OpenAI's claimed refutation of ETH: what it says, and how we checked it

READING LEVEL EVERYONE · EXPERTS

Active · Preliminary

proved in writing 2 · Lean-checked 1

CHECK IT YOURSELF

The check guide for this book · How to check a result yourself

---

What you can rerun: Everything is from-files, from OpenAI's public repository at the pinned commit; the build took about 9.5 hours and needs about 16 GB of memory. Each result page gives its own rerun value and 5-minute check.

What a match shows: a match tells you that the published files really produce the published numbers. It doesn't tell you that the question was the right one to ask, or that a written argument around the numbers is complete. Each recipe says plainly what its steps show and what they don't. For a result whose evidence is logs (logs-only), a match checks the bookkeeping of the logs, not the search that wrote them.

Cite: Sonny (2026). OpenAI's claimed refutation of ETH: what it says, and how we checked it (preliminary). Go check yourself. https://gocheckyourself.org/read/eth-audit/everyone/

1. Start here
2. OpenAI's claimed refutation of ETH, in plain words
3. Should I worry? Questions people are asking
4. What the Lean statement says, line by line
5. How we checked it
6. What it means, and what it does not
7. What is still open
8. Statement
9. Theorems in this book

10. Check it yourself: OpenAI's claimed refutation of ETH
11. Trust and verification
12. People and works
13. Files and rights

1 Start here

What this book shows

- What I checked. OpenAI's claimed refutation of ETH, the Exponential Time Hypothesis: whether its published Lean statement says what ETH says, and how far Lean accepts the proof. (Lean is a language for writing mathematics so exactly that a small program, its kernel, can check every step.)
- The statement matches the standard reading of ETH. With $k = 3$, a correct proof of it would contradict ETH. Checked (re-derived in writing): 12 of the 14 rows of the statement table.
- The proof builds, on the standard axioms only. All 3,091 modules build under Lean 4.34.1 with Mathlib (the Lean community's library of formal mathematics), and the final theorem depends only on the standard three axioms (starting assumptions Lean accepts without proof). Checked by machine. A kernel replay of each module (every stored proof step of the module is sent through the kernel a second time) has finished: all 3,091 accepted, none rejected.

- Why it matters. Hundreds of papers use ETH to explain why a problem probably cannot be solved much faster; if the claim holds, those explanations lose their support. The manuscript's mathematics has not been checked line by line, and five items are open (chapter 4).
- What our tools showed. The same list of checks (statement, build, axioms, replay) can be used on other "checked in Lean" claims. The statement and our two small facts about it are in the theorem library.

The question

OpenAI says one of its AI models has refuted ETH. I am not an author of that work. I read the Lean theorem, rebuilt the proof with public tools, and wrote down what I could and could not check.

Why this book exists. The subject of this book is ETH. When OpenAI released its proof, we checked it carefully, in the spirit of shared knowledge. OpenAI also did something we want to see more of: it released its proof in a form anyone can check. We applaud that. We think releases like this would be stronger still with agreed checks before publication, like the responsible release threshold we propose, and we are encouraged by the open discussion that has followed.

What does this actually mean?

- Not Ethereum. "ETH" here is the Exponential Time Hypothesis, a conjecture in computer science about how long certain puzzles take to solve. It has nothing to do with the cryptocurrency.
- The speed-up is asymptotic. The claimed bound describes how running time grows as puzzles get larger, and it comes without published constants, so no practical speed-up has been shown at any size. OpenAI's own post says turning such advances into practical gains "will require significant further research".

- The claimed running-time bound. ETH implies that the polynomial-time question has a negative answer (not every problem whose answers can be checked quickly can be solved quickly), but refuting ETH says nothing about the polynomial-time question itself. The claimed algorithm is still not polynomial time (running time that grows like a fixed power of the input size). Problems that convert to 3-SAT (a yes-or-no puzzle: can a list of conditions all hold at once?) with only a linear increase in size would inherit a speed-up if the claim holds.
- Encryption. As far as we know, mainstream encryption (AES, RSA, elliptic curves) does not rest on ETH, so this indicates no weakening of it. Chapter 0 says what we base this on.

- Research. If the claimed refutation holds, results that assumed ETH lose that support. Results resting on other hypotheses are not touched.
- Status. It is a claimed refutation. The Lean proof builds with only the standard axioms on our machine, and the review group's examination is ongoing.

The question this book answers: what does the Lean statement OpenAI published say, does it match what ETH says, and how far have we been able to check that the proof is accepted by Lean?

Why it matters

A computer-science puzzle called k-SAT asks: given a list of conditions ("clauses"), each an "or" of at most k yes-or-no variables, such as "x1 or not x2 or x5", can every variable be set to true or false so that all the conditions hold? (A formula written this way is a k-CNF.) With n variables there are 2^n settings, and trying them all (brute force) doubles in cost with each added variable. For decades the best algorithms for 3-SAT ($k = 3$) still took time that grows exponentially in n.

The Exponential Time Hypothesis (ETH), proposed by Impagliazzo, Paturi and Zane around 1999 to 2001, says this is unavoidable: no algorithm solves 3-SAT in time $2^{o(n)}$, meaning time growing more slowly than $2^{c \cdot n}$ for every fixed c above zero. Put plainly, 3-SAT needs some fixed exponential amount of time, however clever the algorithm. A stronger form, SETH, says that for every ε above zero there is some clause width k for which k-SAT cannot be solved in time $2^{(1-\varepsilon)n}$.

ETH matters because hundreds of papers, in graph theory, computational biology, databases and computational geometry, use it as a working assumption to explain why a problem probably cannot be solved much faster than it is now. If ETH fails, those explanations lose their support.

What answering it would help with

- Readers worried by headlines may use the plain answers in chapter 0 to see what changes and what does not.
- Researchers who relied on ETH may use the statement table in chapter 1 to see exactly which version of the claim has been formalised.
- Anyone who meets a "checked in Lean" claim may use the same list of checks (statement, build, axioms, replay) on other results.

What our research shows

What this is

A BOOK ON OPENAI'S CLAIMED REFUTATION OF ETH (26 OCTOBER 2026): WHAT ITS PUBLISHED LEAN STATEMENT SAYS, LINE BY LINE, AND HOW FAR WE HAVE CHECKED IT WITH PUBLIC TOOLS (SONNY, 2026).

Checked by

THE PROOF WAS BUILT WITH LEAN 4.34.1 AND THE PINNED MATHLIB VERSION (ONE EXACT VERSION, LISTED IN THE CHECK GUIDE); ITS AXIOMS WERE PRINTED; A KERNEL REPLAY OF ITS 3,091 MODULES HAS FINISHED, WITH ALL 3,091 ACCEPTED; A SECOND READING WITHIN THIS PROJECT RE-DERIVED THE STATEMENT-MATCHING POINTS OR CITED THEIR SOURCES. THE MANUSCRIPT'S MATHEMATICS HAS NOT BEEN CHECKED LINE BY LINE.

Labels used throughout: checked means re-derived in writing; checked by machine means a program confirmed it; cited means we opened a source; open means not settled. (A module is one Lean file of the proof.)

Why we audit published results: About this research

<!--BOX-->

What's in this book

Status	Count
Checked by machine: the whole proof builds in Lean, and its axioms are the standard three and no others	1 (the final theorem)
Checked by machine: kernel replay of each module	3,091 of 3,091 accepted, none rejected
Checked (re-derived in writing): rows of the statement table in chapter 1	12 of the 14 rows
Cited: sources we opened for the definitions of ETH and for the known proof-system lower bounds	see chapter 3 and the credits
Open	6 (listed in chapter 4)

Item	Count
Chapters	5 (numbered 0 to 4)
Statement files	1
Written check recipes	1 (the check guide)

<!--/BOX-->

What OpenAI claims

OpenAI reports an algorithm that counts the satisfying settings of any k-CNF formula (k fixed, 2 or more) with n variables exactly, deterministically and in polynomial memory, in time $2^{O(n (\log \log n)^2 / \sqrt{(\log n)})}$ times a polynomial, which is $2^{o(n)}$, with a proof formalised in Lean. The manuscript, "The Exponential Time Hypothesis is False" (17 October 2026, revised 25 October), claims sharper exponents, which are not in the Lean statement. A review group led by Russell Impagliazzo, Ramamohan Paturi and Ryan Williams is verifying and explaining the result, and OpenAI has made the proof public at github.com/openai/math (we used the pinned commit listed in the check guide).

