof. Run the machine of Theorem 1 on N , divide out the prime it returns, and repeat until 1 is left. There are at most $\log_2 N \leq n$ rounds, each on an input of at most n bits, so the total time is at most $n(C_k 2^{n/k} + \text{poly}(n))$ for every k , which is $N^{o(1)}$. $\square$

We have audited this result, not the proof. We checked the statement, 13 lines of Lean, reproduced in §3; the proof is checked by Lean. The result came about as follows. We asked AI to apply the result of the ETH paper [31] to factoring, and then to conjecture how much the exponent could be easily pushed down. The first request produced Theorem 1. The second is recorded as Conjecture 1. All the ideas were AI-generated: the reduction, the choice of method, the proof plans and all of the Lean code were produced by AI agents (Claude, Anthropic). Our role was to pose the questions and design the statement. The algorithm is galactic: its advantage over known methods appears only for astronomically large N .

2 Background

Deterministic factoring. Times count bit operations, i.e. multitape Turing machine steps [18]. Trial division takes $N^{1/2+o(1)}$. Lehman reached $N^{1/3+o(1)}$ [25]; Pollard [32] and Strassen [37] reached $N^{1/4+o(1)}$, and Bostan, Gaudry and Schost [4] and Costa and Harvey [10] improved Strassen’s method by factors of logarithmic size; methods based on Coppersmith’s lattice technique [9] reach $N^{1/4+o(1)}$ in polynomial space [18]. Hittmeir improved the exponent-1/4 bounds by a superpolynomial factor [21] and then broke the exponent 1/4 with $N^{2/9+o(1)}$ [22]. Harvey proved $O(N^{1/5} \log^{16/5} N)$ [18], and Harvey and Hittmeir improved it by a factor $(\log \log N)^{3/5}$ [19]; this is the record for every N . A recent paper gains a further factor $\log^{3/5} N$ for balanced semiprimes [17]. Every one of these bounds has a fixed positive exponent, and so do the conditional ones: under the generalized Riemann hypothesis, the class group method of Shanks [35] runs in time $N^{1/5+o(1)}$ (as reported in [18], after [11, §5.6.4]), and Umans and Wang show that a number-theoretic conjecture would give exponent 1/6 [39] (a sufficient arithmetic-progression form of it has since been refuted [20]). We know of no earlier deterministic bound of the form $N^{o(1)}$, conditional or not.

Faster, but randomized or heuristic. With $L_N[\alpha, c] = \exp((c + o(1))(\log N)^\alpha (\log \log N)^{1-\alpha})$: Dixon’s random squares method [14] and the class group relations method, in the variant of Lenstra and Pomerance [28], are proved but randomized, the latter in expected time $L_N[1/2, 1]$. The continued fraction method [29] and the quadratic sieve [33] are deterministic, with running times near $L_N[1/2]$ that rest on unproved smoothness heuristics. Lenstra’s elliptic curve method [27] is randomized and heuristic, with a running time governed by the smallest prime factor. The number field sieve [26] is heuristically $L_N[1/3, (64/9)^{1/3}]$ and is behind the recent records for numbers without small factors. On a quantum computer, Shor’s algorithm factors in polynomial time [36]. For a survey see [11]. Theorem 1 beats none of the classical methods above: $N^{o(1)}$ is far slower than $L_N[\alpha]$ for any $\alpha < 1$. What is new is a proven deterministic bound of $N^{o(1)}$.

