hexagon:2611.00116v1

# THE PARITY OF THE PRIME-COUNTING FUNCTION IN TIME $x^{o(1)}$, ASSUMING THE NEW #SAT ALGORITHMS

VAIBHAV TULSYAN

Abstract. Computing the prime-counting function $\pi(x)$ exactly takes $x^{1/2+o(1)}$ steps by the analytic method of Lagarias and Odlyzko, and to our knowledge its parity is known faster only on short intervals: Tao, Croot and Helfgott decide whether an interval of length at most $x^{1/2+c}$ near $x$ contains an odd number of primes in time $x^{1/2-c+o(1)}$. However, their identity needs the Möbius function $\mu(d)$ for all $d \le \sqrt{x}$, and we know of no bound of the form $x^{o(1)}$ for the parity, even conditionally. We present deterministic algorithms that output $\pi(x)$ mod 2 and the parity of the number of primes in any interval $[A, B]$ in time $2^{O(n/\sqrt{\log n})} = x^{o(1)}$, where $n$ is the input bit length. Our approach replaces $\mu(d)$ by its residue modulo 2, which is the squarefree indicator again: the number of chains $d_1^2 \mid m$, $d_2^2 \mid d_1, \dots$ of depth $L$ is odd exactly when $m$ is squarefree, for $m < 2^{2^L}$. The parity of $\pi$ on $[A, B]$ is then the parity of one count of tuples satisfying $O(\log n)$ multiplications, which Anthropic's counting engine evaluates through a network whose variables have geometrically shrinking widths. We prove both theorems in Lean 4 (`piParity_TM_rate`, `piIntervalParity_TM_rate`; axioms `propext`, `Classical.choice`, `Quot.sound`). The result does not give deterministic prime finding or the exact value of $\pi(x)$: the parity can be even on every interval one queries, and exact counts seem to need iterated products, for whose product-tree verifiers we prove a lower bound in Lean. The constants are astronomical, and the result rests on Anthropic's Lean development of its claimed refutation of the Exponential Time Hypothesis, which is under review.

## 1. Introduction

How fast can one count primes? The prime-counting function $\pi(x)$ is computed exactly in $x^{2/3+o(1)}$ steps by the combinatorial method of Meissel, Lehmer, Lagarias, Miller and Odlyzko [6, 4], and in $x^{1/2+o(1)}$ steps by the analytic method of Lagarias and Odlyzko [7]. The same bound $x^{1/2+o(1)}$ is the best known for deterministically *finding* a prime between $x$ and $2x$, the question of the Polymath4 project [15, 16]. Tao, Croot and Helfgott [16] showed that one bit of $\pi$ can be had faster: they decide whether an interval $[a, b] \subseteq [N, 2N]$ of length at most $N^{1/2+c}$ contains an odd number of primes in time $O(N^{1/2-c+o(1)})$, for an absolute $c > 0$ (their Theorem 1.2), and Lelechenko [9] made $c$ explicit.

However, both results are for short intervals, and the obstacle to longer ones is structural. The identity behind [16] sums a divisor-sum function over $d \le \sqrt{N}$ with weights $\mu(d)$, and $\mu(d)$ for every $d \le \sqrt{N}$ is itself an $N^{1/2}$ computation. We are not aware of a published bound below $x^{1/2+o(1)}$ for $\pi(x)$ mod 2; Sladkey [14, ref. 5] cites a manuscript by Oliveira e Silva on computing the parity of $\pi$ as in preparation, and we could not find it.

In October 2026 OpenAI [11] and Anthropic [2] (below [O] and [A]) each claimed a deterministic algorithm that counts the satisfying assignments of a $k$-CNF in subexponential time, refuting the Exponential Time Hypothesis; both claims come with Lean 4 developments and are under review. Rudich and Rousseau [12] used the engines to factor in $N^{o(1)}$, and our earlier notes [19, 18] proved in Lean, on [A]'s engine, bounds of the form $2^{O(n/\sqrt{\log n})}$ for factoring, class numbers, square roots and divisor sums. We found no derivation of a parity result for $\pi$ from either refutation (Section 8).

### 1.1. The problem and the running example.

We consider the *interval prime parity* problem: given $A$ and $B$, output $\pi([A, B]) \bmod 2$, where $\pi([A, B]) = \#\{p \text{ prime} : A \le p \le B\}$. The case $A = 1$, $B = x$ is $\pi(x) \bmod 2$. We write $\tau^{<}(m) = \#\{(a, b) : ab = m,\ a < b\}$ and

$$C_L(m) = \#\{(d_1, \dots, d_L) :\ d_1^2 \mid m,\ d_2^2 \mid d_1,\ \dots,\ d_L^2 \mid d_{L-1}\}.$$

*Date*: November 2026.

2020 *Mathematics Subject Classification.* 11Y11, 11Y16, 11N05, 68Q25, 03B35.

*Key words and phrases.* Prime-counting function, parity, squarefree numbers, Möbius function, deterministic prime finding, exact counting, Exponential Time Hypothesis, Lean 4, formal verification.

**Example 1.1** (Running example)**.** Take $[A, B] = [1, 10]$ and $L = 2$. Table 1 lists $\tau^<(m)$, $C_2(m)$ and their product. The sum of the last row is 16, which is even, and indeed $\pi(10) = 4$. The summand is odd exactly at the primes $2, 3, 5, 7$: at $m = 6$ and $m = 10$ the factor $\tau^<(m) = 2$ is even, and at $m = 4, 8, 9$ the factor $C_2(m) = 2$ is even.

| $m$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | sum |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $\tau^<(m)$ | 0 | 1 | 1 | 1 | 1 | 2 | 1 | 2 | 1 | 2 | |
| $C_2(m)$ | 1 | 1 | 1 | 2 | 1 | 1 | 1 | 2 | 2 | 1 | |
| $\tau^<(m)\,C_2(m)$ | 0 | 1 | 1 | 2 | 1 | 2 | 1 | 4 | 2 | 2 | 16 |
| $m$ prime | | $*$ | $*$ | | $*$ | | $*$ | | | | 4 |

TABLE 1. The running example: the one-count identity on $[1, 10]$ with $L = 2$.