What we checked and found

- The statement matches the standard reading of ETH. For each k, the Lean theorem gives one fixed machine (a Turing machine, the textbook model of a computer) that, on every k-SAT formula, halts with the

exact count and the yes-or-no answer, within the stated time and using only polynomial space ("polynomial" means growing like a fixed power of the input size). The machine is chosen before the formula, it is deterministic (no coin flips), and the count is exact. With $k = 3$, a correct proof of it would contradict ETH. Checked; chapter 1.

- The whole proof builds. All 3,091 modules in the proof build under Lean 4.34.1, with Mathlib, the Lean community's library of formal mathematics. Checked by machine.
- The axioms are the standard three and no others. An axiom is a starting assumption Lean accepts without proof. The final theorem depends on the three standard axioms that Mathlib normally uses (named in the check guide), and on no placeholder axiom. Checked by machine.

- Kernel replay. A replay with the tool leanchecker sends every stored proof step of a module back through the kernel to catch anything that slipped past the build. The replay of all 3,091 modules has finished: every one was accepted, none rejected. It replays each module's own steps and does not by itself cover Mathlib or core Lean. Checked by machine.
- The second reader. A second reading within this project re-derived the statement-matching points in writing (checked) and cited the sources for the definitions of ETH and for what the comparator tool guarantees (cited). See chapter 2.

What it means for the polynomial-time question

The polynomial-time question asks whether every problem whose answer can be checked quickly can also be solved quickly. ETH implies a negative answer to the polynomial-time question, but refuting ETH does not imply a positive answer to the polynomial-time question. The claimed algorithm is not polynomial time, and the manuscript derives nothing about the polynomial-time question. The related question NP vs coNP, whether "no solution exists" can always be backed by a short proof, is not settled by it either. If the claim holds, problems that convert to 3-SAT with only a linear increase in size,

such as 3-Colouring, Independent Set and Vertex Cover, would also get 2^{o(N)} algorithms. See chapter 3.

What is still open

Five items, listed in chapter 4: a second, independently written kernel with the unpatched checker (one outside run with a patched build accepted; another with the unpatched checker did not finish); the manuscript's 67 pages of mathematics, not checked line by line; the manuscript's other claims (its sharper exponents, corollaries and appendix on local maps), not re-derived; whether the algorithm's correctness gives short proofs in a strong proof system (a fixed set of rules for writing proofs), a conjecture, untested; and any indirect consequence for the polynomial-time question, which nobody can rule out. The comparator tool, which compares the published challenge statement with the proved theorem, has been run by others and accepts; we have not run it.

Check it yourself

Everything needed is public: OpenAI's repository at the pinned commit, Lean, Mathlib at the pinned version, and Lean's own tools. The check guide (VERIFY.md) gives the steps: build the module, print the axioms of the final theorem (expected: the standard three), and replay modules with leanchecker. You need about 16 GB of free memory and many hours. In outline:

```
lake build OAI.Algorithms.SubexponentialSAT.Main
lake env lean Axioms.lean
lake env leanchecker OAI.Algorithms.SubexponentialSAT.Algorithm
```

The last command replays one module, as an example. To replay all 3,091, run the same replay command once for every module that has a compiled file in the project's build folder (VERIFY.md gives the exact command and folder).

The chapters

- Should I worry? Questions people are asking: calm answers on encryption, passwords, AI, the polynomial-time question, and whether a computer proof can be trusted.
- What the Lean statement says, line by line: the statement against the standard form of ETH.
- How we checked it: the build, the axioms, the replay and the second reader, with counts.
- What it means, and what it does not: the algorithm in plain words, other problems, proof systems, the polynomial-time question.
- What is still open: what remains to check.

The statement, the check guide and the credits page complete the book.

We stand on the shoulders of giants: OpenAI, the review group, the originators of ETH and the Lean community are credited on the credits page.

Where to go next

- Does this change anything for passwords or payments? See Should I worry?.
- If a computer accepted the proof, why is the review still going? See How we checked it and What is still open.

Go check yourself.

Have a question about this? Ask it

2 OpenAI's claimed refutation of ETH, in plain words

OpenAI's claimed refutation of ETH, in plain words

The question: OpenAI says one of its AI models has refuted ETH, the Exponential Time Hypothesis. What does its published Lean statement (Lean is a language for writing mathematics so exactly that a small program can check every step) say, and what does the claim mean for everyone else?

The answer in brief: the statement matches the standard reading of ETH, and the proof builds in Lean. As far as we know, it does not break passwords, payments or encryption, and it does not settle the polynomial-time question (whether every problem whose answer can be checked quickly can also be solved quickly).

What ETH is

In plain words. A computer-science puzzle called 3-SAT asks whether a list of yes-or-no conditions can all hold at once. With n variables there are 2^n settings, and trying them all doubles in cost with each added variable. ETH, proposed by Impagliazzo, Paturi and Zane around 1999 to 2001, says this is unavoidable: 3-SAT needs some fixed exponential amount of time, however clever the algorithm. Hundreds of papers use ETH to explain why a problem probably cannot be solved much faster. "ETH" here has nothing to do with the cryptocurrency Ethereum.

What I checked

In plain words. I am not an author of OpenAI's work. I read the Lean theorem, rebuilt the proof with public tools, and wrote down what I could and could not check. The statement matches the standard reading of ETH: with $k = 3$, a correct proof of it would contradict ETH. Checked (re-derived in writing): 12 of the 14 rows of the statement table. The whole proof builds in Lean. Checked by machine. The manuscript's mathematics has not been checked line by line; the open items are listed in the expert version.

Should I worry?

In plain words. As far as we know, no. Passwords, payments and mainstream encryption (AES, RSA, elliptic curves) do not rest on ETH. The claimed speed-up describes how running time grows as puzzles get larger, and it comes without published constants, so no practical speed-up has been shown at any size. It does not settle the polynomial-time question: ETH implies a negative answer to the polynomial-time question, but refuting ETH says

nothing about the polynomial-time question itself. If the claimed refutation holds, results that assumed ETH lose that support.

Why it is worth checking

OpenAI released its proof in a form anyone can check. We applaud that. It is a claimed refutation, and this book is preliminary, not yet refereed.

Experts: the files and checks are here: check guide.

3 Should I worry? Questions people are asking

Should I worry? Questions people are asking

> The short answer. ETH here means the Exponential Time Hypothesis. As far as we know it does not break passwords, payments or encryption. Experts are still reviewing OpenAI's claim, and it matters mainly to researchers who used the Exponential Time Hypothesis as an assumption.

The question

OpenAI says one of its AI models has refuted ETH. I am not an author of that work. I read the Lean theorem, rebuilt the proof with public tools, and wrote down what I could and could not check. Many people have asked what the claim means for them. This chapter answers the common questions in short form.

Why it matters

Calm answers need to be accurate, so each answer here says what it rests on.

What answering it would help with

A reader may find which worries the claim touches (research that used ETH as an assumption) and which it does not (everyday computing, passwords and payments, as far as we know).

What our research shows

OpenAI's result, if it holds, is a remarkable achievement: an algorithm for a famous puzzle that many researchers had believed could not exist. No practical speed-up has been shown. The short answers follow.

Is this about Ethereum?

No. "ETH" in this book is the Exponential Time Hypothesis, a conjecture in computer science, proposed by Impagliazzo, Paturi and Zane around 1999 to 2001, about how much time a certain kind of logic puzzle (3-SAT) needs. The cryptocurrency Ethereum uses the same three letters and is unrelated. Ethereum's security rests on elliptic-curve signatures and hash functions (see the next question), not on this hypothesis.

Does this break my passwords, payments or encryption?

As far as we know, no. Here is what we base that on.