The ETH refutations (as interpreted by Claude while writing the proof). The Exponential Time Hypothesis [24] says that 3-SAT needs time $2^{\Omega(n)}$. In October 2026 two unrefereed preprints refuted it by counting satisfying assignments in time $2^{o(n)}$. OpenAI’s preprint [31] deforms the count polynomially, evaluates it where a few shared linear measurements split the sum into groups, and recovers it by Hermite interpolation, in polynomial space. The second, by Claude (Anthropic) [5], uses a recursion on *networks*: nodes with finite setting sets and weights, and a factor on each pair of nodes that factors through a stated number of shared bits [5, Defs. 1–2]. Grouping $g = 3K^8$ nodes into K balanced groups turns a network into a clique with equality tests on its edges [5, Lemma 1], a variant of the grouping move of Williams [41] after Nešetřil and Poljak [30]. The eight-square identity of Degen and Graves [2, §1], [6, §6.10], in the sign form of Albuquerque and Majid [1, Props. 3.7, 4.4], gives a crossing identity (the construction of [23, Prop. 4.1] applied to it) that splits two crossing equality tests into pieces whose costs add up to 11/12 of theirs [5, Lemma 2]. One level puts g^2 nodes on a grid, expands every rectangle into such pieces (in the spirit of [16, 12], which replace a gadget by a linear combination of available ones), merges rows or columns and regroups, so that the exponent obeys $e' = (11/12 + O(1/K))e + O(1/K)$ and tends to 0 with the number of levels [5, Prop. 3]. Its main theorem counts k -CNF solutions in time $2^{O(\min\{k^2 n \log \log n / \sqrt{\log n}, km / \sqrt{\log m}\})}$, and it is itself formalized in Lean.

What we use from these papers. The proof uses Sections 3–5 of [5] (Definitions 1–3 and 5, Lemma 1, Lemma 2 with its eight-square and crossing identities, Proposition 3 and the conclusion

$e^* = 0$), re-proved from scratch. It does not use that paper’s Definition 4, its sparsification, its CNF network, or its Appendices A–D. Nothing of [31] enters the proof. The Lean package imports no statement, definition or code from either paper.

3 The statement

The Lean development is registered in the Palomar registry as entry PALOMAR-2026-10-28-000015 (<https://palomar-registry.org/entry?id=PALOMAR-2026-10-28-000015&version=1>).

The audit surface is one file of the registered development, `Challenge.lean`, built with Lean v4.35.0 and Mathlib v4.35.0. It imports only Mathlib (`Mathlib.Computability.TuringMachine.Computable`, `Mathlib.Data.Nat.Prime.Defs` and `Mathlib.Data.Nat.Log`). The declarations below are verbatim; doc comments, the module header and the namespace lines are omitted. The definitions:

```
def factor : ℕ → ℕ := Nat.minFac

abbrev FactoringMachine := TM2ComputableInTime encodeNat encodeNat factor

def FactoringMachine.FiniteAlphabets (M : FactoringMachine) : Prop :=
  ∀ k, Finite (M.tm.Γ k)

def FactorsWithin (M : FactoringMachine) (T : ℕ → ℕ) : Prop :=
  M.FiniteAlphabets ∧ ∀ n, M.time n ≤ T n

def FactoringInTime (T : ℕ → ℕ) : Prop :=
  ∃ M : FactoringMachine, FactorsWithin M T
```

The result, as the Challenge states it (its `sorry` is filled by `Solution.lean`), and the conjecture, stated as a proposition:

```
theorem subexponentialRate :
  ∃ M : FactoringMachine, ∀ k : ℕ, 1 ≤ k → ∃ C : ℕ,
    FactorsWithin M (fun n => C * 2 ^ (n / k)) := by
  sorry

def SqrtLogLogConjecture : Prop :=
  ∃ C : ℕ, FactoringInTime (fun n => C * 2 ^ (C * n / (Nat.sqrt (Nat.log 2 n) + 1)))
```

The proof is `Solution.lean`, which states the same declarations with the same names and types and proves the theorem by one line naming a lemma of the proof files. For `Factoring.subexponentialRate`, `#print axioms` prints [`propxt`, `Classical.choice`, `Quot.sound`].

How to read it.

- *Factoring* means `minFac` (Mathlib’s `Nat.minFac`). Applying it repeatedly gives the full factorization with at most $\log_2 N$ calls (Theorem 2).
- *The machine model* is Mathlib’s Turing machine model `TM2`. A `TM2ComputableInTime` structure bundles a finite-state machine (`FinTM2`), its input and output alphabets, a time function, and a proof that on every input a it outputs the encoding of $f(a)$ within time $(|encode(a)|)$ steps. Numbers are written by `Computability.encodeNat` (binary), so the input length n is the number of bits of N and $2^n \approx N$. Mathlib’s `TM2` machines are Carneiro’s, and `FinTM2` with its time notion is Spelier and van Gent’s (file headers; we know of no paper on them). Mathlib requires only the input stack’s alphabet to be finite; `FiniteAlphabets` requires it of every stack.