1.2. **The naive route and the principle.** At first one might count primes exactly, through a signed identity such as $[m \text{ squarefree}] = \sum_{d^2 \mid m} \mu(d)$, and read off the parity. However, a counting engine sums non-negative table entries over labellings; a weight $\mu(d)$ on a label $d$ of $\Theta(n)$ bits needs the factorisation of $d$ inside one node, and the nodes of [A]'s networks hold $o(n)$ bits. We call this the *Möbius barrier*. Instead, we rely on a different principle: *modulo* 2, $\mu(d) \equiv [d \text{ squarefree}]$, *so the Möbius weight is the problem itself at half the size, and the recursion closes after* $\log_2 n$ *halvings.* The resulting chain count is a plain count of divisibility witnesses, and the witnesses shrink geometrically, so their total size is $O(n)$.

1.3. **What we prove.** In this paper we prove that one count of a fixed system of $O(\log n)$ multiplications has the parity of $\pi([A, B])$, build the network that evaluates the count with $O(n)$ shared bits, and prove in Lean that one Turing machine per problem outputs the parity.

**Theorem 1.2** (Section 5; machine-checked)**.** *There are Turing machines $M_1, M_2$ in Mathlib's* `TM2` *model, with finite alphabets on all stacks, and for each of them a constant $C \in \mathbb{N}$, such that on every input word $w$ of length $n$ the machine halts within $\lfloor C \cdot 2^{Cn/\sqrt{\log_2(n+2)}} \rfloor$ steps with the following output:*

*(i) on $w =$* `encodeNat` *$x$, the machine $M_1$ outputs* `encodeNat` *$(\pi(x) \bmod 2)$ (*`piParity_TM_rate`*);*

*(ii) on $w =$* `encodePair` *$A\ B$, the machine $M_2$ outputs* `encodeNat` *$(\pi([A, B]) \bmod 2)$ (*`piIntervalParity_TM_rate`*).*

Since $n = \log_2 x + O(1)$ in (i), the bound is $x^{O(1/\sqrt{\log\log x})} = x^{o(1)}$, against $x^{1/2+o(1)}$ for the exact $\pi(x)$ [7]. In (ii) it is $\max(A, B)^{o(1)}$ for every interval, against $N^{1/2-c+o(1)}$ for intervals of length at most $N^{1/2+c}$ in $[N, 2N]$ [16]. To our knowledge, no bound of the form $x^{o(1)}$ was known for either problem, even conditionally.

1.4. **Contributions.** Our contributions are:

- We prove that the chain count $C_L(m)$ is odd exactly when $m$ is squarefree, for $1 \le m < 2^{2^L}$, and that $\sum_{m \in [A,B]} \tau^<(m) C_L(m) \equiv \pi([A, B]) \pmod 2$ for $B < 2^{2^L}$ (Section 3).
- We present a leveled description of this count, with $3L + 3$ variables at levels $0, \dots, L$, and a version of our network library in which a level-$k$ variable has about $n/2^k$ bits, so that $\Theta(\log n)$ multiplications share $O(n)$ bits in total (Section 4).
- We prove the two theorems of Theorem 1.2 in Lean and list what a reader must trust (Sections 5 and 7).
- We show what the result does not give: an adversary argument (on paper) for why parities alone do not find primes, and a Lean-checked lower bound for product-tree verifiers of the iterated product that exact counting by this route would need (Section 6).

The number-theoretic content is small: the chain lemma, and the observation that working modulo 2 bypasses the Möbius barrier. The network is engineering on top of our earlier library [18]; it is what makes the rate hold.

1.5. **Status of each claim.** We separate four kinds of claim. *Machine-checked in Lean*: the two theorems of Theorem 1.2, with the axioms `propext`, `Classical.choice` and `Quot.sound` only, and every lemma whose Lean name we give, including the tree bound of Theorem 6.2. *Proved on paper only*: the adversary statement of Theorem 6.1; the link from network cuts to the tree model, and the circuit simulation, in Section 6.2, where the need for long products is argued informally; the modulo-$k$ variant of Theorem 6.3; and the comparisons with earlier bounds. *Assumed*: nothing beyond the three axioms; [A]'s results enter as Lean theorems of its development, which the kernel checks with ours. *Checked numerically*: Theorem 1.1; the chain lemma for all $1 \le m < \min(2^{2^L}, 5000)$, $L \le 3$, and the one-count identity on every interval $[A, B]$ with $1 \le A \le 300$, $A \le B \le 3000$, $L = 4$, by a short exact computation that is not part of the proof; and the same identities for $m \le 255$ and intervals in $[1, 120]$ by `#eval` in Lean.

**Outline.** Section 2 recalls [A]'s counting theorem. Section 3 proves the chain lemma and the one-count identity, Section 4 builds the network and the machine, and Section 5 quotes the statements. Section 6 says what the result does not give, Section 7 describes the Lean development, Sections 8 and 9 cover related work and limitations, and Section 10 concludes.

## 2. The engine counts accepted labellings

In this section we recall the one theorem of [A] that we compose with, in the form used by our factoring note [19, §2]. We use [A]'s Lean development at commit `16a5ce4` (`anthropics/formal-math`, folder `eth`, Lean v4.33.1, Mathlib `0df444a`).