1. Passwords. Websites normally store a password only as a scrambled fingerprint (a hash), and your login travels over an encrypted connection. Neither depends on ETH, and nothing in the claim is a method for guessing a password or opening an encrypted connection.
2. What encryption rests on. RSA relies on the difficulty of factoring large numbers (Rivest, Shamir and Adleman, 1978). Elliptic-curve cryptography relies on the difficulty of the discrete logarithm problem on elliptic curves (Koblitz, 1987; Miller, 1985). AES, the standard cipher for encrypting data (NIST FIPS 197), is a fixed design whose security rests on decades of attempts to break it. Factoring already has algorithms faster than brute force (the number field sieve) and is not known to be as hard as 3-SAT, so ETH is not how its security is judged. ETH is also a statement about the worst case over a whole family of puzzles, while a cipher is one fixed function.

1. The size of the numbers. Guessing a 128-bit key by brute force takes 2^128 steps. To use the claimed algorithm against a cipher, the cipher would have to be written as a logic formula with thousands of variables. For 10,000 variables, an exponent of the form $N/\sqrt{(\ln N)}$ is about 3,300,

far above 128, even with the constant taken as 1. It would help an attacker only if the unpublished constants were very small, and nobody has shown that. We ran no experiment.

1. One related claim. The manuscript's Appendix C.2 gives algorithms that find an input for a given output, and count the inputs, for maps in which every output bit depends on only a bounded number of input bits ("constant locality"), in time $2^{o(n)}$ in the number n of input bits. The manuscript says this applies to constant-locality versions of Goldreich's candidate one-way functions, a topic in theoretical cryptography, and that it rules out inversion hardness of the form $2^{\Omega(n)}$ for those families but not one-wayness against polynomial-time attackers. AES, RSA and elliptic-curve systems are not of this form, since their outputs depend on many input bits. We have not checked this appendix.

We did not audit any cryptographic system; that judgement belongs to cryptographers.

Did an AI find this, and should I worry about that?

OpenAI reports that one of its AI systems found the argument (on 16 October 2026) and that it was then formalised in Lean (on 17 October). We cannot confirm how it was found. It does not change what needs checking. A Lean proof is accepted or rejected by Lean's kernel step by step, whoever or whatever wrote it, and the mathematics is being reviewed by human experts. The practical worry is not that an AI wrote it but that such claims need careful human review, which is why the review group exists.

Will computers get much faster?

No change to your computer follows from this. The claim is about a method of solving one family of puzzles in fewer steps as the puzzles grow without limit. It does not make processors faster, and no program you use today gets quicker because of it. OpenAI's post says finding such algorithms and turning theoretical advances into practical improvements "will require significant further research".

Does this solve the polynomial-time question?

No. The polynomial-time question asks whether every problem whose answer can be checked quickly can also be solved quickly. ETH implies that P is different from NP, but the reverse does not hold: refuting ETH says nothing about the polynomial-time question itself. The claimed algorithm takes more than polynomial time, and the manuscript says its result does not give subexponential-time algorithms for general NP problems. See chapter 3.

Is math broken?

No. ETH is a hypothesis: a widely believed guess that nobody had proved. Guesses sometimes fail; that is how mathematics normally works. If the claimed refutation holds, a result of the form "if ETH is true, then this problem is hard" remains logically valid, but it would no longer tell us the problem is hard.

What happens to research that assumed ETH?

If the claimed refutation holds, it loses that support. Hundreds of papers in graph theory, computational biology, databases and computational geometry used ETH to explain why a problem probably cannot be solved much faster. Each such result will need to be re-examined to see whether another assumption covers it. The manuscript itself notes that some of the same lower bounds follow from weaker hypotheses that may still hold, for example the Orthogonal Vectors Conjecture, and that the Lean statement covers each fixed clause width, not formulas whose width grows with the number of variables. This is work for the research community over months and years.

Can I trust a computer-checked proof?

A computer check is strong, and it is also narrow. Lean's kernel checks that a proof proves the statement exactly as written in Lean, from the axioms in scope. It cannot check that the Lean statement says what an English sentence says. That is why this book reads the statement line by line (chapter 1): we found it matches the standard form of ETH. We also built the proof, printed its axioms (only the standard three) and replayed every module through the kernel (chapter 2). What we have not done is check the 67-page

paper by hand, run the comparator tool, or run a second kernel. The expert review covers the mathematics.

What happens next?

According to OpenAI's post of 26 October 2026, the work will be verified and explained by a review group led by Russell Impagliazzo, Ramamohan Paturi and Ryan Williams, whom the post describes as experts in algorithm design and complexity theory. We do not speak for them, and their conclusions are their own. OpenAI says it has agreed to their requests to publish the raw material and to support a conference on what the result means, and the group invites the community to help examine the claim. Until that review reports, the right description is a claimed refutation.

Check it yourself

The testable claims in this chapter are tested in VERIFY.md: the build, the axioms and the replay. Sources for statements about other people's work are on the credits page.

Where to go next

- What exactly did OpenAI prove? See What the Lean statement says, line by line.
- What would a reader need to check next? See What is still open.

4 What the Lean statement says, line by line

What the Lean statement says, line by line

The question

Does the theorem OpenAI proved in Lean say what the Exponential Time Hypothesis denies, or something narrower or different?

Why it matters

A Lean proof only proves the statement written in Lean. If that statement were weaker than ETH's standard form, the Lean proof would not refute ETH even if every step were accepted. So the statement has to be read against the usual definition.

What answering it would help with

A reader may use the table below to see which parts of the English claim sit in the Lean text, and which parts (such as the stronger forms of the hypothesis) are derived outside it.

What our research shows

The answer in one line: the Lean statement is a faithful and slightly stronger form of "k-SAT and counting its solutions can be done in time 2^{o(n)} times a polynomial, using polynomial space, for every fixed k from 2 up", with the weaker of the manuscript's two exponents. Checked (re-derived), from the statement as published at the pinned commit (listed in the check guide).

The statement in plain words

The theorem is the main theorem of OpenAI's proof. It is stated in the challenge file of github.com/openai/math and proved in the final module of the proof (exact names are in the statement file and the check guide). In words: pick any width k of 2 or more. Then there is one fixed Turing machine (a simple model computer with several tapes, each cell holding blank, 0 or 1; in this version each state looks at one tape and acts on that tape only, which is equivalent to the usual multitape machine up to a constant factor) and fixed constants such that, for every number n of variables and every k-CNF formula F on them, written in a standard encoding, the machine

1. halts;
2. leaves on one tape the exact number of satisfying settings, in binary (least significant bit at the start), using at most n + 1 bits;

3. leaves the yes-or-no answer to "is F satisfiable?" on another tape and in its halting state;
4. takes at most C · (L + 1)^d · 2^{A · n · (ln ln (n + 16))² / √(ln (n + 16))} steps, where L is the length of the encoded formula; and
5. at every moment of the run, has used at most S · (L + 1)^e cells in total, counting for each tape the width of the stretch its head has covered (the input tape counts as covered from the start), which is polynomial memory.

(The theorem file ends with the placeholder word "sorry". That is the normal shape of a challenge file: the statement with the proof left blank. The proof lives in the module named above, and the comparator tool is meant to match the two.)

The table

Each row is a question we asked of the statement, the answer, and its label. "Checked" means re-derived in writing by a second reading within this project; "checked by machine" means a program confirmed it; "cited" means a source was opened; "open" means not settled here.