Conjecture 1 (SqrtLogLogConjecture). *Factoring can be done in time $\exp(O(\log N/\sqrt{\log \log N}))$, written in the input length as $C \cdot 2^{\lfloor Cn/(\lfloor \sqrt{\log_2 n} \rfloor + 1) \rfloor}$.*

This is a conjecture made by the AI when we asked it what it could easily prove. We did not ask it to follow through and prove this conjecture. The Challenge states it as a proposition, not as a theorem.

4 How the proof works (according to Claude, edited by humans)

The proof is about 36,000 lines in 143 files outside the audit surface. About 5,000 of them are copied from an ongoing factoring-related project the authors are still working on. At a high level the proof has five parts.

1. **Factoring from counts.** Write each divisor of N in binary using exactly n bits, where n is the number of bits of N , adding zeros on the left of shorter numbers (so with $n = 7$, the number 7 is written 0000111). Suppose we can answer one question: given a string x of bits, does N have a divisor other than 1 whose binary form begins with x ? Then we can find the smallest such divisor, which is $\text{minFac}(N)$, one bit at a time from the left. Start with the empty string. At each step, append 0 if some divisor begins with the result, and append 1 otherwise. (Some divisor always begins with the current string, so if none begins with it followed by 0, one begins with it followed by 1.) After n steps the string is $\text{minFac}(N)$. For example, the divisors of 91 other than 1 are 7, 13 and 91, written 0000111, 0001101 and 1011011, and the steps end at 0000111 = 7. Our machine answers each question by counting the divisors that begin with x and checking whether the count is zero (`search_eq_minFac`). This is a standard way to turn a search into counting; we know of no single source for it.
2. **Counting as a network.** Each divisor d of N pairs with exactly one e such that $d \cdot e = N$, so the count in part 1 is the number of pairs (d, e) with $d \cdot e = N$ and d beginning with x (the divisor 1 is subtracted separately). We build a large system of small checks, each looking at only a few bits, whose ways of filling in all bits that pass every check correspond exactly to these pairs (`mulNet_value`). The checks carry out the fast multiplication method of Schönhage and Strassen [34]: the product $d \cdot e$ is computed with a Fourier transform on numbers modulo $2^m + 1$, split into two layers of smaller transforms [8, 3], and then the carries are added up. Each check only stores the values it is given, so once d and e are chosen every other bit is determined. Grouping the checks into clusters gives a network in the sense of [5], whose value is the count we want, and in which the clusters share at most a constant times n bits in total (`clusterNet`).
3. **The counting engine.** Evaluating such a network by trying every filling takes about 2^n steps. The counting engine of [5], Sections 3–5, does better by repeating two moves. First, it merges the nodes into fewer, larger groups chosen so that the shared bits are spread evenly between groups; such a grouping always exists, and our machine finds one by trying them all. Second, it arranges the groups in a grid and uses two algebraic identities from [5] to rewrite the value as a sum of smaller problems, each costing a fixed fraction less in the exponent. After ℓ rounds it finishes by direct enumeration, obtaining the count multiplied by a known power of two, 2^E . More rounds give a smaller exponent. Our machine computes ℓ from n while it runs, and ℓ grows without bound as n grows, so a single machine runs within $C_k 2^{n/k}$ steps for every k (`uni_exp`). We re-proved the engine instead of importing it, in a variant that writes every table out in full and does all arithmetic modulo small numbers.
4. **Deciding whether the count is zero.** The number $V = 2^E \cdot (\text{count})$ can be far too large to handle cheaply, so the engine only computes its remainder after division by each $m = 1, 2, \dots, 2M$, which keeps every number small. If all these remainders are zero, then

V is divisible by the binomial coefficient $\binom{2M}{M}$, because $\binom{2M}{M}$ is a product of prime powers that are each at most $2M$. Since $\binom{2M}{M} \geq 4^M/(2M+1)$ and M is chosen so that V is smaller than this, V must be 0 (`eq_zero_of_all_dvd`). Both facts about $\binom{2M}{M}$ come from Erdős’ proof of Bertrand’s postulate [15].