A *network on* $[g]$ *with parameters* $P, B, \tau$ has, for every node $a$, settings of at most $P$ bits and a weight $W_a$, and for every pair of nodes an interaction that factors through $c_{a,b}$ shared bits, with $\sum_b c_{a,b} \le P$; its value is $Z = \sum_z \prod_a W_a(z_a) \prod_{\{a,b\}} \Psi_{a,b}$ and its total is $U = \sum_{a \ne b} c_{a,b}$. We use only *label systems*: wires placed on pairs of nodes, each carrying a label seen by both ends, with $W_a$ a 0/1 table. The value is then the number of labellings accepted by every table.

**Imported Theorem 2.1** ([A], Lemma 8(i) and Theorem A(b); see [19, Imported Thm. 2.2])**.** *Let $K = K_\ell \ge 2$ be a level of* [A]*'s schedule, $g = 3K^8$, and $e_\ell \le (3x_0 + 6)/\sqrt{\log_2 K}$ for an absolute $x_0$. A network on $[g]$ with parameters $P, B, \tau$ and total $U$ can be evaluated within $\big(K^g + 2^{e_\ell K^2 L}(K(L + B'))^{C_0 K^2} 2^{2gP/K}\tau\big)(g(P + B))^{C_1}$ operations, where $L = \lceil (U + gP/K)/K^2 \rceil$, $B' = \lceil 2gP/K \rceil + g^2 B$ and $C_0, C_1$ are absolute.*

So a network family with $U = O(n)$ and $gP = O(n)$ is evaluated in $2^{O(n/\sqrt{\log n})}$ steps at the level $K^{40} \lesssim n$ of [19, §5]. The cost is exponential in $U/\sqrt{\log n}$: a network with $U = \Theta(n \log n)$ would cost $2^{\Theta(n\sqrt{\log n})}$, which is superpolynomial in $x$. Everything below is about keeping $U$ linear.

## 3. Squarefree chains and the one-count identity

In this section we prove the two counting facts. Both are in the pure Mathlib module `ChainParity`, whose definitions are

```
def chainCount : ℕ → ℕ → ℕ
  | 0, _ => 1
  | L+1, n => ∑ d ∈ n.divisors.filter (fun d => d * d ∣ n), chainCount L d
def tauLt (m : ℕ) : ℕ := ((Nat.divisorsAntidiagonal m).filter fun p => p.1 < p.2).card
```

so `chainCount` $L\ n = C_L(n)$ for $n \ge 1$ and `tauLt` $m = \tau^{<}(m)$.

**Lemma 3.1** (Chain lemma; `chainCount_mod_two`)**.** *For $L \ge 0$ and $1 \le n < 2^{2^L}$, $C_L(n) \equiv [n \text{ squarefree}] \pmod 2$. In Lean:*

```
theorem chainCount_mod_two {L n : ℕ} (h1 : 1 ≤ n) (h : n < 2 ^ 2 ^ L) :
    chainCount L n % 2 = if Squarefree n then 1 else 0
```

*Proof.* Write $n = st^2$ with $s$ squarefree (`Nat.sq_mul_squarefree_of_pos`). Then $d^2 \mid n$ if and only if $d \mid t$ (`dvd_of_mul_self_dvd`), and $n$ is squarefree if and only if $t = 1$. Read the identity $\sum_{d \mid t} \mu(d) = [t = 1]$ modulo 2, where $\mu(d) \equiv [d \text{ squarefree}]$:

$$\#\{d :\ d^2 \mid n,\ d \text{ squarefree}\} \ \equiv\ \sum_{d \mid t} \mu(d) \ =\ [n \text{ squarefree}] \pmod 2 \tag{3.1}$$

(`key_zmod`). Now induct on $L$. For $L = 0$, $n < 2$ forces $n = 1$, and $C_0(1) = 1$. For $L+1$, $C_{L+1}(n) = \sum_{d^2 \mid n} C_L(d)$, and every such $d$ satisfies $d \le \sqrt{n} < 2^{2^L}$, so by induction $C_L(d) \equiv [d \text{ squarefree}]$, and (3.1) finishes. □

The hypothesis $n < 2^{2^L}$ cannot be dropped: $C_1(16) = 3$ counts $d_1 \in \{1, 2, 4\}$, but 16 is not squarefree, while $C_2(16) = C_1(1) + C_1(2) + C_1(4) = 4$ is even. The depth must grow like $\log_2 n$.

**Proposition 3.2** (One-count identity; `primeParity`). *If $B < 2^{2^L}$, then $\sum_{m=A}^{B} \tau^{<}(m)\, C_L(m) \equiv \pi([A, B]) \pmod 2$. In Lean:*

```
theorem primeParity {L A B : ℕ} (hB : B < 2 ^ 2 ^ L) :
    (∑ m ∈ Finset.Icc A B, tauLt m * chainCount L m) % 2 =
      ((Finset.Icc A B).filter Nat.Prime).card % 2
theorem primeCounting_eq_card (x : ℕ) :
    Nat.primeCounting x = ((Finset.Icc 1 x).filter Nat.Prime).card
```

*Proof.* We show that each summand is odd exactly when $m$ is prime (`term_zmod`). For $m = 0$ and $m = 1$, $\tau^{<}(m) = 0$. For $m \ge 2$ not squarefree, $C_L(m)$ is even by Theorem 3.1. For $m \ge 2$ squarefree, $C_L(m)$ is odd, and the swap $(a, b) \mapsto (b, a)$ pairs the factorisations with $a < b$ and those with $a > b$, with none on the diagonal, so $\tau(m) = 2\tau^{<}(m)$ (`card_divisors_eq_two_tauLt`). Hence $\tau^{<}(m) = 2^{\omega(m)-1}$, which is odd if and only if $\omega(m) = 1$, that is, $m$ is prime (`tauLt_prime`, `tauLt_even`). The second statement relates Mathlib's `Nat.primeCounting` to the count on $[1, x]$. □

In Theorem 1.1, $L = 2$ suffices since $10 < 2^{2^2}$. The machine of Section 4 takes $L$ to be the bit length of $n$, here $L = 3$; on $[1, 10]$ the values $C_3(m)$ equal $C_2(m)$, and the count is again 16.