Question	Answer	Label
Do the machine and constants depend on the formula?	No. The machine and every constant are chosen after k and before n and the formula, so one machine serves all sizes, with no hidden per-size table. The run is the machine's single deterministic run, and the number of steps is its exact length.	Checked
Is the length L of the encoded formula hiding an exponential?	No. For a fixed width k, a formula on n variables has at most $(2n+1)^k$ distinct clauses, and each literal costs at most $4n + 2$ bits, so L is at most $n + 2 + (2n+1)^k (8kn + 4k + 4)$, a polynomial of degree $k + 1$. So $(L + 1)^d$ is a polynomial and cannot contain an exponential factor. The encoding writes numbers in unary, which only makes the stated bound weaker than a binary encoding would.	Checked; checked by machine (a small replica of the encoder agreed on 4,000 random formulas)
Is the exponent really o(n)?	Yes. With $x = \ln(n + 16)$, the ratio of the exponent to n equals $(\ln x)^2 / \sqrt{x}$, which tends to 0. So for every δ above 0 the whole bound is at most $2^{\delta \cdot n}$ for large n, and finitely many small n cost only a constant.	Checked; checked by machine
Does the exponent behave at small n (logs of logs)?	Yes. Since 16 is larger than e^e, $\ln \ln (n + 16)$ stays above 1 for every n, so no junk values arise.	Checked
Does it refute ETH as defined?	Yes, if proved. Taking $k = 3$, it solves 3-SAT in time (polynomial in the input) $\cdot\ 2^{\delta n}$ for every δ above 0, which contradicts "ETH: 3-SAT needs 2^{cn} for some c above 0". ETH is sometimes stated in the number of clauses m instead; after dropping variables that appear in no clause, n is at most $k \cdot m$, so $2^{o(n)}$ is also $2^{o(m)}$.	Checked; the definitions of ETH are cited from Impagliazzo–Paturi (1999), Impagliazzo–Paturi–Zane (2001) and Calabro–Impagliazzo–Paturi (2010)
Does it refute counting ETH?	Yes, if proved. The statement is the exact count over all 2^n settings of the n declared variables, so it contradicts the version of ETH for counting solutions (Dell and others, 2014).	Checked
Does it refute SETH (the stronger Strong ETH)?	Yes, if proved, by a short derivation outside Lean: every fixed k gets $2^{n/2}$, which beats the $2^{(1-\varepsilon)n}$ lower bound SETH asks for at every k.	Checked (derivation); this is not a line of the Lean statement
NSETH, Gap-ETH, randomised versions?	NSETH: a deterministic $2^{o(n)}$ decision procedure for unsatisfiability is also a nondeterministic one. Gap-ETH says no $2^{\varepsilon n}$-time algorithm can tell a satisfiable 3-CNF from one in which every assignment violates a fixed fraction of the clauses; an exact $2^{o(n)}$ satisfiability test does that, so it refutes Gap-ETH. The randomised versions are refuted because a deterministic algorithm is a randomised one with no error. None of these is in the Lean text, and the definitions are taken from the manuscript.	Checked (one-line derivations); definitions cited from the manuscript

Question	Answer	Label
Does the machine model change anything?	No. A multitape machine with a fixed number of tapes can be run on a RAM (the usual computer model) with at most a small logarithmic factor extra, and on one tape with a square, and a square of 2^{o(n)} is still 2^{o(n)}.	Checked in the direction the refutation needs
Does counting also give the yes-or-no answer?	Yes. The statement puts the answer on a tape and in the halting state.	Checked
Does "canonically encoded" hide something?	No. The theorem covers inputs written in one sorted, duplicate-free form. Any ordinary input is converted to that form in polynomial time, which is outside the Lean statement and costs only a polynomial.	Checked
Is the memory bound honest?	Yes. It counts every cell any head has visited, including the whole input, at every moment of the run.	Checked
Does the exponent match the manuscript?	The Lean statement is narrower. It uses $n\,(\ln\ln n)^2 / \sqrt{(\ln n)}$; the manuscript's Theorem 1.1 has the sharper $n \ln\ln n / \sqrt{(\ln n)}$, and $n / \sqrt{(\ln n)}$ when the number of clauses is at most a fixed multiple of n. Both are o(n), so the refutation does not need the sharper ones.	Cited (the manuscript and the repository's page for this result)
Does the proof file define things differently from the challenge file?	We compared the 33 definitions in the challenge file with the proof's, as text: 25 are identical, and the other 8 differ only where a comment or section line was attached to the chunk; we read each of those by eye and the bodies agree. Equal text is not proof that the compiled definitions are equal, which is what the comparator tool would check.	Checked by machine (text comparison); the definition-level comparison is open

What the statement leaves out

The statement is about each fixed width k, not width growing with n; it gives no explicit constants; it says nothing about problems other than k-CNF counting and satisfiability (the manuscript's corollaries about colourings and matchings are not in Lean); and it does not say the algorithm is faster than brute force at any size anyone will meet. As an illustration only: if the exponent were exactly the Lean one, $n\,(\ln\ln n)^2 / \sqrt{(\ln n)}$, with every constant equal to 1, its ratio to n would peak at about 2.17 near $n = 10^{23.7}$ and fall below 1 only past about 10^{2390} variables. The real constants are not published and depend on k, and the manuscript's sharper exponents (its own claim, not in Lean) behave differently, so this is not a statement about where the algorithm beats brute force. It shows how o(n) works: an asymptotic

statement promises nothing at any particular size, and no practical speed-up has been shown.

Net verdict

This indicates that the Lean statement is what is needed to refute ETH and counting ETH, and that SETH, NSETH and Gap-ETH follow from it by short arguments. Whether the proof of the statement is correct is a separate question, covered in chapter 2.

Check it yourself

Download the challenge file from the pinned commit of github.com/openai/math and read the theorem near the end of it (line 194). The statement file in this book repeats it with commentary: statement/STATEMENT.md.

Where to go next

- How was the proof itself checked? See How we checked it.
- What could still be wrong with the definitions? See What is still open.

5 How we checked it

How we checked it

The question

Beyond reading the statement, how far can an outside reader check that OpenAI's Lean proof of it is accepted, and what has been done so far?

Why it matters

A Lean proof is only as trustworthy as the checks around it: that it builds, that it adds no hidden assumptions, and that Lean's kernel, the small program that checks every step, accepts each part. Each check catches a different kind of problem.

What answering it would help with

The same four checks (build, axioms, scan for escape hatches, kernel replay) may be applied to any large Lean proof a reader wants to understand.

What our research shows

The answer in one line: at the time of writing, the whole proof builds, its axioms are the standard three and no others, and the kernel replay of all 3,091 modules has finished: 3,091 accepted, none rejected. Checked by machine.

1. Scan for escape hatches

Lean has a few ways to get a proof accepted without really proving it: the placeholder "sorry", newly declared axioms, native_decide (a command that trusts compiled code), unsafe code, and switches that turn kernel checking off. We searched all 3,977 Lean files in the algorithm's folder, with comments removed, for these and for many similar patterns. None occurs, apart from the single placeholder that closes the challenge file, which is how a challenge file is meant to look. Module names containing "Native" or "compiled", which an expert will notice, refer to the proof's own explicit counting program and its run, constructed inside Lean (we read the headers of two such modules); they are not Lean's native_decide, which trusts compiled code and would show up in the axiom list as an extra axiom about compiled evaluation, and the axiom list below does not contain it. The folder holds the 3,091 modules the final theorem depends on (the dependency list was computed from the imports, and every import resolves to a file in the checkout; the only outside imports are Mathlib). This scan is a search of text, not a proof: it cannot rule out a flaw in Lean's kernel, in Mathlib, or in a definition that makes the statement easy (chapter 1 looked for that last one). Checked by machine.

2. The build

We copied the proof files and the challenge file into a fresh Lean project on one Windows PC, with Lean 4.34.1 and Mathlib at the pinned revision (listed in the check guide), the two versions the repository pins, and built the final

module. The build finished successfully (12,014 jobs). It took about 9.5 hours of elapsed time, stopped and resumed three times along the way, so this is elapsed time and not a clean single-run figure. There is no placeholder or error text in the build output. Checked by machine.

3. The axioms

An axiom is a starting assumption Lean accepts without proof. Lean can list every axiom a theorem depends on. For the final theorem (named in the check guide), the list is exactly:

```
[propext, Classical.choice, Quot.sound]
```

These are the three that Lean's standard mathematics library uses, and they are the three the challenge file's configuration permits. There is no placeholder axiom. We also read the 19 axiom lists the proof prints for itself while building, including five at the end of the final file; all list only these three, and 19 is the number of such print commands in the 3,091 modules. For contrast, the challenge file on its own, before any proof is attached, lists the placeholder axiom, which is expected, since its placeholder is the blank to be filled. Checked by machine.