5. **Running it on a Turing machine.** The engine is written in a small language of arithmetic expressions and compiled in three steps, each proved correct in Lean: to a simple register-machine program, then to a lower-level program, and finally to a Mathlib Turing machine. Each step slows the program down by at most a polynomial factor. Since the bound holds for every k , this costs nothing: a polynomial slowdown of $C 2^{n/k}$ is at most $C' 2^{n/k'}$ for a larger k' . That machine models simulate each other with polynomial slowdown is classical [7, 40]; the simulations here were written and proved from scratch.

5 Conclusion

We have provided a Lean certificate that one deterministic Turing machine finds the smallest prime factor of every N in $N^{o(1)}$ steps, and the complete factorization in the same time. This improves on the best known deterministic bound, $N^{1/5+o(1)}$. Relying on the result only requires a reader to check a 13-line statement that uses nothing but Mathlib.

The algorithm reduces factoring to counting divisors with prescribed leading bits and counts them with the engine behind the refutations of the Exponential Time Hypothesis, re-proved here without importing anything from those developments. Those refutations are recent, and their Lean proofs are long and not yet explained in a form a human reader can follow. We hope this application, with its short statement, serves as a stepping stone for those working to understand the engine. A follow-up paper will explain this proof in detail.

References

- [1] H. Albuquerque and S. Majid. Quasialgebra structure of the octonions. *Journal of Algebra*, 220(1):188–224, 1999. Numbering cited from arXiv:math/9802116.
- [2] J. C. Baez. The octonions. *Bulletin of the American Mathematical Society*, 39(2):145–205, 2002.
- [3] D. H. Bailey. FFTs in external or hierarchical memory. *The Journal of Supercomputing*, 4(1):23–35, 1990.
- [4] A. Bostan, P. Gaudry, and É. Schost. Linear recurrences with polynomial coefficients and application to integer factorization and Cartier–Manin operator. *SIAM Journal on Computing*, 36(6):1777–1806, 2007.
- [5] Claude (Anthropic). A $2^{o(n)}$ time algorithm for 3-SAT. Preprint, unrefereed; Lean development in the repository anthropics/formal-math, folder eth, 2026. Dated 26 October 2026.
- [6] J. H. Conway and D. A. Smith. *On Quaternions and Octonions: Their Geometry, Arithmetic, and Symmetry*. A K Peters, Natick, MA, 2003.
- [7] S. A. Cook and R. A. Reckhow. Time bounded random access machines. *Journal of Computer and System Sciences*, 7(4):354–375, 1973.
- [8] J. W. Cooley and J. W. Tukey. An algorithm for the machine calculation of complex Fourier series. *Mathematics of Computation*, 19(90):297–301, 1965.
- [9] D. Coppersmith. Small solutions to polynomial equations, and low exponent RSA vulnerabilities. *Journal of Cryptology*, 10(4):233–260, 1997.
- [10] E. Costa and D. Harvey. Faster deterministic integer factorization. *Mathematics of Computation*, 83(285):339–345, 2014.
- [11] R. Crandall and C. Pomerance. *Prime Numbers: A Computational Perspective*. Springer, New York, second edition, 2005.
- [12] R. Curticapean and M. Xia. Parameterizing the permanent: genus, apices, minors, evaluation mod 2^k . In *Proceedings of the 56th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2015)*, pages 994–1009, 2015.
- [13] L. de Moura and S. Ullrich. The Lean 4 theorem prover and programming language. In *Automated Deduction – CADE 28*, volume 12699 of *Lecture Notes in Computer Science*, pages 625–635. Springer, 2021.
- [14] J. D. Dixon. Asymptotically fast factorization of integers. *Mathematics of Computation*, 36(153):255–260, 1981.