## 4. One count of a leveled network

In this section we write the sum of Theorem 3.2 as the number of solutions of a system of products, explain why our earlier network library cannot evaluate it at the rate, describe the leveled version that can, and assemble the machine.

4.1. **The description.** A *description*, in the sense of our class-number note [18, §3], lists variables, products of a variable by a variable or a constant, chains (equations between sums of products) and comparisons with constants or other variables. For depth $L$ and interval $[A, B]$ the description `primeDescL` has, for $k = 0, \dots, L$, the variables $d_k, s_k, e_k$, with $d_0 = m$, $s_0 = a$, $e_0 = b$, and the constraints

$$ab = m, \qquad d_k \cdot d_k = s_k, \quad s_k \cdot e_k = d_{k-1} \ \ (1 \le k \le L), \qquad m \ge \max(A, 1), \quad m \le B, \quad a < b.$$

So $s_k = d_k^2$ and $e_k = d_{k-1}/d_k^2$ witness $d_k^2 \mid d_{k-1}$, and they are determined by the chain. A solution is a factorisation $m = ab$ with $a < b$ together with a chain of depth $L$ from $m$, so the number of solutions is $\sum_{m \in [A,B]} \tau^{<}(m) C_L(m)$ (`primeDescL_count`). The condition $m \ge 1$ excludes $m = 0$, which every $d$ would divide. In Theorem 1.1 the 16 solutions include, for $m = 8$, the four tuples with $(a, b) \in \{(1, 8), (2, 4)\}$ and $d_1 \in \{1, 2\}$, $d_2 = 1$.

4.2. **Why the library of the class-number note does not suffice.** ArithNet [18] gives every variable the full box $[0, 2^{n_B})$ with $n_B > n$ bits, and its parameter bound is $U \le 2^{11} c_A(D)(n + 2)$ with $c_A(D) = 64(\mathrm{nV} + \mathrm{nP} + \mathrm{nQ} + 1)(\mathrm{nC} + 1)$. Our description has $\mathrm{nV} = 3L + 3$ variables, $\mathrm{nP} = 4L + 2$ products and $\mathrm{nC} = 2L + 1$ chains, so $c_A(D) = \Theta(L^2)$ and the bound is $U = \Theta(n \log^2 n)$, which makes the engine cost $2^{\Theta(n \log^{3/2} n)}$, worse than trial division. Even with every per-chain copy removed, $3L+3$ variables of $n$ bits carry $\Theta(n \log n)$ bits, which costs $2^{\Theta(n\sqrt{\log n})}$. The witnesses themselves are small: along a chain, $d_k^{2^k} \le m < 2^n$ (`chain_pow_le`), so $d_k$ has at most $n/2^k$ bits and all of them together fewer than $2n$. The network should pay for that.

4.3. **Levels.** A *leveled description* (`LDesc`, module `ArithLDesc`) is a description together with a level for every variable and every chain. A variable of level $k$ ranges over $[0, 2^{b\ell_k})$, where $b$ is the limb width of ArithNet and $\ell_k = \ell / 4^{\min(\lfloor k/2 \rfloor, r-1)}$ is the number of limbs at level $k$ (here $\ell = 4^{r-1}$ is the number of limbs of a full-width variable in [19]); `lbox_spec` shows that a level-$k$ variable $y$ with $y^{2^k} < 2^n$ fits. A level-$k$ chain is checked by the two-dimensional Schönhage–Strassen pipeline of [19] at the transform shape of level $k$, with the limb width and the Fermat modulus of the top level, so a level-$k$ variable is a prefix of the top-level limb

format and the levels need no glue. Well-formedness (`LDesc.WFB`) asks that the operands of a product have level at least that of its chain, that comparisons be between variables of one level or against constants, and that variables enter chains only as products with the constant 1. In `primeDescL`, $m, a, b$ have level 0, $d_k$ has level $k$, $s_k, e_k$ have level $k-1$, and the chains $d_k \cdot d_k = s_k$ and $s_k \cdot e_k = d_{k-1}$ have level $k-1$.

Every object (variable, product, chain, comparison) has its own family of wires and vertices, so the cost is a sum over objects of a function of the level. The number of limbs is geometric, $\sum_{k<K} \ell_k \le 4\ell + K$ (`sum_kl_le`), and a description whose $i$-th object of each kind has level at least $i/c$ (`LDesc.Spread` $c$; `primeDescL_spread` gives $c = 8$) has $O(c)$ objects per level.

**Proposition 4.1** (`arithNet_value`, `arithSys_U_le`, `arithSys_is`)**.** *Let $g \ge 2$ and let $D$ be a leveled description that is well formed at size $n$, has spread $c \ge 1$ and at most $\lfloor \log_2(n+2) \rfloor + 3$ levels. The network of $D$ has value equal to the number of solutions of $D$ in its boxes, total $U \le 2^{19} c\,(n+2)$, and node load $P = 2^{22} c\big((n+2)/g + d(n)\big)$ with $d(n) = O(n^{3/4})$ the vertex degree of* [18].

For `primeDescL` at depth $L = \lfloor \log_2 n \rfloor + 1$ this gives $U \le 2^{22}(n+2)$. We call this version of the library ArithL. It is a re-parameterisation of ArithNet with one more coordinate, not a new idea; its proofs reuse the factoring and class-number pipelines unchanged, and its value lies in the bookkeeping that keeps $U$ linear for $\Theta(\log n)$ multiplications.