4. The kernel replay

A normal build checks a proof as it is compiled, and then stores it. A replay with leanchecker, a tool that ships with Lean, reads the stored proof of a module and sends every step back through the kernel. It exists because a stored proof could in principle differ from what was checked. We replayed one module at a time. The replay of all 3,091 modules has finished: every one was accepted (exit code 0), none rejected. Without its option for also replaying imports, leanchecker replays only the declarations of the module you name and trusts what that module imports, so the finished replay of all 3,091 modules does not by itself cover Mathlib or core Lean.

5. The second reader

After the earlier notes were written, a second reading within this project went back over every step of reasoning in them and either re-derived it in writing, cited a source that was opened, or marked it open. In plain words:

- Checked (re-derived): the order of the quantifiers; the polynomial size of the encoded formula; the step from the exponent to 2^{o(n)}; that the statement contradicts ETH, its counting version and SETH; that NSETH, Gap-ETH and the randomised versions fail if the algorithm is correct (one-line derivations, with the definitions taken from the manuscript); the machine-model argument; that counting also gives the yes-or-no answer; that "canonically encoded" hides nothing; that the algorithm cannot be combined with standard reductions to give subexponential algorithms for general NP problems; and the composite verdict of chapter 1. Chapter 1 and chapter 3 give the arguments.
- Checked by machine: the build, the axioms, the escape-hatch scan and the recalculations of the size bounds.
- Cited: the definitions of ETH and SETH (from Impagliazzo–Paturi, Impagliazzo–Paturi–Zane, Calabro–Impagliazzo–Paturi and Dell and others, all opened); what the comparator tool guarantees (its public description); the known lower bounds for weaker proof systems (chapter 3).
- Open: see chapter 4.

What this adds up to

A reader who repeats the build and the axiom check should see the same output. Together with the statement reading in chapter 1, this indicates that the machine-checkable part of OpenAI's claim holds up, so far, to every check we could run with public tools. It does not replace the review group's examination of the mathematics.

Check it yourself

VERIFY.md gives the commands, the expected output, and the rough time and memory.

Where to go next

- What do the checks leave unchecked? See What is still open.
- What does the algorithm do, in plain words? See What it means, and what it does not.

6 What it means, and what it does not

What it means, and what it does not

The question

What kind of algorithm is OpenAI claiming, how does it relate to the known limits on proofs of unsatisfiability, and what does it imply, if anything, for the polynomial-time question?

Why it matters

A claim that a famous hypothesis is false invites wider conclusions. Each should be tested against what is known. Sources are listed on the credits page, and where we rely on one paper's report of another's result the text says "as reported by". Everything below assumes the manuscript is correct, which we have not verified.

What answering it would help with

A reader may use this chapter to place the claim among related results, and researchers working on proof complexity may find which lower bounds the claim touches (none known).

What our research shows

The answer in one line: the claimed algorithm is an algebraic, deterministic method that is still far from polynomial time; it does not contradict the known lower bounds for weak proof systems, and the audit found no consequence for the polynomial-time question.

The algorithm in plain words

We read the manuscript's introduction, its technique overview and parts of its main sections; for some later sections and an appendix we read only the headings and the statements. In outline, as the manuscript describes it:

1. Make the formula tame. An older technique (the sparsification lemma of Impagliazzo, Paturi and Zane, in the counting form of Dell and others) turns the count into a sum over at most 2^{εn} simpler formulas in which each variable appears in only a few clauses.
2. Write the count as a sum of products. For such a formula the number of solutions equals a sum, over all 2^n settings, of a product of small polynomials, one per clause.
3. Split the sum into groups, with shared summaries. The variables are divided into groups. If a few "measurements" (linear summaries of each group) are fixed, each clause depends on only one group, so the big sum breaks into smaller sums that multiply independently. That alone does not yield a saving, so a parameter T is added to each clause input, and the sum is evaluated at several values of T. Interpolation (the same idea as recovering a curve from a few points) then gives back the true count at $T = 0$.

1. The hard part. The saving depends on replacing each variable by a low-degree polynomial form that needs few measurements. The manuscript builds it from a polynomial in commuting "swap" matrices, in the spirit of earlier work on matrix rigidity by Alman and Williams.
2. General formulas. For general k-CNF at a fixed density, the method is applied to sparse parity equations, using a polynomial of Beigel and Tarui that amplifies a modulus, and equations chosen deterministically from expander codes.
3. A sharper form. An appendix recasts the method as contracting Boolean tensors to reach $n/\sqrt{(\log n)}$ in the exponent when the number of clauses is at most a fixed multiple of n.

The manuscript calls the algorithm "algebraic, based on polynomial interpolation of carefully chosen auxiliary variables". It compares itself with Williams's split-and-list algorithm, Koivisto's sum-product formulation and

Scott and Sorkin's work. A text search of the manuscript finds no "random restriction" and no "probabilistic polynomial" in its body, though the Beigel–Tarui polynomial is used; that is a statement about the text, not a check of the mathematics. The algorithm is deterministic and uses polynomial memory.

How it relates to known limits on proofs

An algorithm that says "this formula has no solutions" produces, in effect, a long computation. A natural question is whether that computation is a proof of unsatisfiability in some standard proof system, and whether known lower bounds on proof size forbid that. Several are known for restricted systems:

- Resolution. Random 3-CNF formulas with a linear number of clauses, and Tseitin formulas on expander graphs, need exponential-size resolution proofs (Chvátal–Szemerédi 1988; Urquhart 1987, as reported by Ben-Sasson and Impagliazzo; Ben-Sasson and Wigderson relate size to width).
- Polynomial calculus (algebraic proofs): degree linear in n for random k-CNF over fields of characteristic greater than 2 (Ben-Sasson and Impagliazzo 2010), with the characteristic-2 case by Alekhnovich and Razborov (2003).
- Regular resolution (a restricted form): size lower bounds of the form $2^{(1-\varepsilon_k)n}$ for some k-CNF families (Beck and Impagliazzo, 2013). The manuscript says its algorithm "does not conflict with" lower bounds of this kind.
- Bounded-depth Frege (proofs in which every line has a bounded nesting of logical operations): exponential in a power of the size for the pigeonhole principle (Håstad, 2023), a different shape of bound from $2^{\Omega(n)}$.
- Frege and Extended Frege (strong, general proof systems): no superpolynomial lower bound is known; Hrubeš and Tzameret wrote around 2012 that the question "remains completely open". We did not check later literature.

There is no contradiction to resolve. A lower bound on proof size in one system conflicts with a fast algorithm only if the algorithm's runs can be turned into proofs in that system of comparable size, and the manuscript makes no such claim. A well-known example shows that no such conversion exists in general: Tseitin formulas are decided in polynomial time by ordinary linear algebra over a two-element field, yet resolution needs exponential size for them. Checked (logic), with the lower bounds cited. A size upper bound in a strong system also says nothing about proof size in a weaker system.

One open conjecture is worth stating. The algorithm's argument is arithmetic: sums over assignments, polynomial identities, exact linear algebra, interpolation. If its correctness could be formalised in a theory strong enough to reason about such arithmetic, its runs would give Extended Frege proofs of size about $2^{o(n)}$ for every unsatisfiable k-CNF. This is conjectured, not shown: nobody has formalised the correctness proof that way, and the known results on determinant identities (Hrubeš–Tzameret; Tzameret–Cook) are about other statements. If the conjecture held, any future proof of a $2^{\Omega(n)}$ lower bound on Extended Frege proofs of unsatisfiable k-CNFs would be false, but no such lower bound is known, so this would only mean that no such lower bound exists. This indicates a direction for research and no more.

The polynomial-time question and NP vs coNP

- The polynomial-time question. ETH implies that the polynomial-time question has a negative answer (if 3-SAT needs time 2^{cn}, it has no polynomial-time algorithm), so ETH was one possible route to a negative answer to the polynomial-time question. Refuting ETH removes that route; it does not answer the polynomial-time question either way. The claimed running time, $2^{O(n\,(\log\log n)^2\,/\,\sqrt{(\log n)})}$ in the Lean statement (the manuscript claims the sharper $n \log\log n\,/\,\sqrt{(\log n)}$ in general, and $n\,/\,\sqrt{(\log n)}$ when the number of clauses is at most a fixed multiple of n), is more than polynomial, so the algorithm does not give a positive answer to the polynomial-time question by itself. The manuscript says that bounds of the form $2^{o(n)}$ "do not

transfer under general polynomial-time reductions", and that its result does not show that general NP problems have subexponential algorithms. Our arithmetic agrees: the standard way to turn an arbitrary problem into 3-SAT (the Cook–Levin construction) makes a formula with between about T log T and T^2 variables for a computation of length T, and the claimed bound then exceeds the 2^N that brute force already takes on the original problem (N being its size), so it gives nothing subexponential. The manuscript derives no statement about the polynomial-time question, and we found none. Nobody can rule out an indirect implication nobody has found, and we assert only that none is stated or used.