- [15] P. Erdős. Beweis eines Satzes von Tschebyschef. *Acta Litterarum ac Scientiarum Regiae Universitatis Hungaricae Francisco-Josephinae, Sectio Scientiarum Mathematicarum (Szeged)*, 5:194–198, 1932.
- [16] A. Galluccio and M. Loeb. On the theory of Pfaffian orientations. I. Perfect matchings and permanents. *Electronic Journal of Combinatorics*, 6:R6, 1999.
- [17] Y. Gao, Y. Feng, H. Hu, and Y. Pan. On factoring and power divisor problems via rank-3 lattices and the second vector. *Mathematics of Computation*, 2026. Published online 18 March 2026; arXiv:2512.19076.
- [18] D. Harvey. An exponent one-fifth algorithm for deterministic integer factorisation. *Mathematics of Computation*, 90(332):2937–2950, 2021.
- [19] D. Harvey and M. Hittmeir. A log-log speedup for exponent one-fifth deterministic integer factorisation. *Mathematics of Computation*, 91(335):1367–1379, 2022.
- [20] X. He and A. Sahai. Refuting a conjecture of Umans and Wang on arithmetic-progression divisor covers. arXiv:2608.06681, 2026.
- [21] M. Hittmeir. A babystep-giantstep method for faster deterministic integer factorization. *Mathematics of Computation*, 87(314):2915–2935, 2018.
- [22] M. Hittmeir. A time-space tradeoff for Lehman’s deterministic integer factorization method. *Mathematics of Computation*, 90(330):1999–2010, 2021.
- [23] P. Hrubeš, A. Wigderson, and A. Yehudayoff. Non-commutative circuits and the sum-of-squares problem. *Journal of the American Mathematical Society*, 24(3):871–898, 2011. Numbering cited from ECCC TR10-021.
- [24] R. Impagliazzo and R. Paturi. On the complexity of k -SAT. *Journal of Computer and System Sciences*, 62(2):367–375, 2001.
- [25] R. S. Lehman. Factoring large integers. *Mathematics of Computation*, 28(126):637–646, 1974.
- [26] A. K. Lenstra and H. W. Lenstra, Jr., editors. *The Development of the Number Field Sieve*, volume 1554 of *Lecture Notes in Mathematics*. Springer, 1993.
- [27] H. W. Lenstra, Jr. Factoring integers with elliptic curves. *Annals of Mathematics*, 126(3):649–673, 1987.
- [28] H. W. Lenstra, Jr. and C. Pomerance. A rigorous time bound for factoring integers. *Journal of the American Mathematical Society*, 5(3):483–516, 1992.
- [29] M. A. Morrison and J. Brillhart. A method of factoring and the factorization of F_7 . *Mathematics of Computation*, 29(129):183–205, 1975.
- [30] J. Nešetřil and S. Poljak. On the complexity of the subgraph problem. *Commentationes Mathematicae Universitatis Carolinae*, 26(2):415–419, 1985.
- [31] OpenAI. The Exponential Time Hypothesis is false. Preprint, unrefereed, 2026. Dated 17 October 2026, revised 25 October 2026. <https://github.com/openai/math/blob/main/preprints/The-Exponential-Time-Hypothesis-is-False-October-25-2026/paper.pdf>.
- [32] J. M. Pollard. Theorems on factorization and primality testing. *Proceedings of the Cambridge Philosophical Society*, 76(3):521–528, 1974.
- [33] C. Pomerance. Analysis and comparison of some integer factoring algorithms. In H. W. Lenstra, Jr. and R. Tijdeman, editors, *Computational Methods in Number Theory, Part I*, volume 154 of *Mathematical Centre Tracts*, pages 89–139. Mathematisch Centrum, Amsterdam, 1982.
- [34] A. Schönhage and V. Strassen. Schnelle Multiplikation großer Zahlen. *Computing*, 7(3–4):281–292, 1971.
- [35] D. Shanks. Class number, a theory of factorization, and genera. In *1969 Number Theory Institute (Proc. Sympos. Pure Math., Vol. XX, Stony Brook, N.Y., 1969)*, pages 415–440. Amer. Math. Soc., 1971.
- [36] P. W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, pages 124–134. IEEE Computer Society Press, 1994.
- [37] V. Strassen. Einige Resultate über Berechnungskomplexität. *Jahresbericht der Deutschen Mathematiker-Vereinigung*, 78(1):1–8, 1976/77.
- [38] The mathlib Community. The Lean mathematical library. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP 2020)*, pages 367–381, 2020.
- [39] C. Umans and S. Wang. A number-theoretic conjecture implying faster algorithms for polynomial factorization and integer factorization. arXiv:2511.10851, 2025.
- [40] P. van Emde Boas. Machine models and simulations. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity*, pages 1–66. Elsevier, Amsterdam, 1990.
- [41] R. Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theoretical Computer Science*, 348(2–3):357–365, 2005.