4.4. **The machine.** A counting package [18, §4] provides, for every input word $w$ and query word $\pi$, a network whose value is $Q(w, \pi)$, with $U$ and $gP$ linear in $|w|$ and polynomial-time readers. From any polynomial-time map $w \mapsto D(w)$ to well-formed leveled descriptions with spread $c$, `ArithLSpec.pkg_exists` builds a package with constant $2^{22}c$ and $Q(w, \pi) = \mathrm{count}(D(w))$. We use it twice, with $n = |w|$: `arithPi` reads $x$ from $w$ and takes $[A, B] = [1, x]$, and `arithPiInt` reads $A$ and $B$ from the two halves of `encodePair` $A$ $B$. Both take $L$ to be the bit length of $n$, so that $B < 2^n < 2^{2^L}$ (`primeL_spec`), and `primeDesc_count_parity` gives count mod $2 = \pi([A, B])$ mod 2.

The query specification `specPi` asks the single query $\pi = [\,]$ and outputs the binary word of the count modulo 2. The generic one-query machine `GR.queryM_TM` of [18, Thm. 4.1] turns a package and a polynomial-time specification with output of length $O(|w|)$ into a finite-alphabet Turing machine that outputs the specified answer within $\lfloor C \cdot 2^{C|w|/\sqrt{\log_2(|w|+2)}} \rfloor$ steps. Composing the two gives `piParity_TM_of` and `piIntervalParity_TM_of`, which the module `Final` restates as the theorems of Theorem 1.2.

## 5. What a reader must trust

A reader who wants to trust Theorem 1.2 must check the following definitions and nothing else of ours. The statements, verbatim from `Final` up to line breaks, are below; we render Unicode symbols as mathematics.

```
theorem piParity_TM_rate :
    ∃ (M : Turing.TM2ComputableAux Bool Bool) (C : ℕ), (∀ i, Finite (M.tm.Γ i)) ∧
      ∀ x : ℕ, GraphProblems.Outputs M (Computability.encodeNat x)
        (Computability.encodeNat (Nat.primeCounting x % 2))
        ⌊(C : ℝ) * 2 ^ ((C : ℝ) * (Computability.encodeNat x).length /
          Real.sqrt (Real.logb 2 ((Computability.encodeNat x).length + 2)))⌋₊
theorem piIntervalParity_TM_rate :
    ∃ (M : Turing.TM2ComputableAux Bool Bool) (C : ℕ), (∀ i, Finite (M.tm.Γ i)) ∧
      ∀ A B : ℕ, GraphProblems.Outputs M (encodePair A B)
        (Computability.encodeNat (((Finset.Icc A B).filter Nat.Prime).card % 2))
        ⌊(C : ℝ) * 2 ^ ((C : ℝ) * (encodePair A B).length /
          Real.sqrt (Real.logb 2 ((encodePair A B).length + 2)))⌋₊
```

The only definitions of ours they use are those of the pair encoding (module `ModP.Encode` of [18]):

```
def padR (n : ℕ) (l : List Bool) : List Bool := l ++ List.replicate (n - l.length) false
def encodePair (m a : ℕ) : List Bool :=
  padR (max (encodeNat m).length (encodeNat a).length) (encodeNat m) ++
    padR (max (encodeNat m).length (encodeNat a).length) (encodeNat a)
```

`Outputs` is the one-line definition of [A]'s statement file, which we also trust:

```
def Outputs (M : TM2ComputableAux Bool Bool) (x y : List Bool) (T : ℕ) : Prop :=
  Nonempty (TM2OutputsInTime M.tm (x.map M.inputAlphabet.invFun)
    (some (y.map M.outputAlphabet.invFun)) T)
```

Behind these are Mathlib's `TM2OutputsInTime`, `Computability.encodeNat` (little-endian binary, the empty word for 0), `Nat.primeCounting`, `Nat.Prime`, `Finset.Icc`, `Real.logb` and `Real.sqrt`, from Mathlib `0df444a`. The definitions `chainCount` and `tauLt` of Section 3 do not occur in the statements; they are part of the proof.

## 6. What the parity does not give

In this section we say plainly what does not follow: a deterministic prime finder, the exact value of $\pi(x)$, and anything practical (the last is in Section 9).

6.1. **Prime finding.** Bisection finds a prime from any interval with an odd number of primes: one of its halves again has an odd number, and after $O(n)$ queries the interval is a single prime. So a fast parity oracle gives a fast prime finder as soon as one knows *some* interval of odd parity. Tao, Croot and Helfgott make the same remark and add that they "do not know of a method to quickly and deterministically locate an interval with an odd number of primes" [16, §1]. Our oracle is faster, but it does not change this. The parity of $\pi$ on $[x, 2x]$ may be even, and nothing we know about the primes forces an odd count on any explicit family of intervals of small size. The following statement makes the gap precise in an oracle model; it is proved on paper only.

**Proposition 6.1** (Adversary; on paper)**.** *Let $H \geq 4$. Consider deterministic algorithms that learn about an unknown set $S \subseteq [x, 2x]$ only by adaptive queries of two kinds, the parity of $|S \cap I|$ for an interval $I$, and membership $y \in S$, and that are promised only that $S$ meets every interval of $H$ consecutive integers in $[x, 2x]$. Every such algorithm that outputs an element of $S$ makes at least $(H-3)/2$ queries on some admissible $S$.*

*Proof.* Answer "even" to every parity query and "no" to every membership query, and suppose the algorithm stops after $q < (H-3)/2$ queries, $q_{\mathrm{p}}$ of parity and $q_{\mathrm{m}}$ of membership, with output $y$. The endpoints of the queried intervals cut $[x, 2x]$ into at most $2q_{\mathrm{p}}+1$ cells, and every queried interval meets $[x, 2x]$ in a union of cells. Let $S$ consist of all integers of $[x, 2x]$ except the membership-queried points and $y$; then, in each cell where $S$ has an odd number of points, remove one more point of $S$. Every queried interval now meets $S$ in an even number of points, every membership answer is correct, and $y \notin S$. At most $q_{\mathrm{m}} + 1 + (2q_{\mathrm{p}} + 1) \leq 2q + 2 < H$ integers of $[x, 2x]$ are missing from $S$, so $S$ meets every window of $H$ consecutive integers. So $S$ is admissible, consistent with all answers, and the output is wrong. □

With $H = x^{0.525}$, the gap bound of Baker, Harman and Pintz [3], the matching strategy (test the integers from $x$ upwards by a polynomial-time primality test [1]) takes $x^{0.525+o(1)}$ steps, worse than the $x^{1/2+o(1)}$ of [7]. The proposition does not say that parities cannot find primes; it says that a proof that they do must use more about the primes than a gap bound, and we know of no such argument. For residues modulo $M$ (Theorem 6.3) the same argument, removing up to $M-1$ points per cell, gives about $H/(2M)$ queries.