- NP vs coNP. NP versus coNP asks whether the answer "no solution exists" can always be backed by a short proof. The manuscript says its algorithm refutes a hypothesis called NSETH, because unsatisfiability of a k-CNF can be decided deterministically in 2^{o(n)} time; anyone can re-run the algorithm to verify the answer. That is a long check, not a short certificate, and Cook and Reckhow showed that NP = coNP is equivalent to the existence of a proof system with polynomial-size proofs of all tautologies. So nothing follows about NP versus coNP.

- What the manuscript states as consequences. ETH, counting ETH, SETH, Gap-ETH, NSETH and the randomised versions are claimed refuted; there are subexponential algorithms for counting independent sets, proper q-colourings (q fixed) and perfect matchings (Corollary 1.3); randomised subexponential approximation of Max-k-SAT (Corollary 1.2); a circuit lower bound for a class called ENP (Corollary 1.4); and algorithms for inverting and counting preimages of maps in which every output bit depends on a bounded number of input bits (Appendix C.2). We re-derived the ETH, counting ETH, SETH, NSETH, Gap-ETH and randomised consequences from the Lean statement (chapter 1); the rest are cited from the manuscript and not re-derived.

Other problems that depend on ETH

The reason ETH matters beyond 3-SAT is that many other problems were shown to be as hard under ETH. Impagliazzo, Paturi and Zane (2001) proved, with their sparsification lemma, that 3-SAT converts in linear size into other NP-complete problems, so that under ETH problems such as 3-Colouring, Independent Set and Vertex Cover on N-vertex graphs need time $2^{\Omega(N)}$. The same conversions work in the other direction for the claimed algorithm: if the Lean statement is proved, a $2^{o(n)}$ algorithm for 3-SAT passes along those conversions, and these problems would also get $2^{o(N)}$ algorithms. The manuscript states this directly for counting independent sets and q-colourings. This is the standard consequence of sparsification; we cite it and did not re-derive it. Problems whose conversions from 3-SAT enlarge the instance more than linearly do not inherit the speed-up, which is why general NP problems are not covered.

Check it yourself

The manuscript is in the OpenAI repository at the pinned commit (in its preprints folder, under the title of the paper, dated October 25, 2026; the path is in the check guide). The passage on reductions is in Section 1, just before Corollary 1.3. The proof-system sources are listed on the credits page.

Where to go next

- Which parts of all this remain unchecked? See What is still open.
- Does any of this affect encryption? See Should I worry?.

7 What is still open

What is still open

The question

After the statement reading, the build, the axiom check and the kernel replay, what has not been checked, and what would settle each point?

Why it matters

Every check has limits. Naming them once, plainly, lets a reader see how much weight the machine checks can carry and where the experts' review has to do the work.

What answering it would help with

A reader who wants to extend the checks may pick any item below; a reader who only wants to judge the claim can see which parts rest on machines and which on the review.

What our research shows

The answer in one line: five items are open: a second-kernel run with the unpatched checker, the manuscript's mathematics, the manuscript's other claims, one conjecture about proof systems, and any indirect consequence for the polynomial-time question.

Machine checks not run

The kernel replay has finished: all 3,091 modules were replayed through Lean's kernel, and every one was accepted, none rejected (chapter 2). Two machine checks have not been run by us; others have run them (items 1 and 2).

1. The comparator tool. We have not run it ourselves. Busnello ran it on the same package and reports that it accepts the challenge's own declaration, with Lean's default kernel and only the three standard axioms; his record and logs are public (doi.org/10.5281/zenodo.23820111). The tool checks the theorem statement against the challenge file, so it settles the question our text comparison left open for the statement itself.
2. A second kernel. The repository's configuration leaves the second checker (nanoda) off. Toledo switched it on and reports that it accepted, using a patched build of nanoda with a larger working stack and about 80 GiB of memory (carlostoledo.co/eth). Another replication

with the unpatched checker ran out of stack and did not finish. We have not run either, and we have not read what the patch changes. A run with an unpatched second kernel would settle this item fully.

Mathematics not checked

1. The manuscript. The paper is 67 pages. We read the introduction, the technique overview and the opening parts of some sections; for Section 4, Section 6 and Appendix D we read only headings and statements. The Lean proof is the formalisation of the argument, so Lean accepting it is the strongest evidence available, but we have not re-derived the mathematics ourselves. The review group's examination is the process meant to do that.

1. The manuscript's other claims. The sharper exponents (n log log n / $\sqrt{}$(log n) in general and n / $\sqrt{}$(log n) for linearly many clauses), the approximation, counting and circuit corollaries, and the appendix on inverting maps with bounded output locality are not in the Lean statement. The Lean statement's exponent carries $(\ln \ln n)^2$, a factor ln ln n more than the manuscript's general bound n·log log n/$\sqrt{}$log n; the Lean form is enough for every consequence in this book, and the sharper form rests on the manuscript alone. O'Donnell's public exposition derives $(\log \log n)^2$ for formulas with linearly many clauses and $(\log \log n)^3$ for general formulas by a randomised route. We did not re-derive them or compute where the sharper exponents would beat brute force.

Implications not settled

1. The proof-system conjecture. Whether the algorithm's correctness can be formalised so that its runs give Extended Frege proofs of size about 2^{o(n)} (chapter 3) is untested.
2. Indirect consequences. The manuscript states no consequence for the polynomial-time question, and we found none, but nobody can rule out one that has not been found.

What would change this chapter

A published review by the review group, or a run of the second kernel with the unpatched checker, would each change the corresponding item. Anyone who runs the steps in VERIFY.md and reaches a different result from the one recorded there is asked to say so. Further research is recommended on all five open items.

Check it yourself

Items 1 and 2 can be run with public tools: the comparator's own instructions are in its repository (github.com/leanprover/comparator), and VERIFY.md gives the replay steps. Items 3 to 6 need an expert reader.

Where to go next

- What does the claim say, in plain words? See Should I worry?.
- Where did the statement table come from? See What the Lean statement says.

8 Statement

Statement

The question: does the Lean theorem that OpenAI published say what the Exponential Time Hypothesis (ETH) denies, and how far has its proof been checked? The answer: the statement is a faithful and slightly stronger form of "k-SAT and counting its solutions are solvable in time 2^{o(n)} times a polynomial, in polynomial space, for every fixed k of 2 or more" (checked, re-derived); the whole proof builds under Lean 4.34.1 and its axioms are the standard three and no others (checked by machine); a kernel replay of all 3,091 modules has finished (3,091 accepted, none rejected); the comparator tool and a second kernel have not been run, and the manuscript's mathematics has not been checked line by line (open).