6.2. **Exact counts, and the iterated-product barrier.** The exact $\pi([A, B])$ would give prime finding at once. By this route it needs an identity that isolates $\omega(m) = 1$ exactly, not modulo a small number. We argue informally, as in our working notes [20, `research/prime-finding.md`, §4]: on squarefree $m$, the weights built from $O(1)$ multiplications of divisors are combinations of $t^{\omega(m)}$ for small $t$; the indicator of $\omega = 1$ on $0 \leq \omega \leq n/\log n$ needs $\Omega(n/\log n)$ such terms, unless one works modulo $t^2$, which yields only residues. So exact counting asks for the $k$-fold divisor function with $k = \Theta(n/\log n)$: a verifier of $m = d_1 \cdots d_k$ with $\sum_i \mathrm{bits}(d_i) = n$. The product tree verifies it with $U = \Theta(n \log k) = \Theta(n \log n)$, and then Theorem 2.1 gives $2^{\Theta(n\sqrt{\log n})}$, which is superpolynomial in $x$. A verifier with $U = o(n\sqrt{\log n})$ would give the exact $\pi(x)$ in $x^{o(1)}$; we do not know one.

In the restricted model of product trees we can prove that no tree shape helps. A *tree verifier* multiplies the factors along a binary tree $T$ with leaf sizes $\ell_i$; its cost is $\mathrm{cost}(T) = \sum_v \mathrm{wt}(v) = \sum_i \ell_i \,\mathrm{depth}(i)$, where $\mathrm{wt}(v)$ is the total size of the leaves below the internal node $v$.

**Theorem 6.2** (`BTree.cost_ge_entropy`, `BTree.cost_ge_equal`; machine-checked)**.** *For every binary tree with positive leaf weights $\ell_i$ and $L = \sum_i \ell_i$, $\mathrm{cost}(T) \geq \sum_i \ell_i \log_2(L/\ell_i)$. If all $k$ leaves have weight $w$, then $\mathrm{cost}(T) \geq kw \log_2 k = n \log_2 k$ for $n = kw$.*

The proof is an induction whose step is the bound 1 on binary entropy (`Real.binEntropy_le_log_two`). The link to networks is on paper: in a tree verifier the subtree of $v$ is separated from the rest by a cut across which the product indicator has rank $2^{\mathrm{wt}(v)}$ (`rank_prodMat_pow`, Lean-checked), so by the cut-rank lemma (`rank_le_two_pow_of_eq_sum_mul`, module `Barriers.RankBounds` of [20]) the cut carries at least $\mathrm{wt}(v)$ bits, and summing over $v$ gives $U \geq \mathrm{cost}(T)$. A lower bound for *all* networks is out of reach: the engine simulates any Boolean circuit of size $O(n^{1+1/40})$ with $U$ of the order of its size, so $U = \omega(n)$ for every verifier of $m = \prod_i d_i$ would be a superlinear circuit lower bound for an explicit function, which is open.

*Remark* 6.3 (Residues modulo $k$; on paper). Modulo $k$, $\mu(d) \equiv \tau_{k-1}(d)[d \text{ squarefree}]$, since $\tau_{k-1}(d) = (k-1)^{\omega(d)}$ on squarefree $d$. Iterating as in Theorem 3.1 gives a $\tau_{k-1}$-weighted chain $W_k(m) \equiv [m \text{ squarefree}] \pmod k$, and with $W = W_{k^2}$ one gets $\sum_{m\in I} \tau_k(m)W(m) \equiv [1 \in I] + k\,\pi(I) \pmod{k^2}$, hence $\pi(I) \bmod k$. We checked this by `#eval` for $k = 2, 3$ on all intervals in $[1, 100]$. The verifier needs $(k^2-1)$-fold products, so its cut grows like $n \log k$; we have not built it and claim no running time. Residues modulo small $M$ do not escape Theorem 6.1.

## 7. The Lean development

In this section we list the modules and state the trusted base. The development is in the public repository [20]: directories `lean/EthFollowup/Primes` (pure Mathlib; the files compile with Lean v4.33.1 and v4.34.1) and `lean-bridge-anthropic/bridge/EthFollowup/Primes` (on top of [A]'s development; the three pure files are linked into it). Table 2 lists the modules.

| module | content | lines |
|---|---|---|
| `ChainParity` | chain lemma, one-count identity, $\pi(x)$ as a card | 249 |
| `ArithLDesc` | leveled descriptions and their count | 124 |
| `PrimeDesc` | the description `primeDescL`, its count, well-formedness, spread | 589 |
| `ArithLShape`, `ArithLParams` | level shapes and boxes; wires, vertices, checks, canonical run | 1104 |
| `ArithLNetDefs`, `ArithLNetParsimony` | the label system and its value | 643 |
| `ArithLNetDeg`, `ArithLNetCount` | degrees; $U$ and $P$ (geometric sums) | 779 |
| `ArithLProcs`, `ArithLPolyChk`, `ArithLPoly` | procedures, checks on codes, polynomial time | 1425 |
| `ArithLPackage` | counting packages from leveled descriptions | 250 |
| `PrimePoly`, `SpecPrime`, `PrimePkgs` | readers, the one-query specification, the two packages | 279 |
| `Final` | the two final theorems | 46 |
| total | | 5488 |
| `IterProdTree`, `IterProdCutRank` | Theorem 6.2; rank of the product indicator (directory `Barriers`; not imported by `Final`) | 142, 148 |
| `Check`, `LemmaA` (in `Scratch/Primes`) | numerical checks by `#eval`, not part of the proof | 66, 253 |

Table 2. The modules, with line counts (`wc -l`). The first three are pure Mathlib. The development reuses, unchanged, the factoring modules of [19], the ArithNet, package and machine modules of [18] (`ArithDesc`, `ArithParams`, `GRate`, `ArithPkgs`, `Encode`, ...) and the bridge modules of [17].