Setup: OpenAI's repository at the pinned commit, Lean 4.34.1 and the pinned Mathlib version. The theorem is the main theorem of OpenAI's proof,

stated in the challenge file and proved in the final module of the proof. The exact names are:

```
Repository:     github.com/openai/math
Commit:         7ef5c0cc5ca4b0e072b7a050fca09ab24bc6fa3a
Mathlib:        d13f23b723b8a846827a245b89c10fc7d3f11612
Theorem:        OAI.Algorithms.SubexponentialSAT.SqrtLogSpace.exact_count_a
Stated in:      ComparatorChallenges/SubexponentialSAT.lean
Proved in:      OAI/Algorithms/SubexponentialSAT/SqrtLogSpace/Final.lean
```

The theorem as published

```
theorem exact_count_and_sat (k : ℕ) (hk : 2 ≤ k) :
    ∃ (machine : Computation.PrimitiveMachine)
      (countTape answerTape : Fin machine.tapeCount)
      (C A S : ℝ) (d e : ℕ),
      0 < C ∧ 0 < A ∧ 0 < S ∧ 0 < d ∧ 0 < e ∧
      ∀ (n : ℕ) (F : SAT.CNF (Fin n)), SAT.WidthAtMost k F →
        ∃ (time : ℕ) (final : LocatedCfg machine),
          Computation.Trace (locatedStep machine) time
            (locatedInit machine (Encoding.encodeCNF F)) final ∧
          final.config.tapes countTape =
            Turing.Tape.mk₁ ((countWord (SAT.count F)).map some) ∧
          final.config.tapes answerTape =
            Turing.Tape.mk₁ [some (decide (0 < SAT.count F))] ∧
          final.config.state = Sum.inl (decide (0 < SAT.count F)) ∧
          BinaryArithmetic.toNat (countWord (SAT.count F)) = SAT.count F ∧
          (countWord (SAT.count F)).length ≤ n + 1 ∧
          (time : ℝ) ≤ OAI.Algorithms.SubexponentialSAT.SqrtLog.Statement.:
            (Encoding.formulaBitLength F) n ∧
          ∀ (prefixSteps : ℕ) (intermediate : LocatedCfg machine),
            prefixSteps ≤ time →
            Computation.Trace (locatedStep machine) prefixSteps
              (locatedInit machine (Encoding.encodeCNF F)) intermediate →
            (visitedSpace intermediate : ℝ) ≤
              S * ((Encoding.formulaBitLength F : ℝ) + 1) ^ e
```

The text above is copied from the file in OpenAI's repository, which is released under the Apache 2.0 licence. In the challenge file the proof is replaced by the placeholder word "sorry", the normal shape of a challenge.

What each part says

Part of the statement	In plain words	Label
The choice of machine, tapes and constants, placed before "for every n and every formula"	One fixed machine and fixed constants for each width k, chosen before the size n and the formula	Checked
The run of the machine on the encoded formula	The machine's own run reaches a final configuration after exactly the stated number of steps	Checked
The count tape and the equation tying it to the count	The count tape holds the exact number of satisfying settings, in binary (least significant bit at the start)	Checked
The length bound on the count	The count fits in n + 1 bits	Checked
The answer tape and the halting state	The yes-or-no answer is on its own tape and is the halting state	Checked
The time bound	Time at most C·(L+1)^d·2^{A·n·(ln ln (n+16))²/√(ln (n+16))}, where L is the encoded length	Checked: the exponent is o(n) and L is polynomial in n for fixed k
The last line, on space	At every moment of the run, the total over all tapes of the width of the stretch each head has covered (the input tape counts as covered from the start) is at most S·(L+1)^e	Checked

The checks on the proof

Check	Result	Label
Scan of 3,977 Lean files for placeholders, new axioms, trusted compiled code and checking switches	none found outside the challenge file's own placeholder	Checked by machine
Build of the final module with Lean 4.34.1 and the pinned Mathlib version	completed successfully, 12,014 jobs	Checked by machine
Axioms of the final theorem	the three standard axioms and no others (listed in the check guide)	Checked by machine
Kernel replay with leanchecker, module by module	3,091 of 3,091 modules accepted, none rejected	Checked by machine
Comparator tool	run by Busnello: accepted (Lean's kernel)	Cited
Second kernel	run by Toledo with a patched nanoda: accepted; unpatched run not finished	Cited
The manuscript's mathematics	not checked line by line	Open
The manuscript's other claims (sharper exponents, corollaries, local-map appendix)	not re-derived	Open

Where to go next

- How was each line of the statement read? See What the Lean statement says.
- How do I repeat the checks? See Check it yourself.

9 Theorems in this book

Theorems in this book

This page sets out, in mathematical notation, the statement this book examines and the two small facts we derived about it. The main theorem and its proof are OpenAI's; we did not write them. Every label below says how far the statement has been checked. For the plain-words version, see What the Lean statement says. For the exact names to type into Lean, see the check guide.

Theorem 1: exact counting of k-SAT in subexponential time (OpenAI, 2026)

Credit: OpenAI (2026), stated and proved in Lean in their public repository. This is the claimed refutation of the Exponential Time Hypothesis. The review group (Impagliazzo, Paturi and Williams) has not finished its review.

For every integer $k \geq 2$ there exist:

- a deterministic multitape Turing machine M over the tape alphabet $\{\sqcup, 0, 1\}$;
- constants $C, A, S > 0$; and
- integers $d, e \geq 1$,

such that for every $n \geq 0$ and every formula F in conjunctive normal form on the variables $x_0, \ldots, x_{n-1}$ whose clauses each have at most k literals, the following holds. Write L for the length of the canonical binary encoding of F, and $\#F$ for the number of assignments in $\{0,1\}^n$ that satisfy F. Started on that encoding, M halts after some number T of steps, and:

1. one tape holds $\#F$ in binary, using at most $n + 1$ bits;

2. another tape, and the halting state, hold the answer $[\#F > 0]$;
3. the running time satisfies

$$T \;\le\; C\,(L+1)^{d}\; 2^{A\,n\,\frac{(\ln\ln(n+16))^{2}}{\sqrt{\ln(n+16)}}}\;;$$

4. at every step of the run, the total width of tape visited is at most $S\,(L+1)^{e}$.

Status:

- The whole proof (3,091 modules) builds under Lean 4.34.1, and its axioms are the standard three and no others. Checked by machine.
- A kernel replay with leanchecker of all 3,091 modules has finished: 3,091 accepted, none rejected. Checked by machine.
- The comparator tool and a second, independent kernel have not been run by us. The manuscript's mathematics has not been checked line by line. Open.
- The comparator tool has been run by others and accepts; a second kernel accepted in one run with a patched build. Cited, not repeated here.

The Lean name of this theorem:

```
OAI.Algorithms.SubexponentialSAT.SqrtLogSpace.exact_count_and_sat
```

Lemma 2: the encoding length is polynomial in n (this project)

Credit: Builds on: the encoding defined in OpenAI's Lean statement (2026). This result (Sonny): proved (written).

For fixed k, every formula F as in Theorem 1 satisfies

$$n+2 \;\le\; L \;\le\; n+2+(2n+1)^{k}\,(8kn+4k+4).$$

So the factor $(L+1)^{d}$ in Theorem 1 is bounded by a polynomial in n, and it cannot hide an exponential.

Status: checked (re-derived in writing from the Lean definitions of the encoding).

Corollary 3: Theorem 1 contradicts the Exponential Time Hypothesis (this project)

Credit: OpenAI (2026) state this consequence of their Theorem 1.1, for ETH and #ETH. ETH: Impagliazzo and Paturi (2001); #ETH: Dell, Husfeldt, Marx, Taslaman and Wahlén (2014). Written out in this project.

The Exponential Time Hypothesis says that there is a $\delta > 0$ such that 3-SAT has no algorithm running in time $2^{\delta n}$ on formulas with n variables. If Theorem 1 holds, then for every $\delta > 0$, k-SAT (in particular 3-SAT) is decided in time $2^{\delta n}$ for all large n, because

$$\lim_{n \to \infty} \frac{(\ln\ln(n+16))^2}{\sqrt{\ln(n+16)}} = 0$$

and, by Lemma 2, $(L+1)^d$ is bounded by a polynomial in n. The same argument applies to counting solutions (#ETH).

Status: checked (re-derived in writing). It depends on Theorem 1, whose status is above.

What this page does not contain

The scope is an audit of OpenAI's statement and an explanation of its k-SAT claim.

10 Check it yourself: OpenAI's claimed refutation of ETH

Check it yourself: OpenAI's claimed refutation of ETH

This guide is a reader's recipe for the machine checks in this book, using only public tools: git, elan (the Lean version manager), Lean 4.34.1, Lake (Lean's build tool), Mathlib and leanchecker (which ships with Lean). Write your own commands from the recipe. You can follow every step with public tools or code you write yourself. Each recipe also comes as a step-by-step page with

commands for Windows, macOS and Linux and the exact output to expect. Exact names, versions and paths are in the code blocks.

What is claimed

The question these checks answer: is OpenAI's Lean proof of its subexponential SAT theorem accepted by Lean, under which assumptions, and does the stored proof pass a kernel replay?