Trusted base and axioms. `#print axioms` reports `propext`, `Classical.choice` and `Quot.sound` for the two theorems of Theorem 1.2, for `ArithLSpec.pkg_exists`, and for the two theorems of Theorem 6.2. No module contains `sorry`, a new axiom or `native_decide`. A fresh `leanchecker` replay of the dependency closure of `EthFollowup.Primes.Final`, the module holding the two final theorems, passed on 4 November 2026. What must be trusted is the Lean kernel, the Mathlib definitions named in Section 5, our definitions quoted there, and [A]'s definition of `Outputs`. [A]'s development, Mathlib's proofs and our other modules are checked by the kernel; they are part of the checked closure, not of the trusted base.

One design point. Polynomial time must hold on *every* context the reader may be given, not only on the contexts it produces. The stored levels of a description are therefore clamped to the number of variables before the cell layout uses them (`ACtx.L` in `ArithLProcs`); otherwise a context with a large binary level would make the table procedure exponential.

## 8. Related work

We organise related work by idea: computing $\pi(x)$ exactly, parities on short intervals, deterministic prime finding, and the counting engines with their formal proofs.

Computing $\pi(x)$ exactly. The combinatorial method of Meissel and Lehmer, as extended by Lagarias, Miller and Odlyzko [6], computes $\pi(x)$ in $O(x^{2/3+\varepsilon})$ steps, and Deléglise and Rivat [4] save a factor $\log x$. The analytic method of Lagarias and Odlyzko [7] takes $x^{1/2+o(1)}$ steps. Both give the parity as a by-product. In contrast, we compute only the parity, but in $x^{o(1)}$ steps.

Parities on short intervals. Tao, Croot and Helfgott [16] prove that one can decide whether an interval $[a,b] \subseteq [N,2N]$ of length at most $N^{1/2+c}$ contains an odd number of primes in time $O(N^{1/2-c+o(1)})$ (their Theorem 1.2), from the identity $\sum_{d^2\mid m}\mu(d)\tau(m/d^2) = 2^{\omega(m)}$ and a fast divisor sum; they also treat the parity of primes in progressions to small moduli. Lelechenko [9] gives an explicit exponent $O(x^{\max(c,7/15)+\varepsilon})$ for the same intervals (his Theorem 2). Sladkey [14] computes the divisor summatory function in $O(x^{1/3})$ steps and mentions the parity of $\pi$ among its applications, citing a manuscript of Oliveira e Silva in preparation. Closest to our work is [16], whose reduction of the parity to divisor counts we share. In contrast, we remove the Möbius sum by Theorem 3.1 instead of evaluating it, which is what allows intervals of any length.

Deterministic prime finding. Polymath4 [15] asked for a deterministic algorithm that finds a $k$-bit prime in time polynomial in $k$, or at least $\exp(o(k))$; the best unconditional bound remains $x^{1/2+o(1)}$, by computing $\pi$ analytically and bisecting [7, 16]. Scanning up from $x$ with a primality test [1] and the gap bound of [3] gives $x^{0.525+o(1)}$. In contrast, we give no prime finder, and Section 6 explains why the parity alone does not yield one.

Counting engines and formal proofs. OpenAI [11] and [A] [2] claim deterministic subexponential #SAT algorithms, with Lean developments, under review. Rudich and Rousseau [12] factored in $N^{o(1)}$ with them, Shunia [13] proved the rate $2^{O(n/\sqrt{\log n})}$ on paper, and [5] and our factoring note [19] proved it in Lean; our class-number note [18] built the network library ArithNet, which we extend here by levels. We use Lean 4 [8] and Mathlib [10]. As of 4 November 2026 we found no derivation of a parity result for $\pi$ from either ETH refutation. We searched the web for the parity of $\pi$ together with "ETH", "#SAT" and "2026"; searched GitHub repositories and code for prime-counting parity (the only hits were our own files); and checked the follow-ups known to us [12, 13, 5], which treat factoring. Hexagon's listing is behind a login, so this negative is weak. In contrast to our earlier notes, the number-theoretic content here is the chain lemma and the observation that working modulo 2 bypasses the Möbius barrier; the rest is the network bookkeeping.

## 9. Limitations

- *The result stands or falls with* [A]*'s development.* The proof rests on Anthropic's Lean development of its claimed refutation of the Exponential Time Hypothesis, which is under review. Lean checks that development as part of our proof, and our statements use only Mathlib, our quoted definitions and [A]'s `Outputs`. We do need the development, built from source at the pinned commit, to be sound, and the toolchain to be sound.
- *One bit only.* The machines output $\pi(x) \bmod 2$ and $\pi([A,B]) \bmod 2$, not $\pi(x)$, and not a prime. Residues modulo $k > 2$ are on paper only (Theorem 6.3).
- *No prime finding.* The parity may be even on every interval one queries; Theorem 6.1 is an oracle-model statement on paper, and the barrier of Section 6.2 is proved only for product trees.
- *The algorithms are galactic.* [A]'s levels start at $K_0 = 2^{x_0}$ with $x_0 \geq 18$, the level needs $K^{40} \lesssim n$, and the constants ($2^{22}\cdot 8$ for the package, and $C$) are astronomical. The machines are of no practical use and beat $x^{1/2}$ only for astronomically large $x$.
- *The machines use exponential space.*
- *No human has checked the work.* The Lean code and this text were written with AI assistance. No human expert has yet checked that Section 5 lists everything a reader must trust.