Check	Claim
Build	The main module of the proof (3,091 modules in its dependency list) builds under Lean 4.34.1 and the pinned Mathlib version
Axioms	The final theorem of the proof depends only on the three standard axioms listed under "Expected output"
Kernel replay	Each module passes leanchecker. On our machine all 3,091 modules were replayed: 3,091 accepted, none rejected

The names used below:

```
Repository:       https://github.com/openai/math
Pinned commit:    7ef5c0cc5ca4b0e072b7a050fca09ab24bc6fa3a
Lean toolchain:   leanprover/lean4:v4.34.1
Mathlib revision: d13f23b723b8a846827a245b89c10fc7d3f11612
Main module:      OAI.Algorithms.SubexponentialSAT.Main
Final theorem:    OAI.Algorithms.SubexponentialSAT.SqrtLogSpace.exact_count
Proof folder:     lean/OAI/Algorithms/SubexponentialSAT/
Challenge file:   lean/ComparatorChallenges/SubexponentialSAT.lean
Manuscript:       preprints/The-Exponential-Time-Hypothesis-is-False-Octobe
```

Status

Active · Preliminary. The build and the axiom report were produced on one Windows PC. The kernel replay of all 3,091 modules finished on our machine: 3,091 accepted, none rejected. We have not run the comparator tool or a second kernel; others have run them (see chapter 4). No reader outside this project has repeated the checks yet.

Commands

You need elan, git, about 16 GB of free memory and many hours of time (see the last section). The whole repository is large (the proof folder alone holds 3,977 Lean files), so a sparse checkout of the proof folder and the challenge file is enough.

1. Get the source at the pinned commit:

```
git clone --filter=blob:none --sparse https://github.com/openai/math.git
cd math
git sparse-checkout set lean/OAI/Algorithms/SubexponentialSAT lean/Compa
git checkout 7ef5c0cc5ca4b0e072b7a050fca09ab24bc6fa3a
```

1. Make a small Lake project. The repository's own project file also pulls in about forty unrelated libraries; the proof's imports (computed from the files) are only the proof folder and Mathlib, so a minimal project is enough. In a new project folder put: a toolchain file with the Lean toolchain line above; a Lake project file that requires Mathlib at the revision above, declares a library named OAI whose root is OAI, and sets the option autoImplicit to false (Lake's own documentation shows the format); and the copied proof folder, placed so that the main module is the file below.

```
OAI/Algorithms/SubexponentialSAT/Main.lean
```

1. Fetch Mathlib's prebuilt files, which saves many hours. If Lake reports that no manifest exists, an update command writes one when you fetch; check that it pins Mathlib at the revision above.

```
lake exe cache get
```

1. Build the main module:

```
lake build OAI.Algorithms.SubexponentialSAT.Main
```

1. Print the axioms. Save these two lines in a new Lean file and run it as in the second block:

```
import OAI.Algorithms.SubexponentialSAT.Main
#print axioms OAI.Algorithms.SubexponentialSAT.SqrtLogSpace.exact_count_
```

```
lake env lean Axioms.lean
```

1. Replay a module through the kernel. leanchecker takes one module name, and the module below is one example out of 3,091:

```
lake env leanchecker OAI.Algorithms.SubexponentialSAT.Algorithm
```

To replay all 3,091, run the same command once for every module of the proof, and record each exit code. The 3,091 are exactly the modules that have a compiled file (the kind Lean writes when it builds a module) under the folder below after step 4; the repository folder holds 3,977 Lean files, and about 900 of them are not part of this proof's dependencies. A module name is the file path under the OAI folder with slashes replaced by dots and the ending removed.

```
.lake/build/lib/lean/OAI
```

We have not run leanchecker with its option for also replaying everything a module imports on this proof, so we do not know its memory use.

1. Scan for escape hatches. Search every Lean file in the proof folder, after removing comments, for each of the words below. We found none outside the challenge file's own placeholder, which is the normal blank of a challenge file. (A text search is not a proof, and the module names containing "Native" are explained in chapter 2.)

```
  sorry   admit   axiom   native_decide   ofReduceBool
  implemented_by   extern   unsafe   skipKernelTC   debug.
```

1. Compare the statement with the proof. The Lean community's comparator tool (github.com/leanprover/comparator) is made for this: it checks that the proved theorem has the same statement as the challenge file. We have not run it. We compared the 33 definitions of the challenge file with the proof's as text, and read the eight that differed by eye; a reader who wants a stricter check can compare the compiled form of each definition in the two versions.

The repository's own route (the project file in its lean folder, then the cache fetch and a build) also exists. We did not use it, because it fetches the other libraries.

Expected output

The build ends with a line like this (the number of jobs depends on what the Mathlib download supplied; ours was 12,014):

```
Build completed successfully (12014 jobs).
```

During the build you will also see axiom lines printed by the proof itself (19 in all), each listing only the same three axioms. The axiom report from step 5 is:

```
'OAI.Algorithms.SubexponentialSAT.SqrtLogSpace.exact_count_and_sat' depends
 Classical.choice,
 Quot.sound]
```

If the list contains the placeholder axiom, or any axiom not in this list, the result differs from ours. The line breaks in the axiom report may differ on your machine. leanchecker prints nothing and exits with code 0 when it accepts a module, and prints a message and exits with code 1 when it rejects.

Warnings from Lake that the copied Mathlib or Batteries folders have "local changes" can appear when the project is copied between systems. On our copy both were at the revisions in the manifest, and the only differences were three non-Lean files that changed type from a link to a plain file.

Time and memory

On our machine the build took about 9.5 hours of elapsed time, with Mathlib already downloaded, and was stopped and resumed three times; we ran the heaviest modules on one thread. Plan for about 16 GB of free memory. While it ran, the replay went at about a hundred modules an hour on the same machine, one module at a time; later modules may be slower, so a full replay of all 3,091 modules is likely to need a day or more. These are rough figures from one machine.

What is not claimed

These checks show that the theorem, as written in Lean, is accepted under the three standard axioms, and that all 3,091 stored modules pass a kernel replay. They do not show that the Lean statement is the right one (that is the reading in the book), that the manuscript's mathematics is correct, or that a second kernel agrees; the comparator tool and a second kernel have not been run.

Supporting records are available on request.

We hope it helps others find more results and keeps their curiosity going. We'd love to collaborate with anyone.

11 Trust and verification

This chapter is the book's file VERIFY.md.

12 People and works

This chapter is the book's file CREDITS.md.

13 Files and rights

Licence

The files Sonny wrote for this book are under the Creative Commons Attribution 4.0 International licence (CC BY 4.0). You may copy, share and adapt them for any purpose, as long as you give credit.

How to give credit: "Sonny (Sonny), gocheckyourself.org", with a link to the page or file you used, and a note saying what you changed, if anything.

What it covers: the chapters, the statement pages, the check guide and the credits. The book contains no Lean files of its own and no code copied from Lean, Mathlib or the checkers.

What it does not cover:

- The quoted theorem statement from github.com/openai/math, which is released under the Apache 2.0 licence; the quotation is credited on the credits page.
- Lean, leanchecker and the comparator tool named in CREDITS.md, which have their own licences (Lean and leanchecker are Apache 2.0). None of their code ships in this book.
- Programs that are not part of this book, and work that is not published on this site.
- Trade marks, and the other rights that section 2(b) of the licence leaves out.

The licence file is LICENSE-CC-BY-4.0.txt; the full legal code is at https://creativecommons.org/licenses/by/4.0/legalcode.

Where to go next

- Which tools and texts does this book rely on, and under what licences? See Credits.
- How do you repeat the checks yourself? See Check it yourself.

14 Each result on its own page

1. Theorem 1: exact counting of k-SAT in subexponential time STATUS: LEAN-CHECKED
2. Lemma 2: the encoding length is polynomial in n STATUS: PROVED IN WRITING
3. Corollary 3: Theorem 1 contradicts the Exponential Time Hypothesis STATUS: PROVED IN WRITING

© 2026 Sonny. The books are under CC BY 4.0 (Creative Commons Attribution 4.0 International): reuse them with credit and a link. Each book's licence file lists the files that keep their own licences.