## 10. Conclusion

We have shown that one bit of $\pi$ on any interval is a single count of divisibility witnesses whose sizes halve along a chain, and that the counting engine evaluates it in $x^{o(1)}$ steps once the network pays for the witnesses at their true sizes. The result is a step towards, not a solution of, deterministic prime finding. An interesting direction is whether the iterated product $m = d_1 \cdots d_k$ with $k = \Theta(n/\log n)$ has a parsimonious verifier with $o(n\sqrt{\log n})$ shared bits; a positive answer would give the exact $\pi(x)$, and hence a prime in $[x, 2x]$, in time $x^{o(1)}$, while a negative answer for all networks would imply a circuit lower bound that is out of reach today.

## Acknowledgements

This work was done with AI assistance (Claude Code, Anthropic): the chain lemma, the Lean development, the numerical checks, the literature and citation checks and the text were produced in sessions with Claude Code. The Lean theorems are checked by the Lean kernel; everything else awaits human expert review. Responsibility for any errors rests with the author. This note continues the project of [17, 19, 18].

## References

[1] M. Agrawal, N. Kayal, and N. Saxena, *PRIMES is in P*, Ann. of Math. (2) **160** (2004), no. 2, 781–793.

[2] Claude (Anthropic), *A $2^{o(n)}$ time algorithm for 3-SAT*, preprint dated 26 October 2026. Paper and Lean formalisation in the repository https://github.com/anthropics/formal-math, folder `eth`; we use commit `16a5ce41304fcdaea4e0be7c132e1c361dc26da5`.

[3] R. C. Baker, G. Harman, and J. Pintz, *The difference between consecutive primes, II*, Proc. London Math. Soc. (3) **83** (2001), no. 3, 532–562.

[4] M. Deléglise and J. Rivat, *Computing $\pi(x)$: the Meissel, Lehmer, Lagarias, Miller, Odlyzko method*, Math. Comp. **65** (1996), no. 213, 235–245.

[5] 1337isnot1337 (GitHub user) and contributors, *integer-factoring: deterministic integer factoring bounds*, public repository https://github.com/1337isnot1337/integer-factoring; Lean formalisation in `formal-appendix/` (pull request #20, merged 30 October 2026).

[6] J. C. Lagarias, V. S. Miller, and A. M. Odlyzko, *Computing $\pi(x)$: the Meissel–Lehmer method*, Math. Comp. **44** (1985), no. 170, 537–560.

[7] J. C. Lagarias and A. M. Odlyzko, *Computing $\pi(x)$: an analytic method*, J. Algorithms **8** (1987), no. 2, 173–191.

[8] L. de Moura and S. Ullrich, *The Lean 4 theorem prover and programming language*, Automated Deduction – CADE 28, Lecture Notes in Comput. Sci., vol. 12699, Springer, 2021, pp. 625–635.

[9] A. V. Lelechenko, *Parity of the number of primes in a given interval and algorithms of the sublinear summation*, Visn. Odes. Nats. Univ., Mat. Mekh. **18** (2013), no. 1(17), 104–114; arXiv:1305.1639.

[10] The mathlib Community, *The Lean mathematical library*, Proc. 9th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP 2020), 2020, pp. 367–381.

[11] OpenAI, *The Exponential Time Hypothesis is false*, preprint dated 17 October 2026, revised 25 October 2026. Paper and Lean formalisation in the repository https://github.com/openai/math.

[12] I. Rudich and L.-M. Rousseau, *Deterministic factoring in time $N^{o(1)}$*, Hexagon preprint hexagon:2610.00600v1, published 29 October 2026, https://hexagonmath.org/2610.00600v1.

[13] J. Shunia, *An improved subexponential bound for deterministic integer factorization*, Hexagon preprint hexagon:2610.00711v1, published 2 November 2026, https://hexagonmath.org/2610.00711v1.

[14] R. Sladkey, *A successive approximation algorithm for computing the divisor summatory function*, arXiv:1206.3369, June 2012.

[15] T. Tao, *Deterministic way to find primes – discussion thread*, The polymath blog, 28 July 2009, https://polymathprojects.org/2009/07/28/deterministic-way-to-find-primes-discussion-thread/.

[16] T. Tao, E. Croot III, and H. Helfgott, *Deterministic methods to find primes*, Math. Comp. **81** (2012), no. 278, 1233–1246; arXiv:1009.3956 (as D. H. J. Polymath).

[17] V. Tulsyan, *Counting Hamiltonian cycles in sparse graphs in subexponential time, assuming the new #SAT algorithms*, Hexagon preprint hexagon:2610.00672v1, October 2026; source and Lean development at https://github.com/xennygrimmato/eth-hamiltonicity.

[18] V. Tulsyan, *Class numbers, square roots and quadratic non-residues in time $2^{O(n/\sqrt{\log n})}$, assuming the new #SAT algorithms*, preprint, November 2026; source in the directory `classnumber-note/` of [20].

[19] V. Tulsyan, *An independent machine-checked proof of deterministic factoring in time $2^{O(n/\sqrt{\log n})}$, via the new #SAT algorithms*, preprint, November 2026; source in the directory `factoring-note/` of [20].

[20] V. Tulsyan, *eth-hamiltonicity: Lean 4 formalisation and research notes*, public repository https://github.com/xennygrimmato/eth-hamiltonicity, 2026; Lean sources as of commit `f5aad1a`.